Applied Statistical Analysis in R
A graduate course for Psychology, Human Factors, and Data Science

Shane T. Mueller
shanem@mtu.edu
Michigan Technological University

November 13, 2024

Contents

Forward

This book is a collection of lecture notes and materials prepared for a graduate level statistics course for Human Factors/Cognitive Science students at Michigan Technological University. It is meant as a practical course in learning to do statistics using R, and will sometimes point to other texts for better background on statistical theory. One important text it makes substantial reference to is Julian Faraway's *Practical Regression in R*, which is available as a free download on many R websites. Other useful resources include John Fox and Sandford Weisberg's "An R Companion to Applied Regression", which cover a much of the same material as this book, and Robert Kabacoff's "R in Action".

This book is intended as a roadmap for teaching graduate students in psychology and social science, who have introductory-level statistics behind them, and will typically have experience using SPSS to conduct their statistical tests. The class is taught using R Commander as a basic interface, which is polished and provides a lot of nice features that students are able to take advantage of. The philosophy of this book recognizes that the actual statistical tests you run are usually at the tail-end of a lot of informal data processing, visualization, head-scratching, and data management. Thus, it is organized first to give a gentle introduction to the R syntax, data handling, and graphics abilities. This is where huge benefits can be gained, because frequently a task that would otherwise take days of copy-pasting in a spreadsheet can be done with a line or two of R code. The second third of the book covers a core set of statistical tests, focusing on regression as a generalization of ANOVA. The final third of the book includes appendix chapters contributed to by students in the class, and involve special topics that each might rightly take an entire course to explore fully.

Chapter 1

Introduction

Most books on statistics start with basic probability theory and simple hypothesis testing. In my experience, the actual statistical tests encompass only part (maybe less than half) of the work involved in designing an experiment and determining what happened. There are many other processing involved centered on handling data, aggregating, sorting, coding, etc., that are as important as the statistical tests.

Consequently, this book will start not with introductory statistics, but rather introduction to using R for data management, programming, graphing, and simulation. Once you are comfortable with its syntax and usage, learning statistical tests will be much more clear and intuitive. I recommend using R 3.0 or later, and RStudio, which provides a modern front-end.

1.1 Why R?

R is an open source statistical computing platform. It has become the de facto platform for statistical computing in many fields. It is free, and has very good support (from user groups, mailing lists, blogs, and published books, companies offering support, and example source code is abundant). Also, there are hundreds (maybe thousands) of free add-on packages available for doing almost any advanced statistical procedures. R is available at http://http://www.r-project.org/. R is based on a proprietary software system called *S+*, but it has essentially overtaken *S+* in its use and capability. However, should you want a commercial stats package, you can typically use *S+* with little learning costs.

The de facto standard stats package in the field of psychology is probably SPSS. SPSS is expensive, and must be licensed annually. Consequently, there is no guarantee that the work you do today using SPSS will be available to you in the future. Individual additional packages are more expensive, and aspects of the system's design can prevent you from learning about the statistical tests it supports. Furthermore, IBM (the latest owner of SPSS) appears has focused SPSS on being a 'Business Analytics' tool, whose goals and cost structure differ from what is useful in academic and applied research contexts.

In terms of features, *R* does not offer as simple of a menu interface for selecting statistical tests as does *SPSS*, and so in terms of its learning curve, *R* can be a bit more challenging. This is partly because it is so much more powerful, and partly because once you learn how to use command-line R, it is actually usually simpler and easier than a menuing system. Furthermore, R is unparalleled in its ability to manage and process data, the R graphics system is much more flexible and powerful than anything available elsewhere, and because

the source code is available, you can always determine exactly what steps are being taken. There was a time when open source stats software lagged behind what commercial products were producing, but the opposite is true today. Researchers will release their analytics software to R, with no guarantee that commercial tools will ever be released.

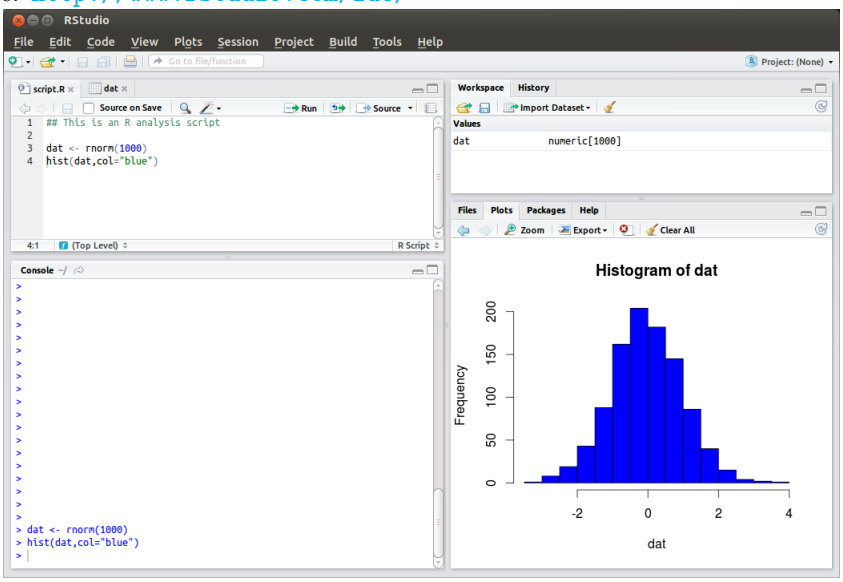
As you are learning, you will get the best benefit from diving into the software completely. So, while you are doing this course, I would encourage you to use R for for all of your data analyses. This investment will pay dividends for the rest of your career.

1.2 Installing R on your computer

To begin, you will need to install R on your computer, or have it done for you by your IT staff. R itself has its own (sort of clunky) front end, but a company called RStudio has created an open source graphical front-end that I recommend using. You may have to install both R and RStudio in order for them to work together. You may prefer a different front-end to R, such as emacs or the standard R GUI, but RStudio is probably the most advanced and professional front-end that exists. A few projects have attempted to create a menu-driven stats front-end with R akin to SPSS, but most seem to have limited usage and be a bit clunky.

1.2.1 Walkthrough of RStudio functions

A screenshot of RStudio is shown below. A video walkthrough of its functions is available here: http://www.rstudio.com/ide/



RStudio is referred to as an 'Integrated Development Environment (IDE)'. There are several parts important to data analysis it brings together:

1. *Command console*: The lower left, this will allow you to type commands into R that get executed. You can also see, copy, and paste from the history of commands you typed. This is the most important way of interacting with R.

2. *Script window* One tab on the upper left. This allows you to save sets of commands for later use.

3. *Data views.* Another tab of the upper left. Data objects (tabular data like spreadsheets) will be displayed here.
4. *Workspace* One tab on the upper right, this shows a listing of all the data objects you have created so far.
5. *History.* Another tab on the upper right, this shows your past history of executed commands.
6. *Plots.* On the lower right, graphics you create will be displayed here. From here, they can be exported into files you can insert in other documents.
7. *Files* A tab on the lower left; lets you look for files (data and .R) you have created
8. *Packages* A list of available packages, allowing you to load them into the workspace, and also access help documentation
9. *Help* Basic help search window.
10. *Project.* In the upper right, this allows you to save a set of files as an all-in-one project.

There are a few basic caveats:

- RStudio will save the current state of your analysis as a workspace when you exit. This might include a number of open source files, and a workspace containing specific objects. I recommend that you avoid relying on this. You should be sure that when doing an analysis, every step is recorded in a .R file, so that it could be re-created from scratch.
- Unlike SPSS, you can have many data tables loaded at the same time. This data is listed in the upper right. This means that you generally need to tell an R command which data object you are interested in using for an analysis.
- Along this line, a number of menu options are available that will make it easier to do things like visualize and input data from files. This can be handy, but it will always create a command that gets copied to the command-line window. I recommend copying that line and saving it in a text .R file for later use.
- R looks for and saves files in a so-called 'working directory'. You should be diligent about setting this (under the 'session' menu), to help manage your analysis projects. For example, each project might have its own directory. This will make it easier to share, archive, and understand if you ever come back to the project.
- Hitting ctrl-enter on a line in an R script will automatically copy that line over to the console window. Hitting ctrl-enter when you have selected a range of lines will copy and execute the entire selection. ctrl-alt-R will run the entire text file using the `source` command.
- R is case-sensitive. That means that `Data` is different from `data`. Furthermore, function names can be identical to data names, so you might have a function and a data structure called `plot`. This usually does not cause trouble. Finally, it is typical practice to use the '.' inside variable names to indicate related information, such as `data.mean` and `data.sd`. This is because the hyphen character '-' is the minus operation, and the underscore character '_' was traditionally overloaded as an assignment operator.

1.3 Getting Started

On the lower left is the command window or console. This command window will wait for you to type a command, and then execute it. It also allows you to examine data. It is really useful for exploratory data analysis, and also to incrementally process data, allowing you to do multiple-step operations and examine the intermediate results.

As a simple first step, the following command displays many different built-in graphics capabilities:

```
demo(graphics)
```

Here, `demo()` is a function, which takes the command called `graphics`. Other options are available for demo as well, which you can see if you type `demo()` into the console. After typing `demo(graphics)` and hitting enter, watch the results in the 'Plots' tab as you press enter on the console. For each graph, a set of commands was automatically copied into the console and executed. Take a look at the commands needed to create each graph. When you are done, use the arrow buttons on the plots tab to scroll back through graphics. Try the 'zoom' button, and then export the graphic to an image. Try also copying it to your clipboard and pasting it into a word processing document.

You can get details about any function by typing ? or `help()`

```
help(plot)
?barplot
```

1.3.1 Simple Math Calculations

R supports doing simple and complicated math, and can be useful as a calculator. Some examples follow:

```
 3 + 3
[1] 6
 9 * 6*3.45
[1] 186.3
 9 + 6*3.45
[1] 29.7
 (9 + 6)*3.45
[1] 51.75
 (9 + 6)^3.45
[1] 11416.02
 (9 + 6)^3.45/33.0
[1] 345.9401
 (9 + 6)^3.45/0
[1] Inf
```

Notice that when we try to divide by 0, we get the number `Inf`, which is code for 'infinity'.

Exercises 1.3.1

- Compute your height in inches, by multiplying your height in feet by 12 and adding the remainder
- Compute your height in meters, using the identity that 1 inch = .0254 meters.
- Compute the average height in meters of at least three people (one who is 5 foot 2, one who is 6 feet 5 inches, one who is 4 feet 8.5 inches).

1.3.2 Numbers and vectors

So far, we have looked at numbers–either integers or real (floating-point) numbers. When doing data analysis, we usually have a series of numbers–which are referred to as a vector. This might be a set of values you have collected; maybe the speed of processing for 100 different people, or their ages, or heights. R actually treats all numbers as vectors of numbers–a single number like we calculated above is just a vector of length 1. This is why output of numbers is preceded by a bracketed [1], like this:

```
 3+1
[1] 4
```

The [1] indicates that 4 is the first element of the vector. If the vector gets longer or more complex, other values will be printed to help you know where the values are. There are many ways vectors are produced. You can create a vector using the `c` function (c is for combine):

```
 c(1,3,5)
[1] 1 3 5
```

This again tells you that the 1 is element 1. You can make a sequence with the colon (:) character:

```
 1:20
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20
```

Here, each printed-out line starts with the element of the vector that the next value starts at. This is just for printing purposes, and does not impact anything else.

1.3.3 Your first graphics

We can easily create random numbers and plot them with a set of graphing functions.

```
par(mfrow=c(1,3))
x <- runif(100)
plot(x)
barplot(x)
hist(x)
```

The commands above do the following: This sets up the display window so it will plot 1 row with three columns. Changing the first or second number can change how many plots are on a single page.

```
par(mfrow=c(1,3))
```

This command does two things. The right-hand part creates a vector with 100 numbers randomly chosen between 0.0 and 1.0. The `x<-` part assign this vector to a named object you can access later. R traditionally used the two-symbol assignment operator `<-`, because the alternative `=` can be easily confused with the test for equality (`==`). However, the `=` will also work for assigning values to variables.

```
x <- runif(100)
```

Now, we can create three plots to examine the data, all on the same page. The first just plots the 100 numbers in sequence. The second is similar, but plots the numbers as a bar-graph. The third one creates a histogram of the numbers, to help see its distribution. Notice that although they were sampled from a uniform distribution, the histogram is probably not exactly flat.

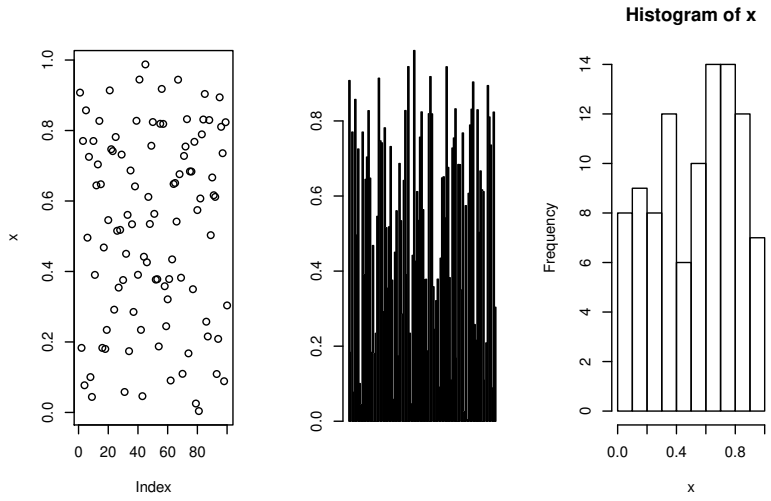
```
plot(x)
barplot(x)
hist(x)
```

Now, lets suppose we want to create a data structure using the one we just created. We can use the object name in arithmetic just like we did earlier with numbers. R will apply those operations to entire objects, not just single numbers. Notice how below, we can multiple a vector of numbers by .2, and each will be multiplied by .2, or add 3 and 3 will be added to each value. But we can add a vector of 100 numbers to another vector of 100 numbers too. Finally, we will use the `cor()` function to compute the correlation between the two vectors, which will be high because one is composed of the other.

```
 tmp <- runif(100)*.2
 tmp2 <- tmp + 3
 y <- x + tmp2
plot(x,y)
```

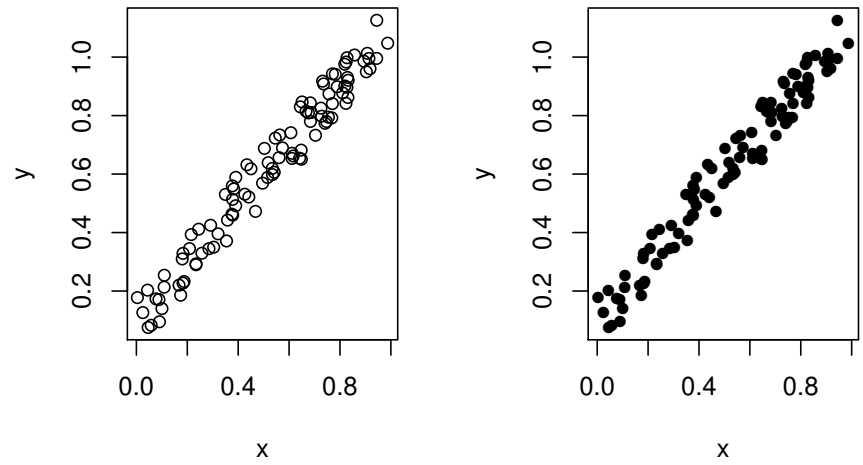
1.3.4 Functions and Function Arguments

Previously, we used functions that took a single argument. Many take multiple arguments, which can change their behavior. For example, the `plot` function used previously will create



a scatter plot when given two arguments. When given just one argument, it will be plotted in order along the x axis and the values along the y axis. When given tow arguments, the first argument will be the x value, and the second argument will be the y value. See the output on the left panel of the figure.

```
plot(x,y)
```



Functions also often return a value or object. For example, if we useth *cor* function, it will compute the correlation between two sets of numbers, and return the value to the command line, printing it out along the way. Correlation is a scaled measure of how two variables covary, ranging from -1 (meaning perfectly negatively associated), to 0 (unassociated) to 1 (perfectly positively associated).

```
1 cor(x,y)
[1] 0.979622
```

If we wanted to save that value for later, we could have assigned it to a variable name:

```
 xycorr <- cor(x,y)
2 xycorr
[1] 0.979622
```

Exercises 1.3.3

- Create `z1`, a set of 100 random numbers like we did above, but make it less highly correlated with x (try to aim for a correlation of .8).
- Create a `z2`, a set of 100 random numbers, but make them negatively correlated with x.
- Compute the correlation between `z1` and `z2`

Most functions take multiple arguments that allow you to control their operation in more detail. They will typically have default values for most of the (optional) arguments. These values are either specified in the order they were defined in, or by writing the argument name in the function execution. Let's say we want to change the symbol used to plot data. This is done using the `pch` argument, which takes an integer between 0 and 32, controlling the symbol used. We can specify the pch we want as below; the results are shown in the right panel of the previous figure. Notice how the circles are filled, rather than empty.

```
1 plot(x,y,pch=16)
```

1.4 Data arrays, frames, and matrices

The above `x` and `y` are arrays or vectors—they represent a sequence of data.

A 2-d or higher matrix can be formed using the `matrix` function. The elements of a matrix must all have the same types of data (which we will discuss next). Thus, it is usually not appropriate for holding tabular data like a spreadsheet, because you often want to mix data types together, with one column being nominal, another being text, a third being numeric, and so on.

The following creates a 10x10 matrix of random numbers, then prints out a rounded-off version of this.

```
1 m <- matrix(runif(100),10,10)
print(round(m,3))
```

1.4.1 Exercise

Exercise 1.4.1

- Create a 10x20 matrix of uniform random numbers.

Most of the time, your data will be in what is called a data frame. This allows putting together a set of tabular data, where different columns may have different data types (text, factors, numbers, etc.). Vectors such as `m` can be transformed into a data frame easily.

```
n <- as.data.frame(m)
```

Unlike a matrix, the columns of a data frame are named. They are given default names if none are provided. Like a matrix, data frames must be 'rectangular'; each column must have an equal number of rows, and each row must have an equal number of columns.

```
print(n)
```

You can examine the data frame n in the data view in RStudio by clicking on it in the workspace. Or you can type the following (note the capital V):

```
View(n)
```

1.5 Accessing sub-elements

Accessing a sub-element of a vector is simple, using the `[]` operator to tell it which element or elements (by number) you want.

```
print(x[20])
```

An entire row or column of a matrix can be accessed too:

```
print(m[1,])   ##get first row
print(m[,3])   ##get third column
print(m[1,3])  ##get a specific element
```

In a Data frame, columns can be accessed by name

```
print(n$V2)
```

or by index:

```
print(n[,2])
```

1.5.1 Accessing elements by name

Elements of a vector (and the rows/columns of a matrix) can be named. This may seem sort of overkill, but it has some nice uses, because elements can be referred to by name instead of position. It makes for an easy lookup table too. For example, let's create a vector with the phonetic alphabet, but name them with their corresponding letter:

```
1
lookup <- c(A = "Alfa", B = "Bravo",
3             C  = "Charlie", D = "Delta")
> lookup
5    A          C          B          D
   "Alfa" "Charlie"   "Bravo"   "Delta"
7
> lookup["B"]
9
B
11 "Bravo"

13 > lookup[2]
B
15 "Bravo"
```

Notice that it returns the labeled responses. The bottom is the actual value, and the top shows what the label was. `lookup[2]` is the same as `lookup["B"]`.

This allows us to do recoding easily. A useful application might be transforming Likert-scale responses in a survey to more uniform values 1 through 7.

```
lookup[c("A","A","C","B")]
2            A          A          C          B          D
       "Alfa"    "Alfa" "Charlie"   "Bravo"   "Delta"
```

1.5.2 Naming columns of a data frame

Similarly, columns of a data frame are accessed by name, but we can add new named columns using the `$` symbol. New data columns can be added to a data frame (but not as easily to a matrix):

```
1 n$new <- 43 #This puts 43 in each row
print(n)
3 n$new2 <- c(44,45) #this recycles the shorter list to fill
print(n)
5 n$new3 <- 1:10
```

1.6 Data types

There are a number of data types you will run into. You can tell what type you are dealing with by using the `typeof()` function.

Integers

These are positive and negative whole numbers. They are only used in a few specific situations in which they are really used in R, but they can save storage room and sometimes be faster to do math on.

```
c(1,3,3,44,5)
typeof(5)
typeof(as.integer(5))
```

numeric/floating point

Most numbers used in R are floating point values that are referred to as a `"double"`. Even integer values are usually stored as double-precision floating point values.

```
33.0
1.223
c(1,3,3,44.3,5)
typeof(33.0)
```

Character strings:

```
bb <- "HELLO"
c("one","two","three")
typeof("H")
```

Logical

Logical values are TRUE and FALSE (T and F). T is actually equal to 1, and F is equal to 0,so adding 1 to T produces 2.

```
a <- c(TRUE,FALSE)
typeof(T)
[1] "logical"
T + 1
[1] 2
```

Factors (nominal-scale values)

When you have categorical data, these will often be stored as factors. A factor can often masquerade as either a numeric or a character string. There are times, especially for some statistical analyses, when using a character instead of a factor to specify a factor will produce different results. So, be careful!

```
 factor1 <- as.factor(c("greg","jan","marsha","bobby",
   "tom","cindy","peter"))
[1] greg   jan    marsha bobby  tom    cindy  peter
Levels: bobby cindy greg jan marsha peter tom
```

Note that the original order is preserved, but there is an implied numeric order which is their alphabetic order. Thus, bobby is level 1, and is equal in value to 1).

Coercing between types

You will sometimes need to change one type to another type. This happens frequently with factors and characters. There are a set of functions that permit transforming between types, just like the `as.factor` function used previously.

For example, the names of factors do not have to be characters, they can be numbers. This can be confusing, especially when the values do not map directly onto their integer values they represent:

```
bb <- as.factor(c(5.5,1,2,3.2,3.3))
 bb
[1] 5.5 1   2   3.2 3.3
Levels: 1 2 3.2 3.3 5.5
```

Notice how the levels are sorted in numeric (actually alphabetic) order. Here, "1" is equal to 1, but "3.3" is equal to 4. What really happened is that the numbers got transformed into character labels, and these got used to label the levels of the factor. Suppose we wanted to get the character values back from the factor. This is pretty simple:

```
 as.character(bb)
[1] "5.5" "1"   "2"   "3.2" "3.3"
```

But to get the numeric value can be tricky. The following won't work:

```
 as.numeric(bb)
[1] 5 1 2 3 4
```

This fails because the factor values get converted to their numeric equivalent, in terms of the alphabetically-sorted factor names. Instead, you need to do:

```
 as.numeric(as.character(bb))
5.5 1.0 2.0 3.2 3.3
```

Sometimes, you have a variable that is a number, but you want to treat it like a factor. For example, suppose you have three different doses of a drug: 0, 50, or 2000 mg. You know that the drug has a non-constant effect–maybe the first 50 mg has the greatest impact, and it then levels off, but by 2000 mg the immune system reacts to the drug and produces negative effects. If you wanted to use this in an analysis, you might not want to try treat this as a categorical variable with three levels, rather than a numeric variable.

```
doses <- c(0,2000,50,2000,50,2000,0,50,50)
dosefactor <- as.factor(doses)
benefit <- c(5,6,20,3,18,2,4,22,17)
dosefactor
 dosefactor
[1] 0    2000 50   2000 50   2000 0    50   50
Levels: 0 50 2000
```

Notice that the levels of the factor appear in numerical order, which is handy.

1.7 Filtering and Selecting or Removing Data Points

If you have slightly complicated data set, you will need to sort and filter data. You might want to remove outliers, or select one condition, or compare values graphically. Let's suppose that you want to compare low and high values of x, and display these subsets graphically. graphically. The low/high split might be at 0.5. So, first, make a filtering variable. This will be TRUE or FALSE depending on the value of x (output is edited for space)

```
set.seed(1111)
x <- rnorm(100)
y <- x + runif(100,-.3,.3)
tmp <- (x>.5)
tmp
  [1] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  ...
 [18] FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE  ...
 [35] FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  ...
 [52] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  ...
 [69] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  ...
 [86] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  ...
```

TRUE and FALSE values are essentially the same as 0/1. A data vector can be filtered in two ways: either by using an equal-length vector of T/F values, or by specifying a integer-vector specifying which elements to select. Here, we will use the first method. Because tmp is a T/F vector, using the [] subset operator will pull out just the elements for which tmp is TRUE:

```
 x[tmp]
 [1] 1.3225244 0.6397020 1.1747866 0.6775081 1.1177719 1.3840475 1.2839409
 [8] 0.9908363 0.8639047 1.4834263 0.6776003 1.5275207 2.2391646 1.7539743
[15] 1.2205099 2.6151287 1.7615008 0.5105179 0.9630118 2.2282912 0.5919149
[22] 0.5851085 1.4398071 2.2851474 1.3073765 2.0673478 0.5210748 1.3947472
[29] 0.5897699 0.8865532 0.7917236 0.6321438 2.0379155 0.8297592 1.5149127
[36] 1.7132959 0.9951573 2.5028453 2.9734652
```

Notice how it picks out all the numbers greater than .5. It does this in their original order. We can use this to create plots that pick out certain values. Here, I will pick out those that are $> .5$, and plot them in one color, then pick out the ones less than .5, and plot them in another color: We could get the values less than .5 by negating tmp with the `!` character, which means 'not':

```
 x[!tmp]
 [1] -0.086580111  0.116290309 -2.930846364 -0.976479813 -1.534281566
```

```
 [6] -1.817001602  0.355633083 -0.080512606 -0.962480325  0.112310964
[11] -0.257651852 -0.718621221 -0.547872283 -0.039941426  0.384087275
[16] -0.786009593 -0.176356978  0.094705820  0.127134851 -1.248214152
[21]  0.307956642 -0.541231591  0.295591914 -1.551084024 -0.679109184
[26] -1.136255216 -1.200430574 -1.753429287  0.159120433 -0.705527410
[31] -0.591079069 -0.279641026 -1.320978222  0.019832320 -0.553447159
[36] -0.175629520 -1.105246204 -0.437523927 -0.655892874  0.387796710
[41] -0.504953477 -0.055572118 -0.641058315  0.285135127 -0.324987285
[46]  0.382917618 -0.471192741  0.369884879  0.200547942 -0.486043023
[51] -0.516570926 -0.196642334 -0.242465523  0.466707755 -1.337070225
[56] -1.517031873 -0.100654867 -0.008265104 -0.216710897  0.216307660
[61] -0.758359383

```

This subsetting trick is incredibly useful for comparing groups, plotting in different colors, and so on. A simple example, shown on the left side of the figure below:

```
plot(x[tmp],y[tmp],col="darkgreen",xlim=c(0,1),ylim=c(0,1))
points(x[!tmp],y[!tmp],col="red",pch=16)
```

Also, R uses a graphics drawing method where you add multiple layers onto a graph in order to add additional detail. Notice that each consecutive command has an additional effect on the figure. This can be used to layer fairly complex data visualizations.

```
plot(x[tmp],y[tmp],col="darkgreen",xlim=c(0,1),ylim=c(0,1),
     xaxt="n",yaxt="n",xlab="",ylab="")
points(x[!tmp],y[!tmp],col="red",pch=16)
axis(1,0:10/10)
axis(2,0:10/10,las=1)
abline(0,1,lty=1)
abline(.5,0,lty=3)
abline(v=.5,lty=3)
```

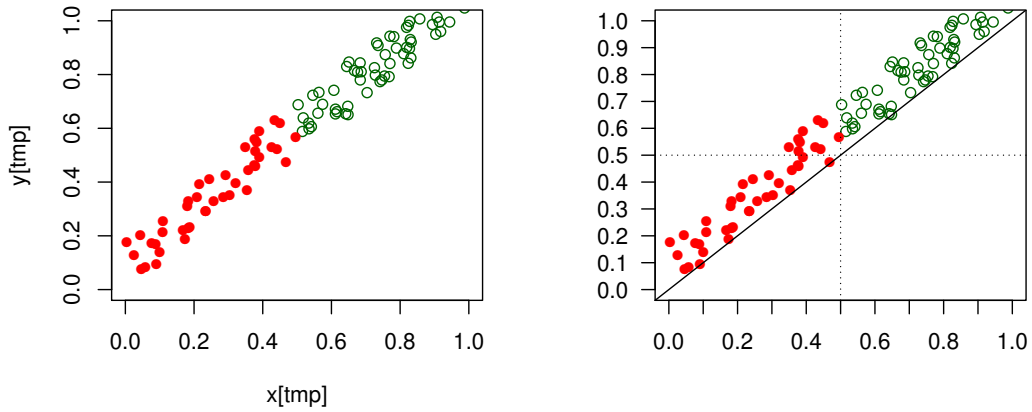
Here, we tell the `plot` function that we don't want it plotting the x and y axes (using `xaxt="n"` and `yaxt="n"`), and also set its x and y labels to "". Then, we add individual axes ourselves. Finally, we use the `abline` to draw lines (the first argument specifies intercept, the second slope, and `v` indicates a vertical line at point x).

We can also use a numerical vector to pull out indices of another vector. For example, just as `x[20]` picked out the 20th value, `x[c(20,21,30)]` picks out a sequence that includes the 20th, 21st, and 30th values. We can use this trick to make the above plot easier. Note that FALSE+0 = 0, and TRUE+0=1. If we executed the following command:

```
c("red","darkgreen")[tmp+1]
```

We get a vector of color names that depend on the value. When plot is given a single color, it will use that color for everything it plots. But if it is given a set of colors the same length as the data, it will set each color individually (the same is true for other graphical parameters such as plot character`pch`, size `cex`, and the like). So, to set the colors more easily:

```
plot(x,y,pch=18,col=c("red","darkgreen")[tmp+1])
```



Maybe we want the symbol dependent on the y value. Use the same trick for `pch`, which controls the plot symbol.

```
plot(x,y,col=c("red","darkgreen")[tmp+1],
     pch=c(1,16)[1+(y>.5)])
```

Exercise 1.7

- The `cex` argument sets the size of the symbol. Use the distance from the origin to impact the size. Choose two distance thresholds. If a point is closer to (0,0) than the smaller threshold, plot it in a small point size. If the point is greater than the larger threshold, plot it in a larger size. If it is between the two thresholds, plot it in a normal size.
- The `points` and `lines` commands will overplot lines or points on an existing plot, whereas `rnorm` acts like `runif` but for the normal distribution. Use this to first plot 500 random normal numbers having mean 0 and sd 1, and then plot any values that are greater than 2 or less than -2 in red. Finally, draw horizontal lines at + and - 2.0, showing this outlier threshold. You can use the `lines` command or the `segments` command for this.

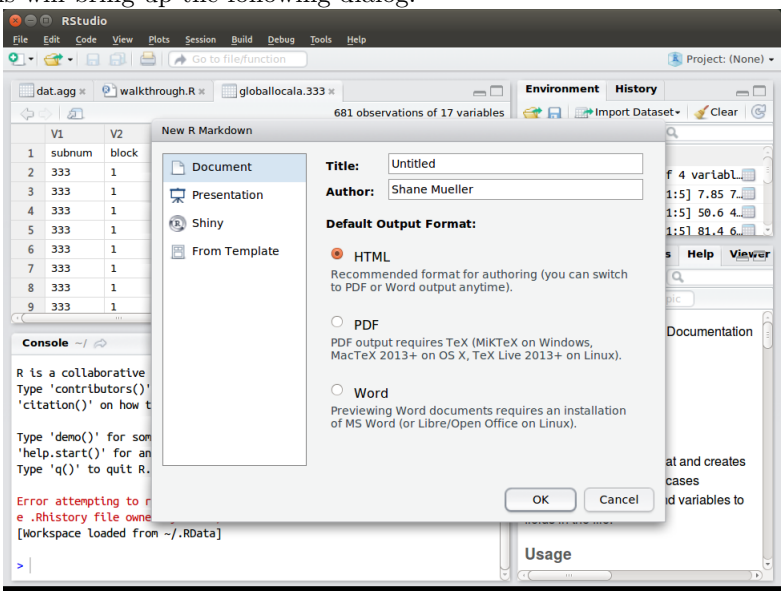
1.8 Report Generation in RStudio

R provides a number of methods for automatically generating high-quality reports from data. The simplest ways are built-in to RStudio, using something called RMarkdown. This section

will cover the basics, but RMarkdown is rapidly improving, so you are likely to get better tutorials on using it elsewhere.

- https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

You begin by creating a new document from the File—New File— R Markdown menu. This will bring up the following dialog:



There are ways to create reports for the web (html), for dynamic web pages (shiny), for presentations, as a pdf, as a word document, and others. Here, select Document and Word, put your name in the author list and edit the title (this can mostly be change after the template is created as well.) By choosing 'word' format, you can go in and edit the report to make whatever tweaks or additions you will eventually need.

It will create a template document that has some example markdown in it already. It starts with the following text:

```
---
title: "Homework Set 1"
author: "Shane Mueller"
date: "08/31/2016"
output: word_document
---
```

After the three dashed line, any text you write will eventually appear as normal text in the report. IF you want headers,italic fontface, lists, and minimal formatting, there are special ways of writing in this text section that will produce it. It is similar to how wikipedia markdown works. For example, headers can be indicated with a line starting with the # symbol, with subheaders multiple ## in a row. A quick reference guide for this markdown is embedded within RStudio, under the Help—Markdown Quick Reference menu option.

One important markdown you will want to use is for R code. If you use three left-quote symbols in a row, this delimits a code block:

```
```
  a <- 1:10
```
```

```
  b <- a + runif(1)
  cor(a,b)
```

In the above example, the lines will appear exactly as you type them, and will display in a fixed-font to demonstrate that it is code. However, an even more powerful markdown will actually execute the code you write. Here, follow the quotes with {r};

```
```{r}
  a <- 1:10
  b <- a + runif(10)
  cor(a,b)
```
```

This will tell the processor to execute the code in the section and send the results to the report file. There are other options available, like suppressing the code completely

```
```{r echo=FALSE}
  a <- 1:10
  b <- a + runif(10)
  cor(a,b)
```
```

This would be useful for a true report you are generating for a client or employer who does not want to see code.

This can also be used to create images. Each back-quoted environment can specify the size of the figures you want to create:

```
```{r echo=FALSE,}
  a <- 1:10
  b <- a + runif(10)
  cor(a,b)
```
```

Exercise 1.8

- Create an R Markdown document. In the document, incorporate the following: section header, section subheader, numeric bulleted lists, italic and bold text, R code that calculates the mean of four numbers, and R code that make a barplot of those four numbers.

1.9 File Management

There are a number of practices you can use to make file management easier. Here are some I have found useful:

- Make a folder or subdirectory containing everything related to a project. You can put your data in a subdirectory called *data*, and other materials there as well. Analysis files can be placed there, output files and figures will be generated there, and R project files will be saved there. Then, you can access the entire project easily, even if you move to a new computer or share your project with someone else.
- Be sure to set the working directory to your R analysis file location so it can find the data.
- Avoid hard-coding file locations on your computer. If you share the analysis folder with someone else, it will not work.
- Even if you use a markdown file, you might want to put functions you create in a library file and load them using the `source` function.
- Try to make any changes to your data through well-documented R file, rather than by hand via excel. You should always be able to redo all of the steps from raw data to analysis, without relying on intermediate hand-copying or sorting.

1.10 Summary

This chapter gave a basic introduction to R, as well as some of its basic plotting and filtering abilities. Many of these will become second-nature in later chapters, but be sure you both understand these functions, and can use them yourself in new contexts.

1.11 Solutions to Exercises

Exercises 1.3.1 Solution

Compute your height in inches, by multiplying your height in feet by 12 and adding the remainder.
If a were 7 feet, 2.5 inches, I would type:

```
> 7 * 12 + 2.5
86.5
```

Compute your height in meters, using the identity that 1 inch = .0254 meters. This is just multiplying the height computed above by the scaling factor.

```
>86.5 * .0245
2.11925
```

Compute the average height in meters of at least three people, (one who is 5 foot 2, one who is 6 feet 5 inches, one who is 4 feet 8.5 inches).

```
> .0245 * ((5*12+2) + (6*12+5) + (4*12+8.5))/3
1.596
```

Exercises 1.3.3 Solution

*Create **z1**, a set of 100 random numbers like we did above, but make it less correlated highly with x.*

```
 x <- runif(100)
 z1 <- runif(100) * .5 + x
 cor(z1, x)
[1] 0.8807643
```

This was too highly correlated. Let's try this:

```
> x <- runif(100)
> z1 <- runif(100)  + 1.4 * x
> cor(z1, x)
[1] 0.804588
```

That's better. I could have gotten here either by increasing the weighting of x, or decreasing the weighting of the random numbers.
*Create **z2**, a set of 100 random numbers, but make them negatively correlated with x.*
Notice that this does not work:

```
z2 <- x -runif(100) * .5
> cor(z2, x)
[1] 0.9133155
```

But this does:

```
z2 <- runif(100) * .5 - x
> cor(z1, x)
[1] -0.8807643
```

The difference here is that the new values must have a negative component of the original values.
*Compute the correlation between **z1** and **z2**.*

```
> cor(z1,z2)
[1] -0.8090315
```

Notice that although these were not directly created based on one another, they were both created to be related to a third variable (x) but in opposite directions. Thus, they themselves will be correlated negatively. This is like how many spurious correlations occur (wearing certain clothing may correlate with being a victim of crime if crime goes up when the weather is hot, and people wear different clothing when the weather is hot). This could lead to things like victim-blaming even if the cause is elsewhere.

Exercise 1.4.1 Solution

Create a 10x20 matrix of uniform random numbers.

```
x <- matrix(runif(10*20),10,20)
```

Exercises 1.7 Solution

The cex argument sets the size of the symbol. Use the distance from the origin to impact the size.
The distance to the origin (0,0) can be computed using the Pythagoras theorem:

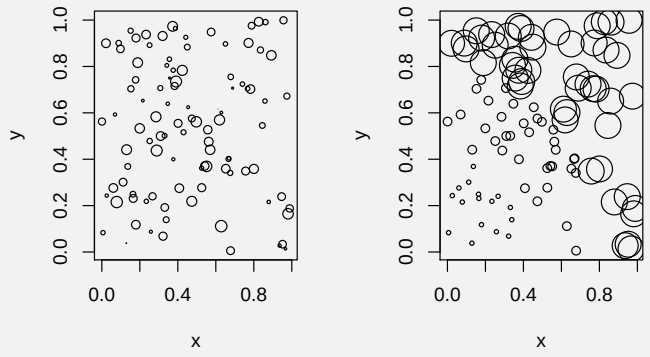
```
x <- runif(100)
y <- runif(100)
distance <- sqrt(x ^2 + y^2)
```

We could use `dist` directly as the cex argument because the distance will always be less than sqrt(2):

```
plot(x,y,cex=distance)
```

However, we want to do this with two thresholds. The simplest way to do that is to create a vector if three cex sizes to use: `c(.5,1,3)`, and then a vector to select which ones.

```
choosepoint1 <- (distance > .4)
choosepoint2 <- (distance > .8)
choosepoint <- choosepoint1 + choosepoint2 + 1
plot(x,y,cex=c(.5,1,3)[choosepoint])
```



Exercises 1.7 Solution cont.

Use the functions plot, points, and lines to create 500 random normal, and highlight the outliers more than 2.5 units from the mean.
First, lets create the numbers, plus an 'index' vector that we will use later.

```
nums <- rnorm(500)
index <- 1:500
```

Now, plot the numbers with open circles:

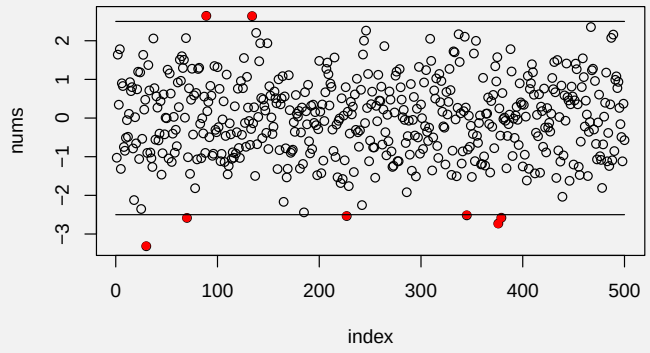
```
plot(index,nums)
```

Now, create a filter like we have done before, and overplot the ones we care about. Here, we filter both the index and the numbers with the filter. Also pch=16 plots in a solid point.

```
filter <- abs(nums)>2.5
points(index[filter],nums[filter],col="red",pch=16)
```

Finally:

```
segments(0,2.5,500,2.5)
segments(0,-2.5,500,-2.5)
```



Exercise 1.8 Solution

Create an R Markdown document. In the document, incorporate the following: section header, section subheader, numeric bulleted lists, italic and bold text, R code that calculates the mean of four numbers, and R code that make a barplot of those four numbers.

The following file should address each of these.

````
---
title: "R Markdown Example"
author: "Shane Mueller"
date: "08/31/2016"
output: word_document
---
# Top-level header

### Subsection title
This is is normal text.  You can also do:

 * **bold**
 * *italic*

Or:
 1. First
 2. Second
 3. Third

## Display and execute code
This will both display and execute the code:
```{r}
(33+55+192+12)/4
barplot(c(33,55,192,12))
```
````

Chapter 2

Handling Data: Reading, Filtering, Aggregating, and Applying functions to data frames

2.1 Reading and Writing in data from files

R is very flexible about reading in data, and can ingest and accept data in many different formats. It is happiest reading in data that are stored in tabular text files with simple separators. A .csv file is a perfect way to save your data so that R can ingest it. But space or tab-separated files are also handled without any hassle. If use the right package, you can also read in simple .xls/.xlsx files (xlsReadWrite), as well as some other stats package data formats such as SPSS using the "foreign" package.

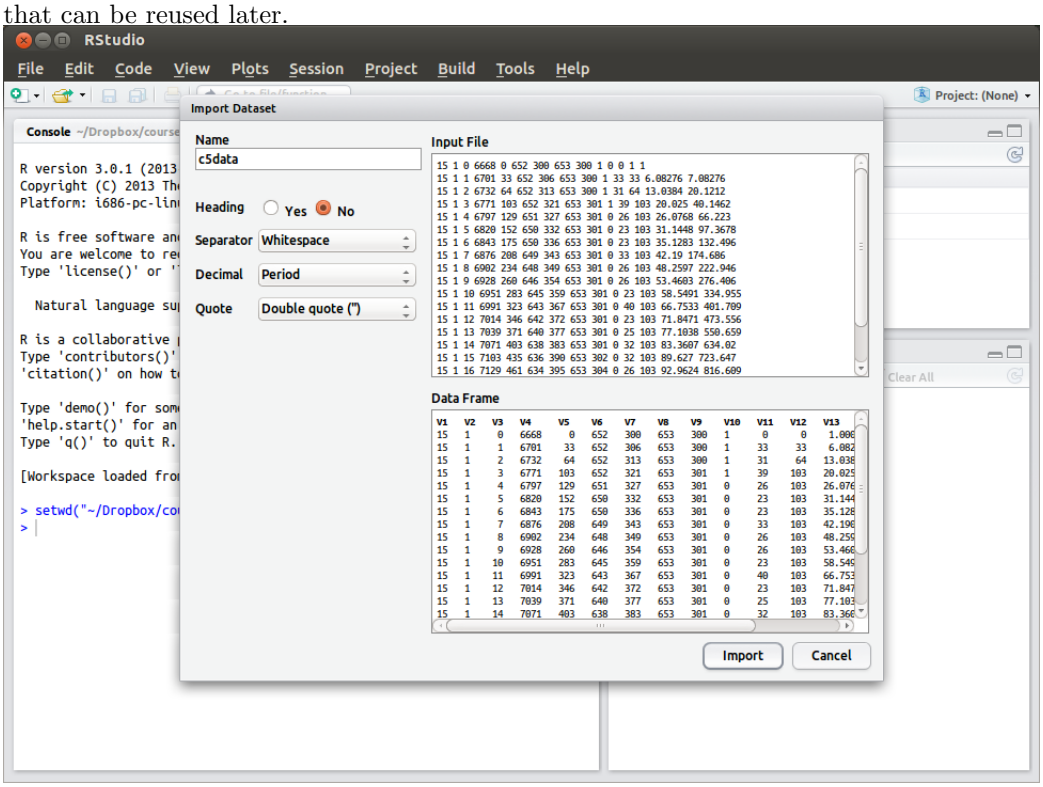
2.1.1 Reading Files

One of the most difficult aspects about reading in data is letting R know where the data is. RStudio has an 'Import dataset' option under the Tools menu that lets you select a file and read it in, helping you through different formatting options. This will write out a command to the console that will let you read in a file, and then save the command for later use.

This can be useful, but it is pretty easy to write the command yourself in an R script. To do this, you need to know where R will look for your files. R has a concept of a 'working directory', which is where it will load and save files during a session. You might want to use a single directory for all of your work, or deliberately set it for each session/project. Typically, you will change the working directory for any distinct data project you have so that different analyses stay separate. You can set the working directory via the `setwd()` function, in RStudio under the workspace menu option. The easiest method is to set the working directory to the source file directory, which is an option under the Session—Set Working Directory menu option.

Suppose you have a file named c5data.txt, that looks like:

Figure 2.1: Screenshot of the RStudio data import dialog. The dialog creates an R command that can be reused later.




```
15 1 0 6668 0 652 300 653 300 1 0 0 1 1
15 1 1 6701 33 652 306 653 300 1 33 33 6.08276 7.08276
15 1 2 6732 64 652 313 653 300 1 31 64 13.0384 20.1212
15 1 3 6771 103 652 321 653 301 1 39 103 20.025 40.1462
15 1 4 6797 129 651 327 653 301 0 26 103 26.0768 66.223
15 1 5 6820 152 650 332 653 301 0 23 103 31.1448 97.3678
15 1 6 6843 175 650 336 653 301 0 23 103 35.1283 132.496
15 1 7 6876 208 649 343 653 301 0 33 103 42.19 174.686
15 1 8 6902 234 648 349 653 301 0 26 103 48.2597 222.946
```

The data file is available for download as a companion to this coursebook. Notice that each row represents an observation or record, and each column represents some distinct independent or dependent variable in an experiment. We would like to read this into a data frame, where each column represents a different variable, and they are all tied together in a tabular format.

Now, to read in the data file: RStudio lets you read this in with a menu command under Workspace—import data set. It also allows you to set various options, including the delimiter–the character that separates each data value (in this case a space). Figure 2.1 is a screenshot of the dialog, showing how you can specify a number of aspects of the data.

Exercise 2.1.1

Read in data file from menu, after setting the working directory. Then, copy the generated command into an .R file, and load it directly from there.

Before you run the command, you can look at the data file just by opening or dragging it into the editor window in RStudio. This doesn't give up access to the data within R, it just lets us look at (and edit) the raw text data file. In the file, we have columns separated by spaces, which used to be pretty common, and still is for some kinds of data like logging data. That format might get us in trouble if some data cell is text with words in it, but in this case we are OK. The `read.table` command is pretty smart about this; it handles whitespace as the default separator of the data. It also chooses a variable name based on the filename, which might not be what we want. When we use the menu, it just creates code that it sends to the console, which in this case looks like this:

```
'c5data' <- read.table("~/Documents/data/c5data.txt", quote="\"")
```

Because the rules for filenames are a bit different than rules for variables within R, it enclosed the variable name c5data in single quotes. There may have been spaces in the filename and it would have preserved that in the variable name, which is only possible with the quotes. But then to use the variable, we also need to use quotes, which is inconvenient. We could have named the data set explicitly as '`data`', and foregone giving the quote argument and the full path of the file (as long as it was in the working directory). In fact, we didn't need to use the menu at all–we could have just typed the command in the console or in the markdown or R file. Then it will automatically reload the data file whenever we run the .Rmd file:

```
data <- read.table("c5data.txt")
```

Notice that when we read the file in, it automatically generates headers with the names `V1` through `V14`.

```
> data
  V1 V2 V3   V4  V5  V6  V7  V8  V9 V10 V11 V12      V13      V14
1 15  1  0 6668   0 652 300 653 300   1   0   0  1.00000  1.00000
2 15  1  1 6701  33 652 306 653 300   1  33  33  6.08276  7.08276
3 15  1  2 6732  64 652 313 653 300   1  31  64 13.03840 20.12120
4 15  1  3 6771 103 652 321 653 301   1  39 103 20.02500 40.14620
5 15  1  4 6797 129 651 327 653 301   0  26 103 26.07680 66.22300
```

But these headers are not very readable. In fact, the headers for this data file should be as follows (even though they didn't appear in the file):

```
subcode trial step time trialtime targx targy
 mousex mousey ontarget dtime totaltime distoff sumdistoff
```

To get the headers on the data frame you can:

1. Edit the file directly, and paste in the header. The file data2.txt is edited using an external editor.

```
data <- read.table("c5data2.txt",header=T)
```

2. Set the column names when you read it in. This sets a variable with the header names, then uses a parameter of read.table to set it on read-in.

```
head <- c("subcode","trial","step","time","trialtime",
          "targx","targy","mousex","mousey","ontarget",
          "dtime","totaltime","distoff","sumdistoff")
data <- read.table("c5data.txt",col.names=head)
```

3. Set the column names afterward. You can give the header as an argument when reading in the data:

```
data <- read.table("c5data.txt")
colnames(data) <- head
```

4. Possible Problems Things can go wrong if you turn the header off but there really is a header, or you specify a header that doesn't exist. As discussed in Chapter 1, a data frame is a collection of vectors, and everything in a vector has to have the same type (even if different vectors/columns of the data frame can have different types). Now, because each column will likely have a text value in its header row, all the numbers will be coded as categorical labels instead of numerical values. For example, you might read in data with a header, but specify `header=F`:

```
data <- read.table("c5data2.txt",header=F)
data[1:10,]
       V1    V2   V3   V4        V5    V6    V7     V8     V9
1  subcode trial step time trialtime targx targy mousex mousey
2       15     1    0 6668         0   652   300    653    300
3       15     1    1 6701        33   652   306    653    300
4       15     1    2 6732        64   652   313    653    300
5       15     1    3 6771       103   652   321    653    301
```

Or, you might read in a file without a header, but accidentally turn it on:

```
data <- read.table("c5data.txt",header=T)
data[1:5,]
  X15 X1 X0 X6668 X0.1 X652 X300 X653 X300.1 X1.1 X0.2 X0.3     X1.2     X1.3
1  15  1  1  6701   33  652  306  653    300    1   33   33  6.08276  7.08276
2  15  1  2  6732   64  652  313  653    300    1   31   64 13.03840 20.12120
3  15  1  3  6771  103  652  321  653    301    1   39  103 20.02500 40.14620
4  15  1  4  6797  129  651  327  653    301    0   26  103 26.07680 66.22300
5  15  1  5  6820  152  650  332  653    301    0   23  103 31.14480 97.36780
```

Notice that it turns the first row of data into the header, prepending an X in front of numbers. Because of little issues like this, the first thing you should do after reading in data is to look at it to make sure the types and header names are correct.

Finally, you can use the 'skip' argument to skip over the first few lines of a data file, in case the headers in the file are inappropriate and you want to create a new set:

```
##Skip the first line of the data file
data <- read.table("c5data2.txt",skip=1, header=F)
colnames(data) <- head
```

But be careful here–you might delete the first few rows of data and not realize it if you skip them.

2.1.2 Other Functions to Read Data

The functions `read.table` and `read.csv` are essentially two names for the same function, just with different default arguments. Consequently, comma-separated values (.csv) files can be read in either with:
`read.table("filename",sep=",")`
or with:
`read.csv()`

```
dat2 <- read.table("c5data.csv",sep=",")
dat2 <- read.csv("c5data.csv")
```

There are also several other related functions: `read.delim` which reads tab-separated files, `read.fwf`, for reading fixed-width formatted files, and `scan`, which is a bit more flexible and will read data into a vector. There are other libraries that read in more sophisticated data tables as well, so be sure that if you use one of these, you know what it is doing and have downloaded/installed the library.

2.1.3 Saving a Data Frame to a text file

You can save files using `write.table` or `write.csv`:

```
> write.csv(data,"data.csv")
> write.table(data,"data.txt")
```

We can then look at the file we created. Here is data.csv:

```
"","subcode","trial","step","time","trialtime","targx","targy",
"mousex","mousey","ontarget","dtime","totaltime","distoff","sumdistoff"
"1",15,1,0,6668,0,652,300,653,300,1,0,0,1,1
"2",15,1,1,6701,33,652,306,653,300,1,33,33,6.08276,7.08276
"3",15,1,2,6732,64,652,313,653,300,1,31,64,13.0384,20.1212
"4",15,1,3,6771,103,652,321,653,301,1,39,103,20.025,40.1462
"5",15,1,4,6797,129,651,327,653,301,0,26,103,26.0768,66.223
```

This can be handy if you have done a lot of processing and want an intermediate data file. Notice that when you do this, it saves, by default, a set of rownames (here numbers 1 through 5) which may not have been in the original. To avoid this, use the argument $row.names = F$ instead of the default $row.names = T$. The result will look like this:

```
    "subcode","trial","step","time","trialtime","targx","targy",
    "mousex","mousey","ontarget","dtime","totaltime","distoff","sumdistoff"
    15,1,0,6668,0,652,300,653,300,1,0,0,1,1
    15,1,1,6701,33,652,306,653,300,1,33,33,6.08276,7.08276
    15,1,2,6732,64,652,313,653,300,1,31,64,13.0384,20.1212
    15,1,3,6771,103,652,321,653,301,1,39,103,20.025,40.1462
    15,1,4,6797,129,651,327,653,301,0,26,103,26.0768,66.223
```

This might be handy if you want to read in data files and save them in the exact format as you read them in. In either case, the headers are all written out, and each record is separated by a comma (because we used write.csv). The file will be saved in your working directory, unless direct it elsewhere using the file name. Any factors will be turned into character strings, and all character strings will be quoted.

Exercise 2.1.3

Generate a matrix of random numbers in a table that is 10 columns and 100 rows. Name the columns after the first ten letters of the alphabet (`letters[1:10]`). Save it out to a .csv data file, and then read it in again.

2.2 Examining data structures

```
data(trees)       #Load the trees data
```

The variable `trees` is a built-in data set in a data frame. It is essentially a spreadsheet data object, where each column has a name, and rows are numbered, and all data in a single column has to have the same type. You can examine trees by using the `attributes()`, `dim`, and `str` functions

```
> dim(trees)
 [1] 31  3

> str(trees)
'data.frame': 31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...

> attributes(trees)  #look at what trees is made of.

$names
```

```
[1] "Girth"  "Height" "Volume"

$row.names
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
    25
[26] 26 27 28 29 30 31

$class
[1] "data.frame"
```

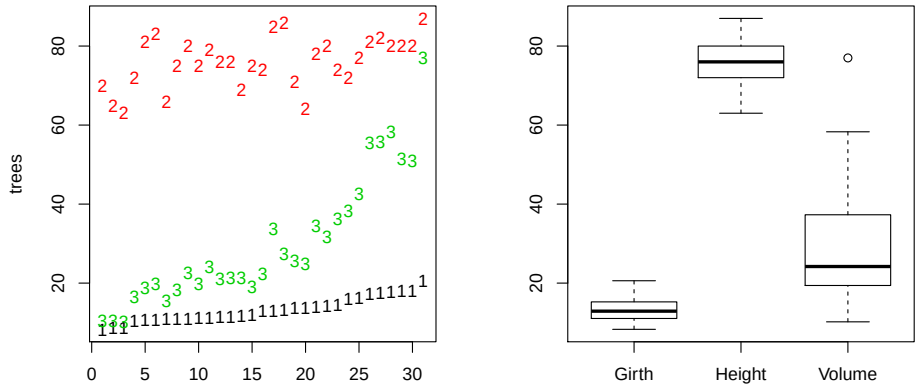
I normally start by just typing the data set name into the console to look at it:

```
  > trees
  Girth Height Volume
1    8.3     70   10.3
2    8.6     65   10.3
3    8.8     63   10.2
4   10.5     72   16.4
5   10.7     81   18.8
6   10.8     83   19.7
7   11.0     66   15.6
8   11.0     75   18.2
9   11.1     80   22.6
10  11.2     75   19.9
11  11.3     79   24.2
12  11.4     76   21.0
13  11.4     76   21.4
14  11.7     69   21.3
15  12.0     75   19.1
16  12.9     74   22.2
17  12.9     85   33.8
18  13.3     86   27.4
19  13.7     71   25.7
20  13.8     64   24.9
21  14.0     78   34.5
22  14.2     80   31.7
23  14.5     74   36.3
24  16.0     72   38.3
25  16.3     77   42.6
26  17.3     81   55.4
27  17.5     82   55.7
28  17.9     80   58.3
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
```

This works if the data frame is not that big, like our 31 observations. If it were thousands of rows, we wouldn't easily be able to look at it like this. Instead we might do a quick plot or subset the data to see parts of it.

We can look at the different data series across the different observations using `matplot`. This does not make a lot of sense, because the horizontal axis is just the order they appear in the data set. A better plotting function would be `boxplot`.

```
par(mfrow=c(1,2))
matplot(trees)
boxplot(trees)
```



It is important to get a sense for what the data will look like, how many observations there are, etc., so you will know what to expect. Suppose you want to look at the data in a tabular format. In RStudio, we can use the `View` function, or click on the data object in the Workspace tab. But we can quickly look at the first few lines of the data using the `head` function:

```
> head(trees)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
```

Which is essentially identical to:

```
> trees[1:6,]
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
```

This type of subsetting was introduced in Chapter 1. You can also use subsetting to *remove* a column, by giving the negative column index. The following two statements are identical for the three-column `trees` data set.

```
plot(trees[,1:2])
plot(trees[,-3])
```

Just by looking at the output of these commands, we can tell that this data set has 31 observations of trees. They seem to be organized from smallest to largest in girth, and although both height and volume are correlated with girth, the relationship between volume and girth seems much clearer.

2.3 Sorting

The same subsetting operator ([]) can be used to sort data frames, vectors, and matrices. Suppose we want to sort the values of two vectors `x` and texttty by the values of `y`. To do this, it will be easiest if they are in a data frame first:

```
x <- runif(500)
y <- x + runif(500,-.3,.3)
dat <- data.frame(x=x,y=y)
print(dat)
```

Now, to sort, we start by feeding the values of y into the `order` function. This will create a set of numbers that indicate the index of the smallest, then next smallest, and then third smallest elements of the vector, up to the largest. For example, notice that the smallest value is in the second position, the next smallest is in the third, and the fourth smallest is in the first position of the original vector:

```
> order(c(2,.5,1,1.5,2,2.5))
[1] 2 3 4 1 5 6
```

Don't get order confused with rank, which is the order of the order (not counting ties). As an analogy, suppose we had people register for a race and recorded their times in the registration table. Rank-order is what we use in races to determine the order people finished in (someone may have finished first or seventh). Order is how we look up which row of the table corresponds to the Nth-fastest time.

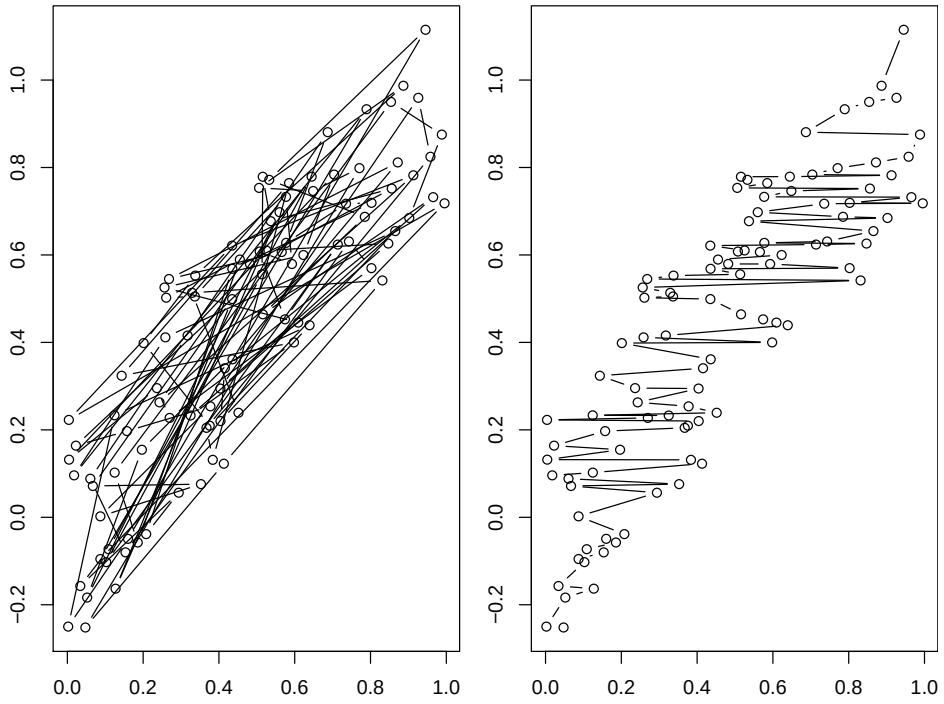
```
> rank(c(2,.5,1,1.5,2,2.5))
[1] 4.5 1.0 2.0 3.0 4.5 6.0

> order(order(c(2,.5,1,1.5,2,2.5)))
[1] 4 1 2 3 5 6
```

Rank replaced each element with its rank order; order specifies the indices in order of their values.

The order function allows us to easily reorder a data set from smallest to largest, or to sort another data set based on the values of the first.

```
ord <- order(dat$y)
dat2 <- dat[ord,]
> head(dat2)
                x           y
309 0.006156245 -0.2875007
128 0.040296487 -0.2252086
158 0.026100303 -0.2203838
449 0.084372839 -0.2135331
47  0.070928196 -0.2050974
466 0.057561985 -0.1802051
```



Notice that when we look at `dat2`, a sorted version of `dat1`, the rows are in a different order. They retain the original row names, and now the smallest value of y is in the first row. When we plot them, if we just plot the points, it won't matter, but if we connect points, they get connected to adjacent points in the figure:

```
par(mfrow=c(1,2),mar=c(3,3,0,0))    ##Make two plots
plot(dat,type="b")   ##Plotted in random order
plot(dat2,type="b")  ##sequentially ordered by x
```

Exercise 2.3

The `type` argument of plot allows you to plot points connected by lines, using the `type="b"` argument. First, plot tree height by volume in its original order, connecting adjacent values, using the `type="b"` argument. Then re-sort them by tree height and re-plot. Finally, re-sort them in a random order, and re-plot.

2.4 Aggregation

2.4.1 Tables

The simplest way of aggregating is with a "crosstab" table. If you think about a set of observations you have categorized along several dimensions, you might want to know how many fit into each combination of categories. For example, you might poll individuals and ask them their gender, their political affiliation, and who they would vote for (candidate A or B). Your raw data might look like this:

```
party <-  c("R","R","D","R","R","D","D","D","R","R","D")
gender <- c("M","M","F","F","F","F","M","M","F","M","M")
vote   <- c("A","B","A","A","A","B","A","A","B","B","A")
survey <- data.frame(party,gender,vote)
```

In `survey`, each row is a person you surveyed. The `table` function allows us to create a count of the categories within a data set.

```
> table(survey$party)
party
D R
5 6
> table(survey$gender)
gender
F M
5 6
> table(survey$vote)
vote
A B
7 4
```

The `table` function will take multiple arguments, and compute counts for every combination of levels of the variables:

```
> table(survey$gender,survey$vote)
  A B
F 3 2
M 4 2

> table(survey$party,survey$vote)
  A B
D 4 1
R 3 3
>
```

Here, we see that both men and women in the sample prefer A slightly to B. On the other hand, Democrats prefer A 4:1, whereas Republicans are equally split.

Table doesn't stop at two dimensions–you can make a table of as many dimensions as you have categories:

```
xtab <- table(survey)
xtab
 , , vote = A

      gender
```

```
 party F M
     D 1 3
     R 2 1

 , , vote = B
      gender
 party F M
     D 1 0
     R 1 2
```

This creates a 2x2x2 table with counts for each cell. We can access slices of this table the same way we accessed rows/columns of a matrix:

```
> xtab[1,,]   #party D
vote
gender A B
F 1 1
M 3 0
> xtab[2,,]   #party R
vote
gender A B
F 2 1
M 1 2
> xtab["D",,] #party = D
vote
gender A B
F 1 1
M 3 0

> xtab[1,2,]   #Democratic Men
A B
3 0
```

Here is a larger example with more observations:

```

 x <- sample(c("a","b","c"),50,replace=T)
 y <- sample(c("unicorns","pegasuses"),50,replace=T)
 z <- sample(c("chocolate","vanilla","strawberry"),50,replace=T)

 table(x)
x
a  b  c
20 18 12

 table(x,y)
y
x   pegasuses unicorns
a        11        9
b         9        9
c         3        9

 table(x,y,z)
, , z = chocolate

y
x   pegasuses unicorns
a         7        3
b         4        6
```

```
c          1        5

, , z = strawberry

y
x   pegasuses unicorns
a          1        4
b          5        2
c          1        2

, , z = vanilla

y
x   pegasuses unicorns
a          3        2
b          0        1
c          1        2
```

Notice that if we give `table` a data frame, it works just as if we gave it each variable individually:

```
 table(data.frame(x,y,z))
, , z = chocolate

y
x   pegasuses unicorns
a          7        3
b          4        6
c          1        5

, , z = strawberry

y
x   pegasuses unicorns
a          1        4
b          5        2
c          1        2

, , z = vanilla

y
x   pegasuses unicorns
a          3        2
b          0        1
c          1        2
```

If we name the output, we can then use [] notation to select particular slices (sub-tables), down to individual entries, either by specifying them numerically or by name:

```
 cross  <-  table(x,y,z)
>
> cross[1,,]
z
y           chocolate strawberry vanilla
pegasuses           7          1       3
unicorns            3          4       2

 cross["a",,]
```

```
z
y           chocolate strawberry vanilla
pegasuses          7          1       3
unicorns           3          4       2

 cross[,2,]
z
x   chocolate strawberry vanilla
a          3          4       2
b          6          2       1
c          5          2       2

 cross[,2,3]
a b c
2 1 2
> cross[1,1,2]
[1] 1
> cross[1:2,2,3]
a b
2 1
```

2.4.2 Functions `aggregate` and `tapply`

A table computes counts of each category. Oftentimes, we want to compute another measure over the data. Two of the most useful functions in R to do this are `aggregate` and `tapply`. In the future, we will also learn about more modern approaches to this using the "tidyverse" set of libraries, but for now, both of these functions create a new data set by applying a function to subsets values, specified by (possibly combined) levels of other variables. The subsetting variable will be treated as a categorical variable, even if it is a number. These functions are similar to what can be done using pivot tables in a spreadsheet or database, but many stats programs have no easy equivalent. R is good at this because it lets you store multiple data frames. So you can have your raw data loaded in one variable, and then an aggregated set in another, without losing the first data set.

An example:

```
set.seed(111)
x <- rnorm(500)
y <- x + runif(500,-.3,.3)
dat3 <- data.frame(x=x,y=y)
dat3$factor <- factor(round(dat3$x/10,1)*10)  ##make bins from x
dat3.agg <-
     aggregate(dat3$y,list(bin=dat3$factor),mean)
dat3.agg$xvals <-
     aggregate(dat3$x,list(bin=dat3$factor),mean)$x
dat3.tab <- tapply(dat3$y,list(bin=dat3$factor),mean)

> dat3.agg
  bin          x       xvals
1  -3 -2.9694215 -3.00802536
2  -2 -1.8622799 -1.84992413
3  -1 -0.8908935 -0.89592586
4   0  0.0382442  0.04730851
5   1  0.9096850  0.91197362
6   2  1.7929266  1.82510517
7   3  2.6943167  2.69485129
> dat3.tab
bin
         -3          -2          -1           0           1           2           3
-2.9694215 -1.8622799 -0.8908935  0.0382442  0.9096850  1.7929266  2.6943167
```

Look at the difference between the aggregated data and the table data. When you have just a single factor, there is not much, but when you start using multiple factors, aggregate keeps your data in a "long" format, with condition tagged for each row, but `tapply` makes a table.

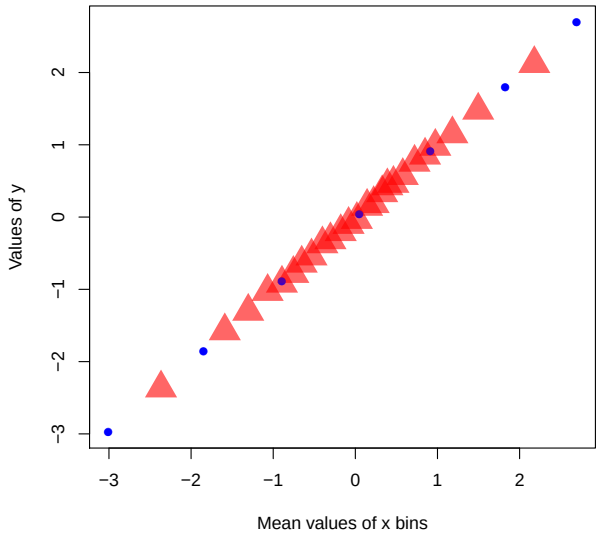
Another way to make bins in a slightly different way uses the `rank` function, which will transform the actual value to a rank (the smallest will equal 1, the next smallest 2, etc.). You can use this to make bins with equal numbers of observations. For example, this will make bins each having ten observations. if we divide the rank by 10 and round down, everything 1 through 20 will be equal to 0 (the first bin), ranks 11 to 20 will be 1, and so on. Here, I will make this new binned factor, then aggregate `y` by this new factor, then aggregate x, extract the aggregated factor and put it on the earlier aggregation `dat3.agg2`.

```
dat3$factor2 <- floor(rank(dat3$x)/10)
dat3.agg2 <- aggregate(dat3$y,list(bin=dat3$factor2),mean)
dat3.agg2$xvals <-aggregate(dat3$x,list(bin=dat3$factor2),mean)$x
```

We can compare the plots made by aggregating over equal widths versus equal numbers of observations. The blue are the original bins, and the red triangles are equal-sized bins.

```
plot(dat3.agg$xvals,dat3.agg$x,col="blue",pch=16,
     xlab="Mean values of x bins",ylab="Values of y")
col2 <- rgb(1,0,0,.6)   # a see-through red
col1 <- rgb(0,0,1,.6)   # a see-through blue
points(dat3.agg2$xvals,dat3.agg2$x,cex=3,col=col2,pch=17)
points(dat3.agg$xvals,dat3.agg$x,cex=1,col=col1,pch=16)
```

Notice how, in the figure, the blue dots are equally spaced, and the red triangles are clustered. Although there are many cases when equal-width bins are preferred, in this case



it hides the fact that the bins on the outside are formed from relatively few observations, whereas the bins in the middle are formed from many.

Exercise 2.4.2

Overplot the mean of each bin on a plot of x versus y.

The difference between `tapply` and `aggregate` are more critical when you aggregate over multiple factors.

Let's create a random independent variable:

```
dat2 <- as.data.frame(1:100)
dat2$factor1 <- sample(factor(c("X","Y","Z")),100,replace=T)
dat2$factor2 <- sample(factor(c("A","B")),100,replace=T)
dat2$y <- rnorm(100)
```

Now, we will use both `aggregate` and `tapply` to compute means by each level of the two factors.

```
dat2.agg2 <- aggregate(dat2$y,list(factor1=dat2$factor1,
                                   factor2=dat2$factor2),mean)

 dat2.agg2
  factor1 factor2           x
1       X       A  0.21732846
2       Y       A -0.18088944
3       Z       A  0.02998482
```

```
4        X        B  0.18830125
5        Y        B -0.00255622
6        Z        B  0.28403348


dat2.tab2 <- tapply(dat2$y,   list(dat2$factor1, dat2$factor2),mean)

 dat2.tab2
                A           B
X  0.21732846  0.18830125
Y -0.18088944 -0.00255622
Z  0.02998482  0.28403348
```

The table is very easy to plot with `matplot`, and it is difficult to get standard `plot` to work on the result of aggregate.

```
par(mfrow=c(1,1))
matplot(dat2.tab2,type="o")
```

A similar plot can be done all-in-one using:

```
interaction.plot(dat2$factor,dat2$factor2,dat2$y)
```

Exercise 2.4.2

For dat2, plot x vs. y, with the color dependent on the value of factor2. Do this in a single plot command.

2.5 The apply function: aggregating by rows or columns

Sometimes, you have a special function you want to apply to each row or column. For example, we may want to compute the variance, standard deviation, min, or max to each row or column. We can use the `apply` function to apply a function repeatedly on each row or column of a matrix—in this case a 7x4 made-up data matrix. The function takes three arguments: the data table, the margin, and the name of the function we want to apply. The margin indicates which index (row=1 and column=2) we want to apply by. If we have a larger table, we can apply to the third/fourth or greater dimension.

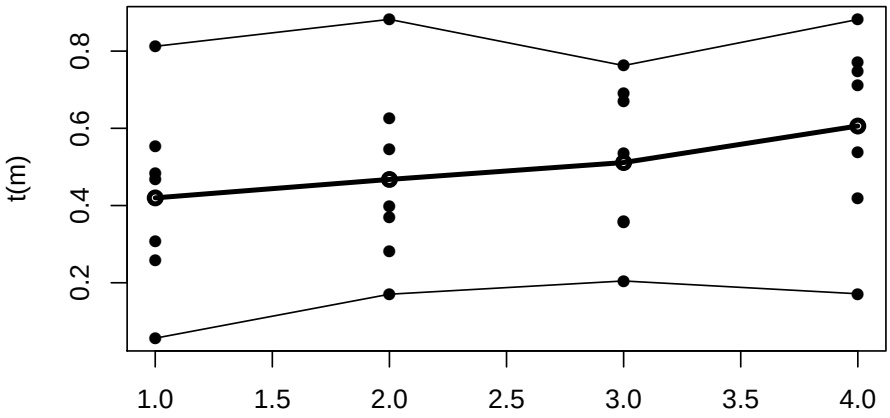
```
m <- matrix(runif(28),7,4)
 m
          [,1]      [,2]      [,3]      [,4]
[1,] 0.21104813 0.4577078 0.7834887 0.7875471
[2,] 0.55996277 0.6619420 0.6980809 0.9733729
[3,] 0.70762780 0.7811945 0.8346006 0.9457420
[4,] 0.54502749 0.9287212 0.5339969 0.7487509
[5,] 0.54972281 0.7647778 0.7075352 0.3891058
```

```
[6,] 0.27248057 0.7018390 0.0667138 0.8204995
[7,] 0.09511412 0.5343559 0.3036206 0.2537405

 round(apply(m,1,var),3) ##By row
[1] 0.084 0.159 0.142 0.068 0.089 0.010 0.092
mins <- apply(m,2,min) ##By column
maxs <- apply(m,2,max) ##By column
 mins
[1] 0.007168949 0.197287780 0.026100020 0.025987033
 maxs
[1] 0.7173192 0.9756417 0.9555009 0.5591875
```

This can be handy for many reasons. Here, we can use it to plot a set of points, and connect the minimums and maximums in each row.

```
par(mfrow=c(1,1))
matplot(t(m),type="p",pch=16,col="black")
points(apply(m,2,mean),type="o",lwd=3)
points(mins,type="l")
points(maxs,type="l")
```



2.5.1 Summing or finding means by row or column

Sometimes you want to aggregate mean or sum over the columns or rows of a matrix. There are some built-in functions for this:

```
rowSums(m)
[1] 2.151442 2.011602 1.676169 1.459006 1.881610 2.224717 2.628655
> colMeans(m)
```

```
[1] 0.4198381 0.4675921 0.5113267 0.6059859
```

2.6 A Complete Example

This end-to-end example uses aggregation, filtering, and multiple layers of plotting to illustrate some of the capabilities of R.

R includes a data set called `ChickWeight`. Load the data set, and look at what it involves:

```
data(ChickWeight)
> attributes(ChickWeight)$labels
$x
[1] "Time"

$y
[1] "Body weight"

> attributes(ChickWeight)$units
$x
[1] "(days)"

$y
[1] "(gm)"
```

An easy way to see the first few lines of a data set is with the `head()` command. Alternately, open with the `View()` command in RStudio, or by clicking on the variable name in the upper right-hand corner of the program.

```
head(ChickWeight)

> head(ChickWeight)
  weight Time Chick Diet
1     42    0     1    1
2     51    2     1    1
3     59    4     1    1
4     64    6     1    1
5     76    8     1    1
6     93   10     1    1
```

It appears to include measures of weight (in grams) over time (in days), for a number of chicks on a number of diets. How many chicks were there? How many diets? A simple way to find this out is to make a table:

```
 length(table(ChickWeight$Chick))
[1] 50

 table(ChickWeight$Diet)

  1   2   3   4
220 120 120 118
```

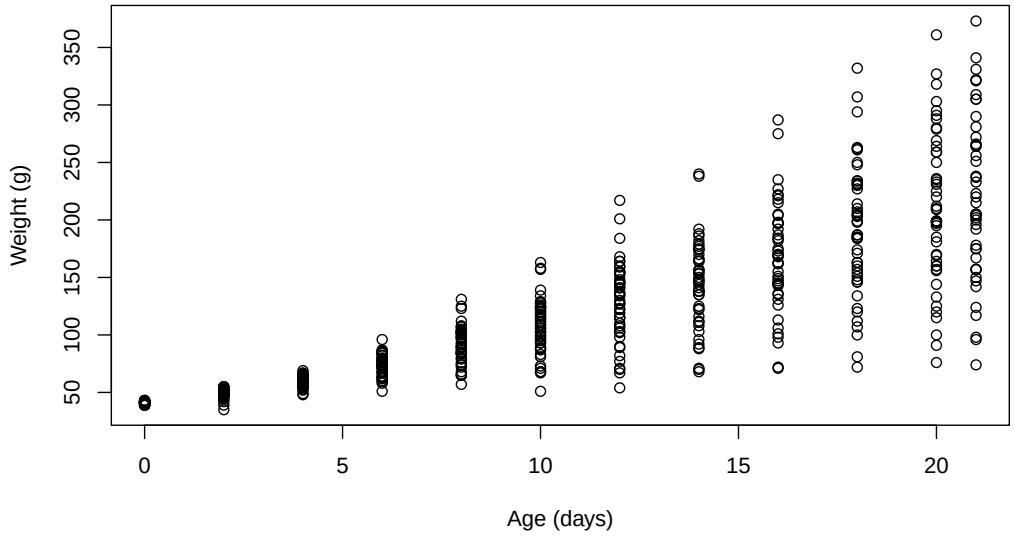
The first command shows we have 50 chicks. The second shows we have 4 distinct diets, with between 118 and 220 observations per diet. But the observations are not distributed equally across diets.

2.6.1 Plot the growth 'cloud'

Let's begin by plotting weight by time. We expect weight to increase by time, and all the points of all the chicks to be overlaid in a growth cloud, shown in Figure 2.2:

```
plot(ChickWeight$Time,ChickWeight$weight,
     ylab="Weight (g)",xlab="Age (days)")
```

Figure 2.2: Initial plot showing the growth of chicks over time.



We want to maybe distinguish different diets; To do this, we can use color. Let's create a vector with four different color names, one for each diet:

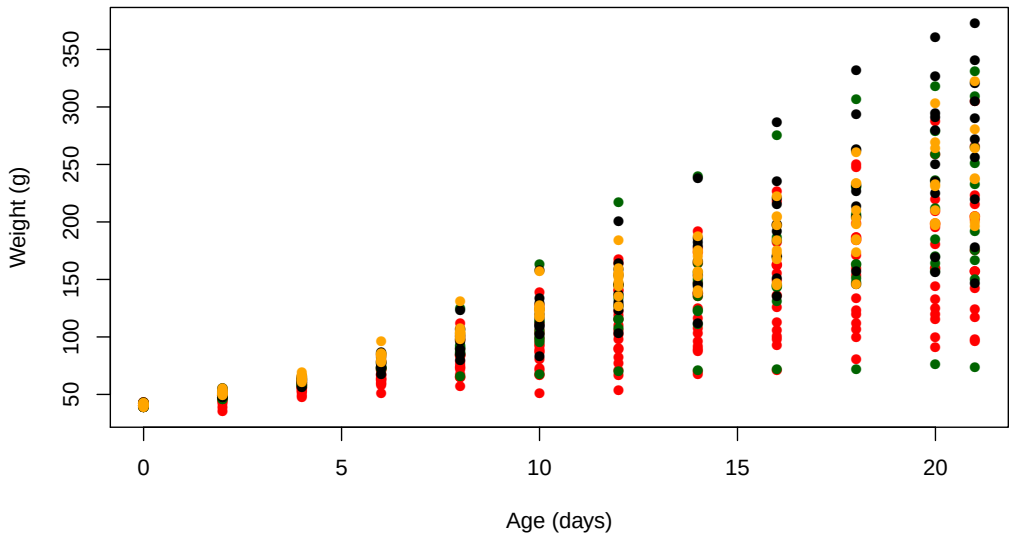
```
cscheme <- c("red","darkgreen","black","orange")
```

There are four diets. Let's plot the cloud by diet (shown in Figure 2.3).

```
plot(ChickWeight$Time,ChickWeight$weight,
     ylab="Weight (g)",
     xlab="Age (days)",
     pch=16,
     col=cscheme[ChickWeight$Diet])
```

Now we can see basic color separations for the different diets. It would be nice to overlay a mean growth curve for each diet. We can use `tapply` to aggregate growth over the diets and times.

Figure 2.3: Figure of color-coded chick weights with average growth trajectories for each weight




```
cw.agg <- tapply(ChickWeight$weight,
                 list(time=ChickWeight$Time,
                      diet=ChickWeight$Diet),mean)
```

```
 cw.agg
    diet
time          1     2     3        4
  0   41.40000  40.7  40.8  41.0000
  2   47.25000  49.4  50.4  51.8000
  4   56.47368  59.8  62.2  64.5000
  6   66.78947  75.4  77.9  83.9000
  8   79.68421  91.7  98.4 105.6000
  10  93.05263 108.5 117.1 126.0000
  12 108.52632 131.3 144.4 151.4000
  14 123.38889 141.9 164.5 161.8000
  16 144.64706 164.7 197.4 182.0000
  18 158.94118 187.7 233.1 202.9000
  20 170.41176 205.6 258.9 233.8889
  21 177.75000 214.7 270.3 238.5556
```

Also, the plot shows that the times recorded were not uniform–starting at Day 0, weight was measured every 2 days until day 20, then again at day 21. To deal with that, we can just manually record the times of measurement, or pull them out of `cw.agg`. But notice that the time column in the table is actually a text label, and so we need to convert them back to numbers.

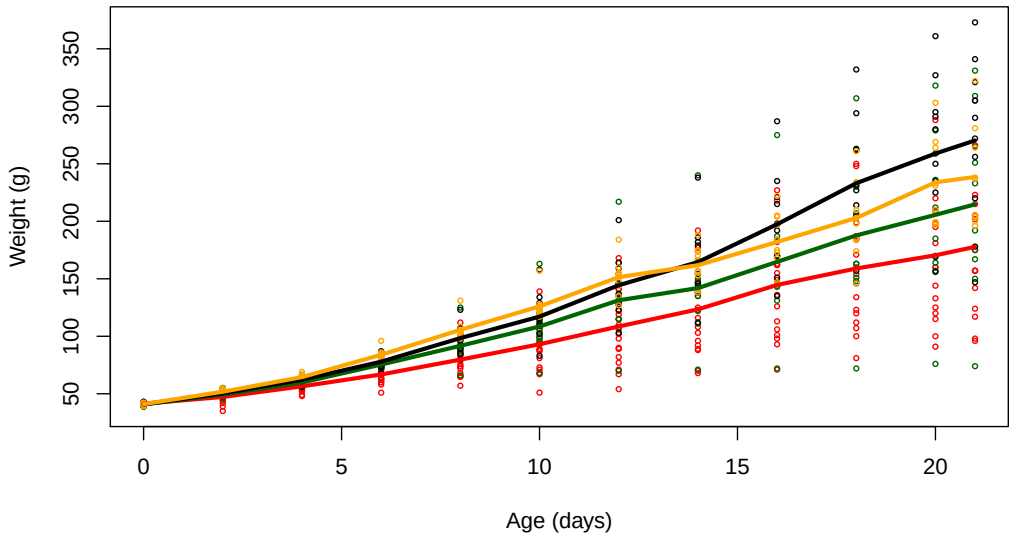
```
##It is easy to just write these down
times <- c(0,2,4,6,8,10,12,14,16,18,20,21)
## We could pull them out of the table, and convert to numbers.
times <- as.numeric(rownames(cw.agg))
```

Now, let's re-plot the points by diet, using a lightly different character scheme, and then overlay the mean values for each diet using the `matplot` function. Notice that in order to overlay, we need to use the `add=T` argument:

```
plot(ChickWeight$Time,ChickWeight$weight,cex=.5,
     ylab="Weight (g)",xlab="Age (days)",
     pch=1, col=cscheme[ChickWeight$Diet])

matplot(times,cw.agg,add=T,type="l",lwd=3,
       col=cscheme,lty=1)
```

Figure 2.4: Basic color-coded chick weight cloud with mean weights for each diet overlaid.



The dot cloud is nice, but it would be helpful to be able to see each individual growth path. To do so, we have sort of hit the limits of `plot` and `points`, but we want to put the growth trends into a matrix, and then plot them as connected series with `matplot`. A way to to this is to use `tapply` again. But this time, we aggregate by both time and chick. Note that there will be only one observation per cell, and so taking the mean of a single number is an easy way to get that number. We could also use the function `I`, which just returns the value that it is passed.

```
cw.bysub<- tapply(ChickWeight$weight,
                  list(time=ChickWeight$Time,
```

```
                        chick=ChickWeight$Chick),mean)

 ## look at just the first ten columns:
cw.bysub[,1:10]
   chick
time 18 16 15 13   9  20  10   8  17  19
  0  39 41 41 41  42  41  41  42  42  43
  2  35 45 49 48  51  47  44  50  51  48
  4  NA 49 56 53  59  54  52  61  61  55
  6  NA 51 64 60  68  58  63  71  72  62
  8  NA 57 68 65  85  65  74  84  83  65
  10 NA 51 68 67  96  73  81  93  89  71
  12 NA 54 67 71  90  77  89 110  98  82
  14 NA NA 68 70  92  89  96 116 103  88
  16 NA NA NA 71  93  98 101 126 113 106
  18 NA NA NA 81 100 107 112 134 123 120
  20 NA NA NA 91 100 115 120 125 133 144
  21 NA NA NA 96  98 117 124  NA 142 157
```

Now we can use matplot to plot this matrix, but what if we want to retain the color-coding. We'd like a list of diets associated with each column. How can we get this? We could use `tapply` again, but aggregating diet instead of weight. Alternately, we could use `aggregate`, and find the minimum, mean, or median diet for each chick (which is OK because they should all be the same). Note that in either case, we need to convert the diet, a category, into a number.

```
diets <- tapply(as.numeric(as.character(ChickWeight$Diet)),
                    list(chick=ChickWeight$Chick),median)


diets <- aggregate(as.numeric(as.character(ChickWeight$Diet)),
                    list(chick=ChickWeight$Chick),median)$x
diets
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
    3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4
```

Now, replot the growth curves, using `matplot`, and use diets to select the color of each line. Then, we will reuse the `plot` command from earlier (turning into `points` so it gets overlaid) to plot the actual points. The results are shown in the first panel of Figure 2.5. I'll add gridlines using the segments command, but do it early so the lines and points get plotted on top of the lines.

```
matplot(times,cw.bysub,col=cscheme[diets],type="l",lty=3,
         ylab="Weight (mg)",xlab="Age (days)")

##Lets add some gridlines
segments(-10,0:15*25,25,0:15*25,lty=3,col="grey")
segments(-10,0:7*50,25,0:7*50,lty=1,col="grey")

points(ChickWeight$Time,ChickWeight$weight,cex=.5,
     pch=1, col=cscheme[ChickWeight$Diet])
```

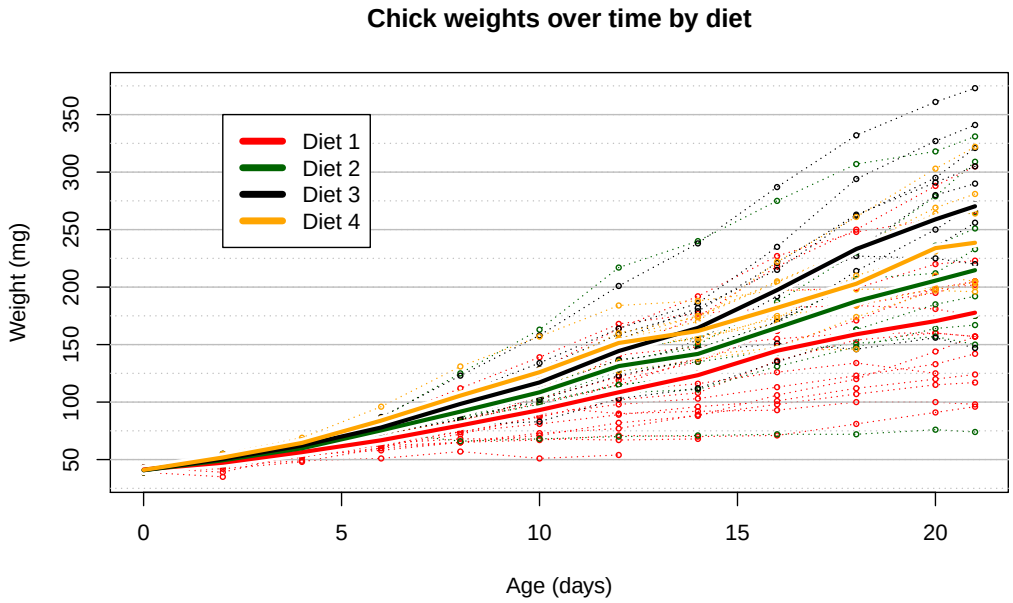
Next, to plot the means, lets first overlay a white line to erase some of the background lines, then overlay the colored mean lines, shown in Figure 2.5.

```
#erase back of line
matplot(times,cw.agg,add=T,type="l",lwd=7, col="white",lty=1)
#replot line:
matplot(times,cw.agg,add=T,type="l",lwd=3, col=cscheme,lty=1)
```

Finally, I'll add a title and other blandishments. The gridlines can be made with a single `segments` function. The results are in the final panel of Figure 2.5.

```
title("Chick weights over time by diet")
##Make a legend here.
legend(2,350,paste("Diet",c(1:4)),col=cscheme,lty=1,lwd=5,bg="white")
```

Figure 2.5: Basic color-coded chick weight cloud with mean weights for each diet overlaid.



2.7 Summary

In this chapter, we covered ways to use aggregate and tapply to make summary data tables. These can be useful in many contexts, including recoding data, aggregating across variables, forming composite score, and they are especially useful for plotting summaries of raw data.

Some of the functions we used in this chapter were:

- `read.table`
- `read.csv`
- `write.table`
- `write.csv`
- `matrix`
- `runif`
- `plot`
- `matplot`
- `points`
- `legend`
- `aggregate`
- `tapply`
- `table`
- `apply`

2.8 Solutions to Exercises

Exercise 2.1.1 Solution

Read in data file from menu, after setting the working directory. Then, copy the generated command into an .R file, and load it directly from there.

Exercise 2.1.3 Solution

Generate a matrix of random numbers in a table that is 10 columns and 100 rows. Name the columns after the first ten letters of the alphabet (`letters[1:10]`). Save it out to a .csv data file, and then read it in again.

```
dat <- matrix(runif(1000),100,10)
colnames(dat) <- letters[1:10]
write.csv(dat,"random.csv")
newdat <- read.csv("random.csv")
```

Exercise 2.3 Solution

The `type` *argument of plot allows you to plot points connected by lines, using the type="b" argument. First, plot tree height by volume in its original order, connecting adjacent values, using the* `type="b"` *argument. Then re-sort them by tree height and re-plot. Finally, re-sort them in a random order, and re-plot.*

```
data(trees)
par(mfrow=c(1,3))
plot(trees$Volume,trees$Height,type="b")
ord <- order(treesheight)
plot(trees$Volume[ord],trees$Height[ord],type="b")
ord <- sample(1:nrow(trees))
plot(trees$Volume[ord],trees$Height[ord],type="b")
```

Exercise 2.4.2 Solution

Overplot the mean of each bin on a plot of x versus y.

```
##the mean xvalue is stored in dat3.agg$xvals
##the mean yvalue is stored in dat3.agg$x:
plot(dat2$x,dat2$y)
points(dat3.agg$xvals,dat3.agg$x,pch=16,col="red")
```

Exercise 2.4.2 Solution

```
plot(dat2$x,dat2$y,col=c("red","blue")[dat2$factor2],pch=16)
```

Chapter 3

Programming in R

This chapter covers several core aspects of R as a programming language. This aspect offers a lot of flexibility, especially for initial data coding and processing.

3.1 Creating functions

We have already used a lot of functions–ones built-in to R. For the most part, these functions are themselves just a set of R commands bound together in a special way that lets you call them. Anything with the form `name(argument)` is a function, and it will generally do computation on the input values, return a value or data structure you can use later, and sometimes have a side-effect–like printing out an analysis or graphic. On rare occasions, a function will change the value of the input parameter, so that usually, the input values will remain unchanged. The return value is generally how information is returned from a function.

You can create functions yourself when you want to encapsulate a set of commands you want to use repeatedly using the `function` command. The last value computed in the function will be returned, or you can use `return()` to return something explicitly. Inside the parentheses of `function`, you include a list of named arguments you want the user to provide. Each of the following functions compute standard error, and produce the same results, but are defined differently.

```
se <- function(x)
{
   stdev <- sd(x)
   se <-  stdev/sqrt(length(x))
   return(se)
}

se <- function(x)
{
   stdev <- sd(x)
   return(stdev/sqrt(length(x))  )
}

se <- function(x)
{
   stdev <- sd(x)
   stdev/sqrt(length(x))
}

```

```
se <- function(x) {sd(x)/sqrt(length(x))}
```

In each of the cases, you would call the function on a data set like this:

```
y <- runif(20)
se(y)
```

A few general facts about functions:

- Any variable created within a function will go away and be inaccessible outside the function. You must return the value to make it available.
- A variable created prior to calling a function is accessible within a function. This power should be used carefully, because if you try to access something that is not defined, the function will crash.
- If you create a variable within a function with the same name as a variable defined outside the function, it will use that local variable instead of the global one.
- Functions usually have names. In the case above, the function name is `se`. The name of a function can be the same as the name of a variable, so it can be a bit confusing.
- You can use the `eval` and `call` functions with the name of another function and arguments to call a function programmatically, like `eval(call("mean",c(1,2,3)))`. This might let you to allow a user to specify a function name as an argument to another function.
- If you use the return keyword, it will return from the function immediately and not execute code after that point.

Exercise 3.1

Make a function that computes the minimum, median, and maximum of a data set `x` and returns them as a list of three elements. Create some fake data from at least two different distributions and test it out.

Many functions that compute statistics, such as `mean` and `sd`, will produce an `NA` result if any of the data it is applied to are `NA`. This makes sense because you want to be able to determine how you want to handle these, but it can be frustrating as well. The `mean` function includes an argument `na.rm`, which if set to `T` will ignore those `NA` values. We can create a wrapper function that uses this argument as its default:

```
cleanmean <- function(x)
    {
       mean(x, na.rm=T)
    }

x <- runif(1000)
```

```
x[x>.95] <- NA  ##Mark the highest ones as NA=missing

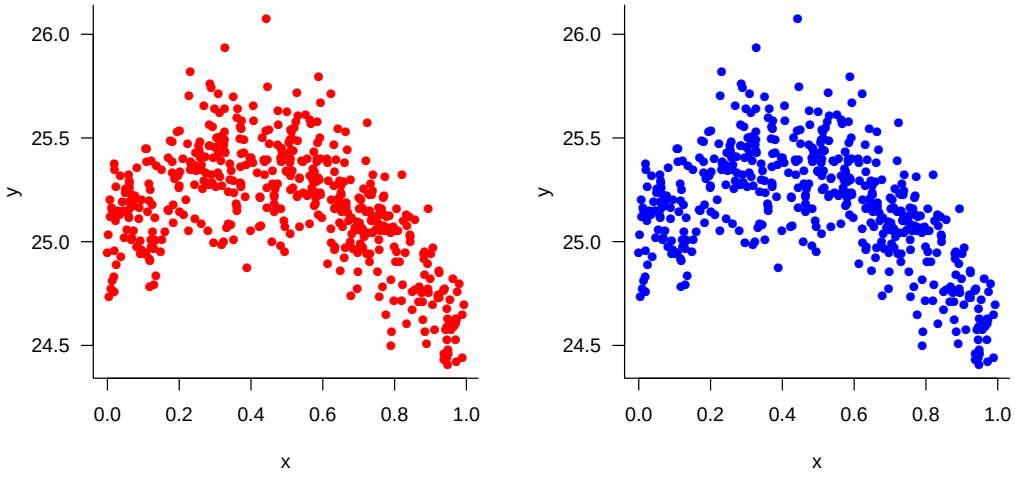
> mean(x)
[1] NA
> cleanmean(x)
[1] 0.485776
```

Functions can have more than one argument. You can specify default arguments within the function command using the equal sign. This allows you to create 'wrapper' functions around functions you use a lot but tend to use different default arguments. An example, `plotnice`, is shown below, whose output is shown in Figure 3.1.

```
x <- runif(500)
y <- 2 *x + 25  - 2.5*x^2 + .2*rnorm(500)
plotnice  <- function(x,y, col="red",
                      lty=3, bty="L", pch=16,las=1)
 {
   plot(x,y,col=col,lty=lty,bty=bty,pch=pch,las=las)
 }

plotnice(x,y)
plotnice(x,y,col="blue")
```

Figure 3.1: Two figures created with the `plotnice` function. `plotnice` is a wrapper for `plot`, which defaults to some specific plotting conditions (via arguments) that can be overridden.



Exercise 3.1

Create a `logplot()` function which plots log(y) versus x. Test it on a new y values computed as in:

```
x <- runif(1000)
y1 <- exp(x * 6 + runif(1000))
y2 <- exp(x * 6 + runif(1000)) + runif(1000)*100
```

3.1.1 Optional and Default values

You can specify optional parameters that have default values, and can check for missing values in functions as well. The typical way to specify a default value is to set its value equal to something in the function definition:

```
increment <- function(value, addend=1)
{
 return(value + addend)
}
```

The optional argument can be computed as well. For example:

```
plotletters <- function(values,labels=LETTERS[1:length(values)])
  {
    barplot(values,names=labels)
  }

plotletters(runif(5))
```

For more complex functions, you can use the `missing` and `is.null()` functions to check whether a value is missing. There is also a special argument named `...`, that scoops up all arguments a user might specify that aren't otherwise named. You can then look through that, or even pass it on to another function, to permit easy pass-through arguments.

```
addup <- function(x,y=NULL)
{
  if(is.NULL(y))
  {
   return(x)
  }else{
   return (x+y)
  }
}

plotxy <- function(x,y)
{
  if(missing(y))
  {
   plot(x)
  } else{
  plot(x,y)
  }
}
plotxy(runif(10))
plotxy(runif(10),runif(10))



plotlabeled <- function(x,y,...)
{
   plot(x,y,main="Special plot",...)
}

par(mfrow=c(1,2))
plotlabeled(runif(10),runif(10))
plotlabeled(runif(10),runif(10),pch=16,cex=2)
```

For more information about optional arguments, see also: http://stackoverflow.com/questions/28370249/correct-way-to-specifiy-optional-arguments-in-r-functions

3.1.2 Wrapping a function

Sometimes, you want to use a function in `aggregate`, `apply`, or `tapply`, but it takes multiple arguments, and these functions will only permit a single argument. In this case, wrap it in a new function that hardcodes the argument(s) you want to fix. For example, to compute the fifth percentile, you need to use `quantile`, which takes two argument–the data and the quantile. Here is a way to define a function that just computes the 5th percentile:

```
q05 <- function(x){quantile(x,.05)}
> q05(runif(1000))
       5%
0.0480189
```

Doing this allows you to use a function that takes multiple arguments in an `apply` function, which requires only one argument.

```
matrix(runif(100,4,25))
mat <- matrix(runif(100),4,25) ##a 4 row x 25 column matrix of random numbers
apply(mat,1,q05)               ##get the 5-percentile from each row:
[1] 0.18653521 0.07055179 0.10283215 0.07423221
```

3.1.3 Nameless (Lambda) Functions

In programming theory, the lambda function has a history that goes back to the early days of logic, and plays an important role in computability theory and systems of logic called "lambda calculus". In R, it is typically used as a shortcut, but can be incredibly powerful in a number of cases.

The key: *YOU DON'T NEED TO NAME A FUNCTION IN ORDER TO USE IT.*

Multiply the numbers in each column by 2:

```
set.seed(111)
a <- runif(10)
b <- runif(10)+a
c <- runif(10)+ b
x2 <- data.frame(a=a,b=b,c=c)

> lapply(x2, function(x){  x * 2})
$a
 [1] 1.18596257 1.45296224 0.74084401 1.02984766 0.75532643 0.83667465
 [7] 0.02131569 1.06459048 0.86432123 0.18736304

$b
 [1] 2.2975224 2.6334192 0.8751263 1.1249434 1.0677315 1.7295302 0.3642031
 [8] 2.9976591 1.4856541 1.4162958

$c
 [1] 3.159644 3.204473 1.559429 1.898199 3.002786 2.373584 1.670662 3.564266
 [9] 3.060510 2.608137

> sapply(x2, function(x){  x * 2})
                a         b        c
 [1,] 1.18596257 2.2975224 3.159644
 [2,] 1.45296224 2.6334192 3.204473
 [3,] 0.74084401 0.8751263 1.559429
 [4,] 1.02984766 1.1249434 1.898199
 [5,] 0.75532643 1.0677315 3.002786
 [6,] 0.83667465 1.7295302 2.373584
 [7,] 0.02131569 0.3642031 1.670662
 [8,] 1.06459048 2.9976591 3.564266
 [9,] 0.86432123 1.4856541 3.060510
[10,] 0.18736304 1.4162958 2.608137
```

Here is an example where we find the mean of the numbers in each column of `x2`, using `sapply` and `lapply`:

```
lapply(x2, function(x){mean(x)})
$a
[1] 0.4069604

$b
[1] 0.7996042

$c
[1] 1.305085

sapply(x2, function(x){mean(x)})
        a         b         c
0.4069604 0.7996042 1.3050845
```

3.2 Conditional Branching

You can execute code conditionally with several methods The simplest and most common of these is the `if()` statement. `if()` executes code within brackets only when the argument in parentheses evaluates to `TRUE`. Any non-zero value is equivalent to `TRUE`

```
if( 0 )
{
  print("This will never print")
}
>

if(1)
{
  print("But this will")
}
>
[1] "But this will"
```

Often, you want to take two different actions, one depending on whether something is true, and another if it is false. To start with, we can choose one of either `T` or `F` using the `sample` function.

```
choice <- sample(c(T,F),1)

a <- NA
b <- NA

if(choice)
{
    print("choice was true")
    a <- 1100

} else {

     print("choice was false")
     b <- 13200
}

[1] "choice was false"
>
> a
[1] NA
> b
[1] 13200
```

You can also chain multiple `if`s together. We will put this inside a function (which we covered previously). Notice that the later else sections never get tested or evaluated if an earlier one is true. So in the case below, a value of 3 will match the first condition, and never get to the third condition.

```
test <- function()
{
 choice <- sample(c(1,2,3,16,40,80),1)

 print(paste("choice was:",choice))
 if(choice > 25 & choice < 90)
```

```
 {
  print("one")
 }else if(choice < 50)
 {
  print("two")
 }else if(choice==3){

   print("three")
 }else{
     warning("This code should never execute")
  }
}

> test()
[1] "choice was: 16"
[1] "two"
> test()
[1] "choice was: 1"
[1] "two"
> test()
[1] "choice was: 1"
[1] "two"
> test()
[1] "choice was: 80"
[1] "one"
> test()
[1] "choice was: 3"
[1] "two"
> test()
[1] "choice was: 80"
[1] "one"
```

3.2.1 Alternatives to if statements

An `if` + `else` chain can handle nearly any situation, but there are a few other similar functions that make things simpler at times. These include the `ifelse`, `switch`, and `which` functions, which are described below

The `ifelse` function

`ifelse` is a function, rather than an R keyword. It works like the if function in most spreadsheets. It will return the second argument if the first argument is true, and otherwise the third argument.

```
x <- sample(c(1,2),1)

##The hard way:
if(x==1)
 {
    y <- "one"
 } else {
    y <- "two"
 }

##The easy way:
y <- ifelse(x==1,"one","two")
```

Recoding vectors using `ifelse`

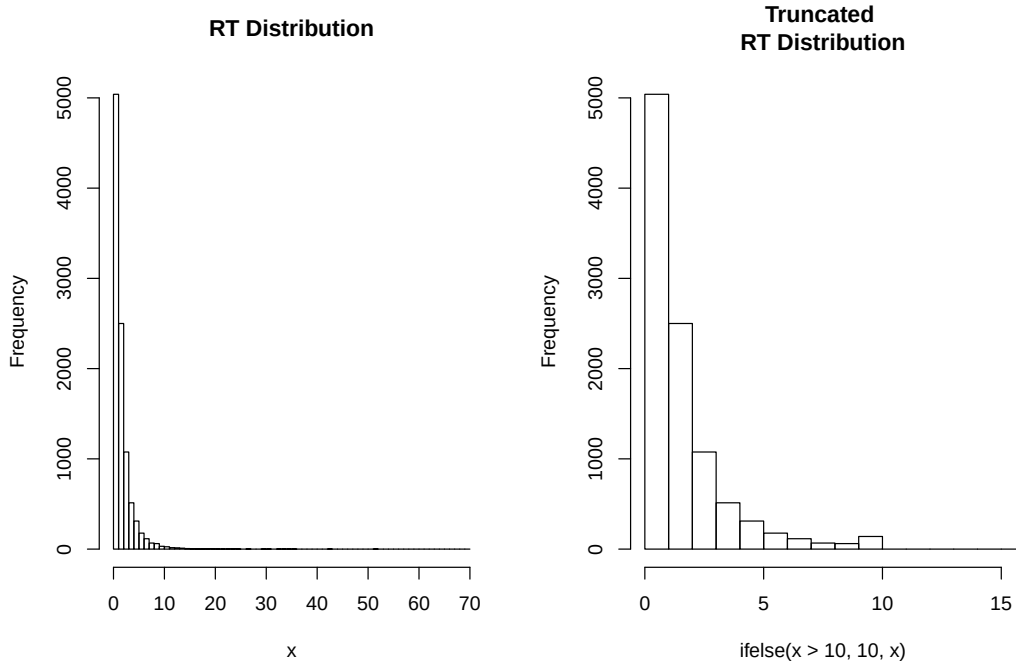
The nice part of ifelse is that it can recode all the elements of a vector in one statement:

```
 x <- sample(c(1,2),10, replace=T)
2 print(x)
 [1] 1 1 1 1 2 1 2 2 1 1
4y <- ifelse(x==1,"one","two")
  y
6 [1] "one" "one" "one" "one" "two" "one" "two" "two" "one" "one"
```

This can come in very handy for easily recoding outliers. For example, suppose I have a task where I recorded response times, and know that any reasonable response should take ten seconds or less. But in my data, there were a handful of responses between 10 and 60 seconds–perhaps the participant was taking a break before continuing on. The actual responses may still be valuable, and so maybe I will recode response times so with a ceiling, so that anything with a time greater than, say, 10 seconds will get recoded to 10 seconds. Notice the tiny bump at 10 for the recoded data in Figure 3.2.

```
x <- exp(rnorm(10000)) #Create a fake RT distribution
2 par(mfrow=c(1,2))
hist(x,breaks=0:70,main="RT Distribution")
4 hist(ifelse(x>10,10,x),breaks=0:60,xlim=c(0,15),
      main="Truncated \nRT Distribution")
```

Figure 3.2: Two histograms for a response time distribution. The right histogram recodes all values greater than 10 to be equal to 10.



`switch`: a one-liner to pick one of multiple conditions

The `switch` function is like a multi-option if statement. It takes many arguments, and the first indicates which of the other argument paths it should take. If the argument is 1, it will go to the second argument, 2 it will go to the third, and so on. Or you can specify the matching values using the = sign, which makes coding fairly easy. However, unlike `ifelse`, `switch` does not work on vector arguments, and so you must use a loop to recode multiple values.

```
x <- sample(1:5,1)
print(x)
switch(x, "one","two","three","four","five")

x <- sample(letters[1:5],1)
print(x)
switch(x, a="one",b="two",c="three",d="four",e="five")
```

`which`: select indices of a vector that match

The `which` function does not execute different branches based on the value, but it will help you select the indices of a vector that match a value. Thus, it can also be used for selective recoding.

```
##this changes lowercase a to uppercase:
x <- sample(letters[1:5],10,replace=T)
x[which(x=="a")]<-"A"
```

Exercise 3.2.1

- Write a function that will take as its first argument a data vector (e.g., something produced by `runif(1000)`), and as its second argument a keyword which tells the function whether to plot a histogram or a scatterplot.

3.3 Iteration and Looping

R allows several methods for looping. Looping or iterating over a data frame is usually the slowest way to process data. You are almost always better off just computing a function directly on the data series, or using something like apply, which we have already learned about. But sometimes slow iteration is necessary...

The simplest is the `while` loop: *Remember: you need to find your own way out of a `while` loop.*

```
continue <- T
i <- 1
while(continue)
```

```
{
   x <- runif(1)
   print(paste(i,"--", x))
   if(x>.99)
   {
     continue <- F
   }

   i <- i + 1
}
```

You can also get out of a while loop using the break keyword

```
while(T)
{
   x <- runif(1)
   print(paste(i,"--", x))
   if(x>.99)
   {
      break
   }
  i <- i + 1
}
print(i)
```

The `while(T)` capability is essentially the same as `repeat`:

```

repeat
{
   if(runif(1)<.1)
     {
        break
     } else {
      cat(".")
     }
}
cat("\n")
```

3.3.1 The `for` loop

The most flexible and common loop is the `for()` loop. it iterates over a set of values within a list.

```
x <- rnorm(100)
for(i in 1:100)
{
  print(paste(i,x[i]))
}
```

Iterate over lists

But it will iterate over any list, not just lists of integers

```
x <- c("one","two","three","four","five")
for(i in x)
{
  print(i)
}
[1] "one"
[1] "two"
[1] "three"
[1] "four"
[1] "five"
```

Iterate through variable names of a data frame

```
set.seed(111)
a <- runif(100)
b <- runif(100) + a
c <- runif(100) + b
x2 <- data.frame(a=a,b=b,c=c)

for(i in x2)
  print(mean(i))

[1] 0.5174984
[1] 0.9647387
[1] 1.469686
```

Iterate through subsets of a factor

Let's add a factor (which could represent an independent variable). Here, we code a group that will depend on the values of `x$c`.

```
x2$group <- as.factor((x2$c>1) +(x2$c>2) + (x2$c>2.5))

par(mfrow=c(2,2))
for(i in levels(x2$group))
{
    tmp <- x2[x2$group==i,]
    plot(tmp$a,tmp$b,ylim=c(0,2),xlim=c(0,1))
}
```

Other loops you may sometimes use include `lapply` and `sapply`, which we already discussed, as well as the simple loop construct `repeat`

1. `repeat: like while(T)`
2. `lapply`: apply a function over a list
3. `sapply`: apply a function over a list;

The functions `lapply` and `sapply` are similar, and are like the `for` example above except they collect the values of the function into another data structure:

```
a <- runif(100)
b <- runif(100)+a
c <- runif(100)+ b
x3 <- data.frame(a=a,b=b,c=c)
>lapply(x3,mean)
$a
[1] 0.4791484

$b
[1] 0.988161

$c
[1] 1.437425
>sapply(x3,mean)
        a         b         c
0.4791484 0.9881610 1.4374250
```

In some ways, these are like the `apply` function reviewed in the previous chapter, except they only apply a function along the different elements of a data frame. Functions such as `apply`, `lapply`, and `sapply` tend to be much faster than a function using explicit looping like `for`, `while`, and `repeat`. For the most part, this increased speed will only have an impact when you have very large data sets (thousands or millions of data points), but it can make enough of a difference that you should be aware of it.

Exercise 3.3.1a

- Write a new mean function that does not return an error when given a factor. Rather, it returns the modal (most common) value of that factor. Then use that function in the `lapply` and `sapply` on `x2`. Use

```
x2 <- data.frame(a=runif(100),b=runif(100),c= as.factor(sample(
    LETTERS,100,replace=T)))
```

- Create a series of 1,000,000 letters of the alphabet using `items <- sample(letters,1000000,replace=T)`. Write functions that will replace all 'a' values with an 'A', and b with a 'B'. Write one that uses an `if` statement, on that uses `ifelse`, and one that uses `which`. Run the function inside a `system.time()` statement, and see which is the fastest.

Exercise 3.3.1b

To find the row means of the first three columns of x2, you can use the command `apply(x2[,1:3],2,mean)` But the mean function won't work if you have `NA` values. To test this, turn any value of `x2$a` that is less than .5 into an NA value, with an statement like:

```
x2[x2<.5] <- NA
```

Do this, then find the row sums of a, b, and c using `apply`, and a lambda (nameless) function that ignores `NA` values.

3.4 Summary

This chapter reviewed basic programming logic available within R, including defining/using functions, conditional logic, iterating, looping and alternatives to functional program structure.

R functions introduced in this chapter:

- `function`
- `length`
- `lapply`
- `sapply`
- `for`
- `while`
- `repeat`
- `break`
- `if`
- `else`
- `switch`
- `which`
- `ifelse`

3.5 Solutions to Exercises

Exercise 3.1 Solution

Make a function that computes the min, median, and max of a data set `x` *and returns them as a list of three elements. Create some fake data from at least two different distributions and test it out.*

```
mmm <- function(x)
{
   min <- min(x)
   med <- median(x)
   max <- max(x)
   c(min,med,max)
}

##See how these change when you have more samples:
mmm(1/runif(1000))
mmm(1/runif(100))

data <- runif(100) + 1/rnorm(100)
mmm(data)
```

Exercise 3.1 Solution

Create a `logplot()` *function that plots log(y) versus x. Test it on a new y.*

```
logplot <- function(x,y)
{
   plot(x,log(y))
}

x <- runif(1000)
y1 <- exp(x * 6 + runif(1000))
y2 <- exp(x * 6 + runif(1000)) + runif(1000)*100
par(mfrow=c(1,2))
plot(x,y1)
logplot(x,y1)

plot(x,y2)
logplot(x,y2)
```

Exercise 3.2.1 Solution

Write a function that will take as its first argument a data vector (e.g., something produced by `runif(1000)`), and as its second argument a keyword which tells the function whether to plot a histogram or a scatterplot.

```
myplot <- function(x,type="scatter")
{
  if(type=="scatter")
  {
    ##Plot a regular plot here
    plot(x)
  }else if(type=="histogram")
    {
      ##Plot a histogram here
      hist(x)
  }else{
     warning("Unable to create specified plot type")
  }
}

myplot(x,"histogram")
myplot(x,"scatter")
```

Exercise 3.3.1a Solution

Write a new mean function that does not return an error when given a factor. Rather, it returns the modal (most common) value of that factor. Then use that function in the ***lapply*** *and* ***sapply*** *on* ***x2****.*

```
a <- runif(100)
b <- runif(100)+a
c <- sample(LETTERS,n=100,replace=T)
x2 <- data.frame(a=a,b=b,c=c)

newmean <- function(data)
{
  if(is.factor(data))
  {
      tab <- table(data)
      names(tab)[which.max(tab)]
  } else {
    return(mean(data) )
  }
}

newmean(x2)
```

Create a series of 1,000,000 letters of the alphabet. Write functions that will replace all 'a' values with an 'A', and 'b' with a 'B'. Write one that uses an ***if*** *statement, one that uses* ***ifelse****, and one that uses* ***which****. Run the function inside a* ***system.time()*** *function, and see which is the fastest.*

```
items <- sample(letters,1000000,replace=T)
f1 <- function(items)
{
   for(i in 1:length(items))
   {
     if(items[i]=="a")
       items[i] <- "A"
     if(items[i] =="b")
       items[i] <- "B"
   }
  return(items)
}
f2 <- function(items)
{
 return (  ifelse(items=="a","A",
                  ifelse(items=="b","B",items)))
}

f3 <- function(items)
{
  items[which(items=="a")] <- "A"
  items[which(items=="b")] <- "B"
  return (items)
}
system.time(f1(items ))
system.time(f2(items))
system.time(f3(items))
```

Exercise 3.3.1b Solution

Find the row sums of ***x2$a***, **b**, *and* **c** *using* ***apply***, *and a lambda (nameless) function that ignores* ***NA*** *values.*

```
x2[x2<.5] <- NA
apply(x2[,1:3],2,function(v){sum(v,na.rm=T)})
```

Chapter 4

Graphics Basics

In this chapter, we will look at using some of the primitive graphics functions–some of which we have already used a bit, to create more interesting graphical displays of data. This chapter will focus on using the following built-in R graphics:

- `plot` A workhorse plotting function.
- `points` Overlay plotting.
- `paste` For combining text.
- `matplot` Plotting a matrix
- `abline` Plotting a line by slope/intercept
- `hist` Histogram
- `segments` Plotting a line segment or segments
- `boxplot` Box plot for visualizing distribution
- `image` Visualize a matrix.
- `barplot` Height of columns
- `text` Plot text in x,y location(s)

This chapter does not provide a complete description of each function. Rather, it demonstrates some of the ways each one can be used. You should consult the R help documentation for details about the arguments available for each graphing function. For each example graphics function, we will discus when it should be used, when it should not, and ways to customize it to help you understand your data better.

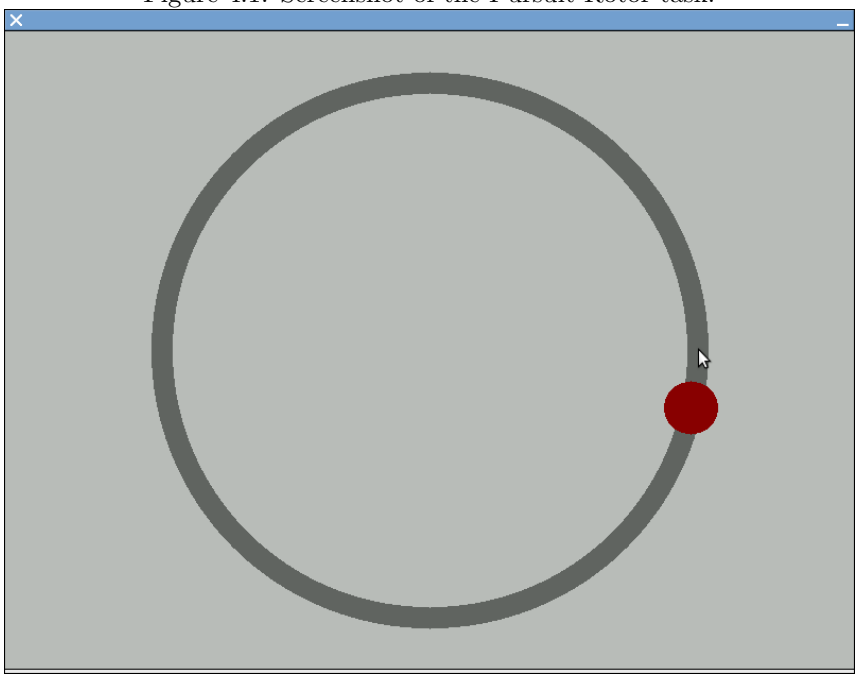
4.1 Cumulative Example: Plotting trials of a multi-trial experiment

One of the great things about R graphics is that you can create your own custom plotting function for your data. Once you customize the function on one condition, you can apply it to all conditions and have identical graphs to compare. We have previously looked at a number of components of plotting, including aggregating, filtering, looping, and simple graphics functions. Our first graphics example will incorporate many of the skills learned so far, to create a way of plotting a small subset of a larger data set. This type of graphic has many uses, including initially looking at data to get a sense for what is going on, as well as publication-ready graphics that give your paper or report a common graph and theme throughout.

4.1.1 The Experiment

. The data we have been reading in is from several trials of a 'Pursuit Rotor' task experiment. (see http://sourceforge.net/apps/mediawiki/pebl/index.php?title=Pursuit_Rotor). On each trial, the participant uses a mouse to follow the target. On every screen refresh (usually about 60 times/second), the current mouse and target location are recorded, as well as information regarding how far off-target the participant is, and whether the cursor is on target. A screenshot is shown in Figure 4.1

Figure 4.1: Screenshot of the Pursuit Rotor task.



Goal: I want to create a function that will plot a single trial of the task in the data, and then allow me to plot all the trials on a single page, and all participants in a single document. We will start by trying to determine a sequence of commands that will achieve this, when satisfied, we can put this in a function.

The data cane be read in from the provided data file:

```
dat2 <- read.csv("pooled-pursuitrotor.csv")
```

First, let's look at the data frame to make sense of it.

```
head(dat2)
     subnum trial steps timeNow timeElapsed targX targY mouseX mouseY ontarget
  1  12887     1     0   17062           0  1213   540   1213    540        1
  2  12887     1     1   17082          20  1212   544   1213    540        1
  3  12887     1     2   17101          39  1212   548   1213    540        1
  4  12887     1     3   17120          58  1212   552   1213    540        1
  5  12887     1     4   17139          77  1212   556   1213    540        1
  6  12887     1     5   17158          96  1212   560   1213    540        1
    tdiff totaltime     diff totaldev
  1     0         0  0.00000  0.00000
```

```
11   2    20         20  4.12311  4.12311
     3    19         39  8.06226 12.18540
13   4    19         58 12.04160 24.22700
     5    19         77 16.03120 40.25820
15   6    19         96 20.02500 60.28320
```

Notice that the first column indicates a participant code. The timeNow and timeElapsed columns record a timer state, in ms, and so with recordings every 17 ms or so, that is about 60 recordings per second (in sync with the refresh rate of the screen). How many participants do we have and how many observations per participant?

```
1  table(dat2$subnum)

3  12818 12837 12841 12861 12870 12887 12896 12916 12931 12938 12962 12963
   3542  3939  3642  3907  3518  3697  3713  3673  3723  3478  3535  3424
```

The numbers 12818, 12837, etc indicate participant codes–there are 12 distinct participants in this data set. Each participant has roughly 3500-4000 samples spread across trials. How are these distributed across trials?

```
   > table(dat2$subnum,dat2$trial)
2
          1   2   3   4
4  12818 896 898 872 876
   12837 989 987 980 983
6  12841 901 920 930 891
   12861 972 973 978 984
8  12870 895 881 887 855
   12887 907 922 933 935
10 12896 930 959 936 888
   12916 934 876 930 933
12 12931 933 917 935 938
   12938 908 876 834 860
14 12962 875 892 936 832
   12963 830 860 871 863
```

About 900 samples per trial. I'd like to isolate a single trial, which is thus going to be about 900 rows of the data frame.

```
1  tmp <- dat2[dat2$trial==1 & dat2$subnum==12818,]
   dim(dat2)
3  dim(tmp)
```

See how tmp is 896 lines–it now consists of a single time series from a single trial, sampled at roughly equal times. Let's see how often and how regular. The tdiff variable records the time between each sample, although we could compute that by hand if we wanted to (see Figure 4.2).

```
1  table(tmp$tdiff)
     0   16   17   19   20
3    1  292  559   43    1
```

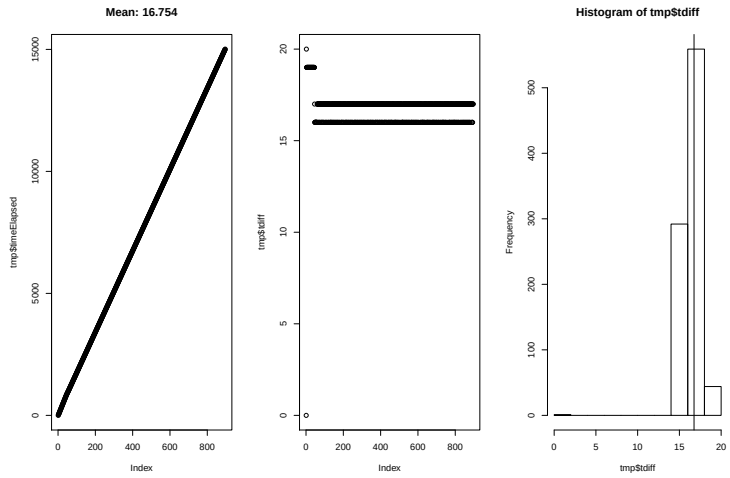
It looks like each record is about 16-17 ms apart. There are a few that are 19 apart and 1 that is 20, and 1 that is 0 (presumably the first recording). Thus, it is probably OK to treat each recording as a regular time series. Looking at the time-differentials more carefully:

```
par(mfrow=c(1,3))
plot(tmp$timeElapsed, main= paste("Mean:" , round(mean(tmp$tdiff),3))
plot(tmp$tdiff)
hist(tmp$tdiff,breaks=50,xlim=c(10,50))
abline(v=mean(tmp$tdiff))
```

The first plots absolute time in series. It looks like a straight line. The second plots the diff. We see that the 'bad' values occurred at the very beginning, which seems reasonable as the computer started working. A histograph illustrates these delta values.

We can use the main title for easily displaying some statistics we compute. Here, the mean tdiff is interesting. But `mean(tmp$tdiff)` has at least 10 digits displayed, so we can round it off to 3 significant digits using the `round` function. Then, we can combine it with a label using the `paste` function, which combines multiple text or text and numbers into one character string. By default, it separates the elements you paste together with a space, which you can change using the `sep` argument.

Figure 4.2: Timing of the Pursuit Rotor data. Each value shows the time between consecutive samples.

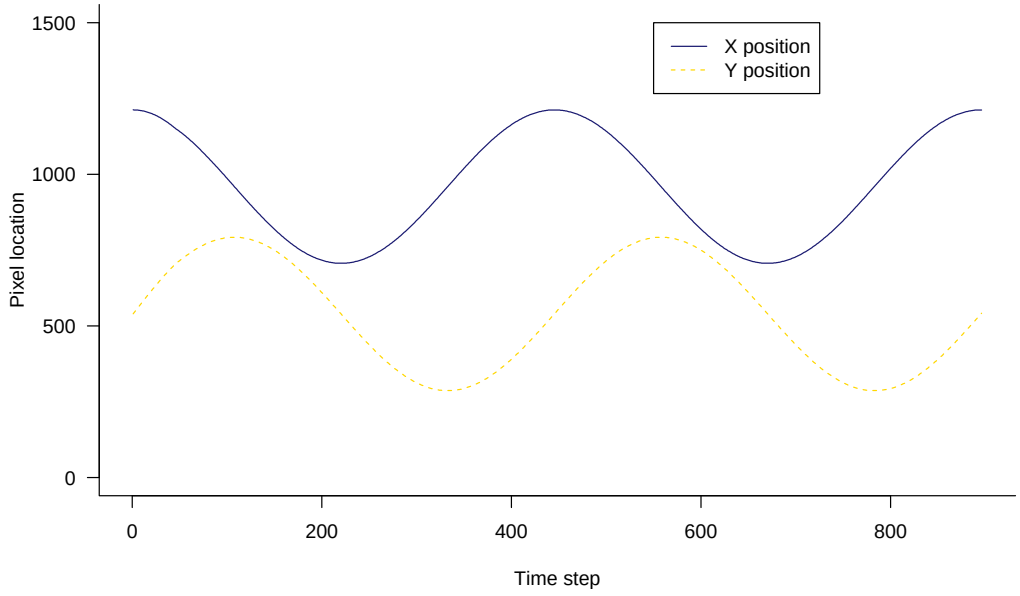


Looking at the absolute time, it seem fairly consistent. But computing deltas between observations, we see that it fluctuates between 16 and 17 ms.

Next, let's do some initial plotting of the mouse coordinates. We can look at the x and y coordinates of the target over time using `matplot`. This function takes a matrix of values and plots each column as a separate series. Notice that matplot takes most of the same arguments that plot takes. In the default plot, the vertical axis gets crowded and one of the labels is removed automatically. We can rotate the direction of labels using the `las=3` argument. We can make the plot a line plot by specifying `type="l"`, and get rid of the surrounding box with the `bty="L"` argument. Plus, add axis labels. A legend will help us interpret what is going on. The result is shown in Figure 4.3

```
matplot(cbind(tmp$targX,tmp$targY),
        type="l",bty="L",las=1,
        xlab="Time step",ylab="Pixel location",
        ylim=c(0,1500))
legend(550,1500,c("X position","Y position"),lty=1:2,col=1:2)
```

Figure 4.3: Target position over time in the Pursuit Rotor data.



The data are screen positions, which are coded from the upper left corner on a 800x600 screen, and so we transform the y coordinate. The results are shown in Figure **??**.

The columns targx and targy specify the screen coordinates of the target (not the mouse) at each time step. Let's plot them against each other A screen will count pixels things from the upper left, and R plots things from the lower right, so we subtract y from its mean to reflect it vertically. We subtract the mean from x so we center it horizontally at 0, without worrying about screen size:

```
xmid <- mean(tmp$targX) ## find the x center
ymid <- mean(tmp$targY) ## find they center
plot(tmp$targX-xmid,ymid-tmp$targY,pch=1,cex=.2,
        xlim=c(-300,300),ylim=c(-300,300),
        xlab="Horizontal pixel",ylab="Vertical pixel")
```

Now, plot the mouse locations as a connected `points()` graph (with `type="l"`)

The mousex and mousey columns specify the mouse coordinates: Plot them to:

```
points(tmp$mouseX-xmid,ymid-tmp$mouseY,type="l",col="grey")
```

This makes two paths, each one circles twice. Let's connect the target to the mouse at each step. This will be about 900 line segments, plotted with a single line of code:

We could have plotted each of these segments one at a time in a for loop, but by giving it all the x/y starting and ending coordinates, it will plot all of the lines. This gives a nice visualization. The ontarget data column indicates whether the mouse was on-target on any sample. Let's use this to draw red circles; I'll do this with a for loop, checking each time, but then show the one-liner. The results are shown in Figure 4.4.

```
for(i in 1:nrow(tmp))
{
  if(tmp[i,]$ontarget)
  {
    points(tmp$mouseX[i]-xmid,ymid-tmp$mouseY[i],col="red",cex=.3)
  }else{
    points(tmp$mouseX[i]-xmid,ymid-tmp$mouseY[i],col="grey",cex=.3)
  }
}

##alternately:
points(tmp$mouseX-xmid,ymid-tmp$mouseY,
            col=c("grey","red")[tmp$ontarget+1],cex=.3)
```

This is nice, but we wouldn't want to repeat this code for every trial we want to plot. So let's put this inside a function:

```
plottrial <- function(tmp,header="")
{
  xmid <- mean(tmp$targX)
  ymid <- mean(tmp$targY)
  meanoffset <- round(mean(tmp$diff),2)
  plot(tmp$targX-xmid,ymid-tmp$targY,pch=1,cex=.2,
     xlim=c(-300,300),ylim=c(-300,300),
     main=paste(header,"\n","mean offset =", meanoffset),
     xlab="Horizontal pixel",ylab="Vertical pixel")

  #mousex, mousey specify the mouse coordinates: Plot them to:
  points(tmp$mouseX-xmid,ymid-tmp$mouseY,type="l",col="grey")

  segments(tmp$targX-xmid,ymid-tmp$targY,
           tmp$mouseX-xmid,ymid-tmp$mouseY,col="grey")

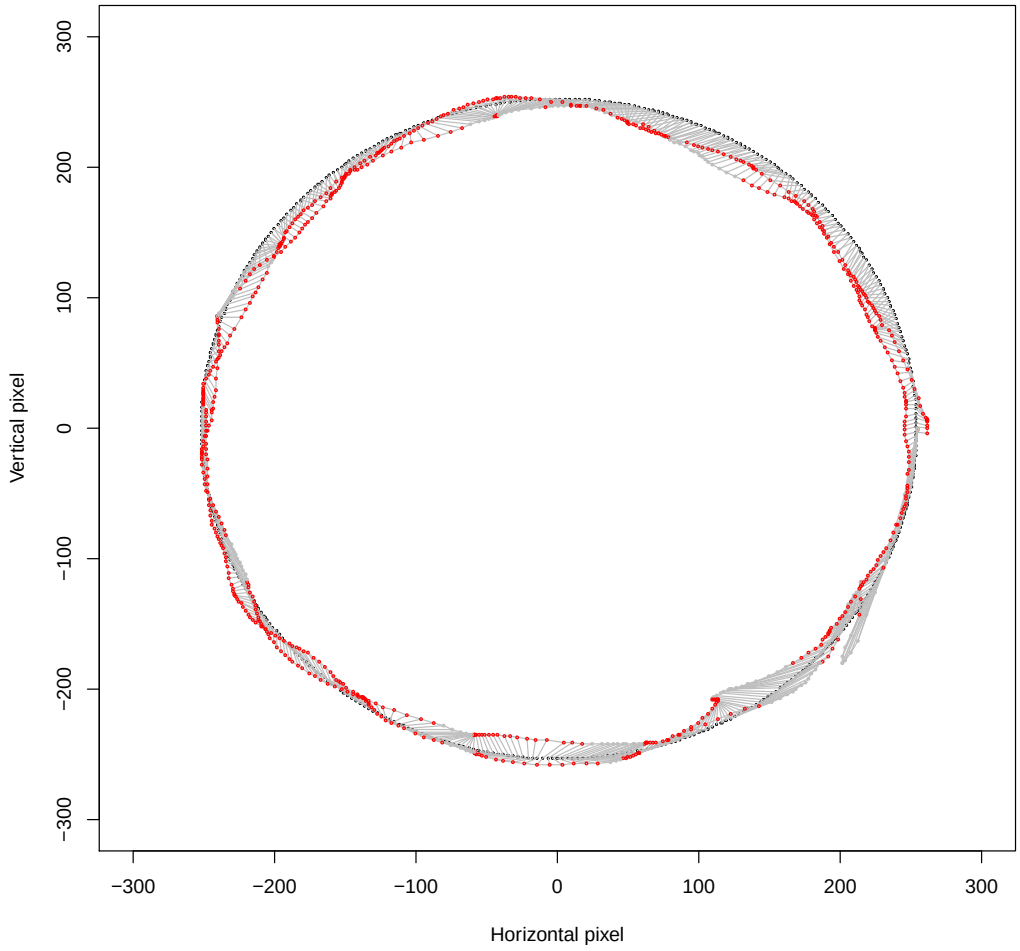
  ##I'll the a faster points method than we did above:
  points(tmp$mouseX-xmid,ymid-tmp$mouseY,
        col=c("grey","red")[tmp$ontarget+1],cex=.3)
}

plottrial(tmp)
```

Now, we have a function that will plot a single trial. This can be handy for initially looking at all the data on a single screen. How can we do this? First, let's see if we can pull out the individual trial numbers and iterate through them. We can try the `levels` function, which will tell us the levels of a factor, and then later use those levels to select subsets.

```
for(i in levels(dat2$trial))
{
  print(i)
}
```

Figure 4.4: The samples when the pointer was on-target are shown in red.



This didn't work–nothing printed. That is because `trial` wasn't a factor:

```
> levels(dat2$trial)
NULL
```

We need to change the trial column into a factor using `as.factor`:

```
for(i in levels(as.factor(dat2$trial)))
{
  print(i)
}
```

```
[1] "1"
[1] "2"
[1] "3"
[1] "4"
```

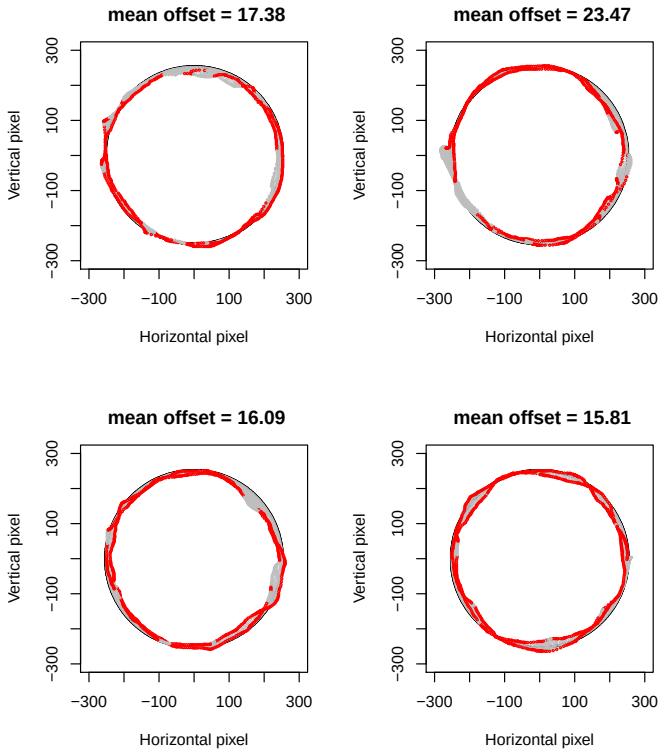
Now it will work. There are 4 trials, so let's plot them on a 2x2 grid. We will set up a loop that plots each trial, and set the printing properties so that it will display one row and two columns of plots:

```
par(mfrow=c(2,2))
for(i in levels(as.factor(sub1$trial)))
{

  tmp <- sub1[sub1$trial==i,]
  plottrial(tmp)
}
```

The results are shown in Figure 4.5. The file had just two trials, but if we had multiple trials, or multiple participants, it would be easy to save these out to a single file and examine each trial later, comparing it across conditions, participants, etc. Notice that we can tell visually that the participant was on-target much more in the later trials than the first.

Figure 4.5: Four trials of the study, each plotted using the command encapsulated within a function.



Finally, maybe I want to look at an entire set of participants. Now, I can just loop through each participant, followed by each trial. Using the `pdf()` command will place plot on a new page of a pdf document, or you can just have them output to a .docx file via RMarkdown.

```
pdf("pr-experiment.pdf",width=7,height=8)
  for(sub in levels(as.factor(dat2$subnum)))
   {
    par(mfrow=c(2,2))
    tmp <- dat2[dat2$subnum==sub,]
    for(trial in levels(as.factor(tmp$trial)))
    {
     print(paste("plotting subject:", sub, " trial: ", trial))
     tmp2 <- tmp[tmp$trial==trial,]
     plottrial(tmp2,header=paste("plotting subject:", sub, " trial: ", trial))
    }
}
dev.off()
```

A file named 'pr-experiment.pdf' should appear in your working directory that you can open and examine in detail.

The custom graphing function we created here is targetted to this specific experiment and data type. When I get a new data set that has a within-participant design like this (i.e., each participant completes the entire set of conditions) , I will almost always write a function like this that attempts to examine the main manipulations in the study for each participant. This way, I get a sense of how systematic the effect is across people, if there are any problems in the data, and so on, before doing any statistical tests.

4.1.2 Summary

In this section, we covered creating a custom graphic that you could use repeatedly on a set of data. Sometimes, a function already exists that does what you want, or that you could customize slightly to do what you need. We will cover some of these next.

Exercise 4.1.2

- Remove the x and y labels from the plot by setting xlab and ylab
- Remove the x and y axes using xaxt/yaxt argument
- Change the box type using the bty argument
- Set the main title after you create the plot so that it tells which trial

4.2 Histograms

Histograms are way to visualize a distribution of data–how many observations fall into different bins. The `hist()` function will create a histogram, and the main argument used is

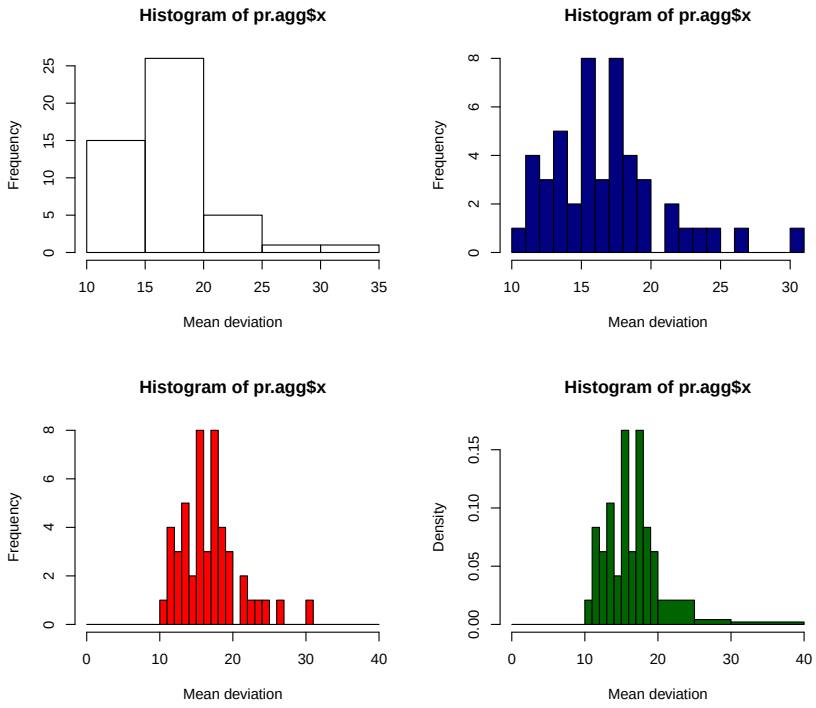
usually `breaks`. If breaks is a number, it will try to divide the data range into equal-sized bins with that many breaks between bins. The documentation says that this is a suggestion only–it will not guarantee exactly that many breaks, and if you want more control, you should compute the breaks yourself and feed them in as a vector of breakpoints. If breaks is a vector, it will use the values given as breaks–even if those breaks are not equal-sized. However, `hist()` will produce an error message if your hand-coded breakpoints do not bracket the data. That is, the smallest break must be smaller than the smallest data value, and the largest break must be larger than the largest data value. If we stay with the pursuit rotor data, we can get a mean offset by trial and subject:

```
pr.agg <- aggregate(dat2$diff,list(dat2$subnum,dat2$trial),mean)

par(mfrow=c(2,2))
hist(pr.agg$x,xlab="Mean deviation")
hist(pr.agg$x,xlab= "Mean deviation", col="navy",breaks=20)
hist(pr.agg$x,xlab= "Mean deviation", col="red",breaks=c(0:40))

##plots density instead of frequency:
hist(pr.agg$x,xlab= "Mean deviation",
       col="darkgreen",breaks=c(0,10:20,25,30,40),freq=F )
```

Figure 4.6: Four alternatives for histograms of pursuit rotor data.



Histograms also return a data structure containing the information that was computed. You can even turn the plotting off and just extract these values if you want to make your own. For example:

```
h1 <- hist(pr.agg$x,breaks=20,plot=F )
print(h1)
```

Overall, histograms are a nice way of examining the distribution of values. This can be critical in understanding which type of test or transformation to apply, and also may give you an idea about why certain things are happening.

Exercise 4.2

Error values like diff are often skewed, because they cannot be smaller than zero, they are difficult to make smaller once they are small, but they can get very large. Transform diff to its natural logarithm using log(), then make two histograms side-by-side (using par(mfrow=c(1,2))) that have a gold filled color, one with raw diff scores and one with the transformed diff scores. Make the histogram filled with gold color, and specify the breaks argument to most clearly see the distributions.

4.3 Box-and-whisker plots

A handy plotting function that helps you see the distribution of a dependent variable over a number of independent variables is called the box and whisker plot, made with the `boxplot` command. Rather than looking at each bin, this will visualize the middle, some aspect of the main body of the data (like the inter-quartile range), and lines indicating something close to the maximum and minumum of the data.

The boxplot in R is not well documented, and many aspects of the size of the box and whiskers can be customized. The center line is the median, and the boxes indicate the inter-quartile range–the 25th and 75th percentiles. The `range` parameter defaults to 1.5, which controls the extent to which the whiskers extend–by default, to the most extreme data point that does not exceed 1.5 times the inter-quartile range from the box. Any points outside this range are plotted as individual points indicating possible outlier.

We will make one based on the `OrchardSprays` data which is built-in to R. The boxplot command uses the symbol to indicate the factors by which the IV should be plotted.

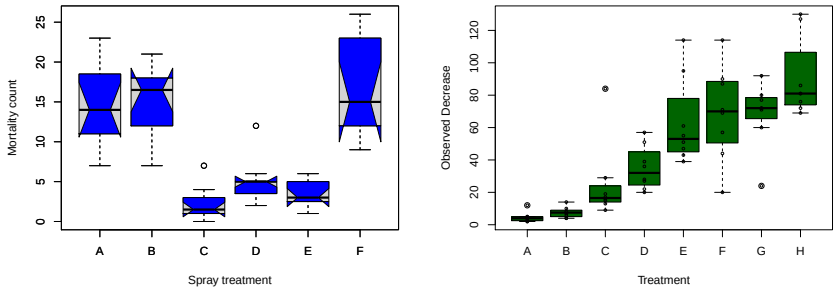
The following example (shown in the left panel of Figure 4.7) makes a boxplot of the `count` variable over the five levels of `spray`. Boxplots show the median value, the interquartile range using a box, and ± 2 inter-quartile ranges as whiskers. Any values outside these are plotted explicitly, to show potential outliers. I've overlaid two boxplots; one standard one in grey, and another blue one that depicts a secondary measure of spread. Presumably, count is a mortality rate for insects. Which ones are the best? Which ones worst?

```
boxplot(InsectSprays$count~InsectSprays$spray)

##simpler to specify a data frame and refer to columns:
boxplot(count ~ spray, data = InsectSprays,
         col = "lightgray",  xlab="Spray treatment",
         ylab="Mortality count")
#This will add a notch layer.
boxplot(count ~ spray, data = InsectSprays,
        notch = TRUE, add = TRUE, col = "blue")
```

The notch is used as a crude difference test–if notches don't overlap, this indicates that the means are likely to differ statistically.

Figure 4.7: Two boxplots, for the insectSprays and OrchardSprays data sets.



Another boxplot is also shown in the right panel of Figure 4.7. Notice how we can easily overplot the actual values, to provide additional information when appropriate.

```
boxplot(decrease ~ treatment, data = OrchardSprays,
         col = "darkgreen",xlab="Treatment",ylab="Observed Decrease")

points(OrchardSprays$treatment,OrchardSprays$decrease,cex=.5)
```

4.3.1 Advanced boxplots

Boxplots can take two IVs. The following example does not show much, but its syntax is correct. It plots treatment by rowpos, which will contain only one observation per cell. To make a boxplot, you should really have a minimum of ten points–ideally more.

```
boxplot(decrease~treatment+rowpos,data=OrchardSprays)
```

In addition to the syntax using the s̃ymbol, boxplot will work with a data frame, plotting each column. If your columns are not on the same scale, it will still plot all the boxes on the same scale. See Figure 4.8–notice how in the plot on the right, the range is compressed visually and you can't tell what is going on. You'd be better off making separate boxplots here.

```
##boxplot also works with a data.frame, plotting each column.
mallsales<-data.frame(Maurices=1000*exp(runif(365)),
   JCPenney =2000*exp(runif(365)),
   Sears = 3000*exp(runif(365)))
boxplot(mallsales,las=3,col=c("red","orange","yellow"))

birthdate <- round(1950 + runif(100)*40)
tenure <- (2010-birthdate - 20 ) - runif(100)* 10
boxplot(data.frame(birthdate,tenure),
      main="Age and years worked\n at ACME corporation",
      bty="n", cex.main=.8)
```

Figure 4.8: Example boxplots made with a data frame. Distinct variables on different ranges will still bo plotted on the same range (right panel), even if the results are not helpful.

Boxplots can be nice to give a quick look at the data, but they often hide detail. First of all, if you only have a few data points, the boxplot can hide this fact and make you think you have a much more systematic distribution than you really do. Also, skewness can be hidden, especially for distributions with long tails, which may get plotted as outliers. You may consider plotting raw data, or histograms instead. As an alternative, the violinplot provides a vertical histogram that is like a boxplot but better represents the shape of the distribution.

4.4 `image` plots

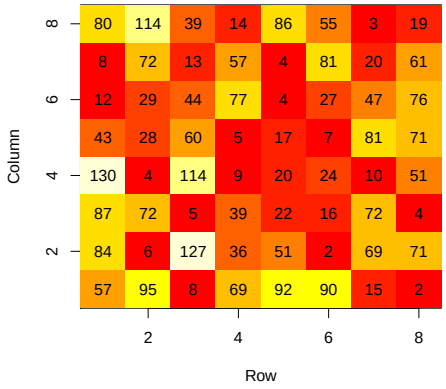
An image() plot can be useful for visualizing spatial data, or for looking at correlation matrices, or simple bitmaps. It takes as an argument a matrix (not a data frame), and so it will return an error message if you try to make an image from a data frame. You may need to convert a data frame to a matrix using `as.matrix(data)`, and the values need to be

numeric. An `image()` plot simply makes a grid from the matrix, with the color dependent on each value of the matrix. One word of caution: a matrix is plotted from the lower left-hand corner of the image. The first row of the matrix is the bottom row of the image. Be careful when labeling the axes so that you get this right.

The orchard spray example is perfect for the image plot. We have observations at each point in an 8x8 grid. To convert the tagged data to a matrix, we can use the `tapply` function.

```
layout <-   tapply(OrchardSprays$decrease,
                   list(OrchardSprays$rowpos,OrchardSprays$colpos),mean)
image(1:8,1:8,layout,xlab="Row",ylab="Column")
text(OrchardSprays$rowpos,OrchardSprays$colpos,OrchardSprays$decrease)
```

Figure 4.9: Effects of different Orchard sprays based on the physical layout of the specimens. 8 treatments were counterbalanced over each row and column.



We can create a matrix by hand as well:

```
data <- matrix(c(8,1,1,1,1,1,8,
                 1,0,0,0,0,0,1,
                 1,0,3,3,3,0,1,
                 1,0,0,0,0,0,1,
                 1,0,0,4,0,0,1,
                 1,0,0,0,0,0,1,
                 1,0,5,0,5,0,1,
                 1,0,0,0,0,0,1,
                 8,1,1,1,1,1,8)       ,7)
image(data,xlab="",ylab="")
```

Another use for image plots is to visualize a correlation matrix. Rows or columns that are highly correlated will brighter (whiter) than those uncorrelated. For example, consider the iris data set, which involves three main groups of species (see Figure 4.11).

```
data(iris)
cor(iris[,1:4])
image(cor(t(iris[,1:4])),
      main="Correlation map of iris data")
```

Figure 4.10: Image created with a matrix command.

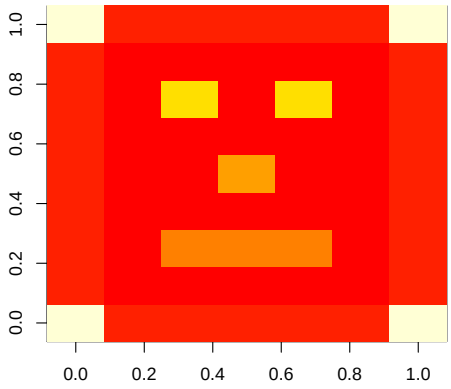
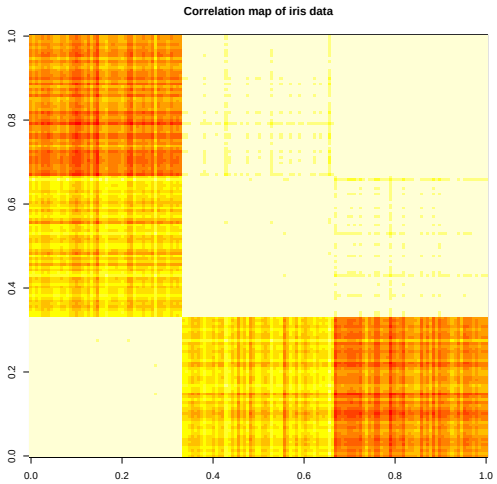


Figure 4.11: Using image to visualize the correlation of a set of irises.



Notice that there the first group (the first 1/3 of the rows or columns; the bottom left corner of the graph) is dissimilar from the others, but the second and third groups are somewhat similar. This correlation plot can be used to help cluster groups and identify similar participants or stimuli.

Yet another use is to create a 2-dimensional histogram. We will use this to visualize the density of a bivariate normal distribution. To do this, we take our raw data, and create bins so that we can count how many observations fall into each bin. A simple way of binning is to

use `ceiling()`, which will round up to the next highest number (this will be a little better than round, because the boundaries of each bin will be at the whole number). We could bin into larger bins as well. Once we create the bins, then just create a contingency table using `table()` and plot the resulting count matrix. For extra clarity, we can plot the raw data on top of the image plot.

```
##Create a random bivariate normal with correlation between x and y
x <- rnorm(500,4,1.5)
y <- rnorm(500,2,1.5) + x*.2

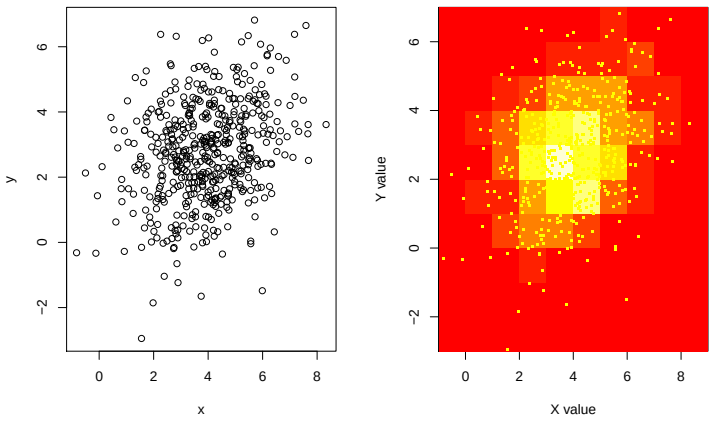
par(mfrow=c(1,2))
plot(x,y)

##Create a 2-D histogram with 1-wide cell grids.
txy <- table(ceiling(x),ceiling(y))

image(as.numeric(rownames(txy))-.5,
     as.numeric(colnames(txy))-.5,txy,xlab="X value",ylab="Y value")

points(x,y,col="yellow",pch=".",cex=3)
```

Figure 4.12: Image created showing density of bivariate normal.



Exercise 4.4

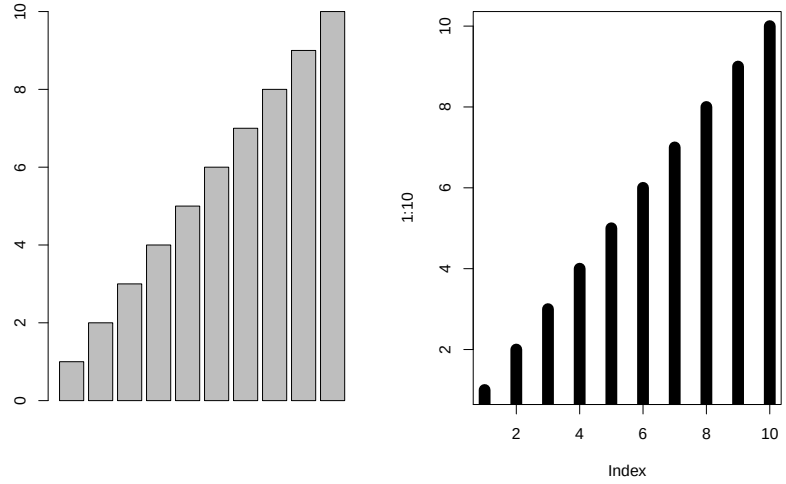
Add gridlines to an image plot using the `segments` command.

4.5 Barcharts/Barplots

Another common plot function is the barchart or a barplot. These are useful especially when you have a data type that is on a scale where the 0 value matters, because the height of the bar represents the size of the value. Most people recommend only using bar charts when your independent variable is categorical, but that advice is often violated. A simple bar chart is available the plot function, and `type="h"`, but more control is available with the barplot command:

```
barplot(1:10)
plot(1:10, type="h",lwd=3)
```

Figure 4.13: Simple bar chart examples.

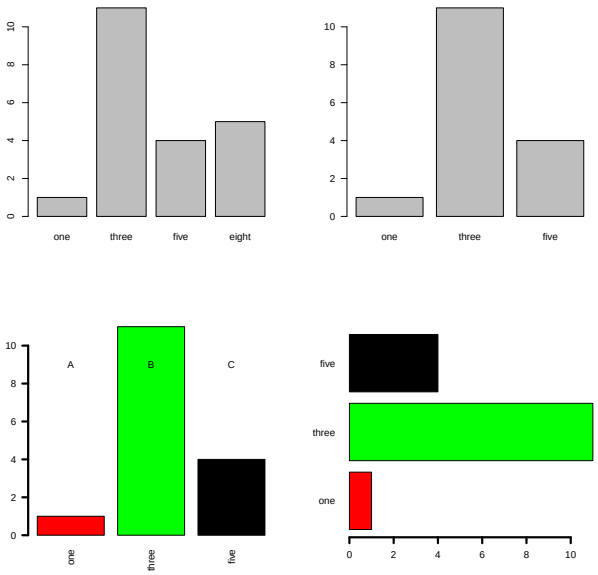


We can make a number of improvements, by adding labels to the axis, changing colors, and using `las` to change the orientation of axis labels, or try a horizontal barplot. Some examples are:

```
par(mfrow=c(2,2))
barplot(c(1,11,4,5),names=c("one","three","five","eight"))
barplot(c(1,11,4),names=c("one","three","five"),las=1)
#barplot(c(1,11,4),names=c("one","three","five"),las=2)
xs <- barplot(c(1,11,4),names=c("one","three","five"),
              col=c("red","green","black"),las=2, lwd=3)
text(xs,c(9,9,9),c("A","B","C"))
barplot(c(1,11,4),names=c("one","three","five"),
        col=c("red","green","black"),las=1, lwd=3,horiz=T)
```

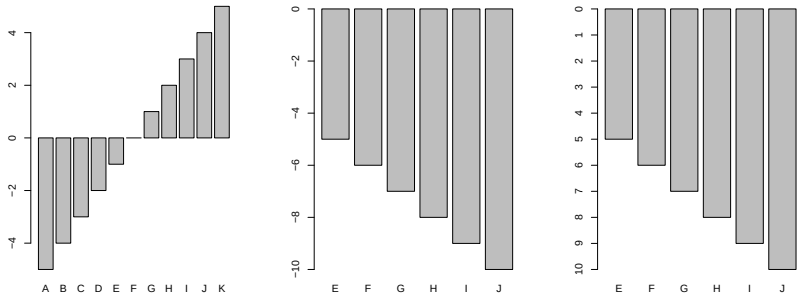
If you have a bar chart with negative values, it will also work, but produce a 'hanging' chart. When the values are close to 0, even if they are both positive and negative, everything will plot fine. You can also use the axis command to relabel your vertical axis so that negative values are labeled as positives, as shown in Figure 4.15.

Figure 4.14: More bar chart examples.




```
par(mfrow=c(1,3))
barplot(-5:5,names=LETTERS[1:11])
barplot(-5:-10,name=LETTERS[5:10])
barplot(-5:-10,name=LETTERS[5:10],yaxt="n")
axis(2,0:-10,0:10)
```

Figure 4.15: Hanging bar chart examples, with the bars dropping from the top.

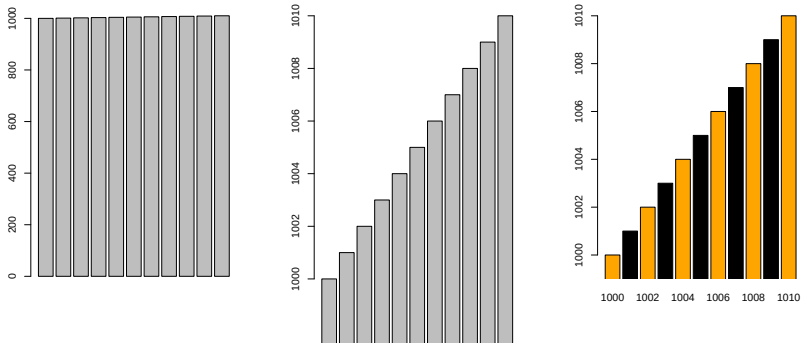


When your range of values is small and differs a lot from 0, barplots don't work well. For example, a barplot of the numbers 1000 to 1010 show up as identical. This might be a good reason to avoid a barplot, because the bar implies that the magnitude from 0 matters a lot. Maybe the Figure is showing you that the small differences is sort of not relevant. However,

you might consider zooming in the vertical range. For example, suppose this is elevation above sea level. Relative large absolute differences may be dwarfed by the baseline elevation difference from sea level, so it could make sense truncating the bottom of the y axis using a custom `ylim` argument. However, this does not usually turn out well, as in the center panel of figure 4.16. The `xpd` argument fixes this bleed-over, although you have to be careful that you are not deliberately misleading the reader–in many cases, the difference between 1009 and 1010 might not matter (e.g., average time in ms); whereas in other cases it might (year of an important historical event).

```
par(mfrow=c(1,3))
barplot(1000:1010)
barplot(1000:1010,ylim=c(1000,1010)) ##ugh
barplot(1000:1010,ylim=c(999,1010),names=1000:1010,
     col=c("orange","black"),xpd=F)  ## chops off, but might be OK
```

Figure 4.16: Bar chart examples, truncating the bottom of the graph.



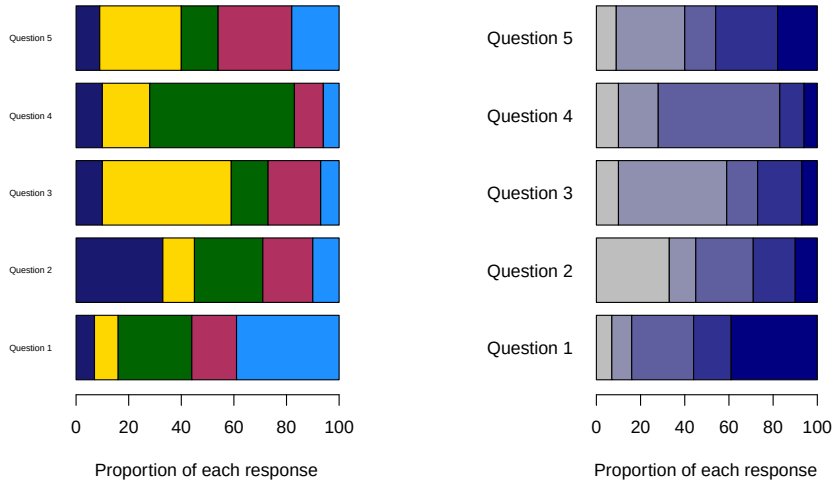
4.6 Barcharts with multiple series

Barcharts work well to compare values across multiple series. You can give the barchart a matrix of values, and it will create separate series.

One way to use this is if you a set of survey questions with common answers. Suppose you have a number of non-scalar survey questions with answers a through d–in which there is not a correct answer, and thus 'mean' does not mean anything. Instead, you might want to show the distribution of responses for each question. The following shows a five-item multiple-choice test.

```
set.seed(100)
data <- data.frame(q1=sample(letters[1:5],100,replace=T,
prob=c(1,1,3,2,5)),
q2=sample(letters[1:5],100,replace=T,prob=c(3,1,3,2,1)),
q3=sample(letters[1:5],100,replace=T,prob=c(1,5,1,2,1)),
q4=sample(letters[1:5],100,replace=T,prob=c(1,2,8,2,2)),
```

Figure 4.17: Two bar plots representing five-option survey questions horizontally.




```
q5=sample(letters[1:5],100,replace=T,prob=c(1,5,2,4,3)))

datatable<-apply(data,2,table)
> datatable
q1 q2 q3 q4 q5
a  7 33 10 10  9
b  9 12 49 18 31
c 28 26 14 55 14
d 17 19 20 11 28
e 39 10  7  6 18


par(mar=c(5,18,1,2)) ##change margins
barplot(datatable,names=paste("Question",1:5),
       col=1:5,horiz=T,las=1,
       xlab="Proportion of each response",
       cex.names=.5)
```

You can see that this created a stacked barplot. Stacked barplots imply that the absolute number of observations in each column don't really matter or are the same. This seems appropriate for a test like this, but what if you were plotting ethnicity breakdown by state. Looking at just five states for convenience, (see https://www.ojjdp.gov/ojstatbb/ezacjrp/asp/State_Race.asp), we can examine "Juvenile residential placements" by state:

```
juvy <- read.table(text="State White  Black   Hispanic  Native  Asian   Other
Alabama   300   507   27  0   3   9
Alaska   78   30  3   75  3   21
Arizona   237   114   255   54  9   51
Arkansas  198   315   33  0   6   3
California  900   1863  3729  42  138   54",header=T)

cols=c("darkgreen","bisque","navy","red","violet","orange")
juvymat <- as.matrix(juvy[,-1])
rownames(juvymat) <- juvy[,1]
```

```
##normalize within each state.
juvymat.normed <- juvymat/rowSums(juvymat)*100

par(mfrow=c(1,2),mar=c(8,5,3,0))

barplot(t(juvymat),las=3,legend=T,
        args.legend=list(x=4,bty="n"),
        ylab="Number of residents",
        col=cols, main="Juvenile Residential Placements")

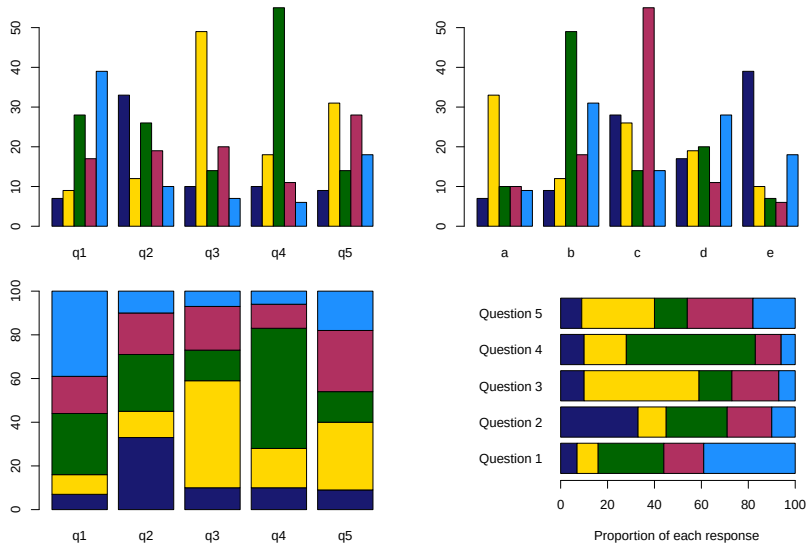

barplot(t(juvymat.normed),las=3,legend=F,
        ylab="Percentage of residents",
        col=cols, main="Juvenile Residential Placements")
```

Here, the stacked bar on the left of Figure **??** illustrates both the differential population across state and the relative size of the sub-population within each state. The stacked bar will not normalize on its own–but dividing the matrix by its rowsums will easily create this.

Sometimes you want a mini bar plot for each column (or row) of your matrix. the beside=T argument will do this. You need to be careful about rows versus columns though. If you want the opposite grouping, you can plot the transpose of a matrix using `t()`. These are shown in the left to panels of Figure 4.18.

```
barplot(datatable,col=1:5,beside=T)
barplot(t(datatable),beside=T,col=1:5)
barplot(datatable,col=1:5,beside=F)
```

Figure 4.18: Alternates for plotting survey questions. Each plot represents the same data, either by row or by column.



These first barplots are fine, but they hide the fact that the questions add up to the same number. If we use beside=F (the default), a stacked barplot will be produced, and it will show the relative proportion of each answer. Also, it is more compact. But it could be difficult to read the question, which is why a horizontal plot might be better (lower right panel). Here, you might need more room on the left margin, if you want to add the entire text of the question. Use the following command to control this. Also, you could use `mtext` to add additional text in the margin.

```
par(mar=c(5,18,1,2))
barplot(datatable,names=paste("Question",1:5),
   col=1:5,horiz=T,las=1,
   xlab="Proportion of each response")
```

Overall, bar charts permit plotting by multiple categorical variables, and usually are best when the absolute difference from 0 is meaningful. Truuncating the y axis is possible in other cases, but can be misleading.

Exercise 4.6

- For the OrchardSprays data set, compute using tapply the average effect of each treatment, each row, and each column position.
- Plot average decrease by each of these variables, on a single 1x3 plot.
- Use tapply to compute the average effect of each treatment x colpos and treatment x rowpos.
- Make two stacked barplots with treatment as the main axis, and either row or column position as the division within each bar (use the matrix just created)
- Make two stacked barplots with row or column position as the main axis, and treatment as the division within each bar. Use the same matrix you just created.
- Look across the 7 plots you created and describe the relative importance of the spray, versus the row and column position in the orchard grid.

4.7 Answers to exercises

Exercise 4.1.2 Solution

- Remove the x and y labels from the plot by setting xlab and ylab
- Remove the x and y axes using xaxt/yaxt argument
- Change the box type using the bty argument
- Set the main title after you create the plot so that it tells which trial

```
## To find out about the parameter settings for plots, try help(par)
##

plottrial <- function(tmp,header="")
{
  xmid <- mean(tmp$targX)
  ymid <- mean(tmp$targY)

  meanoffset <- round(mean(tmp$diff),2)
  plot(tmp$targX-xmid,ymid-tmp$targY,pch=1,cex=.2,
       xlim=c(-300,300),ylim=c(-300,300),
       main=paste(header,"\n","mean offset =", meanoffset),
       xlab="",ylab="",
       xaxt="n",yaxt="n",
       bty="n")

  #mousex, mousey specify the mouse coordinates: Plot them to:
  points(tmp$mouseX-xmid,ymid-tmp$mouseY,type="l",col="grey")

  segments(tmp$targX-xmid,ymid-tmp$targY,tmp$mouseX-xmid,ymid-tmp$mouseY
    ,col="grey")

  ##I'll use a faster points method than we did above:
  points(tmp$mouseX-xmid,ymid-tmp$mouseY,col=c("grey","red")[tmp$
    ontarget+1],cex=.3)


}
dat2 <- read.csv("pooled-pursuitrotor.csv")
sub1 <- dat2[dat2$subnum==12887,]
par(mfrow=c(2,2))
for(i in levels(as.factor(sub1$trial)))
{
  trial <- i
  tmp <- sub1[sub1$trial==i,]
  plottrial(tmp,header=paste("Trial",i))
}
```

Exercise 4.2 Solution

Error values like diff are often skewed, because they cannot be smaller than zero, they are difficult to make smaller once they are small, but they can get very large. Transform diff to its natural logarithm using log(), then make two histograms side-by-side (using par(mfrow=c(1,2))) that have a gold filled color, one with raw diff scores and one with the transformed diff sores. Make the histogram filled with gold color, and specify the breaks argument to most clearly see the distributions.

```
par(mfrow=c(1,2))
hist(dat2$diff,breaks=50,col="gold")
hist(log(dat2$diff),breaks=50,col="gold")
```

Exercise 4.4 Solution

Add gridlines to an image plot using the `segments` command.

```
tmpx <- rnorm(10000,mean=8,sd=1.7)
tmpy <- tmpx/2 + 3 + rnorm(10000,mean=3,sd=.7)
grid <- table(round(tmpx),round(tmpy))
image(1:nrow(grid),1:ncol(grid),grid,col=grey(1:100/100))
segments(1:20,0,1:20,10,col="red")
segments(0,1:20,20,1:20,col="red")

##this doesn't quite look right. The gridlines are down the center of
    each cell. Adjust by.5
image(1:nrow(grid),1:ncol(grid),grid,col=grey(1:100/100))
segments(1:20+.5,0,1:20+.5,15,col="red")
segments(0,1:20+.5,20,1:20+.5,col="red")
```

Exercise 4.6 Solution

- For the OrchardSprays data set, compute using tapply the average effect of each treatment, each row, and each column position.
- Plot average decrease by each of these variables, on a single 1x3 plot.
- Use tapply to compute the average effect of each treatment x colpos and treatment x rowpos.
- Make two stacked barplots with treatment as the main axis, and either row or column position as the division within each bar (use the matrix just created)
- Make two stacked barplots with row or column position as the main axis, and treatment as the division within each bar. Use the same matrix you just created.
- Look across the 7 plots you created and describe the relative importance of the spray, versus the row and column position in the orchard grid.

```
par(mfrow=c(1,3))
barplot(tapply(OrchardSprays$decrease,
               list(OrchardSprays$treatment),mean),
        main="Decrease by treatment",
        col=1:8)
barplot(tapply(OrchardSprays$decrease,
               list(OrchardSprays$rowpos),mean),
        main="Decrease by Row position",
        col=1:8)
barplot(tapply(OrchardSprays$decrease,
               list(OrchardSprays$colpos),mean),
        main="Decrease by Column position",
        col=1:8)

print(tapply(OrchardSprays$decrease,
             list(OrchardSprays$rowpos,OrchardSprays$treatment),mean))

tab2r <- tapply(OrchardSprays$decrease,
                list(OrchardSprays$rowpos,OrchardSprays$treatment),mean)

tab2c <- tapply(OrchardSprays$decrease,
                list(OrchardSprays$colpos,OrchardSprays$treatment),mean)

par(mfrow=c(2,2))
##first by treatment, then by row/column
barplot(tab2r,col=1:8,main="Treatment")
barplot(tab2c,col=1:8,main="Treatment")

##first by row/column, then by treatment:
barplot(t(tab2r),col=1:8,main="Row")
barplot(t(tab2c),col=1:8,main="Column")
```

Chapter 5

Advanced Graphics Topics

The previous chapter examined how to use some of the most common built-in graphics functions in R, and how to adapt and combine these to make custom graphics. The present chapter expands on this, covering some additional less common graphics functions and plots and functions to add adornments to graphics that are useful.

- `pie` Avoid at all costs.
- `dotchart` Preferred way of displaying categorical distributions.
- `error bars`
- `confidence intervals`
- Advanced use of the `boxplot`
- Use of bitmapped images
- Violin Plots

Libraries used in this chapter:

- gplots
- plotrix
- jpeg
- pixmap
- vioplot

5.1 Pie charts: A Bad Idea

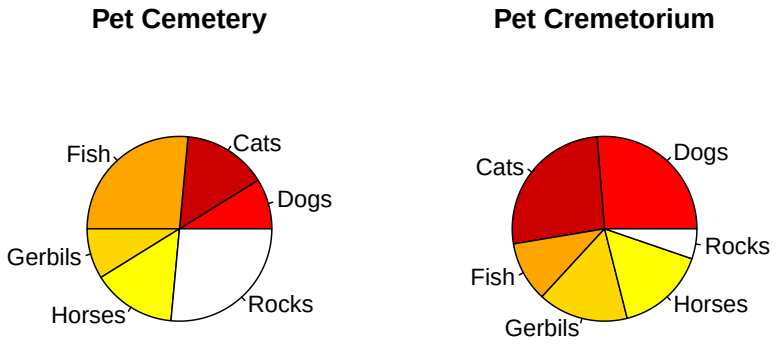
Pie charts have been rightly derided by data bloggers and researchers: http://peltiertech.com/use-dot-plots-for-better-categorical-comparisons/

R has a number of pie chart functions, but they have limitations. Here, we can see the some of the problems with a side-by-side pie chart, shown in Figure 5.1.

```
pie(c(3,5,9),c("Dogs","Cats","Fish"))

cols <- c("red","red3","orange","gold","yellow","white")
## what's worse than comparing category sizes between pie charts?
##Comparing them between two pie charts
par(mfrow=c(1,2))
pie(c(3,5,9,3,5,9),c("Dogs","Cats","Fish","Gerbils","Horses","Rocks"),
    main="Pet Cemetery",
    col=cols)
pie(c(5,5,2,3,3,1),c("Dogs","Cats","Fish","Gerbils","Horses","Rocks"),
    main="Pet Cremetorium",
    col=cols)
```

Figure 5.1: Pie charts are difficult to accurately interpret. The only thing linking corresponding areas on graphs is the name and color



.

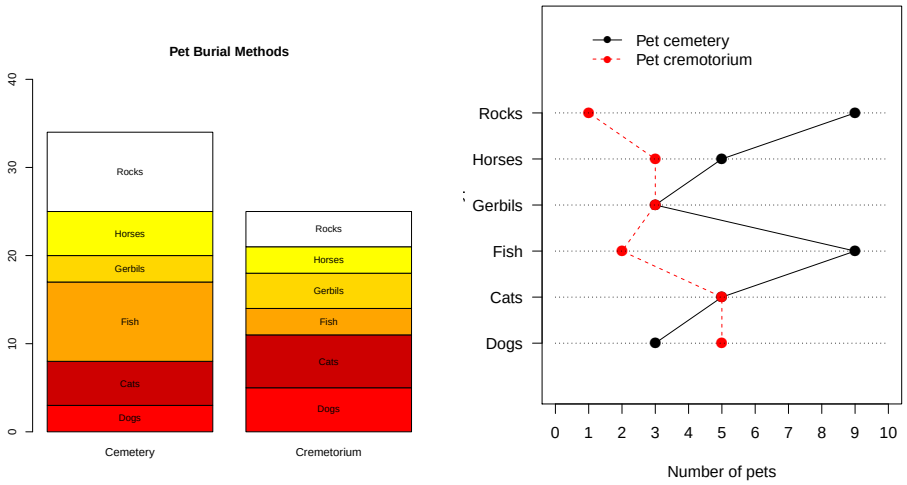
There are many problems with pie charts. They involve comparing the areas of regions that are different shapes, oriented in different directions. They make comparison difficult. They can hide the number observations in each cell. They doesn't work well when you have many categories. And 3D versions are even worse, because they distort the size of the polygon such that the amount of area on the page no longer corresponds to the value they are trying to represent. On top of all this, researchers have confirmed that people do not judge values in pie charts correctly, so don't use them. Anything that can be displayed in a pie chart could be displayed in a bar chart–even a single column stacked bar.

Here is an example of using a pair of stacked bar graphs to show distributions. I have added text directly onto stack using the text command, computing the center of each area. This avoids some of the problems with pie charts, and comparing between categories is easier, but by no means simple (see Figure 5.2).

```
##stacked bar compares area, but is easier to judge:
data <- cbind(c(3,5,9,3,5,9),c(5,6,3,4,3,4))
rownames(data) <- c("Dogs","Cats","Fish","Gerbils","Horses","Rocks")
colnames(data) <- c("Cemetery","Cremetorium")
xs <- barplot(data,legend=F,col=cols,
              ylim=c(0,40),
              main="Pet Burial Methods")

yvals <- apply(data,2,cumsum)
yv2 <- (rbind(yvals,0)+rbind(0,yvals))[1:6,]/2
text(xs[1],yv2[,1],rownames(yvals),cex=.8)
text(xs[2],yv2[,2],rownames(yvals),cex=.8)
```

Figure 5.2: Stacked barplot and dotchart are nice alternatives to pie charts



.

5.2 Dot charts: an alternative to barplots and pie charts

Anything that can be shown in a pie chart can also be shown in what people refer to as a 'dot chart'. This is a horizontally-organized matplot, with each row a different category. Especially with multiple series, it more easily permits comparing between series. The only loss in comparison to a pie chart is that a pie chart may show proportions better, as the total area is equal to 100%. We will start by showing a hand-made dot chart next to the corresponding pie charts they replace. The dotchart uses a matplot function, but puts the data matrix in the 'x' instead of y argument, and a set series of integers the 'y' argument–this is the opposite of what you usually do in a matplot, but it turns the plot sideways. Then, by drawing horizontal lines using `segments` and a few other things, we can create our own dotchart, shown in the right panel of 5.2

```
data <- cbind(c(3,5,9,3,5,9),c(5,5,2,3,3,1))
##do it 'by hand'
matplot(data,1:6,pch=16,xlim=c(0,10),yaxt="n",xaxt="n",
        xlab="Number of pets",ylab="Pet type",
        type="o",ylim=c(1,7),
        col=cols)
segments(0,1:6,10,1:6,lty=3)
axis(2,1:6,c("Dogs","Cats","Fish",
           "Gerbils","Horses","Rocks"),las=1)
legend(1,7,c("Pet cemetery","Pet cremotorium"),
         lty=1:2,pch=16,col=1:2,bty="n")
```

R has a built-in dotchart available with the `dotchart()` command. However, it plots each series vertically, instead of putting them both on the same axis values, which the above code does do. Nevertheless, the built-in dotchart can be useful. We will use it here to show the extent to which different members of the Senate voted in support of 19 issues the AFL/CIO identified as being related to union interests.

These first graphs show various settings I used to try and display the information better or more clearly. The groups function will arrange the values according to the levels of another variable–I'll use party affiliation. Note that dotchart REQUIRES a factor for the grouping variable. As of recent versions of R, string variables are not automatically converted to factors, so we need to change the V2 variable of x to be a factor. We could do it when we read it in, or just when we use it in dotchart.

```
x <- read.table("aflcio-votes.txt")
votes <- x[,3:21]
senator <- paste(x$V1,x$V2)
votes2 <- rowSums(votes=="R")  ##Recode for voting 'Right'
dotchart(votes2)
dotchart(votes2,labels=senator)
dotchart(votes2,labels=senator,groups=as.factor(x$V2))
dotchart(votes2,labels=senator,groups=as.factor(x$V2),cex=.5)
```

At this scale, the senator names are almost readable. It might be interesting to ignore party affiliation at first, and just look at the extent to which each senator is in support of union issues. We can then use color to indicate party affiliation.

```
ord <- order(votes2)
dotchart(votes2[ord],labels=senator[ord],cex=.5,
         col=c("blue","yellow","red")[x$V2[ord]],
          pch=15)
```

Figure 5.3: Iterative changes to the `dotchart` function to improve layout.

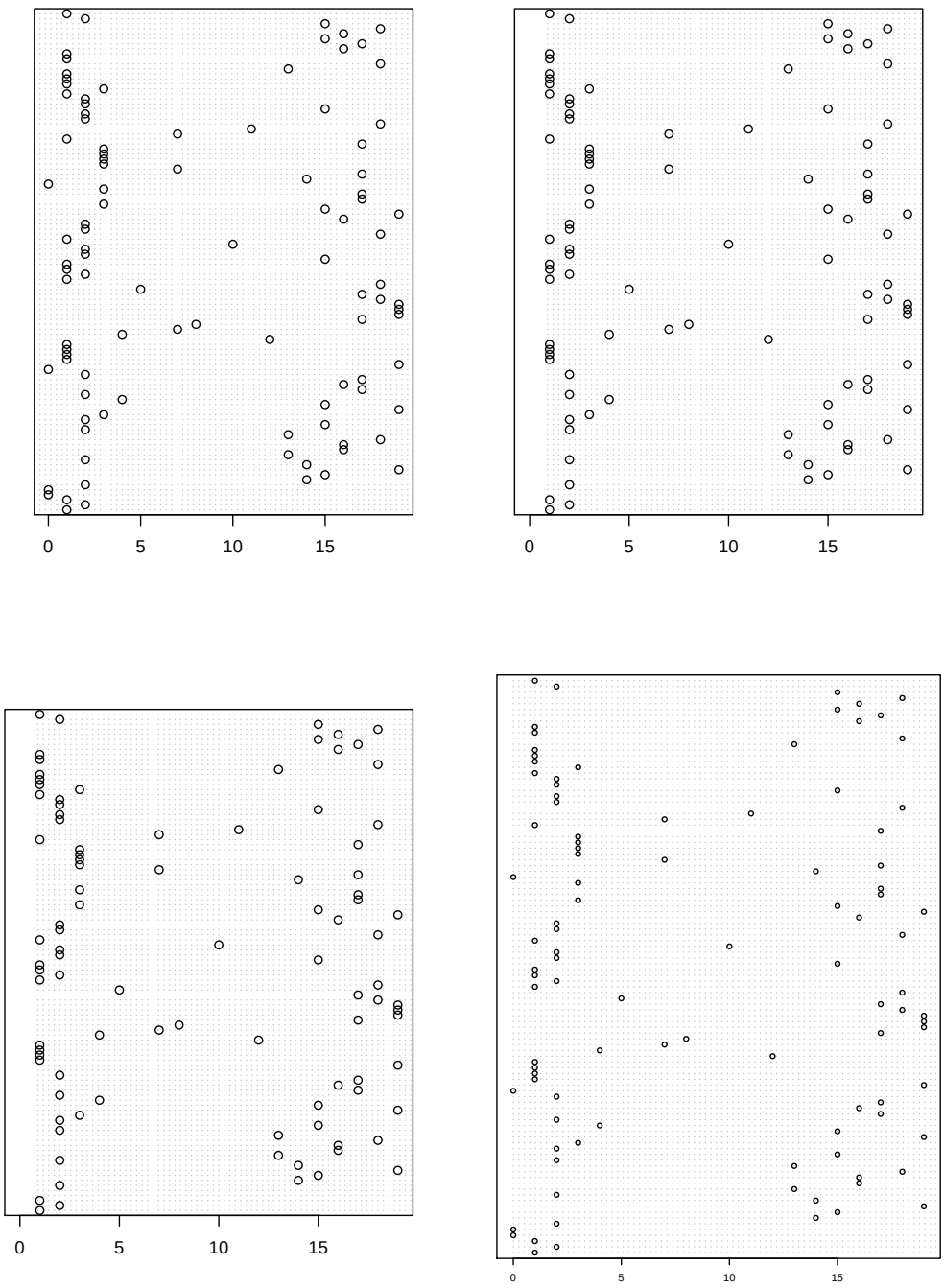
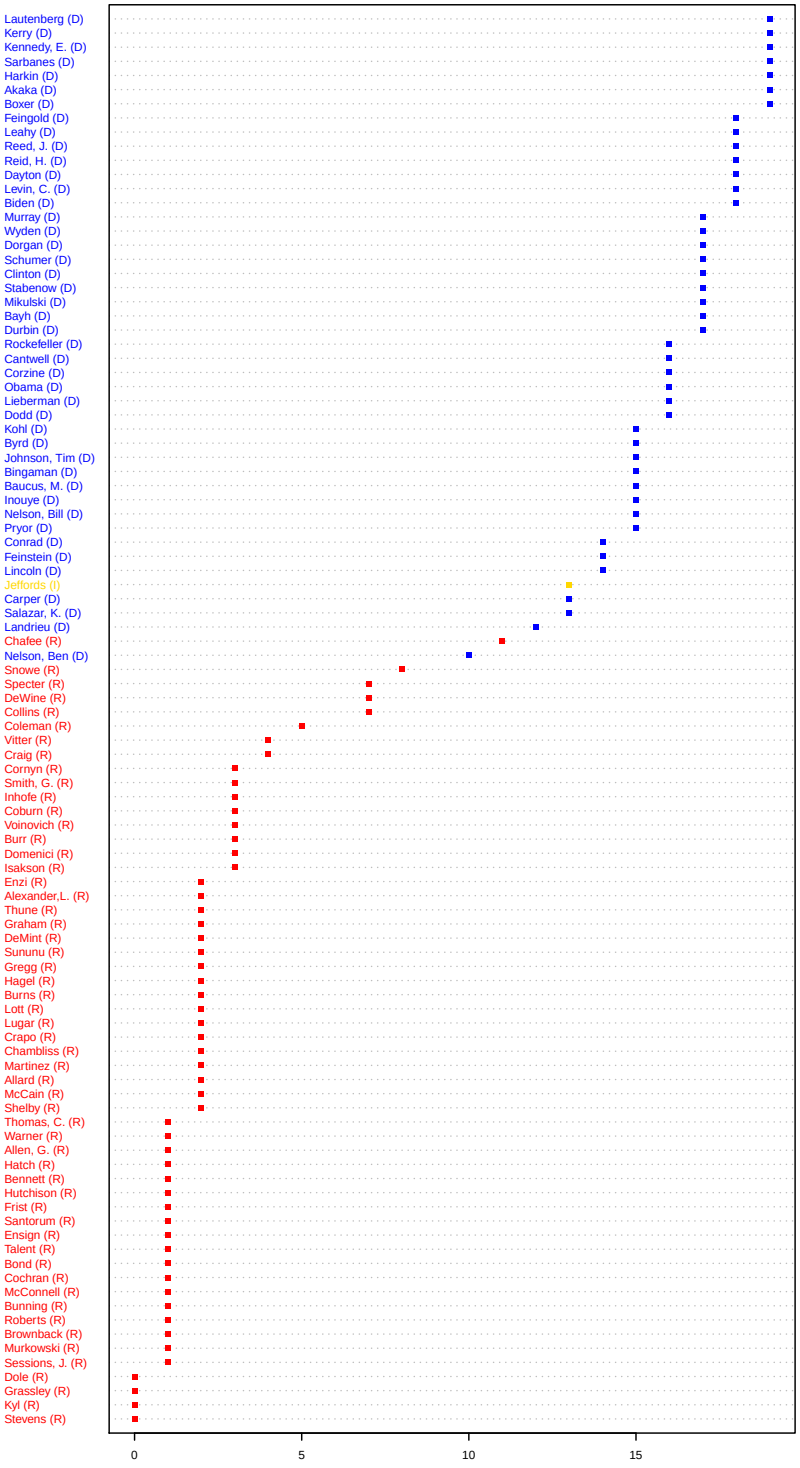


Figure 5.4: Final `dotchart` function on Senate data. Color indicates party affiliation



5.3 Error bars/confidence intervals

Scientific reporting guidelines typically include advice about presenting information about the variability of the data along with the means. People take different approaches to this, and so you should be sure to document which approach you took. Some people plot the confidence interval of the mean–generally representing one standard error unit above or below the mean. Standard error is an estimate of the variability of your estimate of the mean, and is computed as:

$$se = sd/\sqrt{N} \tag{5.1}$$

Another variation is to plot plus or minus *two* standard error units. Yet another is to plot the standard deviation of the distribution, or to plot extreme percentiles of the distribution, which is more like a boxplot. Thus, whatever you do, be sure to document what you did, because some people may assume you did something different, and expect someone to misunderstand. In any case, a simple and crude way to create error bars is using the `arrows()` function, and bending the arrowhead so it has a 90 degree angle.

```
set.seed(100)
x <- rep(1:5,each=25)
y <- x *3 + sqrt(x)*rnorm(25*5)*8 + runif(25*5)*3

se <- function(x){sd(x)/sqrt(length(x))}

means <- aggregate(y,list(x),mean)
sds <- aggregate(y,list(x),sd)
ses <- aggregate(y,list(x),se)

##plot the means:
plot(x,y,pch=16,cex=.8,col="darkgrey",xlim=c(0,5))
points(1:nrow(means),means$x,cex=1.2,col="red",pch=16)
```

Arrows makes a line with arrowheads, which we can make into error bars with the code=3 (arrows on both ends), and angle=90.

Try making an error bars around the first one. Here, we will make both standard error and standard deviation bars, but generally you only want one depending on the context. This is shown in the left panel of Figure 5.5

```
arrows(1,means[1,]$x+ses[1,]$x,1,means[1,]$x-
          ses[1,]$x,code=3,angle=90,length=.1,lwd=2)
arrows(1,means[1,]$x+sds[1,]$x,1,means[1,]$x-
          sds[1,]$x,code=3,angle=90,col="red",lwd=2)
```

We can do all of them in one fell swoop like this (shown in the center panel of Figure 5.5:

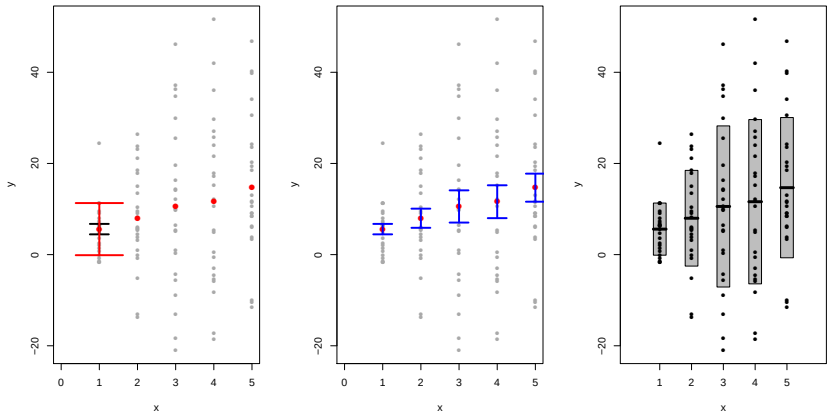
```
arrows(1:nrow(means),means$x+ses$x,
       1:nrow(means), means$x-ses$x,
         code=3,angle=90,length=.1,lwd=2,col="blue")
```

The command is kind of daunting, so let's take a closer look. Each of the first four arguments is a vector that is five elements long. The first and third are just the numbers 1 through 5, to specify the x coordinates of the error bars, matching the x coordinates of the means. The second and fourth arguments are the means, plus and minus the standard error values computed earlier. The `code` value indicates that both ends should have arrows (otherwise just the second end will), and the angle=90 specifies the angle of the arrowheads. Finally, length specifies the size of the arrows, and it is often useful to set this to be attractive.

These types of errors bars are common, but there are a lot of things you could do. What about making rectangles instead? I'm not sure how wide I want this, so we'll make it easy to adjust by setting up a variable `diff`.

```
diff <- .2
plot(x,y,pch=16,cex=.2,type="n",xlim=c(0,5),xaxt="n")
rect(1:4-diff,means$x-sds$x,1:4+diff,means$x+sds$x,col="grey")
points(x,y,cex=.8,pch=16)
segments(1:4-diff,means$x,1:4+diff,means$x,lwd=3)
axis(1,1:4)
```

Figure 5.5: Examples of the use of the `arrows()` and `rect` functions to create error bars by hand.



5.3.1 Built-in error bar functions

Both the plotrix and gplots libraries have the (same) functions called plotCI, which will make a plot with confidence intervals.

```
install.packages("plotrix")
install.packages("gplots")
library(plotrix)
library(gplots)
```

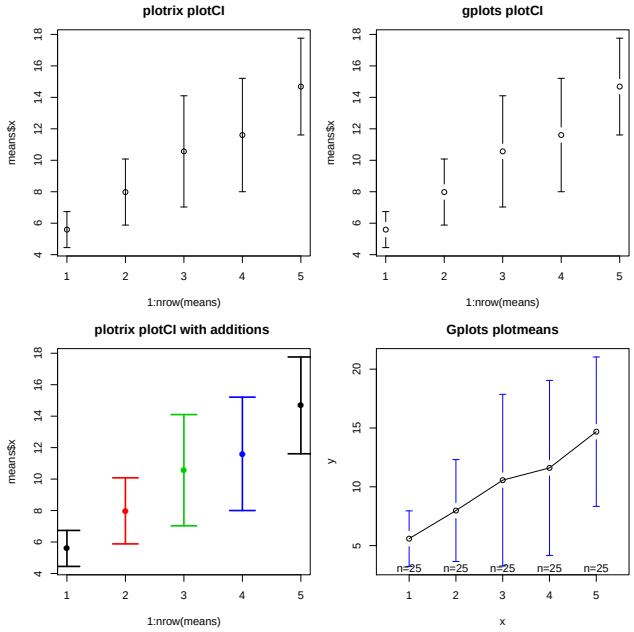
The confidence interval alone is often fine, but sometimes you'd like to plot the actual data with the error bars on top. Here, we'll also adjust some parameters, including the color of the bars.

```
plotrix::plotCI(1:nrow(means),means$x,ses$x,main="plotrix plotCI")  # like
    this one better
gplots::plotCI(1:nrow(means),means$x,ses$x,main="gplots plotCI")   # this one
    not as good
plotrix::plotCI(1:nrow(means),means$x,ses$x,add=F,lwd=2,cex=1.2,sfrac=.04,col
    =1:4,
                pch=16, main="plotrix plotCI with additions ")
```

The plotmeans function in gplots provides a wrapper to plotCI that does some additional plotting and labeling:

```
plotmeans(y~x,main="Gplots plotmeans") #give it the original data!
```

Figure 5.6: Built-in error bar plotting functions, including plotCI from plotrix and plotCI and plotmeans from the gplots library.



5.3.2 Error bars on barplots

How about on barplots? Let's just use the plotCI to add an error bar (see left panel of Figure 5.7), by using the `add=T` argument:

```
barplot(means$x,col="navy")
plotCI(1:nrow(means),means$x,ses$x,add=T)
```

That doesn't quite look right. First off, the error bars are misaligned. Also, they put the actual points in, which we'd like to remove. The misalignment comes because the barplot function creates its bars with a horizontal axis that is offset a bit from the actual numbers you'd expect. But barplot returns a vector of those numbers, which you can then use for plotting (see second panel of Figure 5.7.)

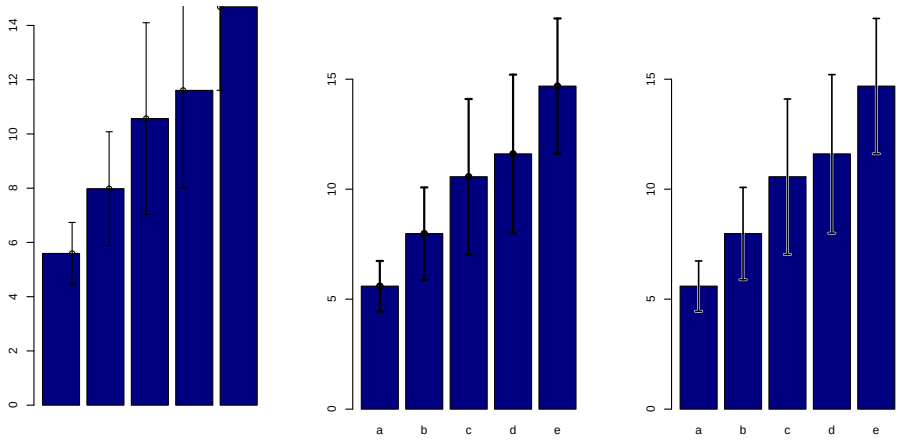
```
newx <-barplot(means$x,names=letters[1:nrow(means)],col="navy",ylim=c(0,max(
    means$x) + max(ses$x)))
print(newx)
     [,1]
[1,]  0.7
[2,]  1.9
[3,]  3.1
[4,]  4.3
[5,]  5.5
#I don't like this:
plotrix::plotCI(newx,means$x,ses$x,add=T,lwd=2)
```

Now, let's clean it up a bit (third panel of Figure 5.7. Here, we change the gap to 0, now that the symbol has been removed (with type="n"). Here, I will overplot a white errorbar and a black error bar, to give the error bar a little contrast from the dark blue barplot. This is shown in the right panel of Figure 5.7

```
##try #3. Add two versions of the error bars with a white outline
newx <-barplot(means$x,names=letters[1:nrow(means)],col="navy",ylim=c(0,max(
    means$x) + max(ses$x)))

gplots::plotCI(newx,means$x,ses$x,add=T,lwd=2.5,type="n",gap=0,col="white")
gplots::plotCI(newx,means$x,ses$x,add=T,lwd=1.5,type="n",gap=0,col="black")
```

Figure 5.7: Three examples using plotCI to add error bars to a boxplot.



Exercise 5.3.2

Write a function that takes a dependent measure as one argument, and a categorical set of levels as the second argument (the IV). You can assume that these are of the same length–no need to do any checking, and that there are at least three observations in each . Within the function:

- Use aggregate to compute the mean value of the dependent measure associated with each level of the independent measure.
- Similarly, compute the standard deviation for each level.
- Compute the number of observations in each level (try using length() as the function in aggregate)
- Based on these values, compute the standard error (s.e.) as sd/sqrt(n) for each group.
- Within the function, create a plot of your choice (bar or point plot), and add error bars of +/- 1 s.e. unit on each side of the mean
- Add optional arguments to the function to control main header, axis labels, and at least four other graphical arguments

5.4 Advanced Boxplotting

In the previous chapter, we covered boxplots against a single independent variable. You can also organize them by multiple IVs. In these cases you should use color and text labels for ease of interpretation. Notice that rather than specifying factors separated by commas, we use the symbol. This symbol is used in many functions, especially models, to specify you want to look at the thing on the left as a function of the things on the right.

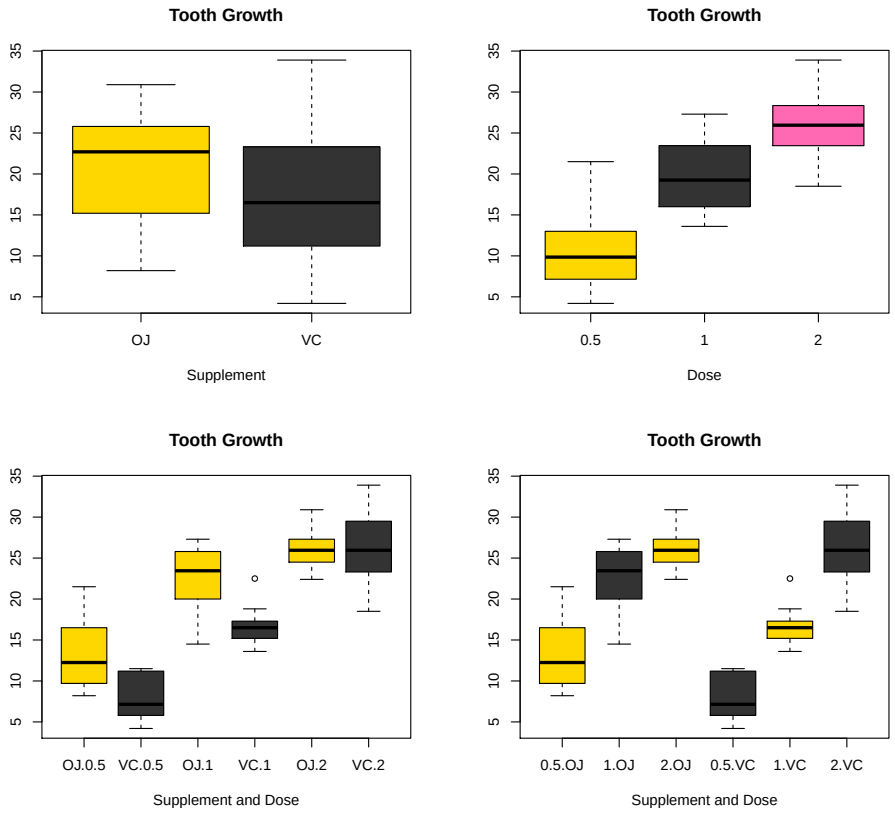
The following code plots tooth grown by the supplement type, and dosage (see top panel of Figure 5.8).

```
boxplot(len ~ supp, data=ToothGrowth, notch=F,
         col=(c("gold","grey20")),
        main="Tooth Growth", xlab="Supplement")
#plot by dosage
boxplot(len~dose, data=ToothGrowth, notch=F,
        col=(c("gold","grey20","hotpink")),
        main="Tooth Growth", xlab="Dose")
```

We can also plot length by both simultaneously:

```
#plot by both factors
boxplot(len~ supp+dose, data=ToothGrowth, notch=F,
       col=(c("gold","grey20")),
       main="Tooth Growth", xlab="Supplement and Dose")
```

Figure 5.8: Four example box plots on the same data. The lower left panel has a display problem, with colors not mapping onto any reasonable property of the data (see exercise)



.

```
5
##Change the order
7 boxplot(len ~ dose+supp, data=ToothGrowth,
    col=  c("gold","grey20"),
9     main="Tooth Growth", xlab="Supplement and Dose")
```

Exercise 5.4

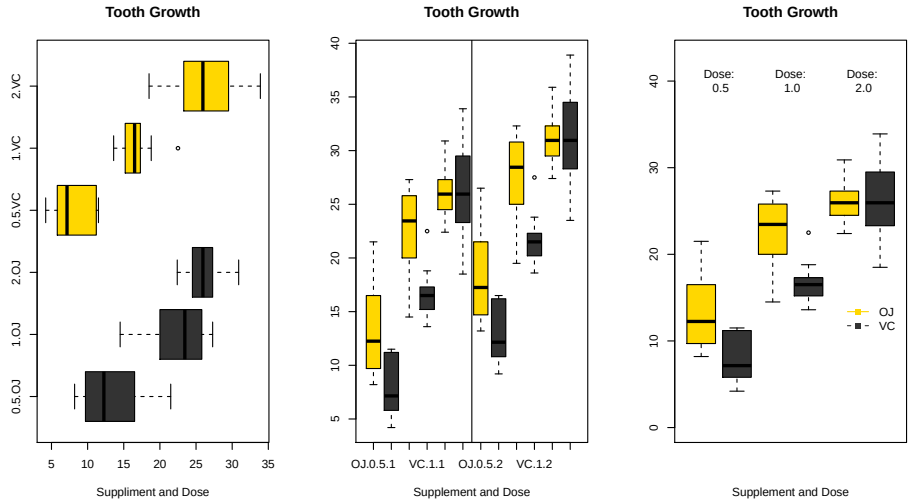
The color scheme is wrong in the last example (lower right panel). Fix it in some coherent way.

5.4.1 Sideways boxplots

Boxplots work sideways too, using the horizontal keyword. See the left panel of Figure **??**.

```
1 boxplot(len~dose*supp, data=ToothGrowth, horizontal=T,
   col=(rep(c("gold","grey20"),each=3)),
3    main="Tooth Growth", xlab="Suppliment and Dose")
```

Figure 5.9: Several additional boxplots.



.

5.4.2 Boxplots with three independent variables

What if we have more than two IVs? Let's try this by just doubling the data set here (see center panel of Figure 5.9). Notice

```
tooth2 <- rbind(ToothGrowth,ToothGrowth)
tooth2$rep <- rep(1:2,each=nrow(ToothGrowth))

boxplot(len~supp*dose*rep, data=tooth2, notch=F,
  col=(c("gold","grey20")),
  main="Tooth Growth", xlab="Supplement and Dose")
abline(v=6.5)
```

5.4.3 Adding your own headers and legend to a boxplot

The group headers are difficult to understand

```
x<-boxplot(len~supp*dose, data=ToothGrowth,
  col=(c("gold","grey20")),
  main="Tooth Growth", xlab="Supplement and Dose",
          xaxt="n",ylim=c(0,43))
text(c(1,3,5)+.5,c(40,40,40),paste("Dose:\n",c("0.5","1.0","2.0")))
legend(5,15, c("OJ","VC"), pch=15,col=c("gold","grey20"),bty="n",lty=1:2)
```

5.5 Adding images to a plot

Sometimes, it can be handy to add bitmap images (jpegs, etc.) to a graph to help graph comprehension. For example, if you are doing an eyetracking study, you can put a screenshot of the actual test in the background and plot eye movements on top of this. We need a few more libraries to handle this:

```
library(jpeg)
library(pixmap)
```

In this case, let's use an image as the plotting symbol–look at overall interest in the Brady Bunch. We'll sntart by reading in image files (named after the different members) using the read.jpeg function, then convert it to a format we can use:

```
bradys <- c("marsha","carol","greg","jan","alice",
            "peter","brady","cindy","mike","bobby")

images <- c()
for(i in bradys)
{
  ##This comes from the jpeg library
  img <- readJPEG(paste("images/",i,".jpg",sep=""))
  ##Convert to a pixmap using from the pixmap library:
  img2<-pixmapRGB(img)
  images <- c(images,img2)
}
```

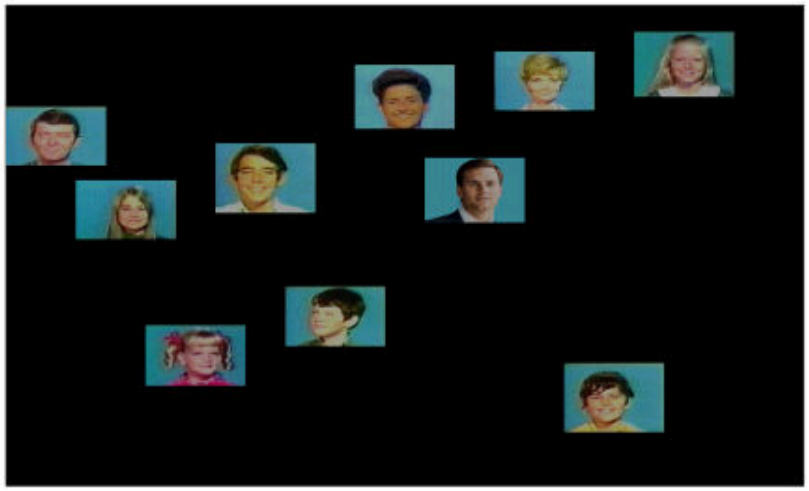
Now, let's plot them in random locations on the graph. We use the addlogo function to plot an image onto a particular region of the figure.

```
##randomize the order
ord <- order(runif(length(bradys)))

plot(0,type="n",xlim=c(0,800),ylim=c(0,600),xaxt="n",yaxt="n",
     xlab="",ylab="",bty="n")
rect(0,0,800,600,col="black")
x <- 0
for(i in images[ord])
{
   y <- round(runif(1)*500)
   addlogo(i,c(x,x+100),c(y,y+60),asp=1) ##addlogo is in pixmaps library
   x <- x + 70
 }
```

Notice that the original is just a normal plot() figure. Addlogo puts a pixmap at a specific location.

Figure 5.10: Example of adding images to a graph.



5.6 Violin plots

An useful alternative to the boxplot is the violin plot. Instead of showing quantiles, it attempts to show the entire distribution, sometimes with a boxplot inside it. To show the distribution, violin plots will generally use some approach to smoothing to show a smooth outline–one that often looks like the shape of a violin. consequently, the smoothing algorithm used can have a large impact and should be understood.

There are several libraries in R for creating violin plots. These include `vioplot`, `violinmplot`, along with a plot option within `ggplot2`.

We can look at the OrchardSprays data set using both methods. Like many third-party libraries in R, the arguments for each function differ, and each has some plotting quirks.

5.6.1 The `vioplot` library

The `vioplot` function in Daniel Adler's vioplot library takes each distribution as a separate argument. To create these distributions for each treatment, I used tapply and so each distribution becomes a row or column of that table. This function also seems to have trouble setting headers and axis labels, so I used the base `title()` function to put these on afterward. Also, I overlayed the actual points on each violin plot using `matplot` with the `add=T` argument.

```
v1 <- tapply(OrchardSprays$decrease,
            list( row=OrchardSprays$rowpos,
                        treatment=OrchardSprays$treatment), mean)
install.packages("vioplot")
library(vioplot)

vioplot(v1[,1],v1[,2],v1[,3],v1[,4],v1[,5],v1[,6],v1[,7],v1[,8],col="gold",
        names=LETTERS[1:8],ylim=c(0,200))
title(main="OrchardSprays violin plot with point overlay: vioplot",
      ylab="Decrease in bees",xlab="Treatment")
matplot(t(v1),add=T,pch=1,col="grey30",cex=1)

# Here is another plot that uses a smaller h value.
vioplot(v1[,1],v1[,2],v1[,3],v1[,4],v1[,5],v1[,6],v1[,7],v1[,8],col="gold",
        names=LETTERS[1:8],h=2,ylim=c(0,200))
title(main="OrchardSprays violin plot: vioplot\nsmoothing kernel h=2",
      ylab="Decrease in bees",xlab="Treatment")
```

Exercise 5.6.1

The following creates intermingled distributions whose mean and variability are correlated. Thta is, as the mean increases, the standard deviation does as well.

```
 conds <- sample(1:4,1000,replace=T)
data <- (rnorm(1000,mean=conds*5,sd=conds+1))
```

Create a violin plot showing the distribution of the four conditions

5.6.2 Adapting a custom violin plot function

Because of the smoothing, neither plot works well when you have data from a small number of integer or ordinal values—perhaps the most frequent type of data collected by psychologists, in the form of Likert-scale responses to questions. The vioplot function cannot turn off smoothing, and if you make it too small, you see bumps at the location of the data, as shown in Figure 5.11. Setting the smoothing argument to be large enough simply washes out any meaningful difference is the distributions. In the following code, I create an example likert-scale data set that also has a non-response category, associated with the factor level 0.

```
set.seed(1000)
levs <-c("Don't Know","Strongly Disagree", "Disagree","Neutral",
         "Agree",
         "Strongly Agree")
gender=as.factor(sample(c("Men","Women"),1000,replace=T))
vals1 <- pmax(0,floor(rnorm(1000,mean=2.5,sd=.8) - as.numeric(gender)*.5))+1
vals1[sample(1000,20)]<- 0  ##ad some don't know responses

resps <- factor(levs[vals1+1],levels=levs)

data <- data.frame(gender=gender,
                   value=vals1,
                   resps)
head(data)

par(mfrow=c(1,3))
vioplot(data$value[data$gender=="Men"],
        data$value[data$gender=="Women"],col="gold")

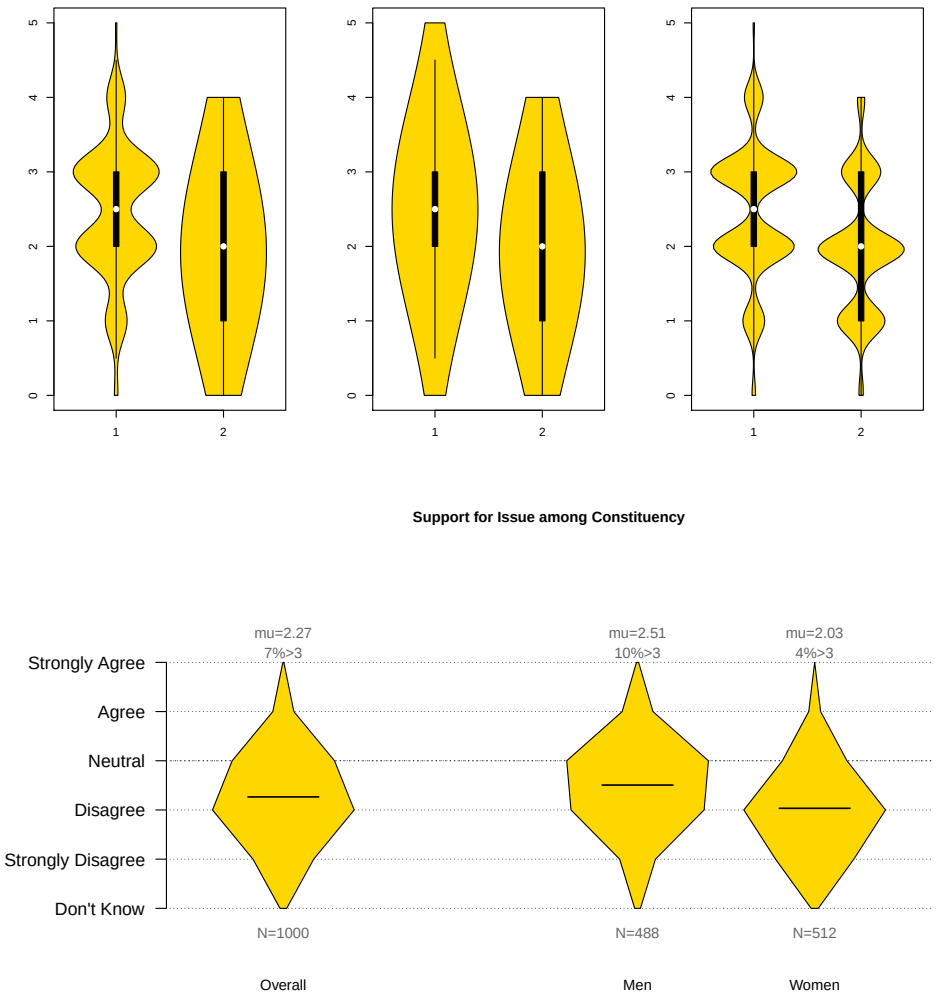
vioplot(data$value[data$gender=="Men"],
        data$value[data$gender=="Women"],h=1.2,col="gold")

vioplot(data$value[data$gender=="Men"],
        data$value[data$gender=="Women"],h=.2,col="gold")
```

Neither the default not two custom values of h are satisfying, as shown in Figure 5.11

The nice thing about R is that the code for creating the graphics is available. You can see it if you just type the name of the function into R. I once was responsible for presenting results from a university-wide survey in which all the responses were likert-scale. The vioplot was ugly, but I wanted a nice visual depiction of the responses. So I copied out the code listing for vioplot, saved it into a file, and gave the resulting function a new name (vplot2). Then, by iteratively changing things, I was able to customize it to create a violin plot suitable for a wide audience. The beginning of the code is shown here, the complete function is available in

Figure 5.11: Vioplot has trouble representing Likert-scale responses. Here, either the smoothing is too large, or it creates bumps around the integer-responses. Top panel shows examples using vioplot. Bottom panel shows results from custom function adapted from vioplot.



the supplemental materials. The changes were initiall fairly minor–removing the code that did the smoothing and using the non-smoothed value. I also added additional plotting to provide more details.

```
1
vplot2  <- function (x, ..., range = 1.5, h = NULL, ylim = NULL,ylabs=NULL,
3                    names = NULL,
                     title="",
5                    horizontal = FALSE, col = "gold", border = "black", lty =
     1,
                     lwd = 1, rectCol = "black", colMed = "white", pchMed =
    19,
7                    at, add = FALSE, wex = 1, drawRect = TRUE,adjust=0,
                     crit=3,
9                    ignore=c(0)
)
11 {
  datas <- list(x, ...)
13
  n <- length(datas)
15  if (missing(at))
    at <- 1:n
17  upper <- vector(mode = "numeric", length = n)
  lower <- vector(mode = "numeric", length = n)
19  q1 <- vector(mode = "numeric", length = n)
  q3 <- vector(mode = "numeric", length = n)
21  .....
  }
23
par(mar=c(2,8,3,1),mfrow=c(1,1))
25 vplot2(data$value,
       data$value[data$gender=="Men"],
27       data$value[data$gender=="Women"],
       at=c(1,3,4),
29       names=c("Overall","Men","Women"),
       drawRect = T,
31       ylim=c(-1,7),
       ylabs=levs,
33       ignore=c(0),
       title="Support for Issue among Constituency"
35       )
```

5.7 Solutions to exercises

Solution to Exercise 5.3.2

Write a function that takes a dependent measure as one argument, and a categorical set of levels as the second argument (the IV). You can assume that these are of the same length–no need to do any checking, and that there are at least three observations in each . Within the function:

- Use aggregate to compute the mean value of the dependent measure associated with each level of the independent measure.
- Similarly, compute the standard deviation for each level.
- Compute the number of observations in each level (try using length() as the function in aggregate)
- Based on these values, compute the standard error (s.e.) as sd/sqrt(n) for each group.
- Within the function, create a plot of your choice (bar or point plot), and add error bars of +/- 1 s.e. unit on each side of the mean
- Add optional arguments to the function to control main header, axis labels, and at least four other graphical arguments

```
1
###Exercises:
3 plotMeansandSE <- function(x,conds,main="",xlab="",ylab="",
                                    col="grey20",cex=2)
5 {
    require(gplots)
7   conds <- factor(conds)
    agg <-     aggregate(x,list(conds),mean)
9   agg$sd <-  aggregate(x,list(conds),sd)$x
    agg$n <-   aggregate(x,list(conds), length)$x
11  agg$se <- agg$sd/sqrt(agg$n)
    xvals <- as.numeric(agg$Group.1)
13  xs <- barplot(agg$x,names=agg$Group.1,
                  ylim=c(0,max(agg$x+agg$se*2)), col=col,
15                main=main,xlab=xlab,ylab=ylab)

17  gplots::plotCI(xs,agg$x,agg$se,add=T,lwd=.5,type="n",gap=0,col="black"
      )
}
19
set.seed(100)
21 conds <- sample(1:4,100,replace=T)
data <- (rnorm(100,mean=1+conds*5,sd=conds+1))
23
plotMeansandSE(data,conds,main="Stinky",col="darkgreen",
25               xlab="Condition",ylab="Observed ability")
```

Solution to Exercise 5.4

The colors in a boxplot are recycled in order, so we need to create a vector of 6 colors, three gold and three grey:

```
boxplot(len~dose*supp, data=ToothGrowth,
 col=(rep(c("gold","grey20"),each=3)),
  main="Tooth Growth", xlab="Supplement and Dose")
```

Or you could color each sub-element of the series a different color:

```
boxplot(len~dose*supp, data=ToothGrowth,
 col=(rep(c("gold","grey20","hotpink"),2)),
  main="Tooth Growth", xlab="Supplement and Dose")
```

Solution to Exercise 5.6.1

The following creates intermingled distributions whose mean and variability are correlated. Thta is, as the mean increases, the standard deviation does as well.

```
 conds <- sample(1:4,1000,replace=T)
data <- (rnorm(1000,mean=conds*5,sd=conds+1))
```

Create a violin plot showing the distribution of the four conditions

```
vioplot(data[conds==1],data[conds==2],data[conds==3],
        data[conds==4],main="Distribution of four conditions")
```

5.8 Additional Resources

- http://addictedtor.free.fr/graphiques/thumbs.php

Chapter 6

Colors and Special-purpose graphics packages

This chapter covers two major themes: colors and special-purpose graphics.

The default color schemes for R are ugly and difficult to get right. Luckily, there are a number of packages that have been developed to allow you to use nice color schemes, color gradients, and color themes. Using a consistent and unique color theme across a set of figures gives a level of professionalism and consistency to your graphics, and can make a great impression.

There are many packages that implement special-purpose useful graphics. This chapter covers a few such packages. There are many more, but a little practice learning to use these packages can be helpful in learning about R and typical uses of R libraries.[1]

6.1 Colors, Color palettes, and Color gradients

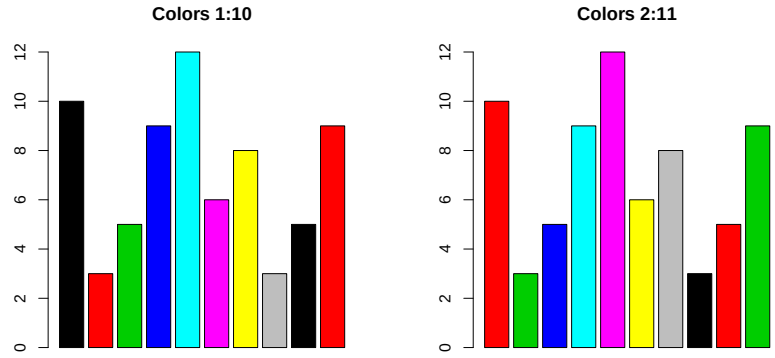
One of the best things you can do to improve the look and feel of your images is to use color. Whenever possible, color should be used to map redundantly onto some other visual indicator of a value, because you cannot rely on the final reader to be seeing your graphic in color, or having full color vision. A publication may convert graphics to greyscale, a reader may make a black-and-white photocopy or scan of your document, their display monitor may show colors differently from yours, or they may be colorblind and unable to distinguish some color pairs. So, use it carefully to improve the look, but you should typically not rely on color on its own to be the only indicator of a factor level or other independent variable.

6.1.1 How R handles color

By default R handles color through a number of complementary systems. Most color arguments will be able to interpret color names, integers representing a palette index, or rgb values. There are more than 650 named colors, which you can see if you use the command `colors()`. Giving a function a named color is an easy way to use color hues. Also, you can use the results of a call to a function `rgb()`, which takes values representing red, green, and blue, and creates a hexidecimal color code that is used in many applications:

[1] The examples provided in this chapter were adapted from examples originally developed by students in the 2012 MTU Graduate Statistics Program: Natasha Hagadone, Saima Ghazal, Krissy Guzak, Alison Regal, Kejkaew Thanasuan, and Wei Zhang.

Figure 6.1: Illustration of the eight standard colors in the palette, and how plot functions recycle the colors.




```
 rgb(.1,.1,.9)
[1] "#1A1AE6"
```

You could also specify this hexcode directly, and there are many websites that let you pick colors out that will display the corresponding hex code.

Finally, the numbers 1 through 8 are interpreted as indices of a color palette which itself can be customized. So, you can give a plot command numbers, and these get interpreted as colors.

6.1.2 Color Palettes

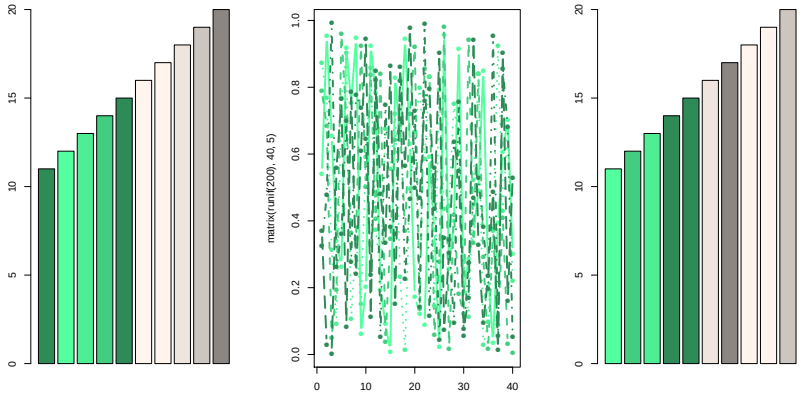
Most plotting functions in R have a `col` argument to specify the color. By default, they will take a number specifying which color in the current palette to use. The default palette has eight colors. You can simply specify number of the palette index to set colors, and if you use a number higher than the number of colors in the palette, it will recycle earlier numbers.

```
dat <- c(10,3,5,9,12,6,8,3,5,9)
barplot(dat,col=1:10,main="Colors 1:10")
barplot(dat,col=1:10+1,main="Colors 2:11")
```

You can hand-pick your own color scheme, and either use it directly by passing it to the `col` argument, or you can set the default palette to that set of colors. For example, maybe I want a sea-related palette. All of the colors whose names R understands are accessible via the `colors()` function. The following will pick out any that start with "sea".

```
seacolors <- colors()[substr(colors(),1,3)=="sea"]
> seacolors
 [1] "seagreen"  "seagreen1" "seagreen2" "seagreen3"
 [5] "seagreen4" "seashell"  "seashell1" "seashell2"
 [9] "seashell3" "seashell4"
par(mfrow=c(1,3))
```

Figure 6.2: Three example plots using the our sea colors pallette. The first plot was given the list of colors as an argument. The second and third plot used those colors automatically after the palette was set.




```
barplot(11:20,col=seacolors)
palette(seacolors[order(runif(10))])
matplot(matrix(runif(200),40,5),type="b",lwd=2,pch=16)
barplot(11:20,col=1:10)
```

Exercise 6.1.2

Create three series of 5 numbers (i.e., a matrix with 3 columns and 5 rows). Create a side-by-side matplot and a stacked barplot with those values, and default color settings. Then, Pick a set of at least five named colors. Set the palette to these colors, and remake the same plots, so that they use these colors.

6.1.3 Built-in color scheme generators

There are a number of functions, either built into R or available as add-on libraries, thta allow you to create sets of colors that are good for showing gradations or showing categorical differences. Four built-in functions are shown below.

```
dat <- c(10,3,5,9,12,6,8,3,5,9)
par(mfrow=c(2,2))
barplot(dat,col=heat.colors(10),main="Heat colors")
barplot(dat,col=terrain.colors(10),main="Terrain colors")
barplot(dat,col=topo.colors(10), main ="Topo colors")
barplot(dat,col=cm.colors(10), main = "Cyan to magenta")
```

These functions are nice because they will give you a color gradient with as many steps as you need, interpolating distinct colors at all the steps in between. Now, these color schemes are not what you typically want for barplots, but they can be useful in other contexts. For example, you can use a color gradient to indicate which element of a sequence you are

Figure 6.3: Examples of four different color gradients

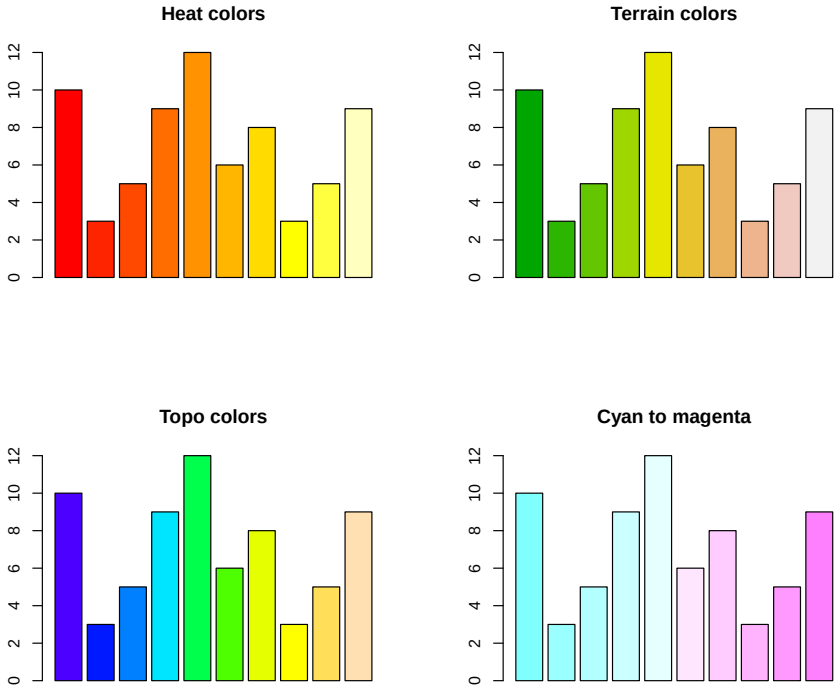
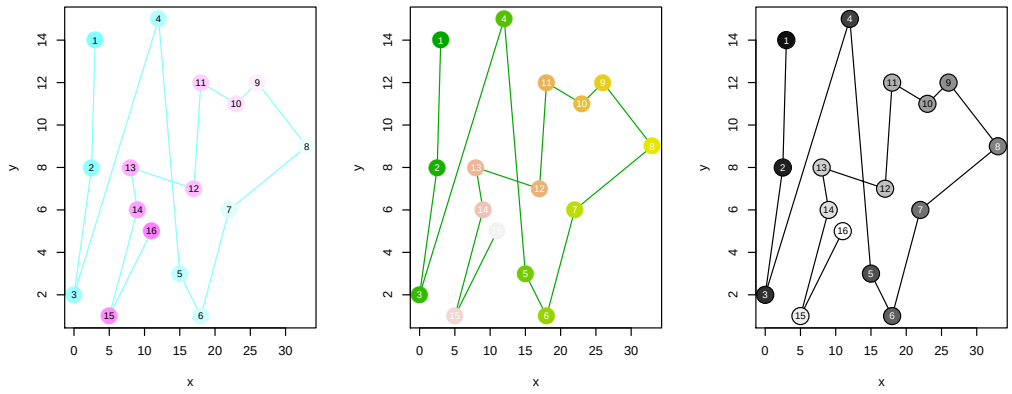


Figure 6.4: Examples of using a color gradient to indicate sequence position.



plotting. Here is a set of 2-D points, which could represent a sequence of eye movements. The last one uses a greyscale color set, which is created a little differently from the other color gradients.

```
x <- c(3,2.5,0,12,15, 18, 22,33,26,23,18,17,8,9,5,11)
y <- c(14, 8,2,15, 3, 1, 6, 9,  12,11,12,7,8,6,1,5)
##there is a start and an end--let's adjust the color:
plot(x,y,type="b",pch=16, col = cm.colors(16),cex=3)
text(x,y,1:16,cex=.8,col="black")

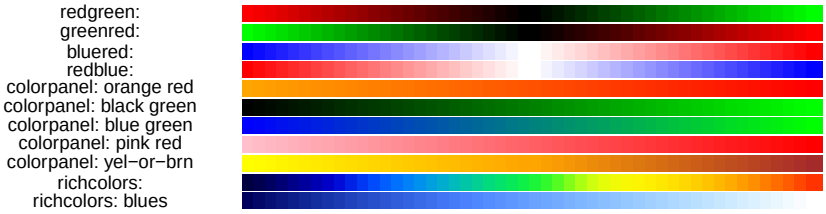
plot(x,y,type="b",pch=16, col = terrain.colors(16),cex=3)
text(x,y,1:16,cex=.8,col="white")

##Greyscale
plot(x,y,type="l",cex=2)
points(x,y,type="p",cex=3, col = grey(1:16/16),pch=16)
points(x,y,type="p",cex=3,col="black")
text(x,y,1:16,cex=.8,col= rep(c("white","black"),each=8)) ##notice reverse
    colors
```

The `gplots` library has some built in color gradients that are nicer. These include `redgreen`, `greenred`, `bluered`, `redblue` and `colorpanel`, which does arbitrary gradients of two or three colors. There are a number of 'named' palettes in gplots as well, accessible via the rich.colors function. Some examples or shown in Figure 6.5.

```
library(gplots)
plot(1:50,rep(10,50),pch=15,cex=2,xlim=c(-20,50),ylim=c(0,12),
    col=redgreen(50), xaxt="n",yaxt="n",xlab="",ylab="",bty="n")
text(-12,10,"redgreen:")
points(1:50,rep(9,50),pch=15,cex=2,col=greenred(50))
text(-12,9,"greenred:")
points(1:50,rep(8,50),pch=15,cex=2,col=bluered(50))
text(-12,8,"bluered:")
points(1:50,rep(7,50),pch=15,cex=2,col=redblue(50))
text(-12,7,"redblue:")
points(1:50,rep(6,50),pch=15,cex=2,
        col=colorpanel(50,"orange","red"))
text(-12,6,"colorpanel: orange red")
```

Figure 6.5: Example using a color gradients in the `gplots` library.

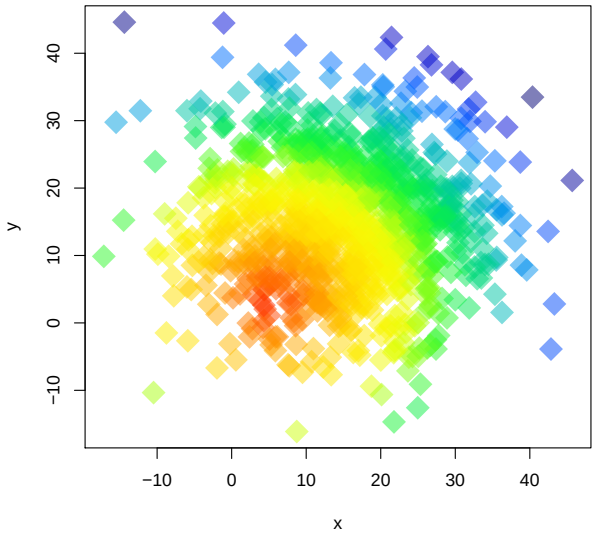



```
points(1:50,rep(5,50),pch=15,cex=2,
        col=colorpanel(50,"black","green"))
text(-12,5,"colorpanel: black green")
points(1:50,rep(4,50),pch=15,cex=2,
        col=colorpanel(50,"blue","green"))
text(-12,4,"colorpanel: blue green")
points(1:50,rep(3,50),pch=15,cex=2,
        col=colorpanel(50,"pink","red"))
text(-12,3,"colorpanel: pink red")
points(1:50,rep(2,50),pch=15,cex=2,
        col=colorpanel(50,"yellow","orange","brown"))
text(-12,2,"colorpanel: yel-or-brn")
points(1:50,rep(1,50),pch=15,cex=2,
          col=rich.colors(50))
text(-12,1,"richcolors:")
points(1:50,rep(0,50),pch=15,cex=2,
          col=rich.colors(50,palette="blues"))
text(-12,0,"richcolors: blues")
```

We can also use a gradient such as this to map onto another value, such as distance from a particular point. In the example in Figure 6.6, distance from the point (5,3) is mapped onto a color gradient. In this example, color is given an alpha value of .5, to make it slightly transparent.

```
#### Color in 2x2 plot based on distance from a location.
n <- 1000
x <-rnorm(n)*10 + 15
y <- rnorm(n)*10+ 15
##Color by distance from (3,2); use alpha channels so
##we can see overlap
dist <- sqrt((x-5)^2+(y-3)^2)
plot(x,y,col=rev(rich.colors(50,alpha=.5))[1+floor(dist)],
         pch=18,cex=3)
```

Figure 6.6: Using color to represent distance from a point.



Exercise 6.1.3

Plot a matrix of values using image, with several different gplots color gradients.

6.1.4 Colorbrewer palettes

The colorbrewer project was designed to create color schemes for maps and other graphical displays. You can visit the website at http://colorbrewer2.org to browse and select color schemes. The RColorBrewer package implements these color palettes, and contains three types of color palettes: sequential, providing a color gradient; qualitative, giving a set of colors that are distinct to map onto different categories, and divisive, scaling between two colors.

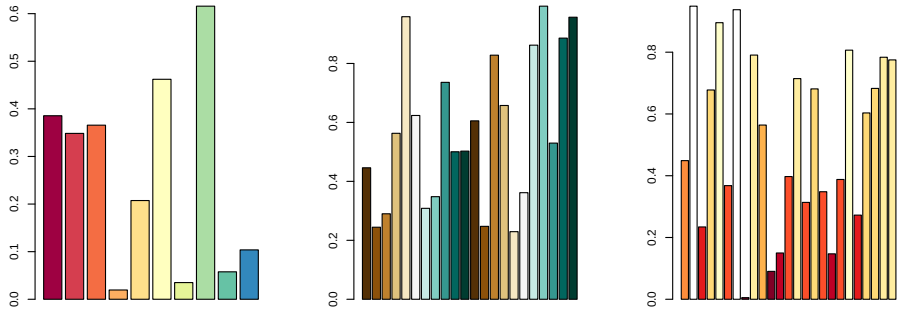
Example code to see the different palettes is here.

```
install.packages("RColorBrewer")
library(RColorBrewer)

##Check out a few palettes available:
display.brewer.all()
display.brewer.all(type="seq")
display.brewer.all(type="qual")
display.brewer.all(type="div")

display.brewer.pal(10,"Blues")
```

Figure 6.7: Using color to represent distance from a point.



```
display.brewer.pal(6,"Greens")
display.brewer.pal(12,"BrBG")
display.brewer.pal(7,"Accent")
```

Here are a few examples of using color brewer palettes on a barplot. Here, I can either set the palette to some scheme, or feed it into the plot function in the `col` argument. The last example shows how to make the color depend on the height of the bar.

```
##set your palette like this:
par(mfrow=c(1,3))
palette <- brewer.pal(11,"Spectral")
palette(palette)
barplot(runif(10),col=1:10,main="Spectral colors")

barplot(runif(22),col=brewer.pal(11,"BrBG"),main="Brown-Green palette")

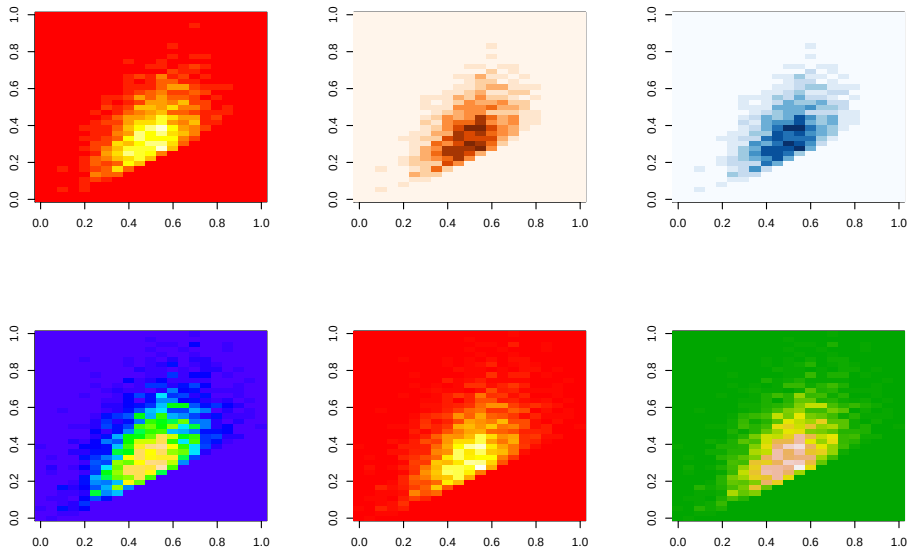
##or, something more meaningful
dat <- runif(25)
pal <- rev(brewer.pal(10,"YlOrRd"))
col <- floor(dat*10)+1
barplot(dat,col=pal[col],main="Color depends on value")
```

Here, we create a matrix showing different counts, and we want to use an image plot to show the count in each cell. We can use a number of palettes to help do this:

```
x <- rnorm(2500)*3
y <- x + sqrt(rnorm(2500)^2) *8
tab <- table(round(x),round(y))
image(tab)
image(tab,col=brewer.pal(11,"Oranges"))
image(tab,col=brewer.pal(11,"Blues"),legend=T)

## Use some built-in color schemes:
##
image(tab,col=topo.colors(50))
image(tab,col=heat.colors(50))
image(tab,col=terrain.colors(50))
```

Figure 6.8: Using color to represent distance from a point.



Exercise 6.1.4

Use a 'divisive' color palette from RColorBrewer. Use it to make a non-stacked barplot from a data table having two columns and five rows, so that the plot has two groupings of bars, and each set of bars has the same color scheme.

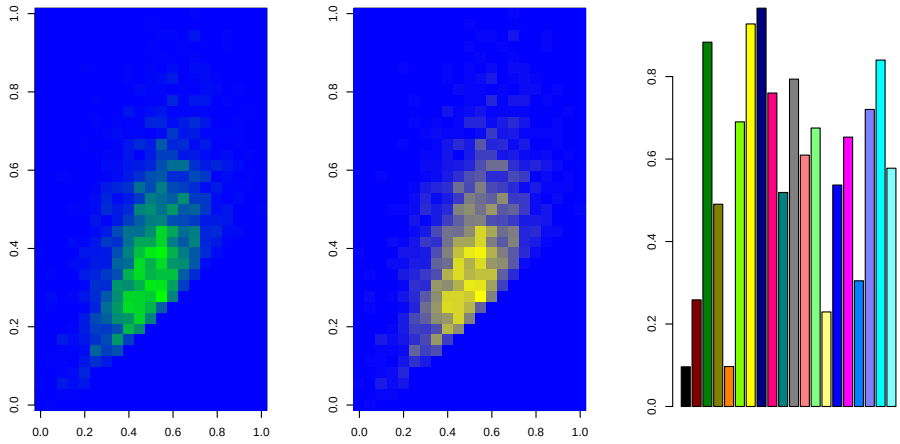
6.1.5 ColorRamps

ColorRamps is another package that builds color palettes. It has a handful of specific gradients, including:

- blue2green
- blue2green2red
- blue2red
- blue2yellow
- cyan2yellow
- green2red
- magenta2green
- matlab.like
- matlab.like2
- primary.colors
- ygobb

Each of these create a color scheme, with N gradations along the color gradient. rgb.tables and table.ramp let you create a custom one. Several examples are shown in Figure 6.9.

Figure 6.9: Using ColorRamps as a color gradient.




```
install.packages("colorRamps")
library("colorRamps")
image(tab,col=blue2green(50))
image(tab,col=blue2yellow(50))
barplot(runif(20),col=primary.colors(20))
```

6.1.6 Building colorblind-visible from RGB space

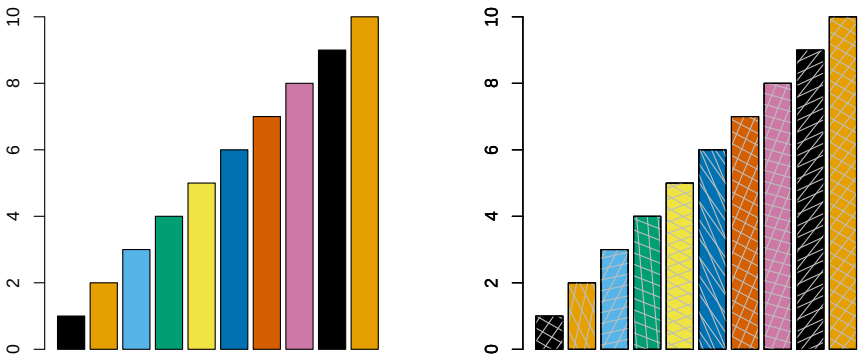
A significant minority of people (mostly men) have colorblindness, with these mostly being an inability to distinguish red and green. This should be considered when making graphics. The following palette has been suggested to be discriminable by red-green colorblind individuals. We can also add lines of different angles to help distinguish colored bars from one another.

```
colorblind <- c(rgb(0,0,0,maxColorValue=255),
                rgb(230,159,0,maxColorValue=255),
                rgb(86,180,233,maxColorValue=255),
                rgb(0,158,115,maxColorValue=255),
                rgb(240,228,66,maxColorValue=255),
                rgb(0,114,178,maxColorValue=255),
                rgb(213,94,0,maxColorValue=255),
                rgb(204,121,167,maxColorValue=255))
palette(colorblind)
barplot(1:10,col=1:10)


##use density and angle to overlay a stipple pattern
barplot(1:10,col=1:10)
barplot(1:10,density=10,angle=runif(10)*180,add=T)
barplot(1:10,density=10,angle=runif(10)*180,add=T)

barplot(matrix(runif(20),4,5))
```

Figure 6.10: Using a custom rgb colors to make a colorblind-visible color scheme.



6.1.7 Some thoughts on color schemes

Using a consistent color scheme across plots in a publication adds unity and coherence, especially when consistently mapped onto the same groups/IVs. The advantage of setting the palette is that you can change the entire plotting scheme with one line:

```
dat <- t(matrix(runif(20),5,4) * (1:5)^2)
cats =c("Fr","So","Ju","Sr","Gr")

par(mfrow=c(2,2))
palette(c("midnightblue","gold","darkgreen","maroon","dodgerblue"))
matplot(dat,type="l",lwd=3,lty=1,xaxt="n",ylim=c(1,25))
legend(2,25,rev(cats),lty=1,lwd=3,col=5:1)
pie(c(1,13,5,2,1),labels=c("Fr","So","Ju","Sr","Gr"),col=1:5)
dotchart(c(1,13,5,2,1),labels=c("Fr","So","Ju","Sr","Gr"),col=1:5,pch=16)
barplot(c(10,5,8,4,6),names=cats,col=1:5)


##Whoops, somebody didn't like the color scheme.
par(mfrow=c(2,2))
palette( brewer.pal(9,"Set1"))
matplot(dat,type="l",lwd=3,lty=1,xaxt="n",ylim=c(1,25))
legend(2,25,rev(cats),lty=1,lwd=3,col=5:1)
pie(c(1,13,5,2,1),labels=c("Fr","So","Ju","Sr","Gr"),col=1:5)
dotchart(c(1,13,5,2,1),labels=c("Fr","So","Ju","Sr","Gr"),col=1:5,pch=16)
barplot(c(10,5,8,4,6),names=cats,col=1:5)
```

You can also use related colors to plot data versus summary stats or related sub-levels of a factor. On the left panel of Figure 6.12, we have plotted the raw data in light green,

Figure 6.11: If we make a set of related graphs, a single palette command can recolor all of them.

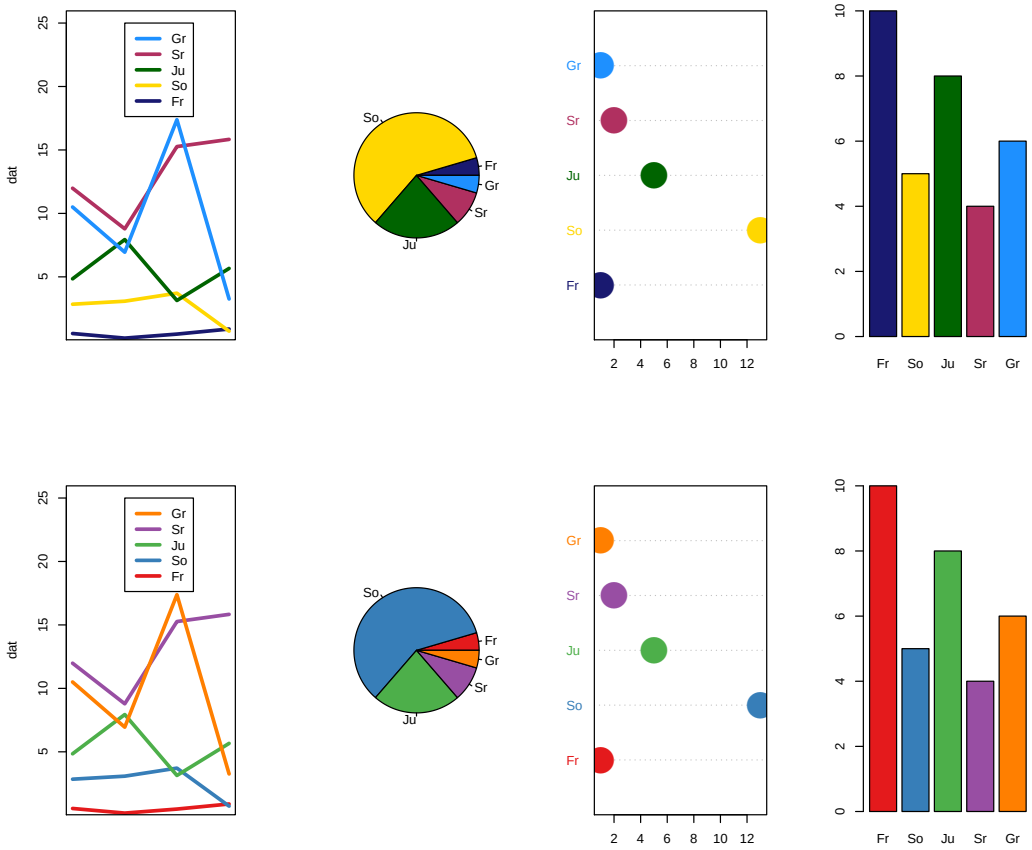
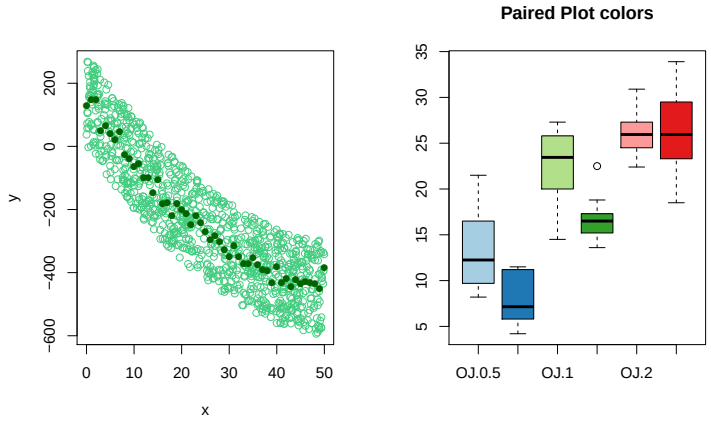


Figure 6.12: Using color themes to represent related aspects of the data can be useful for grouping.



and the means in darker green. On the right, we have arranged sub-levels of a condition in a boxplot so that related sub-levels have a similar color scheme.

```
x <- runif(1000)*50
xbin <- round(x)
y <- x*(x/5-22)+runif(1000)*300
y.agg <- aggregate(y,list(xbin),mean)

par(mfrow=c(1,2))
plot(x,y,col="seagreen3")
points(y.agg$Group.1,y.agg$x,col="darkgreen",pch=16)


boxplot(len~supp*dose,data=ToothGrowth,
        col=brewer.pal(6,"Paired"),main="Paired Plot colors")
```

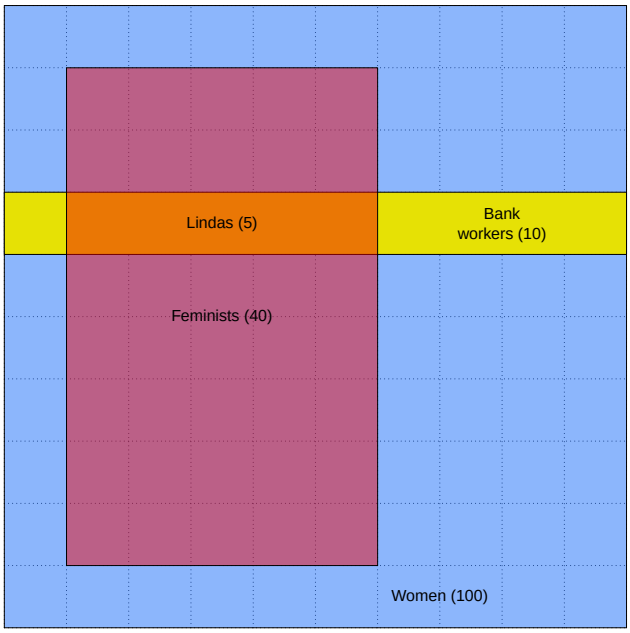
6.1.8 Using Transparency

Using a classic problem in decision heuristics, we can use boxes to visualize relative sizes of groups in a population. In a random sample, of women, we might have 40 self-identified feminists, and 10 employed in the financial industry. The results are shown in Figure 6.13.

```
cols <- c(rgb(240,0,5,maxColorValue=255,alpha=120),
        rgb(230,225,5,maxColorValue=255,alpha=255),
        rgb(12,100,250,maxColorValue=255,alpha=120))

plot(0,type="n",xlim=c(0,10),ylim=c(0,10),bty="n",
      xaxt="n",yaxt="n",xlab="",ylab="")
segments(0:10,0,0:10,10,lty=3)
segments(0,0:10,10,0:10,lty=3)
```

Figure 6.13: Example using rectangles and transparancy to draw spatial graph.




```
rect(0,0,10,10,col=cols[3])
rect(0,6,10,7,col=cols[2])
rect(1,1,6,9,col=cols[1])
text(3.5,5,"Feminists (40)")
text(3.5,6.5,"Lindas (5)")
text(7,6.5,"Bank workers (10)")
text(8,3,"Women (100)")
```

Exercise 6.1.8

When you have a scatterplot of integers points, you tend to get points plotting on top of one another, which hides the number of observations. You can get a better notion of this sometimes if you use large points with transparency. Those locations with more points will show up darker. Use a large point size and

```
x <- sample(1:10,100,replace=T)
y <- round(x + rnorm(100)*2)
```

6.2 Balloon Plots

A balloon plot is a nice way to represent cross-tabulated data. The `balloonplot` function is in the gplots library, which you may need to download separately. You run balloonplot with three main arguments–the category for the x and y margins (rows and columns), and the count associated with each. You could create these from a larger data set using the aggregate command.

```
balloonplot(x,y,z)
```

In this example, suppose we have identified five cat names and five colors, and want to look at whether cats names depend on their colors. The data here are faked, but we could possible collect such data via surveys of cat owners. To start with, the following code will create a data frame with these values.

```
library(gplots)
catnames <- c("izzy", "dilly", "spooky", "bernard", "jack")
catcolors <- c("black", "orange", "gray", "brown")
datavals <- round(exp(runif(20)*5))
data <- data.frame(Cat=rep(catnames,4),
                   Color=rep(catcolors, each=5),
                   Count=datavals )
```

The balloonplot takes the independent variables or factors as its first two arguments, and the dependent variable as its third argument. We can make a simple plot with default arguments, providing a few labels to make it plot nicer. The result is shown in Figure 6.14

```
balloonplot( data$Cat, data$Color, data$Count,
             colmar=2,ylab="Color",xlab="Cat name")
```

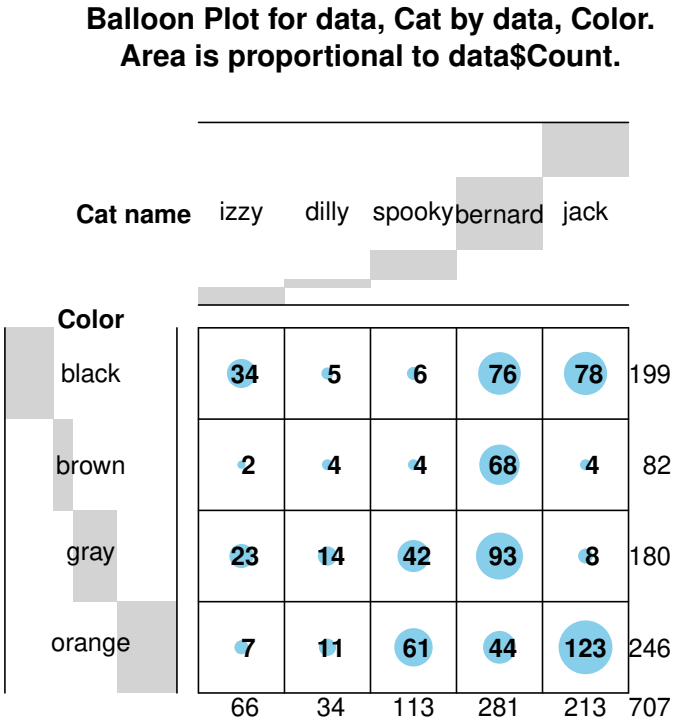
Notice in Figure 6.14 that each cell has a circle whose size is related to the number of cats of that category, but also the marginal proportions are depicted with their relative size on the column and row headers. Finally, the cell values and row/column sums are also shown.

We can change some additional arguments. The `colmar` changes the size of the margins `label=F` will turn off the number counts, `dotcol` will set the color of the dots, which we might want to set to be similar to the cat color; `dotsize` controls the maximum size of the dot in each cell; `scale.range` controls how dot size is mapped onto the numbers. If you have an absolute zero value, as we do, it is best to stick with the default "absolute", but this can be changed to "relative".

```
balloonplot( data$Cat, data$Color, data$Count, colmar=3,
             label=F,ylab="Color",xlab="Cat name",
             dotcol=rep(catcolors,each=5),dotsize=12,
             scale.range="absolute")
```

Finally, we can use parameters to unfill the circles (setting dotchar to 1; the dot character can be any character value you typically give to pch in a regular plot), and adding the labels back in a neutral color.

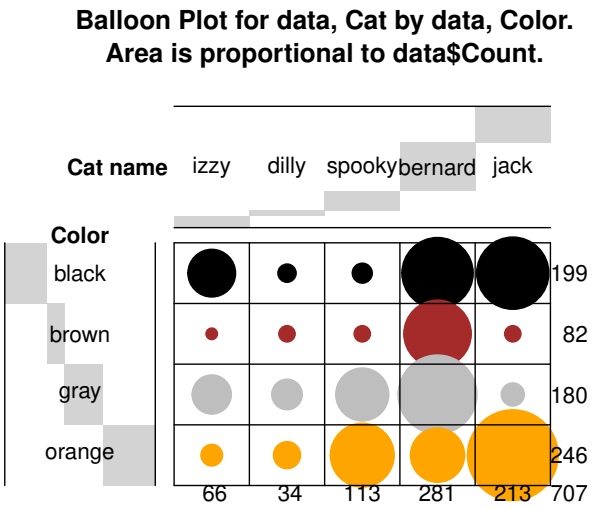
Figure 6.14: Example balloon plot with default parameters.

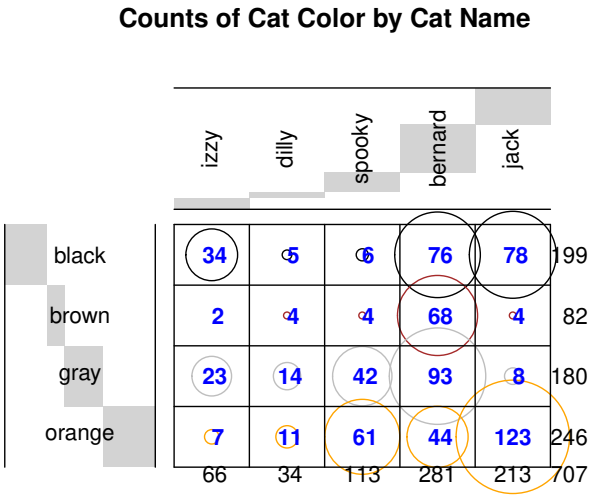



```
balloonplot( data$Cat, data$Color, data$Weight, colmar=3,
           dotchar=1,label.color="blue",
             ylab="",xlab="",main="",
             dotsize=15,colsrt=90,scale.range="relative",
             dotcol=rep(catcolors,each=5))
```

Additional examples can be found at: http://addictedtor.free.fr/graphiques/graphcode.php?graph=60

Figure 6.15: Example balloon plots with additional parameters.





6.3 Gap Plot

A gap plot will displays a plot with one or two missing ranges on one or both axes. The `gap.plot` function is part of the `plotrix` library, which you may have to download and install on your own computer. This can be helpful for plotting several related data series where one has very different values.

The default gap.plot arguments are:

```
gap.plot(x,y,gap,gap.axis="y",bgcol="white",breakcol="black",
brw=0.02,xlim,ylim,xticlab,xtics=NA,yticlab,ytics=NA,
lty=rep(1,length(x)),col=rep(par("col"),length(x)),
pch=rep(1,length(x)),add=FALSE,...)
```

In this example, we will create a series of numbers that have values with means that are around 5 and around 200, and try to plot them with the normal plot functions.

```
#create 40 random numbers
twogrp<-c(rnorm(10)+5,rnorm(10)+200,rnorm(10)*2+5,rnorm(10)+205)

#create color for each group
gpcol<-rep(c(2,3,4,5),each=10)
xvals <- rep(c(1,2,3,4),each=10)

#this is the graph when using "plot" command
plot(xvals,twogrp,col=gpcol,xlab="Index",ylab="Group values",
     main="No Gap on Y axis")
```

Notice the graph that is produced in Figure 6.16 compresses the range within each group, making it hard to tell if there are differences in means or variances.

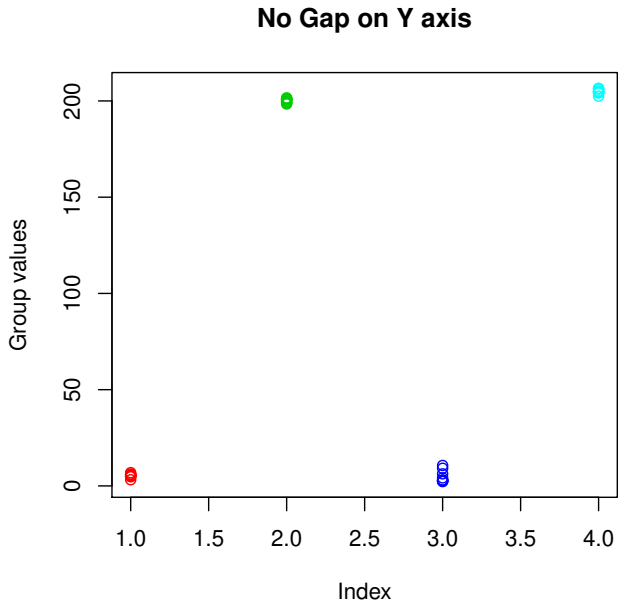
The `gap.plot` function will allow us to cut out the middle sections. The result is shown in Figure 6.17.

```
library(plotrix)
gap.plot(xvals,twogrp,gap=c(10,190),xlab="Index",ylab="Group values",
  main="Gap on Y axis",col=gpcol)
```

The main problem with this graph is we now cannot tell the scale, because the tickmarks get cut off too. This can be fixed with the `ytics` argument. The result is seen on the right panel of Figure 6.17. Also, by adjusting the gap a little so it is just inside the tickmarks, the display is improved.

```
library(plotrix)
par(mfrow=c(1,2))
gap.plot(xvals,twogrp,gap=c(10,190),xlab="Index",ylab="Group values",
  main="Gap on Y axis",col=gpcol,ytics=0:220)
gap.plot(xvals,twogrp,gap=c(11,194),xlab="Index",ylab="Group values",
  main="Gap on Y axis",col=gpcol,ytics=seq(-5,220,5))
```

Figure 6.16: Example plot with data that are separated, making within-group variations difficult to see.



The gap plot will also allow you to add gaps along the horizontal axis, or both simultaneously. Also, you can overlay layers using the `add=T` argument. The code below shows a horizontal gap with a red line overlaid:

```
dat<-rnorm(40)
plot(twogrp,dat,xlab="X values",
     xtics=c(4,7,17,20),ylab="Y values",
     main="Gap on X axis with added lines")

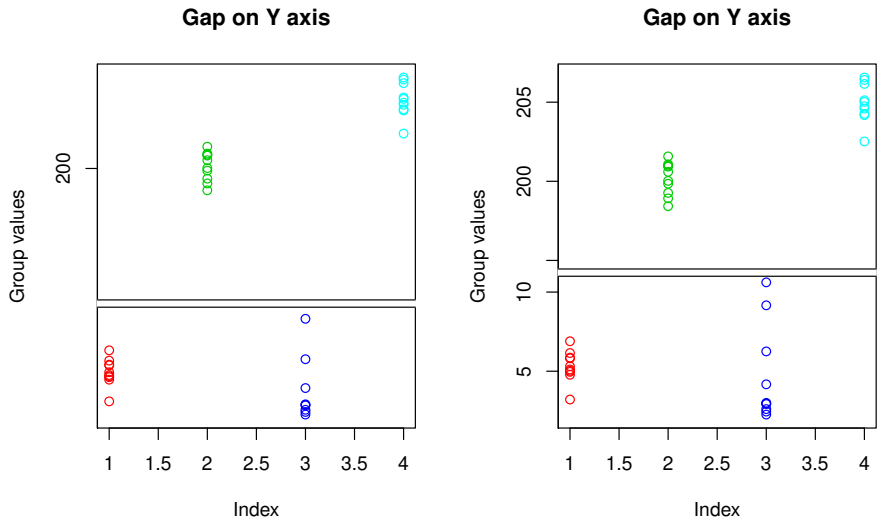
gap.plot(twogrp,dat,gap=c(11,194),gap.axis="x",xlab="X values",
  xtics=c(-2,8,2,seq(190,220,5)),
  ylab="Y values",main="Gap on X axis with added lines")
gap.plot(c(seq(0,7.5,by=0.5),seq(16.5,22.5,by=0.5)),
  rnorm(22),gap=c(11,194),gap.axis="x",type="l",add=TRUE,col=2,)
```

The following shows how you can add two gaps, and how to add overlay lines using `gap.plot`. Note that you must use the same gaps on the overlay function, or it will not plot correctly.

```
dat<-rnorm(40)
gap.plot(twogrp,dat,gap=c(11,194),gap.axis="x",xlab="X values",
  xtics=c(-2,8,2,seq(190,220,5)),
  ylab="Y values",
  main="Gap on X axis with added lines")

```

Figure 6.17: Example plot with the middle cut out, allowing us to see finer differences in the means and variances of the different groups of data.



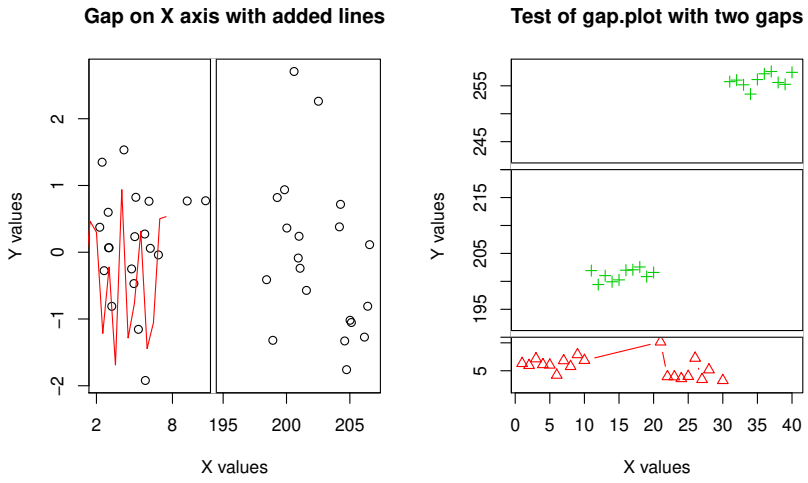

```
gap.plot(c(seq(0,7.5,by=0.5),seq(16.5,22.5,by=0.5)),
  rnorm(22),gap=c(11,194),gap.axis="x",type="l",add=TRUE,col=2,)
```

Now let's alter the data a bit so we can make a two-gap plot. Here, make a copy.

```
twogrp2 <- twogrp+1
twogrp2[31:40] <- twogrp2[31:40]+50

gap.plot(twogrp2,gap=c(11,190,220,240),
 xlab="X values",ylab="Y values",
 main="Test of gap.plot with two gaps",xtics=seq(0,40,by=5),
 ytics=c(seq(0,300,5)),
 lty=c(rep(1,10),rep(2,10)),
 pch=c(rep(2,10),rep(3,10)),
 col=c(rep(2,10),rep(3,10)),
 type="b")
```

Figure 6.18: Example plot with the middle cut out, allowing us to see finer differences in the means and variances of the different groups of data.



6.4 The `barplot2` function

One common complaint of R is that it does not make error bars on your plots. The `barplot2` function, which is found in the `gplots` library, is designed to put error bars on your barplots.

In this example, we will use the VADeaths data set, which record deaths per 1000 people for different age ranges, and is built into R. It does not actually contain confidence regions or measures of variability, but we can estimate those from the data because they are essentially probabilities.

```
> VADeaths
      Rural Male Rural Female Urban Male Urban Female
50-54        11.7          8.7        15.4          8.4
55-59        18.1         11.7        24.3         13.6
60-64        26.9         20.3        37.0         19.3
65-69        41.0         30.9        54.6         35.1
70-74        66.0         54.3        71.1         50.0
```

The first thing we will do, after loading the gplots library, is invert and transpose the order of the table, so it will plot properly:

```
library(gplots)
hh <- t(VADeaths[ 5:1,])
> hh
             70-74 65-69 60-64 55-59 50-54
Rural Male    66.0  41.0  26.9  18.1  11.7
Rural Female  54.3  30.9  20.3  11.7   8.7
```

```
Urban Male    71.1  54.6  37.0  24.3  15.4
Urban Female  50.0  35.1  19.3  13.6   8.4
```

Now, let's set up some variables and precompute some things so that it will be easier to control the plot. First, let's creates confidence interval for the lower side.

One handy thing to know is that the standard deviation of the sampled mean of a binomial distribution is $\sqrt{p \times (1-p)/n}$. That is, if you have a coin that has a probability of coming up heads of p, an experiment with n trials will come up heads with a distribution around p having a standard deviation of $\sqrt{p \times (1-p)/n}$. We can use this in the VADeaths case because these are mortality rates, out of 1000 individuals. So let's compute error bars as ± 1 standard deviation, assuming we are sampling 1000 individuals in each group.

```
ci  <- sqrt(hh/1000 * (1- hh/1000) / 1000)*1000

##creates 95% confidence interval for the upper side
ci.l <- hh -ci*1.96
ci.u <- hh +ci*1.96
```

Now, let's make the barplot. Be sure the `hh`, `ci.u`, and `ci.l` variables are loaded. Explanations for each argument are shown as comments.

```
mp <- barplot2(hh,
  beside = TRUE,  ##TRUE plots them different series beside each other

   ##assigns color to the bar graph (can change colors and orders)
   col = c("lightblue", "mistyrose",
           "lightcyan", "lavender"),
    ##Creates legend using the column names as
    ##the lengend entries and gives min and max for y-axis
    legend = colnames(VADeaths),
     ylim = c(0, 100),
     main = "Death Rates in Virginia per 1000",
     font.main = 4, #Font size of main title
     col.main="darkgreen",  #color of main title

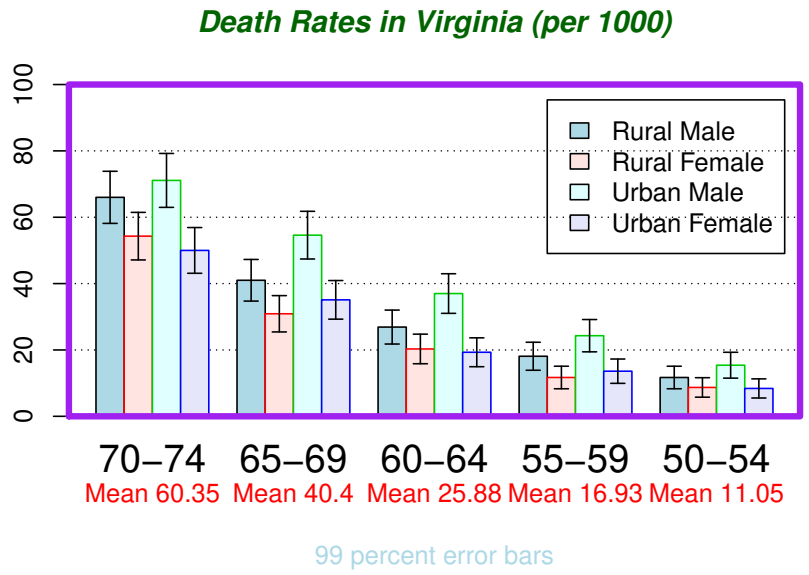
      ##Change the border color of the bars:
        border=c(1:4),
     ##Creates sub title on x-axis, gives color to subtitle
     ## using mybarcol command,
        sub = "99 percent error bars", col.sub = "lightblue",
        cex.names = 1.5,
       ##Plot the confidence intervals:
        plot.ci = TRUE,
        ci.l = ci.l,
        ci.u = ci.u,
        ##Plot gridlines
        plot.grid = TRUE)



##Add margin text to provide summary mean values:
mtext(side = 1, at = colMeans(mp), line = 2,
      text = paste("Mean", formatC(colMeans(hh))), col = "red")
## Also, add a box around the whole graphic
box(col="purple",lwd=4)
```

The resulting plot is found in in Figure 6.19. Think carefully about what the confidence intervals here mean–they may not mean what you think they do. It really is an attempt to indicate where the true mean for each group should lie, assuming we sampled 1000 from each group. But these data may not be a sample from Virginia–they may be the complete population data, and so we may think about the confidence region as if we were using it to estimate the rate in another state, assuming the rates were the same.

Figure 6.19: Example barplot2 with built-in legend and confidence regions.



6.5 The `bandplot` function

The `bandplot` function, found in the gplots library, will plot a scatterplot, and the draw locally smoothed mean and standard deviation lines over the data. The default arguments are:

```
bandplot(x, y, ..., add = FALSE, sd = c(-2:2),
          sd.col=c("magenta", "blue", "red", "blue", "magenta"),
          sd.lwd=c(2, 2, 3, 2, 2),  sd.lty=c(2, 1, 1, 1, 2),
          method = "frac", width = 1/5, n=50)
```

A `bandplot` can be useful to show confidence bounds on data when one value is correlated with another one independent or dependent variable. One example use might be to show norms on some test across an age range, so that one can determine a performance criterion adjusted for age. Note that the current version of gplots (2.10.1), the different smoothing

parameters seem to not operate as expected, but in the future these parameters (method, width, and n) should allow better control over how smooth the estimates are.

For a few simple example, let's create a sets of X and Y values in which the value and/or variance of Y are dependent on X:

Example 1: Variance Scales with square of x Upper left panel of Figure 6.20

```
x<-1:1000
y<-x + .003*x^2 + rnorm(1000,mean=0,sd=1+x^2/1000)
bandplot(x,y,main=paste("Example bandplot\n",
          "Variance proportional to square of value"))
```

Example 2: y and x have a curvilinear relationship. As seen in the upper right panel of Figure 6.20, the bands plotted are robust to the form of the functional relationship, and do not depend on assumptions about polynomials. This figure shows some examples of how to alter the visual appearance.

```

n<-1000
x <- round(runif(n, 0,100))
y <- x - (x-25)*(x-50)+ rnorm(n)*600
# 0 stands for mean
bandplot(x,y,sd=c(-1,0,1),col="grey",sd.col=c(1,4,1),
          sd.lty=c(3,1,3),sd.lwd=c(2,4,2),
          main = "Band plot with only +/- 1 s.d.")
```

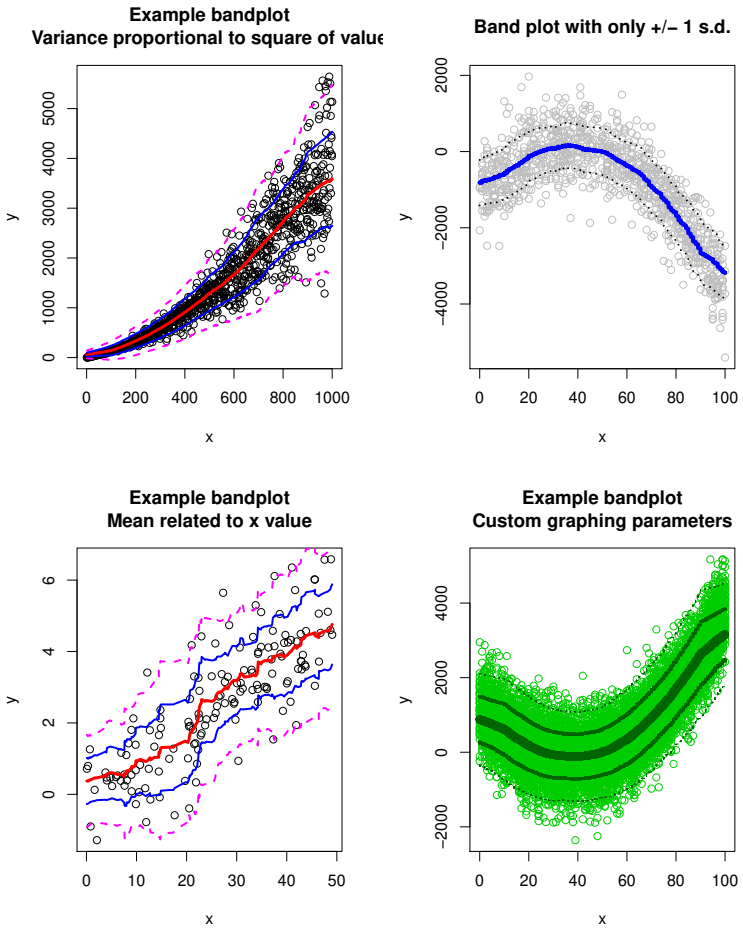
Example 3: y correlated with x with relatively few observations As shown in the lower left panel of Figure 6.20, the bands can be quite rough when you have relatively few observations.

```
n <- 150
x <- runif(n)*50
y <- rnorm(n, mean=x/10, sd=1)
bandplot(x,y,main="Example bandplot\nMean related to x value")
```

Example 4: Custom graphing settings Lower right panel of Figure 6.20

```
n<-10000
x <- round(runif(n, 0,100))
y <- (x-25)*(x-50)+ rnorm(n)*600
bandplot(x,y,main="Example bandplot\nCustom graphing parameters",
         col="green3",sd.col=rep("darkgreen",5),sd.lwd=c(1,3,8,3,1))
```

Figure 6.20: Example bandplot with built-in confidence regions.



6.6 The `pyramid.plot` function

The pyramid plot is designed to show distributions of men and women across age ranges. However, it can be adapted for many situations where you want to compare histograms of two groups side-by-side. It is in the `plotrix` library, and its default arguments are:

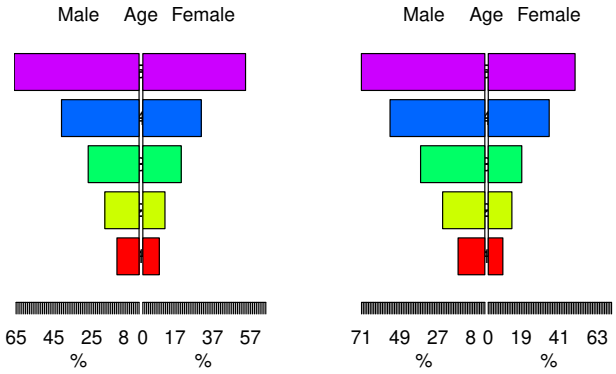
```
pyramid.plot(lx,rx,labels=NA,top.labels=c("Male","Age","Female"),
 main="",laxlab=NULL,raxlab=NULL,unit="%",lxcol,rxcol,gap=1,
 ppmar=c(4,2,4,2),labelcex=1,add=FALSE,xlim,show.values=FALSE,ndig=1)
```

To test this out, we can use the VADeaths data set that was used in the `bandplot2` example. Since there are two groups (urban and rural), we'll make two plots next to one another:

```
par(mfrow=c(1,2))
pyramid.plot(VADeaths[,1],VADeaths[,2])
pyramid.plot(VADeaths[,3],VADeaths[,4])
```

As seen in Figure 6.21, the default settings are not very good for this data set.

Figure 6.21: Two examples of `pyramid.plot` with default settings.



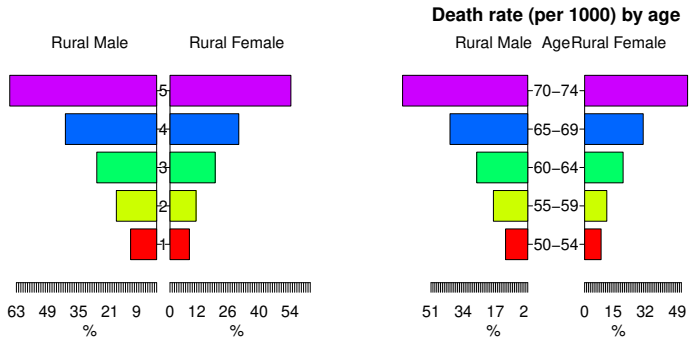
Let's see how we can fix it. To start, we can add labels on the top and increase the gap between the plots to allow us to see the category labels. Although the original stack labels were correct (male versus female), we will put in our column headers, which also tell us urban/rural:

```
pyramid.plot(VADeaths[,1],VADeaths[,2],
             top.labels=c(colnames(VADeaths)[1],
             "",
             colnames(VADeaths)[2]),
             gap=3)
```

The result is shown in the left panel of Figure 6.22. Next, we will add a main title, add the age labels using the labels command, and increase the gap between the two even more to accomodate. The result is found on the right panel of Figure 6.22.

```
pyramid.plot(VADeaths[,1],VADeaths[,2],
             top.labels=c(colnames(VADeaths)[1],
                          "Age",
                          colnames(VADeaths)[2]),
             gap=15, main="Death rate (per 1000) by age",
             labels=rownames(VADeaths))
```

Figure 6.22: Two examples of improvements to the `pyramid.plot`.

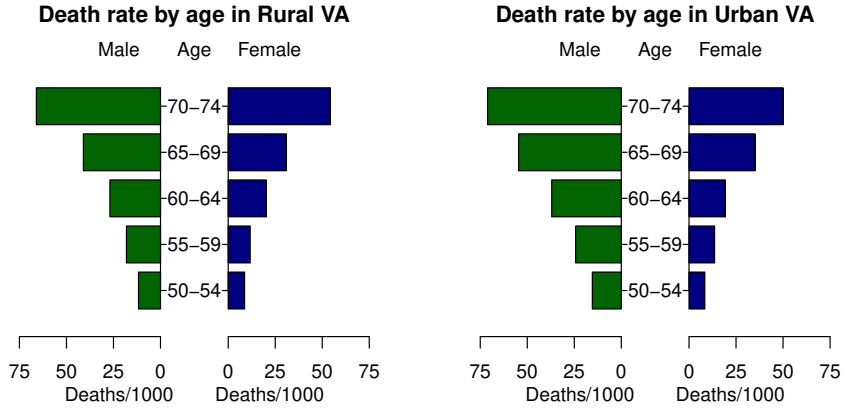


These plots are getting to look respectable. I made a number of other tweaks to captions and x ranges/labels, to show the final plot in Figure 6.23.

```
par(mfrow=c(1,2))
pyramid.plot(VADeaths[,1],VADeaths[,2],xlim=c(80,80),gap=18,
             top.labels=c("Male","Age","Female" ),
             main="Death rate by age in Rural VA",
             lxcol="darkgreen",rxcol="navy",
             laxlab=c(0:3*25),raxlab=c(0:3*25),
             labels=rownames(VADeaths),unit="Deaths/1000")

pyramid.plot(VADeaths[,3],VADeaths[,4],xlim=c(80,80),gap=18,
             top.labels=c("Male","Age","Female"),
             main="Death rate by age in Urban VA",
             lxcol="darkgreen",rxcol="navy",
             laxlab=c(0:3*25),raxlab=c(0:3*25),
             labels=rownames(VADeaths),unit="Deaths/1000")
```

Figure 6.23: Final improved `pyramid.plot` with both urban and rural data.



6.7 Other Graphics Packages of Note

There are many special-purpose and general-purpose graphics packages available in R.

- *ggplot* The most popular are ones related to ggplot2, which provides a completely reworked graphics system for R. It provides flexibility and makes attractive graphics easy, at the cost of a more complex syntax and more difficulty in customization.
- *plot.ly*. The plot.ly library is not tied directly to R, but can be accessed via a number of systems. It creates nice interactive and dynamic graphs suitable for on-line use. They are generally interactive, zoomable, and allow users to explore data.
- *maps*. This package provides ways to plot geographical data.
- *leaflet*. This provides another means for mapping geographical data, with interactive web-based maps.
- *imager*. Provides hooks to fast image-processing routines.
- *gplots*. A Library containing many special-purpose graphics utilities. Along with several functions we have looked at already, it includes:
  - *venn* Make a venn diagram
  - *angleAxis* Plot axis labels at an angle.
  - *heatmap2* A heatmap with a dendrogram.
- *plotrix*. A library containing dozens of specialty graphics and graphical functions. Some of my favorites include:
  - *battleship.plot* Like a bubble plot, but with squares
  - *bumpchart* Track change of a number of values across two time periods
  - *centipede.plot* Ordered dot-chart with confidence regions
  - *clock24.plot* Plot time information in a radial plot; see also radial.plot
  - *draw.circle,draw.arc, draw.ellipse* Graphical elements
  - *gantt.chart* Create timelines for resources
  - *diamond.plot* Create radial axes for multiple series
  - *radial.plot* Radial plot with radians as axis
  - *polar.plot* Radial plot with degrees as axis
  - *histStack* Create a stacked histogram displaying multiple distributions
  - *thigmophobe.labels* Creates labels that don't overlap the points.

6.8 Solutions to Exercises

Exercise Solution 6.1.2

Create three series of 5 numbers (i.e., a matrix with 3 columns and 5 rows). Create a side-by-side matplot and a stacked barplot with those values, and default color settings. Then, Pick a set of at least five named colors. Set the palette to these colors, and remake the same plots, so that they use these colors.

```
data <- matrix((1:15)[order(runif(15))] ,ncol=3)
par(mfrow=c(2,2))
palette("default")
matplot(data,type="b")
barplot(data)
palette(c("seagreen","orange","gold","navy","red4"))
matplot(data,type="b")
barplot(data,col=palette())
```

Exercise Solution 6.1.3

Plot a matrix of values using image, with several different gplots color gradients.

```
library(gplots)
vals <- matrix(runif(100),10,10)
image(vals,col=colorpanel(50,"black","green"))
image(vals,col=redblue(50))
```

Exercise Solution 6.1.4

Use a 'divisive' color palette from color brewer. Use it to make a non-stacked barplot from a data table having two columns and five rows, so that the plot has two groupings of bars, and each set of bars has the same color scheme.

```
pal <- brewer.pal(5,"RdYlBu")
data <- cbind(c(5,2,3,2,1),c(9,1,3,4,1))
barplot(data,beside=T,col=pal)
```

Exercise Solution 6.1.8

When you have a scatterplot of integers points, you tend to get points plotting on top of one another, which hides the number of observations. You can get a better notion of this sometimes if you use large points with transparency. Those locations with more points will show up darker. Use a large point size and

```
x <- sample(1:10,100,replace=T)
y <- round(x + rnorm(100)*2)
mycol <- rgb(.8,.2,.3,.5)
par(mfrow=c(1,2))
plot(x,y)
plot(x,y,col=mycol,cex=2,pch=16)
```

Chapter 7

Random Variables, Probability, and Parameter Estimation

> We buy knowledge with the assumptions we make; all knowledge is paid for; if the assumptions are correct, we have a bargain.
>
> Clyde Coombs, *A Theory of Data*

7.1 Randomness, the unknown, and models of reality

The field of statistics is about trying to draw inferences about things that happen in the world that are unknown. This is the essential quality of randomness–the causes and results are not known with certainty. In the physical world, we believe that you could predict the future exactly if you were able to account for all of the factors and causal relationships–all the particles in the universe and the forces that describe their interactions. There are debates within quantum physics whether this is really true, but for large-scale phenomena, what is true is that the factors that produce all but the simplest phenomena are too complex–and thus unknowable–to make definitive predictions. In a sense, randomness is not a property of the universe–it is a property of our understanding of the universe, and a consequence of information we do not know.

When trying to understand something that is happening in the world, we build a statistical model that approximates as closely as is reasonable what is happening in the world. Since the events in the world we care about have randomness, the model has to represent that uncertainty or unknown aspects. In probability and statistics, these models are generally based on concepts called *random variables*. We tend to pick random variables that are well understood and that are justified models of the phenomenon. Then, we can use mathematical inference based on calculus, probability theory, simulation, or other methods to make predictions about how the events in the world should behave if the model is correct. The model is rarely 100% correct. Even for something like flipping a coin, in reality there is a small chance that during flip, the coin will land on its edge. If we feel that the chances of this are small or the consequences of this outcome are negligible, we might be happy to consider modeling a coin flip as a two-outcome process.

7.2 Random variables and sample spaces

A random variable is a mathematical concept we use to model the outcome of real processes that have uncertainty. We can refer to these outcomes as events. Random variables are defined, in part, by their sample space–a set of possible outcomes of the event. For the coin flip, our model of the world would have two states–heads and tails. The sample space of rolling a die is the values 1 through 6; the sample space of the age of a person chosen at random from the population will be a value greater than 0 and less than the oldest person in the population. The sample space itself does not tell you how likely different outcomes are–just what values they may take on.

Because the sample space refers to the random variable, which is a model of the process, we are usually content with there being some mismatches between the random variable's sample space and reality. In reality, there are other things that could happen besides a coin landing heads or tails (it could roll down a drain and never been found). Similarly, if we measure down to the second, researchers think there are about four babies born every second on earth. So the sample space for ages of people, even if measured down only to the second, is not a continuous variable but really a countably but large set of values. But we are usually content to model the sample space as a continuous variable.

So, a random variable describes a variable that can take on different possible values, and the relative chances of each of the possible values. In order to make further progress, mathematicians have made a set of assumptions (called axioms) about how to assign a value called probability to the sample space. Probability theory is based on these axioms. Curiously, these are assumptions–not proven facts. We have simple made these assumptions, and work within the consequences. In fact, several mathematicians have explored alternative axiom sets, which are at odds with the common assumptions of probability theory.

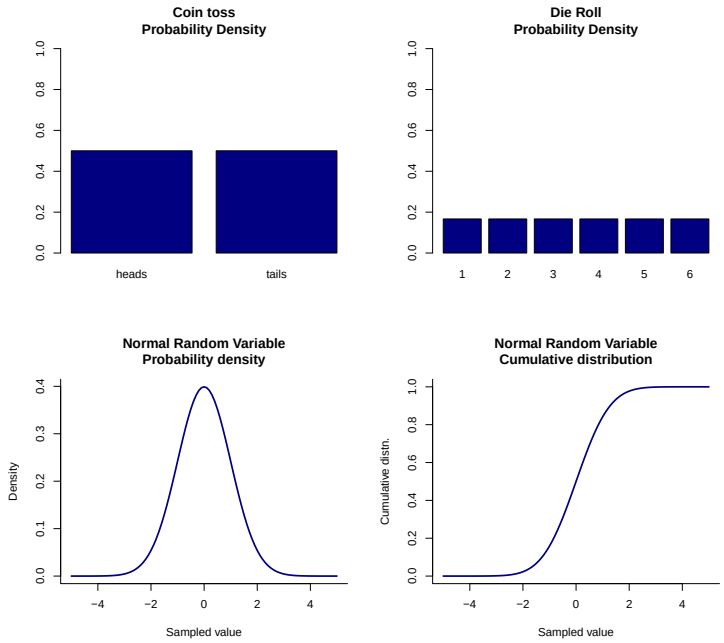
The axioms of probability theory describe how we assign a number portions of the sample space of an event. These portions are referred to as subspaces, and simply mean any possible set of outcomes. The subsets are much like the possible bets one can place at a roulette wheel. In roulette, you can place bets on red, black, any particular outcome, sets of four outcomes, etc. You can place money on every single outcome so that you are sure to win, or no outcomes so that you are sure to not win. The possible subspaces of the sample space are all possible bets you might make, even though the simplest bet is a single number. We can carve up any sample space into subsets. In the case of the flip of a coin, the subsets include just four possibilities: the null set ,heads, tails, or heads or tails. For the roll of a die, the sample space is the values 1 through 6, and any combination of values (e.g., even, odd, $1, 2, 6$, the entire set, or the null set which is empty. For continuous variables, subsets could be any range of values (e.g., 10 years old, a teenager, between the ages of 3 and 26). For continuous variables, a single value (exactly 10 years old to the second) is an empty set, but ranges of values have probability.

The axioms of probability theory tell us how probability may be assigned to any possible subset of the sample space. To assign probability measures to subsets of the sample space, we use these assumptions:

- The probability of an empty subset is 0.
- The probability of the entire set is 1.
- The probability of the union of several mutually-exclusive subsets is the sum of the probabilities of the subsets.

These three assumptions are the basis for all of probability theory. The third axiom gives us the rule to combine probabilities. It says, for example, if the probability of a die coming

Figure 7.1: Density for a fair coin, a fair die, and a normal distribution; cumulative for the normal distribution.



up 1 is 1/6, and the probability of the die coming up 2 is 1/6, then the probability of the die coming up either 1 or 2 is $1/6 + 1/6 = 1/3$.

If we are able to assign probabilities to all possible subsets of the sample space via some mathematical formula or table, this is called the probability distribution. According to the first axiom, any impossible event has probability 0, and according to the second axiom the probabilities of all possible events together must sum to 1.0. For example, the chance of a coin landing heads plus the chance of a coin landing tails adds to 1.0, even if the coin is not fair. In other cases, where the sample space is continuous, it only makes sense to ask about the probability of ranges of values. For discrete sample spaces (like the flip of a coin), this is often called the probability mass function (pmf), whereas for a continuous distribution it is called the probability density function (pdf).

For continuous distributions, like when modeling the chance of a dart hitting the wall, the probability of landing at any specific spot is 0, even though some locations (near the bullseye) are more likely than others. The use of a probability density function allows us to measure the total probability within any specific space on the wall. For discrete random variables, the equivalent is called the probability mass function, because discrete events do take on probability. In many cases, we would like to define or examine the cumulative density or mass function of a distribution (sometimes just called the "cumulative"). The cumulative is a function whose domain is the sample space, and whose range is $[0, 1]$, that describes the probability of a sample occurring that is less than or equal to particular value–the integral from the minimum to a particular value. In contrast, the density and mass functions itself is the first derivative of the cumulative distribution, and also provides a measure of the relative likelihood of each possible value.

So, to summarize, the random variable is a model of an random process in the world, that lets us capture this randomness. It maps values called probability to the subsets of the

sample space (possible outcomes) of the event. The way this mapping is done is referred to as the distribution, and distributions can either be discrete (with a countable number of possible outcomes) or continuous (with an uncountable number of outcomes).

Figure 7.1 shows the probability mass/density and cumulative distribution functions for several distributions.

Some Examples:

- Discrete uniform random variable: takes on a fixed number of values with equal probability. (can be simulated with sample)
- Uniform random variable: takes on a continuous value between two endpoints (often 0 and 1; can use `runif()`)
- Binomial random variable: number of times a trial succeeds out of N, when a given probability of success exists.
- Normal random variable: takes on a bell-shaped curve with a mean and spread.

A random variable is conceptually like a function in that it may take parameters that control its overall shape, such as its center, its shape, etc. The important thing a random variable provides us is a model of a stochastic process–one in which the outcome is not always the same. Typically, in order to model a particular real-world process, we will make an assumption about what type of random variable it is, and then try to estimate the parameters using data, so that we can make inferences about the population we have sampled from.

For example:

- For a discrete uniform random variable, there might be no real parameters we estimate. If we assume a die is fair, our distribution is just given.
- for a more general discrete random variable, there might be a parameter associated with n-1 of the options (with the last fixed or determined by the rest)
- For a uniform random variable: a parameter might determine the minimum and maximum values of the range.
- For a binomial random variable: a parameter might be the probability p of giving response '1', and the number of trials.
- For a Normal random variable: parameters might be the mean and variance of the distribution. The Normal RV has a density function defined by $f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, but there is no formula for its cumulative distribution function, which means we can only estimate it with simulation or numerical estimates.

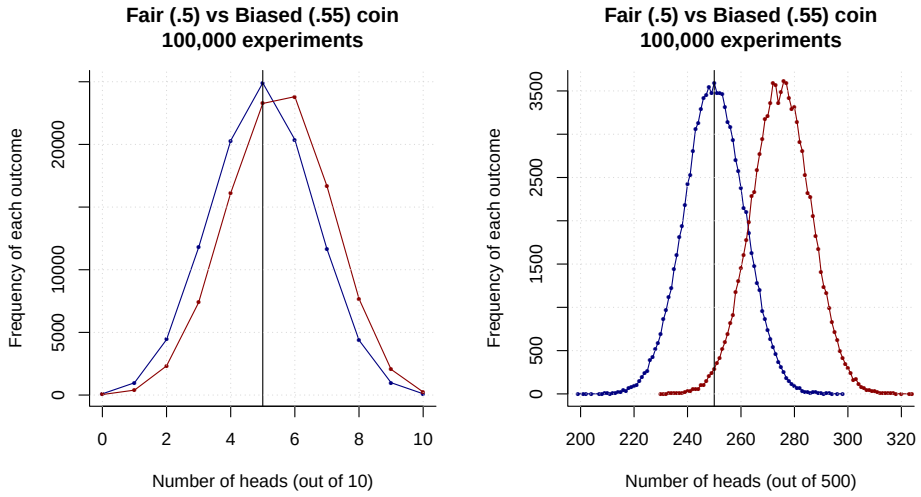
We can typically generate random data from a distribution associated with any these random variables. If we understand the distribution, we can draw inferences about the likelihood of certain observations. A simple example would be trying to make an inference about whether a coin is biased. We assume that the number of times a coin lands as heads in a set of coin flips (an experiment) is a binomial random variable. A fair coin (with 50% chance of landing heads) will produce a roughly even number of heads and tails, but how can we tell if the coin is biased? We sample observations of the coin, use those observations to estimate the distribution (model), and then use the mathematics to determine the likelihood that the observed data could have been generated from a fair vs unfair coin. Even if we can't figure out the mathematical formula to help us understand whether a particular outcome is unlikely, we can often simulate it:

```
heads <- function(prob=.5,flips=1,repetitions=1)
{
  base <- matrix(runif(flips*repetitions)<prob,
               ncol=flips,nrow=repetitions)
  return(rowSums(base))
}
  set.seed(100)
  fair.10 <- table(heads(.5,10,100000))
  biased.10 <-table(heads(.55,10,100000))

  fair.500 <- table(heads(.5,500,100000))
  biased.500 <-table(heads(.55,500,100000))
```

Here, suppose you wanted to know if a coin was doctored so that it was slightly more likely to land heads than tails. Even a slight bias would help a gambler come out on top. So, suppose you flipped the coin 10 times. The left panel of Figure 7.2 shows the distribution of a fair and a slightly biased (.55) coin. For a single experiment of ten flips, even if it landed heads 9 or 10 times out of 10, this would not be overly strong evidence that the coin was unfair. You could never detect a slight bias just by flipping a few times. On the other hand, if you took a few hours and flipped the coin 500 times, the case is slightly better. The fair and biased distributions are shown on the right side of Figure 7.2. Now, there is a difference in the distribution, but even so the difference is not large enough to be highly confident in many cases. Suppose you got exactly 250 heads in this experiment. In only about 1.4% of the biased coin experiments did we see a value this small or smaller, and you might be fairly confident that the coin is not biased. But what if you saw around 264 heads? about 2000 cases in each experiment had 264 heads, and so it was about equally likely to have happened from either coin. But, as the number of heads increase to 270 or 280, the chance that a fair coin could have produced this is very unlikely, and you would be justified in calling the coin biased.

Figure 7.2: Simulation of 100,000 experiments where we flip a fair and a slightly biased coin either 10 or 500 times. For experiment size of 10, it would be difficult to catch a biased coin on a single experiment. For an experiment of size 500, you can be reasonably confident the coin is biased as the number of flips exceeds 280.



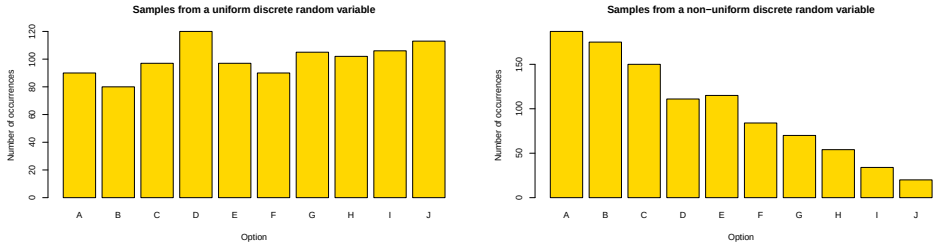
Some common random variables that can be sampled from are described below:

7.2.1 Discrete uniform:

A discrete uniform random variable has an equal chance of sampling each of a number of categories, such as heads vs tails, sides of a die, or lottery/bingo balls. The following will generate 1000 samples from a 10-category discrete uniform distribution, as shown in Figure 7.3.

```
set.seed(100)
data <-   sample(LETTERS[1:10], 1000, replace=T)
barplot(table(data),col="gold",ylab="Number of occurrences",
        xlab="Option",
        main="Samples from a discrete uniform random variable")
```

Figure 7.3: Simulation of 1000 draws from a discrete uniform (left) and non-uniform (right) distribution.



7.2.2 Discrete non-uniform:

A related distribution would assume the different categories are not uniform. Using sample with the `prob` argument will sample from a specific non-uniform distribution, also shown in Figure 7.3. This sort of distribution is normally referred to as simply a categorical distribution.

```
set.seed(101)
data <-   sample(LETTERS[1:10],
                 prob=10:1,1000, replace=T)
barplot(table(data),col="gold",ylab="Number of occurrences",
        xlab="Option",
        main="Samples from a non-uniform discrete random variable")
```

7.2.3 Continuous Uniform Distribution

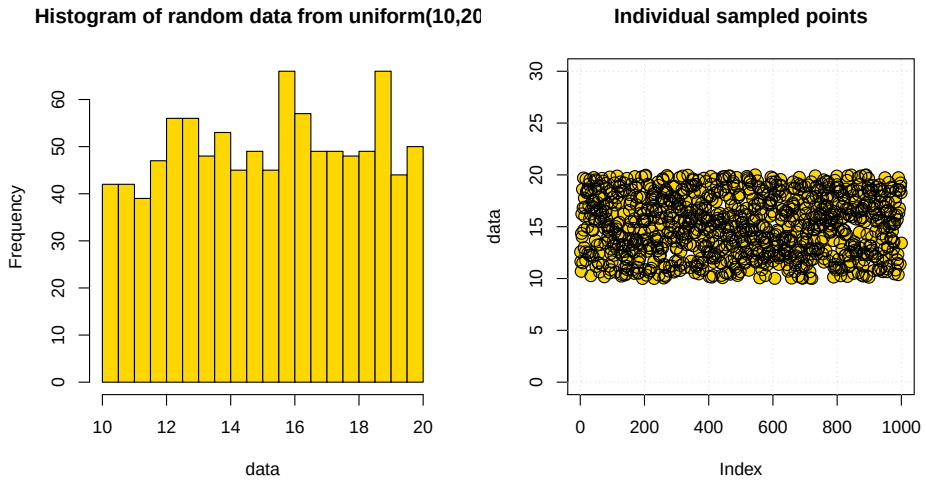
The continuous uniform distribution has an equal chance of any number between a minimum and maximum. For example, a uniform distribution with minimum 10 and maximum of 20 could be generated with the following:

```
data <- 10 +runif(1000)*10
par(mfrow=c(1,2))
hist(data,breaks=20,main="Histogram of random data from uniform(10,20)",col="
    gold")
plot(data,ylim=c(0,30),pch=16,col="gold",cex=1.5,main="Individual sampled
    points")
grid()
points(data,cex=1.5)
```

7.2.4 Binomial distribution

Single coin flip is modeled as a bernoulli distribution; it has one of two states (represented as 0 or 1), and it achieves state 1 with a probability p, and state 0 with probability $1 - p$. If we repeat this N times and count the number of outcomes that are 1, this is a binomial distribution. Thus, a binomial distribution has two parameters, the probability and the number of samples. Generally, the number of samples is fixed by the context of the experiment, and you might try to estimate p from data. Sample repeatedly from a

Figure 7.4: Simulation of 1000 draws from a uniform distribution with min 10 and max 20.



binomial distribution with a function such as seen in the listing below. Here, we compare the distributions with two different p arguments, either $p = .4$ or $p = .8$, and a common $N = 15$ (see Figure 7.5).

```
##Binomial distribution

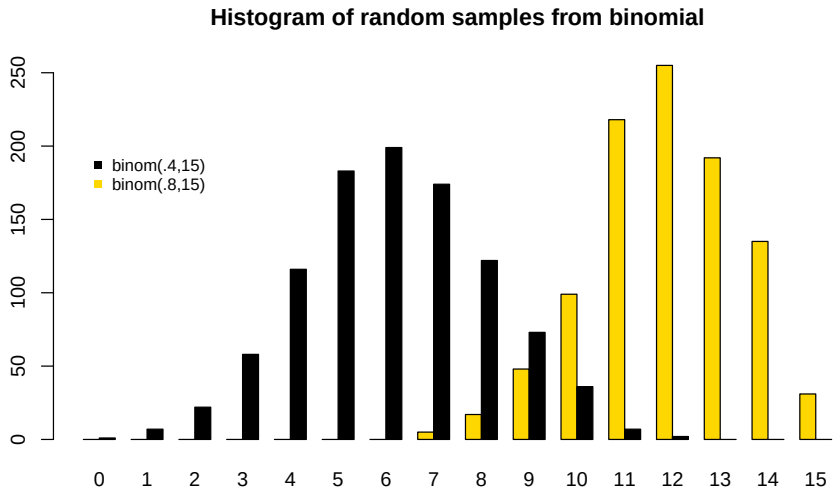
makebinomial <- function(n=100,p=.5)
{
  sum(runif(n) < p)
}

data1 <- sapply(1:1000,function(x){makebinomial(n=15,p=.8)})
data2 <- sapply(1:1000,function(x){makebinomial(n=15,p=.4)})

tab<- rbind(sapply(0:15,function(x){sum(x==data1)}),
              sapply(0:15,function(x){sum(x==data2)}))


barplot(tab,main="Histogram of random samples from binomial",
        beside=T,col=c("gold","black"),names=0:15)
legend(1,200,c("binom(.4,15)","binom(.8,15)"),cex=.8,bty="n",
       pch=15,col=c("black","gold"))
```

Figure 7.5: Simulation of 1000 draws from two binomial distributions, with different p parameters.



7.2.5 The Normal distribution

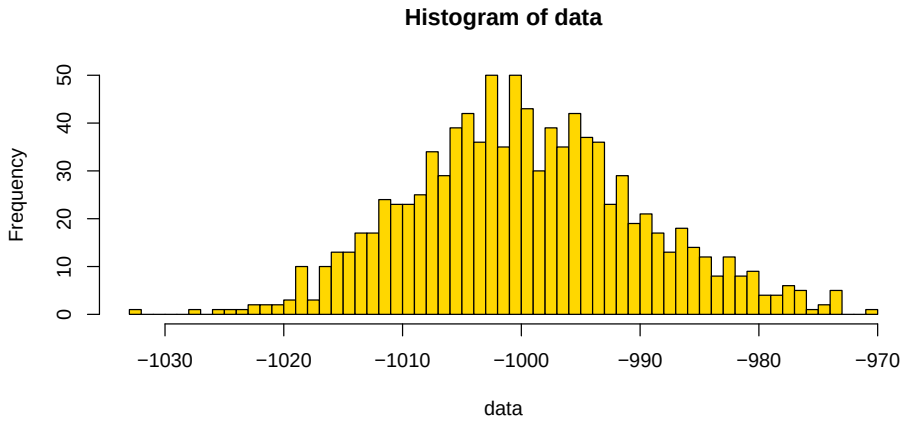
The normal, or gaussian, distribution is a special distribution, used frequently in statistics. It is a continuous distribution that has a mean value, is symmetric around the mean, and has the greatest chance of values being close to that mean, and is defined over the entire range of values from negative to positive infinity–its sample space is the entire line of real numbers. There are other distributions with very similar shapes, but the normal distribution can be mathematically proven to arise naturally out of many processes that involve composites of almost any other random variable. Thus, it is usually a good guess to model any natural process. As discussed earlier, we know the mathematical form of the density function, which is the shape of the well-known bell curve. But the distribution function is usually more important, because it allows us to measure the probability of a subspace of the sample space. For example, we often want to know the area to the left of a particular value we estimate, which we use to compute the 'p-value' of a statistical test. Also, the cumulative distribution would let us sample from the distribution to help do inference or simulation. But unfortunately, there is no closed-form solution to the distribution function, so we need to estimate it with approximations.

To generate samples from a normal distribution with mean -1000 and s.d. 10, whose histogram is shown in Figure 7.6. Notice that the shap is actually quite similar to the binomial distribution examined earlier.

```
data <- rnorm(1000)*10-1000
hist(data,breaks=50)
```

R has a number of related functions for a handful of distributions. For the normal distribution, these include:

Figure 7.6: Simulation of 1000 draws from a normal distribution.



1. `dnorm(x, mean = 0, sd = 1, log = FALSE)` computes the density (or likelihood) of the normal distribution.

2. `pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)` computes the cumulative probability

3. `qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)` computes quantiles of probabilities.

4. `rnorm(n, mean = 0, sd = 1)` generates random values from the specified normal distribution

Examples:

```
> round(dnorm(-4:4),3)
[1] 0.000 0.004 0.054 0.242 0.399 0.242 0.054 0.004 0.000

> round(pnorm(-4:4),3)
[1] 0.000 0.001 0.023 0.159 0.500 0.841 0.977 0.999 1.000

> round(qnorm(1:9/10),3)
[1] -1.282 -0.842 -0.524 -0.253  0.000  0.253  0.524  0.842  1.282
```

Here, dnorm and pnorm are the density and cumulative density function; qnorm is the inverse–it give the sample value associated with a particular point on the cumulative density function.

Exercise 4.1.2

Generate and view 1000 or more samples from:

(a) A normal distribution having mean 100 and s.d. .1

(b) A normal distribution having mean 100 and s.d. 20

(c) A uniform distribution 100 and 101.

(d) A discrete distribution among 20 elements, where element n has a probability proportional to 1/n.

(e) A process where you roll a 10-sided die, and based on the result, you sample a random normal distribution with *both* mean equal to 25 times the result, and standard deviation equal to the result of the die-roll.

7.3 Comparing data to a theoretical distribution

Often, our statistical tests will make assumptions (both strong and weak) about the type of distribution we have. The purpose of using a random variable to model a process in the world is that we can draw conclusions about the random variable by examining it, and thus make inferences about what is likely to have been true about the world.

If our assumptions are violated, our conclusions may not follow. In fact, our assumptions are almost always violated in one way or another, and so it can depend on how badly the violation is. For example, some properties of the normal distribution include:

1. It is continuous–any rational number could be observed (usually this is violated by the precision of our instruments, whether it is a 7-point scale, or a 1-ms timer on the computer)

2. Its range is infinite–there is no minimum or maximum possible value. This is often violated practically on data that are scales (which have to be truncated at the bottom and top), and also on things like accuracy and response time (which can't go below 0)

3. It is symmetric. This is frequently violated on many kinds of data, such as ratings scales, response times, etc.

4. It has a gaussian shape. There are many similarly-shaped distributions (the logistic is one), whose shape differs slightly from the normal distribution. Practically, it would can be impossible to tell which was the more appropriate model without collecting thousands or tens of thousands of observations.

5. The data arose from an unchanging (stationary) process. This is almost always violated, because there are usually sequential tendencies, mixtures of multiple processes (such as a 'paying attention' and 'not paying attention' process that can lead to individual outliers, non-stationarity, dependency, etc.)

If we have data, we can use histograms, boxplots, violin plots, and other graphical tools to help us determine how badly our assumptions are violated. We will look at this more seriously later in the book when we discuss testing assumptions of statistical tests. But one of the most useful (and least-used) ways of examining the distribution is what is known as a q-q plot. A q-q plot compares the quantiles of observed data to the corresponding theoretical quantiles of the distribution you believe it comes from. If these are similar, they form a straight line, and indicate the shape of the distribution conforms to the assumptions. For example, if you had 100 data points, you would compare these values in rank order to the 0:99th quantile of the hypothesized distribution. The nice thing about a q-q plot is that it is scale-invariant, so it doesn't matter what the mean or variance of your data is.

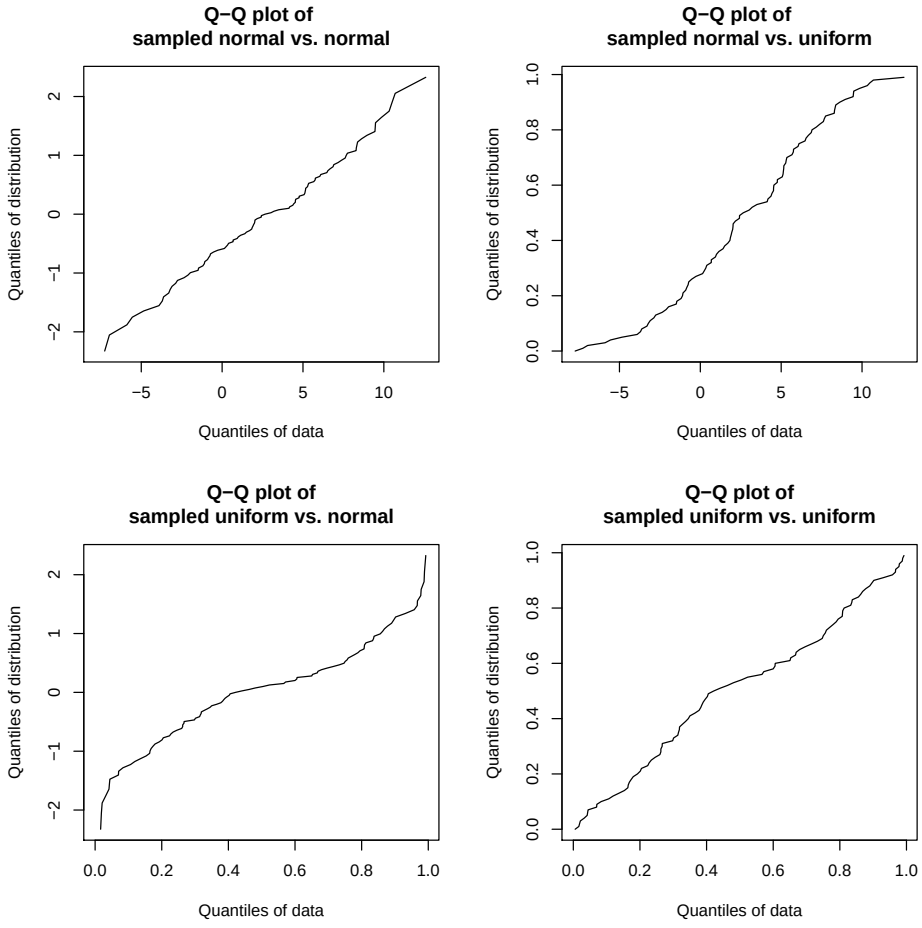
Here, we can compare numbers generated from a normal and a uniform distribution to both the normal and uniform.

```
data <- rnorm(100)*5+3
data2 <- runif(100)
par(mfrow=c(2,2))
plot(sort(data),qnorm(0:99/100),type="l",
     main="Q-Q plot of \nsampled normal vs. normal",
     xlab="Quantiles of data",ylab="Quantiles of distribution")
plot(sort(data),qunif(0:99/100),type="l",
     main="Q-Q plot of \nsampled normal vs. uniform",
     xlab="Quantiles of data",ylab="Quantiles of distribution")

plot(sort(data2),qnorm(0:99/100),type="l",
     main="Q-Q plot of\nsampled uniform vs. normal",
     xlab="Quantiles of data",ylab="Quantiles of distribution")

plot(sort(data2),qunif(0:99/100),type="l",
     main="Q-Q plot of\nsampled uniform vs. uniform",
     xlab="Quantiles of data",ylab="Quantiles of distribution")
```

Figure 7.7: QQ-plots of 100 sampled data points from to different distributions, compared to theoretical normal and uniform. The appropriate distribution shows a straight line.



Notice that when random values sampled from a normal distribution are compared to a normal, the q-q plot is a straight line; similarly when uniform values are compared to the uniform density, it is also straight. This is a sign that you have chosen the correct distribution.

Because the q-q plot is so useful, you can use the built-in `qqplot` and `qqnorm` functions to automatically form q-q plots against arbitrary distributions or against the normal.

Exercise 7.3

Using the data set c7rts.csv, form a histogram and normal q-q plot for the response time (rt) data. Is it well described by the normal distribution? Then, transform the data with a logarithm and plot the histogram and qq-plot. How about now? In what ways might the original and transformed data mismatch the normal distribution. Why?

7.4 Inferential Statistics

We typically make assumptions about what random variable is used to model a specific process. We often don't even test those assumptions, and many times cannot test them well–although they sometimes can be. In fact, we typically assume a single random variable–the normal distribution. We use this assumption to help us answer a specific inferential question: based on what we have observed, what is the 'true' situation among the population we sampled from? More specifically, we might want to know if one of two advertising campaigns is better than another, or if a memory training intervention actually improves measures of memory in comparison to before the training.

The definition of a "statistic" is a function computed on data. For example, mean, min, median, (sqrt(max)-3*min + mean) are all statistics of data. Usually, when someone says they are "doing statistics" on their data, they are doing *inferential* statistics–trying to infer something about the true state of the world based on a sample. In many applications of human factors, psychology, and data science, the true state of the world is something that is true of a given population (of people, or cognitive states, or neurons, etc.), and your sample is the set of observations (people, trials, etc.) drawn to represent that population. Ideally, you don't have to observe the whole population to understand what is the truth–you can observe a relatively small sample and *infer* what the true state of the population is.

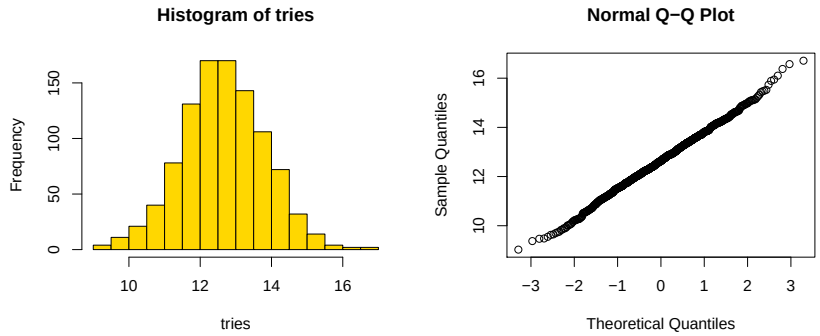
The process of 'doing statistics' is really the process of:

1. Sampling data from a population of data
2. Identifying a random variable as the model of the sample space and probability distribution.
3. Estimating the parameters of the distribution based on a sample of observations.
4. Trying to make an inference about the generating population based on these assumptions.

7.5 Parameter Estimation

In the previous section, we discussed that you must estimate parameters of your distribution in order to make inferences. In psychology, many mathematical theories create distributions

Figure 7.8: The histogram and qq-plot of a mystery distribution



of simulated data, and we'd like to get a best guess at some of those parameters to determine whether they differ across groups whose data we are modeling. For example, parameters of a multi-layer neural network are all of the connection weights between all the nodes, as well as some other tuning and learning parameters. In general, how might we estimate these parameters?

There are a number of ways to estimate parameters, and a lot of the recent work in statistical computing has involved ways to estimate parameters from more and more complex models that presumably better describe the world. A crude, but sometimes effective, way to estimate parameters is an 'eyeball' method. For example, consider the following data, whose histogram is shown in Figure 7.8.

```
x <- runif(100)*10
x2 <- (x-5)^2+ x*5 + 15 + rnorm(100)*30
par(mfrow=c(1,2))
hist(x2,breaks=20)
qqnorm(x2)
```

If we wanted to model this with a normal distribution, we need to estimate two parameters– the mean and variance of the distribution. We might guess it comes from a normal distribution whose mean is about 50, but we may really have no idea about its variance/spread. We can always create a function that simulates data from a given distribution, and visually compare the two:

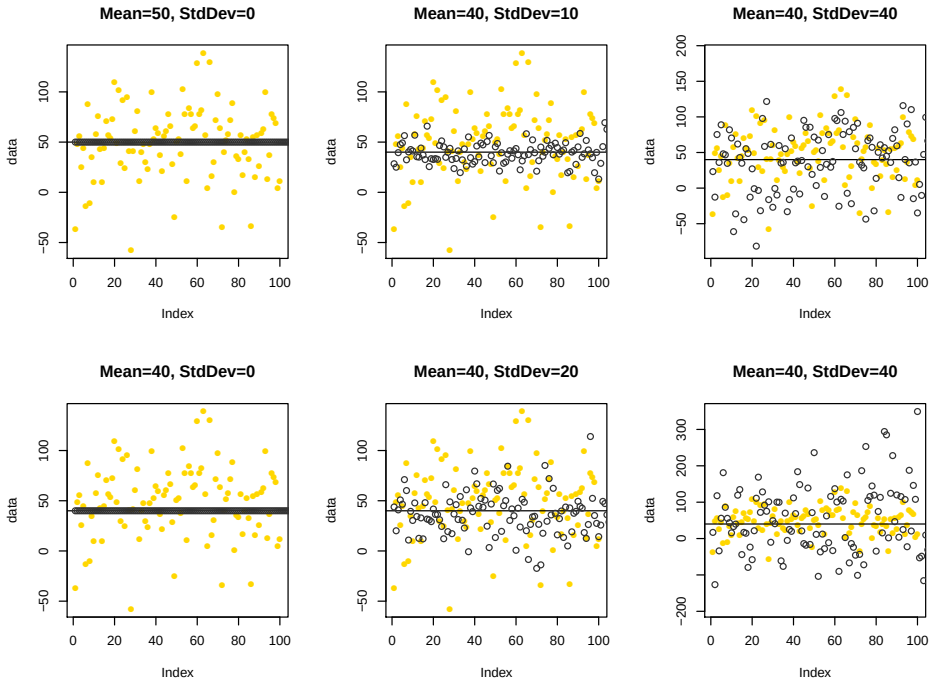
```
plotmean <- function(data,main="",middle,spread=0, samples=1000)
{
  samplex <- rnorm(samples,mean=middle,sd=spread)

  range <- range(c(samplex,data))
  plot(data, pch=16,col="gold",ylim=range,main=main)
  abline(middle,0)

  points(samplex,col="grey20",pch=1)
}

plotmean(x2,50,main="Mean=50, StdDev=0")
plotmean(x2,40,main="Mean=40, StdDev=0")
plotmean(x2,40,10, main="Mean=40, StdDev=10")
plotmean(x2,40,20, main="Mean=40, StdDev=20")
plotmean(x2,40,40,main = "Mean=40, StdDev=40")
```

Figure 7.9: Searching through the mean parameter space, this time guided by the mean square error. Even though the spread parameter is incorrect, we can get a better idea for the correct mean value.




```
plotmean(x2,40,80,main = "Mean=40, StdDev=40")
```

In its simplest form, when we give it our x values and parameters of the normal distribution– starting with one that has a given mean and no variability. But we think 50 is too high, so we can try `plotmean(x2,40)`, which we guess is more reasonable. Then, we can start adjusting the variability, from 0 to 10 to 40 to 80. 80 seems to high, but around 40 seems acceptable.

Exercise 7.5

Through trial-and-error, identify your best estimates for the two parameters that seem to best describe the data:

```
test1 <- 12 + runif(500)/3 + runif(500) + runif(500) + runif(500)*3
test2 <- rnorm(100) - 1/(1+runif(100))
test3 <- 5 + abs(rnorm)*3
```

Of course, this can and should be automated, provided we can clearly define what we mean by "looking reasonable". This is hard to establish. For example, it might be reasonable to try to equalize either the number or amount of error below and above the line. It might be useful to 'punish' large mistakes more than small mistakes, maybe even so that two small mistakes are considered better than one larger mistake that is the same size as the sum of the two small mistakes. It might be useful to identify a set of parameters that are the most likely based on these data (known as the maximum likelihood estimates). Or it might be useful to assume the noise comes from a particular distribution, so that looking reasonable means approximating the shape of the data distribution. There are different strategies people use, but they typically require:

- Establishing a cost function that specifically gives a score to each point, for any given set of parameters
- Finding a way to minimize that cost by picking a good set of parameters.

Perhaps the most common cost function is called 'squared error loss'. In a situation like this, it would identify your best estimate for a value $\hat{y}$ given x and your parameters, and then compute $(y - \hat{y})^2$: the squared difference. In this case, we would ignore the variability and consider only the deviation from the average. A good fit should minimize the total (sum) of the squared error across all points. We can compute this in a revised function, which no longer needs to sample anything; but we will anyway. Note that squared error loss is convex, meaning that the cost for a single large mistake is always greater than the sum of the cost for two smaller mistakes that add up to the large mistake. Now, let's just add this calculation to the mean plotter, and we can maybe search through the parameters better. We can start with a small s.d., and just try to hit the best mean accurately, which we think should be around 40.

```
plotmean2 <- function(data,middle,spread=0, samples=1000,main="")
{
  samplex <- rnorm(samples,mean=middle,sd=spread)
  range <- range(c(samplex,data)) #establish the join range of the data and
    simulation

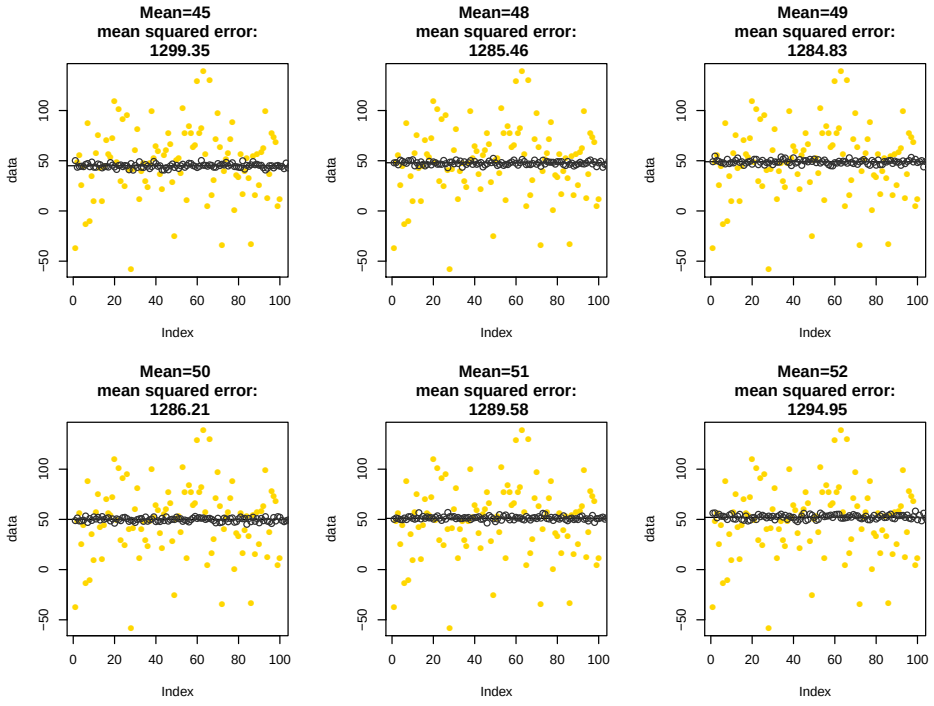
  plot(data,pch=16,col="gold",ylim=range,main="")
  abline(middle,0)

  points(samplex,col="grey20",pch=1)
  ##our best-guess model has mean=middle.  Compute square error:
  squarederror <-  mean((middle-data)^2)

  title(paste(main,"\nmean squared error:\n",round(squarederror,2)))
  return (squarederror)
 }

plotmean2(x2,45,2,main="Mean=45")
plotmean2(x2,48,2,main="Mean=48")
plotmean2(x2,49,2,main="Mean=49")
plotmean2(x2,50,2,main="Mean=50")
plotmean2(x2,51,2,main="Mean=51")
plotmean2(x2,52,2,main="Mean=52")
```

Figure 7.10: Searching through the mean parameter space, this time guided by the mean square error. Even though the spread parameter is incorrect, we can get a better idea for the correct mean value.



Notice by changing the parameter of the function, we reduce mean squared error, meaning the distribution is becoming a better model of the data. And now, guided by the statistic, we find that the true mean of 50 really was closer, and that our best estimate of the mean should be 49.

Exercise 7.5

Using the new `plotmean` function, find the best mean parameter you can, in terms of squared error.

```
test1 <- 12 + runif(500)/3 + runif(500) + runif(500) + runif(500)*3
test2 <- rnorm(100) - 1/(1+runif(100))
test3 <- 5 + abs(rnorm)*3
```

On their own, the error in this plot is compared to the estimate of the mean, and so the MSE statistic does not help us estimate the variability directly–we can still only eyeball it.

How could we estimate the variability better? We could look at the overall distribution of the errors in our model and the data, and compare them. The following function does this, by comparing cumulative distributions of the two, which you can then eyeball, or we could arrive at another cost function to help fit this parameter. Notice that in Figure 7.11, the red estimated distribution function goes from steeper than the data to shallower than the data around sd=35, which might be a good estimate of our standard deviation parameter.

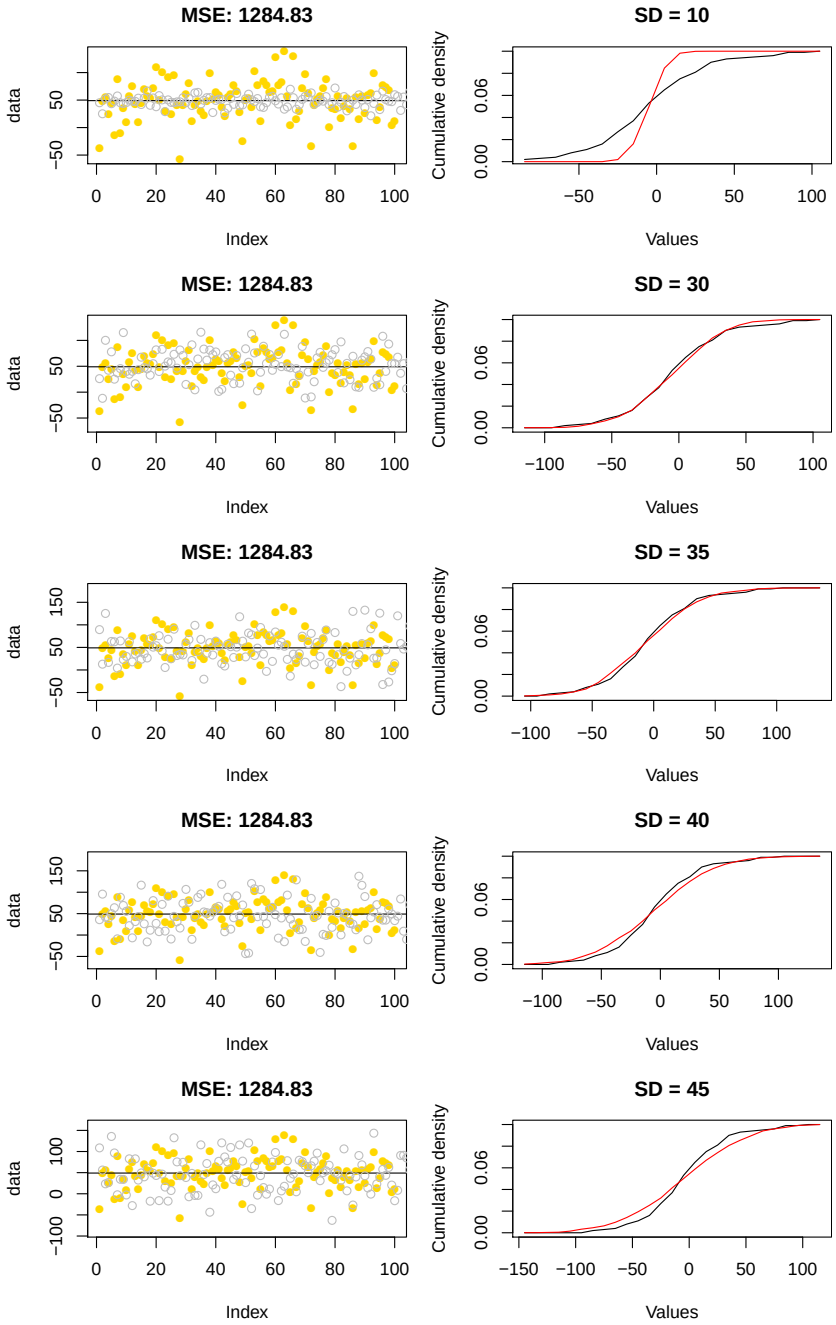
```

plotmean3 <- function(data,middle,spread=0, samples=1000,main="")
{
  samplex <- rnorm(samples,mean=middle,sd=spread)
  range <- range(c(samplex,data)) #establish the join range of the data and
    simulation
  par(mfrow=c(1,2),mar=c(4,5,3,0))
  plot(data,pch=16,col="gold",ylim=range,main="")
  abline(middle,0)

  points(samplex,col="grey",pch=1)
  ##our best-guess model has mean=middle.  Compute mean square error:
  error <-  (middle-data)
   title(paste("MSE:",round(mean(error^2),2)))
    modelerror <- samplex - mean(samplex)
  bothbreaks <- hist(c(error,modelerror),breaks=25,plot=F)$breaks
  x1 <- hist(error,plot=F,breaks=bothbreaks)
  x2 <- hist(modelerror,plot=F,breaks=bothbreaks)
  plot(x1$mids,cumsum(x1$density),type="l",main=main,xlab="Values",ylab="
    Cumulative density")
  points(x2$mids,cumsum(x2$density),type="l",col="red")

  return (mean(error^2))
}
 plotmean3(x2,49,10,main="SD = 10")
 plotmean3(x2,49,30,main="SD = 30")
 plotmean3(x2,49,35,main="SD = 35")
 plotmean3(x2,49,40,main="SD = 40")
 plotmean3(x2,49,45,main="SD = 45")
```

Figure 7.11: Searching through the mean parameter space, guided by a depiction of the cumulative distribution.



Exercise 7.5

Using the new `plotmean3` function, find the best parameters you can, including for the variability.

```
test1 <- 12 + runif(500)/3 + runif(500) + runif(500) + runif(500)*3
test2 <- rnorm(100) - 1/(1+runif(100))
test3 <- 5 + abs(rnorm)*3
```

It should be noted that a different kind of cost function is sometimes used to estimate these parameters: the likelihood, which is then maximized (technically, the negative log-likelihood is typically minimized). Likelihood is a number obtained by assessing the value of the density function of the random variable at the point of the data observed. If you have a cloud of data you wish to model with a two-dimensional normal distribution. Here, the best likelihood answer (maximum likelihood) is found by placing the center of the distribution at the center of the cloud, and adjusting the standard deviations so it best matches your data. If the standard deviation parameter is too small, too many outer points will have too low of a likelihood. If you set the standard deviation parameter too big, too many inner points will have a value too small. The maximum-likelihood estimate is sometimes biased, meaning you are likely to underestimate the variance of the actual process that generated the data, but it has some useful properties.

7.5.1 Summary of ad hoc parameter estimation

Typically, when performing a statistical analysis, we would not resort to these crude ways of estimating parameters. Instead, we will often use other ways that can be proven to give the best answer in one way or another. But at its heart, every test you perform requires parameter estimation in some fashion, and you should recognize that it is always possible to 'search' for the parameters rather than computing them directly, as we will do in the next section.

7.6 Parameter estimation with statistics

Earlier sections discussed the goal of estimating parameters for the random variables we try to model the world with, and showed ad hoc methods for searching for these values, which we sometimes have to resort to. But there are sometimes easier ways. If we are lucky, we can estimate parameters using the right statistic, rather than search.

7.6.1 Statistics

The word STATISTIC has a very simple definition which is familiar to sports fans, but not as much to researchers who conduct statistical analysis: a statistic is a function of data.

For example, the function *MEAN* adds up all the elements and divide by N, and is a function of the data.

Other statistics include:

- Median–the middle point
- Mode –the most common value observed

- Geometric Mean (Nth root of the product; the length of the side of the square whose area is the same as the rectangle formed by the values but in high dimension).
- min–the smallest point
- max –the largest observation
- variance –subtract the mean from each value, square, and compute the average.

Basically, if you think of your data as a vector or a table in R, a statistic is any number that results from a function that takes those data as an argument.

Not only might we model our data as a random variable, we might also model the results of a statistic as a random variable. Another way to think about it is that just as our process is a random variable, a function of that process is also a random variable. Consequently, just as our data might have a distribution, so will our statistics. But, the distribution of the statistic will typically be different than the distribution of the data. The importance of this will be used later, and some special facts about the relationship between a distribution and the distribution of a statistic are central to our ability to make inferences using a small number of tests.

Exercise 7.6.1

In the example above, in terms of squared-error cost function, the best estimate for the middle parameter is the mean of the data, and the best estimate for the spread is the sample standard deviation of the data. Use plot3 and give it mean and sample sd (computed as sd(data), where n is the number of observations).

```
test1 <- 12 + runif(500)/3 + runif(500) + runif(500) + runif(500)*3
test2 <- rnorm(100) - 1/(1+runif(100))
test3 <- 5 + abs(rnorm)*3
```

7.6.2 Using statistics for parameter estimation

As we discussed earlier, we have the problem of wanting to estimate parameters from a random variable based on data. Instead of ad hoc estimating or tedious search, there are sometimes functions we could compute on our data (i.e., statistics) that could estimate the parameters we care about.

This is one of the core aspects of modern statistical testing, and is one that is usually hidden from users by software packages. In many situations, we can compute statistics on data we have that will directly estimate the parameters we care about. If we are lucky, we can identify a function that will estimate the parameters of the distribution we are interested in modeling. Being "lucky" means several things: our data really were generated from the distribution we think they were; we have enough data so that our estimates are not incorrect or biased;, and finally, we have assumed the data came from a distribution whose parameter we know how to estimate with a statistic. This last part was historically usually fairly easy, because statisticians have traditionally focused on distributions for which this is true. In

recent years, however, this tends to not be the case, and we use more complex methods for estimation.

People often confuse the statistic for the parameter it is estimating, because they can be closely linked. Statistics for estimating simple distributions are often fairly straightforward and seem intuitive, but involve non-trivial mathematics to derive and prove their convergence properties. And usually, we may not even know we are doing it, because the statistical tools can hide the estimation process from us.

7.6.3 Example: The Binomial distribution.

One of the most common and simple random variables we encounter is the binomial RV, which can be used to model many yes/no processes, including indirectly whether an act succeeds (accuracy). Suppose we create a data set from a binomial distribution with two parameters with the following function:

```
makebinomial <- function(n=100,p=.5)
{
    sum(runif(n) < p)
}
```

Look at the series of results:

```
makebinomial(100,p=.9)
makebinomial(1000,p=.5)

> makebinomial(100,p=.9)
[1] 87
> makebinomial(1000,p=.5)
[1] 499
```

we could run this many times to visualize the RVs distribution, using this trick, which is visualized in Figure **??**

```
samples <- sapply(1:1000,function(x){makebinomial(n=25,p=.8)})
hist(samples)

slightly more procedural:
samples <- rep(0,1000)
for(i in 1:1000)
{
  samples[i] <- makebinomial(25,.8)
}
hist(samples)
```

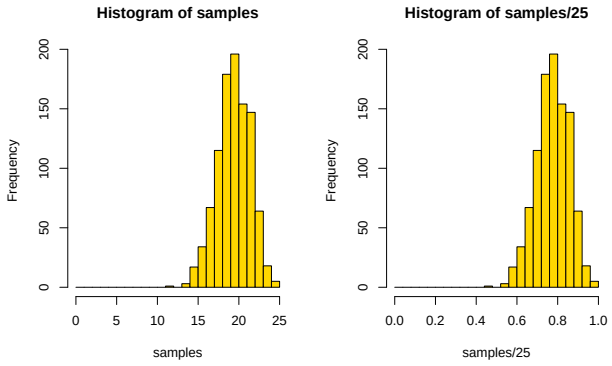
Question: how can we estimate the value p from a set of data? It seems like intuitive that if we find the number of successes, and divide by the number of samples, the answer would be a good estimate. Here, since we had 25 trials, we divide by 25. The results are shown in the right panel of Figure 7.12.

The set of estimates we obtain:

```
hist(samples/25)

mean(samples/25)  ##This should be pretty close to 0.8
```

Figure 7.12: Searching through the mean parameter space, guided by a depiction of the cumulative distribution.



How far off are we likely to be? In my simulation, the mean value was .7952, which seems pretty close, but we can see that on any individual experiment, our estimates generally ranged between .7 and .9.

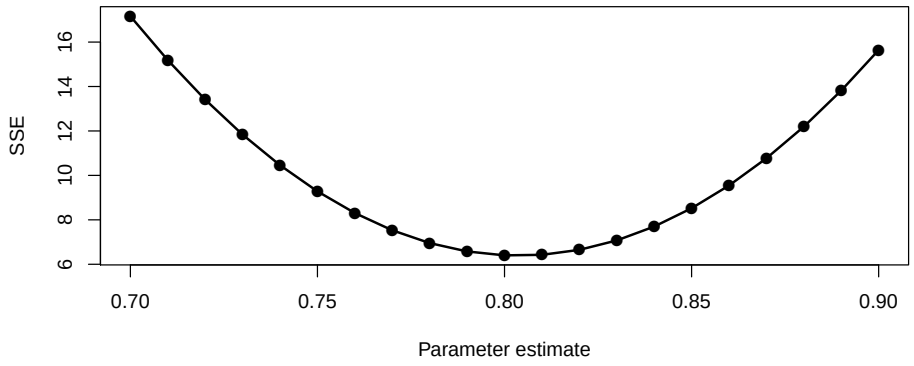
Exercise 7.6.3

- Suppose you were flipping a fair coin 100 times. What is the chance that it comes up heads fewer than 40 times? Although this can be computed exactly, estimate this through 'Monte Carlo' simulation, by simulating the experiment and running it hundreds or thousands of times.
- Suppose you run a gambling operation, and get suspicious that a coin is biased if it landed heads or tails more than 55 times out of 100 flips. If the coin is really biased so that it lands heads on average 60% of the time, what is the probability that on a given 100-flip sequence, you would think it is fair? Biased heads? Biased tails? Simulate at least 1000 of these 100-flip experiments to determine your answer.

Now, suppose we compute sum squared error for any particular estimate. We give it data and N (the parameter of the binomial), and a particular estimate, and it will give us a goodness-of-fit statistic.

```
sse <- function(data,N,estimate)
{
  sum((data/N-estimate)^2)
}
set.seed(100)
dat <- sapply(1:1000,function(x){makebinomial(n=25,p=.8)})
sse(dat,25,.5)
```

Figure 7.13: SSE values for parameter estimates around 0.8.




```
sse(dat,25,.8)

> sse(dat,25,.5)
[1] 98.6816
> sse(dat,25,.8)
[1] 6.4016
```

Now, we can again use trial-and-error to find a value that produces the smallest SSE.

```
> estimate
[1] 0.8038
> sse(dat,25,estimate)
[1] 6.38716

```

This estimate has a SSE of 6.38. Note that we can't get an SSE of 0, because the data are all 1s and 0s, and so each observed value will differ from the mean by .2 or .8. Now, if we adjust our estimate a little bit in either direction:

```

> sse(dat,25,estimate+.01)
[1] 6.48716
> sse(dat,25,estimate-.01)
[1] 6.48716
```

It might not surprise you that the average of the proportions appears to be the best estimate. We can see that with this plot:

```
plot(70:90/100, sapply(70:90/100,  function(x){sse(dat,25,x)}),type="o",
     ylab="SSE",xlab="Parameter estimate")
```

This shows a few things. First, a statistic (and a simple one) might be our best estimate of a parameter. In this case, the mean of the data (a statistic) is the best estimate of the parameter of the binomial distribution describing the data. Second, our estimate itself is not likely to be the same every time–it has its own distribution. This means that if you ran the same experiment again, you'd expect a different answer. Although we will not go through the proofs of theorems in this class, we will simulate these properties so that you can have a more intuitive understanding of how statistical tests work.

What happens when we compute the mean statistic on random variables? We will see in the next section.

7.7 The Normal Distribution

For many of the statistical tests we perform, we are concerned with the normal distribution. Why? For two reasons–it turns out that if you have a bunch of processes and mix them together, they typically begin to approximate the normal distribution. And second, because the underlying distributions we care about (a random variable representing our estimate of different parameters) can often be proven to approximate the normal, via something called the *Central Limit Theorem*. To see this, suppose simply that the time needed to complete a task is made up of a bunch of sub-tasks, each of which has a distribution that is not normally distributed. This could go all the way down to sequences of neural processes.

In this case, suppose each of 50 processes has a uniform distribution with a mean between 0 and 1:

```
means <- runif(50)
```

we can simulate the time for the whole process with this, 1000 times–it is like we have run 1000 experiments of 50 samples, computing the mean each time:

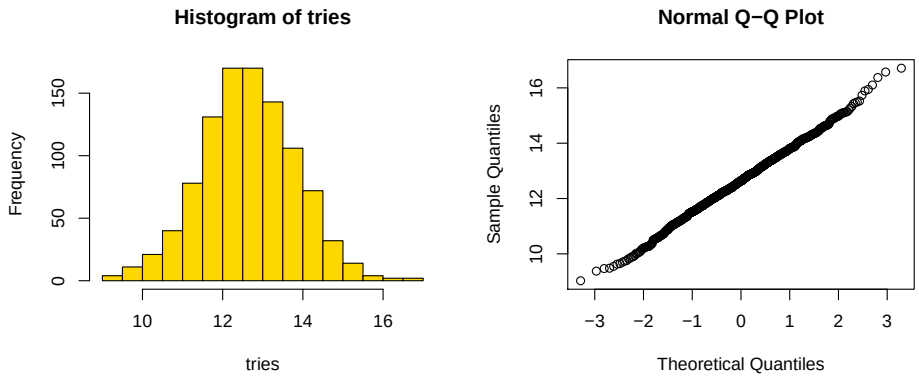
```
means <- runif(50)
tries <-  sapply(1:1000,function(x){sum(runif(50)*means)})
par(mfrow=c(1,2))
hist(tries,col="gold")
qqnorm(tries)
```

Notice that although the individual random variables were uniform, the composite, which is the sum of other random variables, approximates a normal distribution. This is called the *Central Limit Theorem*, and can be proved mathematically. It is the fundamental theorem that leads us to use the normal distribution, and to not worry too much if our original data are generated by a distribution that is non-normal.

7.7.1 More on Comparing Distributions

It can be handy to compare two distributions against one another, or to compare a distribution against a known distribution to determine whether it violates the assumptions. The quantile-quantile plot has been developed to do this. This simply plots the values of quantiles against one another. As we have used already, `qqnorm()` will do this against a normal distribution and qqplot() will compare two distributions. Really, this is the same as plotting the sorted data against one another, assuming you have the same number of points in each data set.

Figure 7.14: The sum of 50 random uniform distributions with different mean values is indistinguishible from a normal distribution




```
dat1 <- rnorm(1000)
dat2 <- 100 + rnorm(1000)*5
```

Because these have the same number of observations, we can plot them against one another simply by sorting each one

```
plot(sort(dat1),sort(dat2))
```

but what if we apply a transformation:

```
dat3 <- log(dat2-min(dat2)+.001)
plot(sort(dat1),sort(dat3),type="o")
abline(0,1)
qqplot(dat1,dat3)
```

This is no longer linear, indicating that the distributions don't match, either by plotting the sorted values against one another or via the `qqplot` function. In general, `qqplot` will be better because it will work even if the two data sets are of different lengths. We can use the qqnorm function to see how quickly the sum-of-uniform random variables converges to a normal, which will be explored in the next exercise.

Run the following code and examine the qq-plots of each distribution. Which are approximated by normal distributions? Which are not?

```
tries <- log(dat2-min(dat2)+.1)
qqnorm(tries)

##Here is a normal distribution
means <- runif(50)
tries <-  sapply(1:1000,function(x){sum(rnorm(50)*means)})
qqnorm(tries)

par(mfrow=c(1,2))
##what about a log-normal distribution.  Look at the base distribution:
```

```
tries <-  sapply(1:1000,function(x){sum(exp(rnorm(1)*means))})
hist(tries,breaks=20)
qqnorm(tries)
##It is highly skewed.  But what if take the sum of 10?

tries <-  sapply(1:1000,function(x){sum(exp(rnorm(10)*means))})
hist(tries)
qqnorm(tries)

##sum of 100?
tries2 <-  sapply(1:1000,function(x){sum(exp(rnorm(100)*means))})
hist(tries2)
qqnorm(tries2)

##sum of 1000?
tries3 <-  sapply(1:1000,function(x){sum(exp(rnorm(1000)*means))})
hist(tries3)
qqnorm(tries3)

par(mfrow=c(1,3))
qqnorm(tries)
qqnorm(tries2)
qqnorm(tries3)
```

The more we add together, the more the result of the function matches the normal distribution. What this means is that many complex processes that stem from a bunch of different sub-processes will tend to be generally normal, and more importantly, if we collect a bunch of observations of the same process, and add them up (possibly dividing by N), it will also be normal, regardless of its source distribution. But adding up the values and dividing by N is just the mean statistic. This means that even if the data distribution itself is not normal, the mean statistic will tend to have a distribution across experiments that IS normally distributed, even though we cannot see it because we only conduct one experiment at a time. (Note, this assumption can be violated, but typically statisticans are less worried about violating the normality assumption as they are about violating other assumptions like homogeneity of variance. Many processes (time, accuracy, precision, ratings scales, etc.) are logically not normally distributed, but they still will produce means that are.

Exercise 7.7.1

Write a function that creates a random variable that is a sum of three random processes; two uniform RVs and a normal RV (you choose how these are mixed). Run a simulation where you sample 100 of these and estimate the mean. Use a q-q plot to examine the distribution of these means, for at least 1000 simulated experiments.

7.8 Biases in Parameter Estimation

The basic question we asked in the previous exercise is at the heart of statistical inference. We often want to know whether, based on observing some data, the group, process, people,

or context we happened to sample is likely to differ more broadly from some other group, process, people, or context. For the coin, we KNOW whether a particular sample is biased (and it is whenever it differs from 50:50), but we might want to infer whether the coin is biased (so we can get rid of it, bet on it, or fix it).

Flipping a coin is a lot like any human process that either succeeds or fails. This includes whether a student succeeds in learning/recalling information, whether the correct response is chosen, whether a job is completed on time, whether a advertising lead is converted to a sale, etc. We may have two products, interfaces, students, tools, contractors, etc., and observe that on some criterion, one succeeds on 90 percent of the questions, whereas another succeeds on 85%. We KNOW that on a particular sample, one did better than the other, but want to know whether that is likely to repeat itself. This is the core of inference–inferring a more broad pattern about the population based on a smaller sample.

Psychology research typically wants to generalize to a broader population. This is why sampling can be important, because if you sample only from university students, your statistical generalization is to the population of university students, which may be fairly limited. On the other hand, many types of studies may really want to generalize to a person based on a sample of their behavior. For example, if you are designing and testing a new interface for a special-purpose software tool used by a handful of people, you may not be as interested in sampling a general population and more interested in convincing yourself that the few people who use it are indeed better. You may not need to test a lot of people, but you should be confident that the test would be repeatable within a single person.

The problem is that typically, we have just a single observation–the experiment. From this observation, we make our best guess at the random variable that generated the data, and ask questions about them. It might seem intuitive that to estimate a parameter associated with the mean of a distribution, we need to find the mean of the outcome variables. This is typically true, but it is not always the case that there is such a simple relationship between the parameter and the statistic used to estimate that parameter. Often, the parameter is given as a greek symbol, and our estimate is written with a 'hat', indicating it is an estimate. The mean parameter is often written μ, whereas the estimate of the mean is $\hat{\mu}$.

When we are estimating other parameters of a distribution, our intuitive methods of estimating can be *biased*. Usually, bias manifests as an under-estimation or over-estimation for small sample sizes.

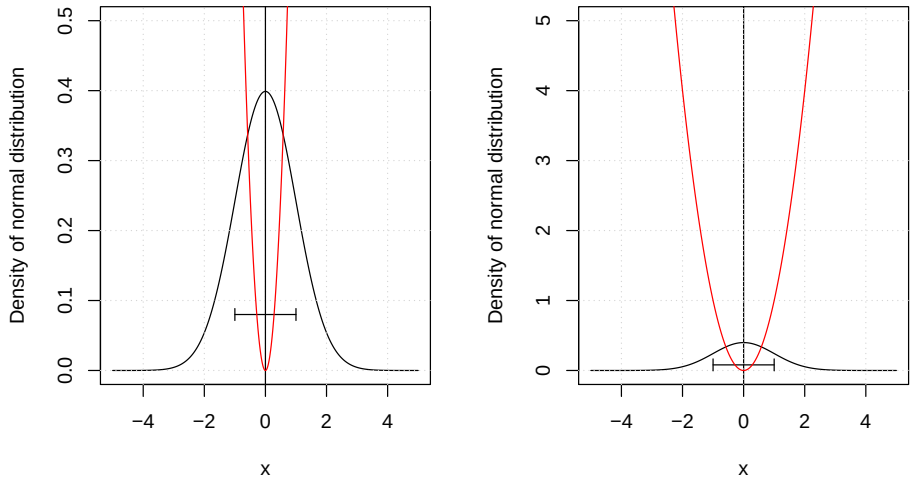
Here are 1000 normal random numbers from a distribution with the σ (sigma or variability) parameter = 10.

```
y <- rnorm(1000)*10
hist(y)
```

Suppose we want to estimate the standard deviation parameter of of the original, by computing the standard deviation statistic. For a distribution function, variance (sd squared) is a weighted function averaged over all points of the function. This computes the exact expected density at all points:

```
x<- -500:500/100
y <- dnorm(x)
plot(x,y,type="l",ylab="Density of normal distribution",ylim=c(0,.5))
abline(v=0)
arrows(-1,.08,1,.08,angle=90,code=3,length=.05)
```

Figure 7.15: Normal density and squared error deviation around mean of distribution.



For this exact distribution, we can compute its standard deviation as the mean of the squared deviations. The red line indicates how much penalty an error accrues as it deviates from the mean. So, for this hypothetical distribution, you would penalize each data point based on the value of the red line. The mean height of the red line over all the data points is the standard deviation.

```
plot(x,y,type="l",ylab="Density of normal distribution",ylim=c(0,5))
abline(v=0)
arrows(-1,.08,1,.08,angle=90,code=3,length=.05)
x<- -500:500/100
y <- dnorm(x)
squaredev <- (x-mean (x))^2
points(x,squaredev,type="l",col="red")
```

We can compute this as a weighted mean using `weighted.mean`. This is a numerical estimate, looking at the function every .1 steps on the x axis In the figure, the red value (squared deviance) is being weighted by the black value (density).

It is essentially asking how big, on average, is the red line, weighed by the values we are most likely to see.

```
weighted.mean(squaredev,y)
```

Notice that it is about 1.0, which is what we'd expect. A weighted mean does basically what we would do by averaging over observations sampled from that distribution. Those values that are sampled more often are represented more often in our data, which get averaged together. So, let's try to compute the average of the squared deviations, using the 'mse' function below. MSE computes the variance of the data, so we will need to take the square root to estimate the standard deviation.

```
  mse <- function(x){mean((x-mean(x))^2)}
  x <- rnorm(20); sqrt(mse(x))
```

This is troubling. the data are generated from a distribution whose standard deviation is 1.0, but appears to be consistently below 1.0. Let's check this for a bunch of examples:

```
hist(sapply(1:1000,function(x){mse(rnorm(20))}),xlab="")
mean(sapply(1:1000,function(x){mse(rnorm(20))}))
```

Here, the mean estimate ended up being .959. If we increase `N` from 20 to 200 , it seems to get better, to .989. If we increase it to 2000, it is almost exactly correct–.999.

```
hist(sapply(1:1000,function(x){mse(rnorm(200))}),xlab="")
mean(sapply(1:1000,function(x){mse(rnorm(200))}))

hist(sapply(1:1000,function(x){mse(rnorm(2000))}),
         xlab="",breaks=100)
mean(sapply(1:1000,function(x){mse(rnorm(2000))}))
```

So, for small experiments, using mse to estimate the standard deviation will be biased, even though mse is identical to the formula for computing standard deviation from data. Standard deviation–computed as *mse*–is a STATISTIC (a function of data), and if we use it to estimate the corresponding parameter of the distribution (often referred to as the greek symbol σ) it is systematically biased, and it seems like it is worse for small samples. Let's test this more systematically:

```
samplesizes <- c(2,3,4,5,6,7,8,9,10,20,50,100,200)
ests <- c()
reps <- 10000

for(samples in samplesizes)
{
   dat <- matrix(rnorm(samples*reps),reps,samples)

   vars <- apply(dat,1,mse)
   print(paste(samples,mean(vars)))
   ests <- c(ests,mean(vars))
}
```

This is not very good until we get to experiments of around 100!!!

```
plot(samplesizes,ests,type="o",main="Underestimation of variance",
       xlab="Number of observations",ylab="Estimated Variance",
       ylim=c(.5,1.1))
abline(1,0,lwd=3,col="grey")
```

What is going on here? Applying the same function to the actual distribution and to data sampled from the distribution produce different numbers, when the sample size is small. This is because our statistic is a biased estimate of the parameter. Is there an estimate that is unbiased? By looking at the under-estimations across sample sizes, we can detect a pattern.

The bias gets smaller as N gets bigger. Through calculus, one can prove that the bias is n/(n+1) times the true variance, and so if we multiply/adjust the estimate by this factor, we can produce an unbiased estimate.

```
adj <- samplesizes/(samplesizes-1)
plot(samplesizes,ests*adj,main="Adjusted estimates",
     ylab="Adjusted value n/(n-1)")

##Incorporate this directly into our MSE
mse.sample <-  function(x){sum((x-mean(x))^2)/(length(x)-1)}

samplesizes <- c(2,3,4,5,6,7,8,9,10,20,50,100,200)
ests <- c()
ests2 <- c()
reps <- 10000
for(samples in samplesizes)
{
   dat <- matrix(rnorm(samples*reps),reps,samples)

   vars <- apply(dat,1,mse)
   vars2 <- apply(dat,1,mse.sample)
   print(paste(samples,mean(vars),mean(vars2)))
   ests <- c(ests,mean(vars))
   ests2<- c(ests2,mean(vars2))
}
```

Here is the 'original' estimate, compared to the unbiased estimate:

```
plot(samplesizes,ests,type="o",
     main="Underestimation of variance",
     xlab="Number of observations",
     ylab="Estimated Variance",
     ylim=c(.5,1.1))
abline(1,0,lwd=3,col="grey")
##The 'adjusted' estimate:
points(samplesizes,ests2,type="o",col="gold",pch=16,lwd=2)
points(samplesizes,ests2,type="p",col="black",pch=1)
```

This bias is why we sometimes use what is called the 'sample variance' versus 'population variance' to estimate standard deviation. If you are trying to estimate a parameter of a population, you should always use the sample variance formula, because it will be, on average, an unbiased estimate. As your number of samples get larger, the bias gets smaller and so the difference between the two formulas goes away, but you should still use the sample variance estimates.

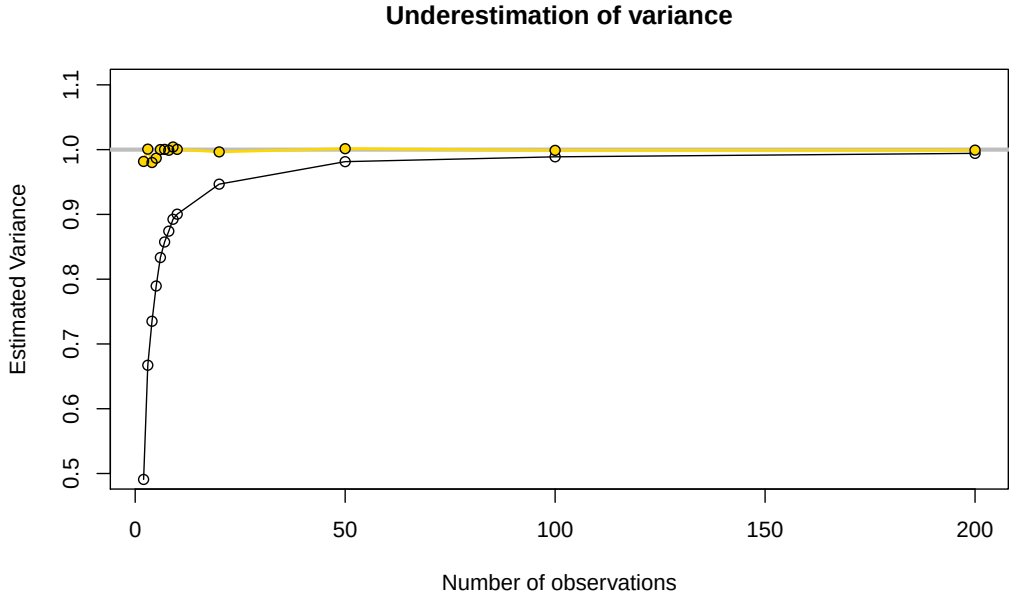
Outcome:

To estimate a parameter of a distribution, we need to compute a statistic on the data. But we cannot expect the parameter to be unbiased. This is especially true for variance estimates, which is why we use the 'sample' variance and sample standard deviation

7.9 Summary

In this chapter, you learned about random variables and their parameters. We looked at the challenge of estimating those parameters, and learned that some parameters can be estimated using statistics of data. However, those estimates can sometimes be biased.

Figure 7.16: The estimate of standard deviation is biased (black line) and depends on the number of observations. We can use a debiased sample standard deviation formala (gold line) instead to account for this biasing.



7.10 Solutions to Exercises

Exercise 7.2.5 Solution

Generate and view 1000 samples from:

(a) A normal distribution having mean 100 and s.d. .1

(b) A normal distribution having mean 100 and s.d. 20

(c) A uniform distribution 100 and 101.

(d) A discrete distribution among 20 elements, where element n has a probability proportional to 1/n.

(e) A process where you roll a 10-sided die, and based on the result, you sample a random normal distribution with *both* mean equal to 25 times the result, and standard deviation equal to the result of the die-roll.

```
hist(rnorm(1000,mean=100,sd=.1))
hist(rnorm(1000,mean=100,sd=20))
hist(runif(1000,min=100,max=101))
barplot(table(sample(1:20,1000,replace=T,prob=1/1:20)))

vals <- (sapply( ceiling(runif(10000)*10),function(x){rnorm(1,x*25,x)}))
hist(vals,breaks=100)
```

Exercise 7.3 Solution

Using the data set c7rts.csv, form a histogram and normal q-q plot for the response time (rt) data. Is it well described by the normal distribution? Then, transform the data with a logarithm and plot the histogram and qq-plot. How about now? In what ways might the original and transformed data mismatch the normal distribution. Why? The following code will examine these qqplots:

```
data <- read.csv("c7rts.csv")
par(mfrow=c(1,2))
hist(data$rt,breaks=50)
qqnorm(data$rt)
hist(log(data$rt))
qqnorm(log(data$rt))

data2 <- data$rt[data$rt<5000]
hist(data2)
hist(log(data2))
qqnorm(log(data2))
```

Looking at the results, it is highly skewed, and not well-approximated by a normal distribution. Transforming with a log transform is a small improvement, but still not perfect.

Exercise 7.5 Solution

Through trial-and-error, identify your best estimates for the two parameters that seem to best describe the data:

```
test1 <- 12 + runif(500)/3 + runif(500) + runif(500) + runif(500)*3
test2 <- rnorm(100) - 1/(1+runif(100))
test3 <- 5 + abs(rnorm)*3
```

Exercise 7.5a Solution

Using the new `plotmean` function, find the best mean parameter you can, in terms of squared error.

```
test1 <- 12 + runif(500)/3 + runif(500) + runif(500) + runif(500)*3
test2 <- rnorm(100) - 1/(1+runif(100))
test3 <- 5 + abs(rnorm)*3
```

Exercise 7.5b Solution

Using the new `plotmean3` function, find the best parameters you can, including for the variability.

```
test1 <- 12 + runif(500)/3 + runif(500) + runif(500) + runif(500)*3
test2 <- rnorm(100) - 1/(1+runif(100))
test3 <- 5 + abs(rnorm)*3
```

The following seem reasonable:

```
plotmean(test1,"Test1",14.5,1)
plotmean(test2,"Test2",-1,1.1)
plotmean(test3,"Test3",8,1.7)
```

Exercise 7.6.1 Solution

In the example above, in terms of squared-error cost function, the best estimate for the middle parameter is the mean of the data, and the best estimate for the spread is the sample standard deviation of the data. Use plotmean3 and give it mean and sample sd (computed as sd(data), where n is the number of observations).

```
test1 <- 12 + runif(500)/3 + runif(500) + runif(500) + runif(500)*3
test2 <- rnorm(100) - 1/(1+runif(100))
test3 <- 5 + abs(rnorm)*3
```

Let's see what the same values look for plot3

```
plotmean3(test1,"Test1",14.5,1)
plotmean3(test2,"Test2",-1,1.1)
plotmean3(test3,"Test3",8,1.7)

plotmean3(test1,"Test1",mean(test1), sd(test1))
plotmean3(test2,"Test2",mean(test2), sd(test2))
plotmean3(test3,"Test3",mean(test3), sd(test3))
```

Exercise 7.6.3 Solution

- Suppose you were flipping a fair coin 100 times. What is the chance that it comes up heads fewer than 40 times? Although this can be computed exactly, estimate this through 'Monte Carlo' simulation, by simulating the experiment and running it hundreds or thousands of times.
- Suppose you run a gambling operation, and get suspicious that a coin is biased if it landed heads or tails more than 55 times out of 100 flips. If the coin is really biased so that it lands heads on average 60% of the time, what is the probability that on a given 100-flip sequence, you would think it is fair? Biased heads? Biased tails? Simulate at least 1000 of these 100-flip experiments to determine your answer.

```
samples <- sapply(1:1000,function(x){makebinomial(n=100,p=.6)})
mean(samples<45)
mean(samples<=55)
```

Exercise 7.7.1 Solution

Write a function that creates a random variable that is a sum of three random processes; two uniform RVs and a normal RV (you choose how these are mixed). Run a simulation where you sample 100 of these and estimate the mean. Use a q-q plot to examine the distribution of these means, for at least 1000 simulated experiments.

```
myrandom <- function(x){runif(1) + runif(1)*3 + rnorm(1)}

means <- rep(0,1000)
for(i in 1:1000)
means[i] <- mean(sapply(1:100,myrandom))

data <- matrix(sapply(1:100000,myrandom),1000,100)
qqnorm(rowMeans(data))

hist(means)
qqnorm(means)
```

Exercise 7.7.1 Solution

Write a function that creates a random variable that is a sum of three random processes; two uniform RVs and a normal RV (you choose how these are mixed). Run a simulation where you sample 100 of these and estimate the mean. Use a q-q plot to examine the distribution of these means, for at least 1000 simulated experiments.

```
myrandom <- function(x){runif(1) + runif(1)*3 + rnorm(1)}

means <- rep(0,1000)
for(i in 1:1000)
 means[i] <- mean(sapply(1:100,myrandom))

data <- matrix(sapply(1:100000,myrandom),1000,100)
qqnorm(rowMeans(data))

hist(means)
qqnorm(means)
```

Chapter 8

Inferential Statistical Tests

Libraries used in this chapter: `BayesFactor`, `vioplot`, `effectsize`, `BSDA`, `ggplot`, `GGally`

In the previous chapter, we discussed how we model the world with random variables, and estimate aspects of those models based on parameter search and estimating statistics. If we have a model of the process, with parameters estimated with samples from the process, we can then begin asking questions about whether what we observe would have been likely to have happened just by chance during our sampling.

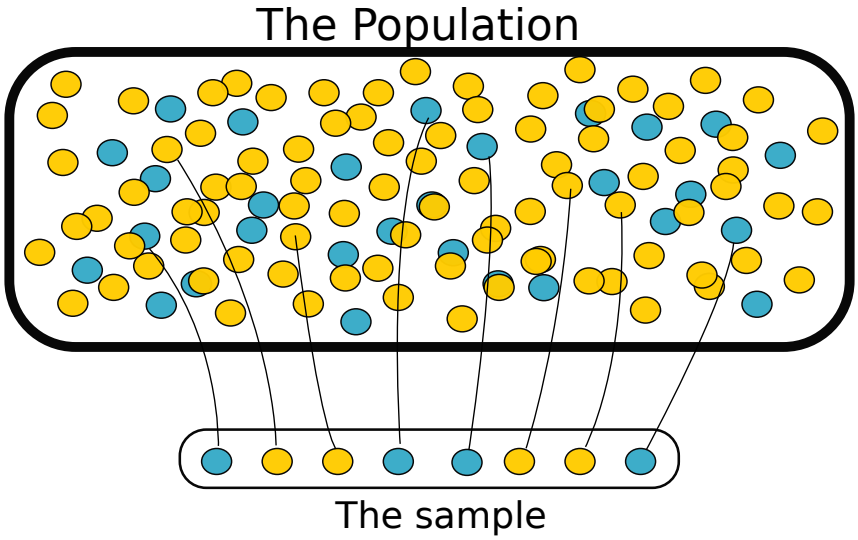
The basic conceptual model of statistical inference is illustrated in Figure 8.1. Here, we hypothesize that there is a set of all possible events, outcomes, people, or processes that we measure. We'd like to know something about that population, or maybe subgroups of the population. For example, maybe we want to know, in the population of all users of Facebook, whether they are likely to respond to an advertising campaign and visit a website or make a purchase (sometimes called "converting"). Or, in the population of all rides in a self-driving car, whether the rate of fatal accidents is more than 1 per 100,000 miles driven. Or, in the population of likely voters, whether they support a republican or a democrat. In each of these cases, it is too costly, logistically difficult, or impossible to measure every case. So, we make a representative sample of the cases, people, or situations we are interested in and see what the sample looks like.

In the figure, the balls represent all the cases we care about–maybe the entire population of a city who will vote in an election. If we want to find out how the election is likely to turn out, we sample from this population. Maybe backers of a tax bill know they need 60% support for the bill to pass, and want to know how many people support this (the yellow circles) in the voters (all of the circles). Because they cannot hold a complete election prior to the election, and even if they went door-to-door and asked each person they would not be able to get answers from everyone, they must sample from the population and see what the result is. So, in the population, about 3/4 of the population supports the bill, but in the sample, 1/2 support it. This can happen just by chance–even if sampling is done correctly and in an unbiased manner. So, if you were a supporter of the bill, would you conclude that you have lost? If you were an opponent, would you conclude that the bill will lose and you do not need to invest more time or money in campaigning? This is the question of statistical inference, and is the basis for all statistical hypothesis testing.

It is easy to forget that your statistical test is always about generalizing to a population–it is not about whether there is a difference in the data you observed, but rather about whether the difference you observed is likely to have been true of the population you sampled from, rather than coming just from chance.

Scientists will often forget this in an experiment, and think that the statistical test

Figure 8.1: Depiction of a sampling process. All statistical tests assume we have a population we are drawing from and a sample we are observing, and try to draw conclusion about the population based on the sample



is about whether the results of a particular study are valid, or "statistically significant". This is not really correct–it is always about generalizing to a population. In more applied settings, analysts may actually be looking at the entire population they care about, and so an inferential statistical test does not make sense there either. For example, if you have a customer database of 1,000,000 records, and want to know whether customers over 40 made more purchases per month that customers under 40, you are almost guaranteed to find a difference in the mean purchase amount. In this case, one might ask, "but is the difference statistically significant", but this is a mis-understanding of how statistics work–because the population you are trying to generalize to is the entire sample. Thus, you are not really asking an inferential question, and inferential statistics will not help directly. You may want to know whether the pattern you saw will be true in the future, but again this is not usually the same question that inferential tests will be able to answer. In all cases, when attempting to do an inferential test, you must first ask, "what is the population I'm sampling from?", because that is the only thing the test is designed to tell us.

8.1 Hypothesis Testing with Statistical Tests

Statistical tests typically work by making assumptions about the random variables that model the processes, attempting to test whether the assumed model is appropriate, estimating parameters of those random variables based on a sampling process, and trying to make inferences about the population that is being sampled from based on the mathematics of the distribution. We reason about the distribution's properties, and try to understand whether the data we see come from different populations of people or processes. In this chapter, we will describe three approaches to doing this inference. The first is the traditional approach referred to as the Null-Hypothesis statistical test (NHST), which is well-understood but increasingly the target of criticism. The second is the non-parametric approach, where

we attempt to do the tests in a way that does not depend on the form of the random variable that has generated the data. Non-parametric statistics relax the need to make strong assumptions about the distribution, but still generally involve a NHST approach. Finally, a method that is increasingly used in research is a Bayesian hypothesis testing approach. There are several versions of the Bayesian approach to modeling data, but we will focus on the simplest version of this, which is Bayes Factor hypothesis testing.

8.1.1 Classic Null-hypothesis statistical tests

Most traditional methods involve parameter estimation about an assumed distribution., and using null-hypothesis statistical test (NHST) scheme to determine whether a sampled difference is likely to have arisen from a true population difference, rather than just by chance if no difference existed in the population. This is sometimes called "parametric" testing, and the well-known t test is the prime example. Here, we typically make assumptions about the population distribution (such as that it is normal), identify a default assumption about this distribution (the Null hypothesis), and then develop a statistic (e.g., t) which has a resulting distribution if the null hypothesis were true (a t distribution). When then compute that statistic for the data we collected, and the statistical test is the process of looking at the hypothetical distribution, and determining how likely the value we observed would be if the null hypothesis were true.

Two aspects of NHST pervade thinking about scientific research: the null hypothesis, and the p-value. The null hypothesis is the stance that the default assumption is that no difference exists between groups, and that we should seek to find evidence that allow us to reject this hypothesis. When we conduct a statistical test, the value of the statistic is examined and we assign a probability associated with the chance that the value observed could have come from the null hypothesis. If this is unlikely (less than 1 in 20), the test is usually deemed "statistically significant". But neither of these are necessary, and we will see when discussing Bayes factor tests.

8.1.2 Non-parametric tests of group differences

A traditional test like the t test assumes the data follow a specific distribution, and a test statistic is computed that determines the likelihood that the observed data arose under the null hypothesis. But what if the assumptions about the distribution do not hold? Because of the central limit theorem, this is sometimes not too big of a challenge, but we still may wonder whether these conclusions still hold. What if the data are skewed, like times and money spent are? What if the data are bounded between 0 and 1 (like accuracy or conversion rate), or between 0 and some large number (like response time), or among a small set of categories (like a likert scale). When you think about it, almost nothing studied directly in psychology and human factors is truly normally distributed. More critically, you often cannot tell whether the violations really exist, and so it is hard to know the impact of violations when you do not have a lot of data.

In response to these concerns–especially for cases in which we know the data are not normally distributed, statisticians will take another approach: non-parametric tests. Non-parametric testing is somewhat of a misnomer, but the general approach is to find a way of looking at the data and create a NHST that do not rely on making strong assumptions about the distribution of the random variable. Non-parametric tests often make use of rank or order to draw conclusions. For example, if you want to know whether the mean of one group of data is greater than the mean of a second, a non-parametric statistic may involve

mixing both sets of data and rank-ordering them. If substantially more than half of one group is in the top half of the data, this might be evidence to reject the null hypothesis.

Non-parametric tests are typically used for data which we know violate assumptions of normality. For example, they can be useful for data I you have ordinal responses, or are highly skewed and so have long tails that will influence the outcomes. That is, some observations will be substantially larger or smaller than the median, making your estimates of variance very large, and your estimates of mean unrepresentative of the central tendency. Most non-parametric tests work by transforming the data into ranks and running parametric comparisons on those ranks. The most common non-parametric tests corresponding to the t-test are the 'Wilcox' and 'Mann-Whitney U' tests.

8.1.3 Bayes Factor Tests

Another alternative to the classic NHST framework is the Bayesian framework. Bayesian tests apply a normative rule for combining evidence–Bayes Rule–to draw inferences about hypotheses and distribution parameters given the data.

Bayes rule is an optimal way of combining evidence with your current strength of hypothesis to determine the balance of evidence in favor or against hypotheses. A simple formula for Bayes rule is written:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{8.1}$$

What this says is that if we want to know the probability of A (an outcome or hypotheses) in the light of evidence or data B, we need to know two things: the probability of the evidence given the hypothesis ($P(B \mid A)$, called the likelihood), and the outright probability of the the outcome or hypothesis ($P(B)$, called the prior). The likelihood is just the height of the density function at the observed data point, and the prior is usually arranged so that it does not bias you in favor or against one of the hypotheses. For some Bayesian inference approaches, we do not estimate single parameter values (in the sense of finding the most likely parameter), but rather estimate a distribution of likely parameter values that may have produced the result. This is called the posterior distribution. We will look at how we can use these for estimating variability, but the Bayes Factor approach generally hides this all from us by combining the total likelihood of each hypothesis into a number called the Bayes Factor. This includes the possibility of evidence in favor of the null hypothesis. Thus, unlike the NHST, you can determine whether the data provide a good case for the null hypothesis. There are quite advanced hierarchical Bayes approaches that involve estimation via Monte Carlo simulation, and allow complex hierarchical relationships to be modeled. In this course, we will focus on a simpler approach that is almost a drop-in replacement for a NHST: a Bayes Factor test. For each classic and non-parametric test we will examine, we will also look at the Bayesian version of the test, which produces as an outcome a Bayes Factor–a relative likelihood of hypotheses–rather than a p-value.

8.1.4 Other Bayesian tests

There are a number of other Bayesian libraries used for Bayesian data analysis. For complex models, researchers build complex networks that might have millions of parameters and use monte carlo methods to do inference and sampling, using BUGS, JAGS, STAN, and other frameworks. These are generally more complicated but more powerful. The BayesFactor library uses fairly simple schemes with the goal of producing a Bayes Factor. The `bain` library extends Bayes Factor analysis for a wider range of hypothesis models and constraints.

Kruschke's BEST approach is a mix of these two, and provides t-tests with a Bayesian underpinning that produce more traditional output. These are available in `BayesianFirstAid` library, and the `BEST` library. There is an `rBesT` library which also does Bayesian tests but may be unrelated to BEST. The `Bolstad` library has Bayesian analysis functions that are a companion to Bolstad's 2007 book *Introduction to Bayesian Statistics, John Wiley & Son*, and the `BaM` library provides functions used in Bayesian Methods: A social and behavioral sciences approach. There are packages for many specific analysis, including `walker` for Bayesian GLMs, Bayesian meta-analysis (e.g., `metabup`), `bsts` library extends this to time series analysis, and many others. These mostly don't work together, but may be handy in particular cases, and you should recognize that the Bayes Factor is only the start of Bayesian analysis.

8.2 Example: Simulating the NULL hypothesis

To start thinking about testing the difference between groups, let's consider the null hypothesis– that there is no difference in the means of two groups of observations. For example, you might want to know whether a drug is effective at producing weight loss. In this case, the null hypothesis would generally be that the drug has no effect on weight loss, with the alternative being either that it has some effect (i.e., the means of two groups are NOT the same), or that the treatment improves weight loss (i.e., the mean of the drug group is lower than the untreated group).

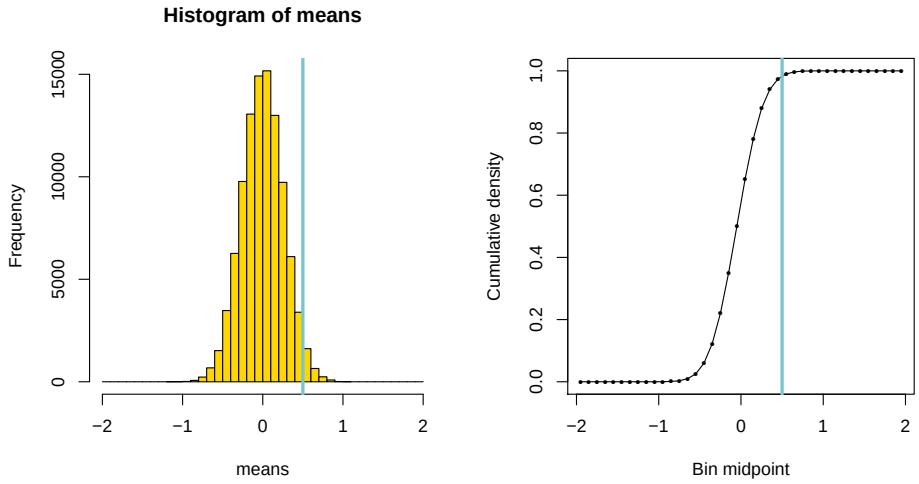
As we have seen before, even if the null hypothesis were true, our observed mean will almost never be exactly 0. And in fact it will have a distribution whose size depends on both the distribution of the original data and the number of samples used to compute the mean. Thus, even if the null were true and you ran the study many times, you would get a distribution of results—a random variable. In general, we cannot know this distribution without conducting thousands of experiments. To start, let's assume the original data are normally distributed with a mean of 0 and standard deviation of 1.0, and we want to see the distribution, supposing we sampled 15 observations.

```
 numexps <- 10000
means <- rep(0,numexps)
sds <- rep(0,numexps)
expsize <- 15
for(i in 1:numexps)
{
  #Generate one experiment:
  data <- rnorm(expsize,mean=0,sd=1)
  #estimate its mean
  means[i] <- mean(data)
  sds[i] <- sd(data)
}
hist(means)
 mean(means)
sd(means)
[1] 0.257
```

8.3 The t-test approach

Here, for an experiment of size 15, the standard deviation of the means is .257, and the mean of the means is close to 0. Suppose we wanted to come up with some ad hoc cut-off

Figure 8.2: Histogram of 10000 experiments with 15 observations of the null hypothesis of N(0,1). Here, a criterion of +.5 would only accept the null hypothesis incorrectly about 2.5% of the time.



criterion to help us decide whether we could reject the null hypothesis (i.e., no difference), but without knowing the whole distribution–after all, we only know one sample from the distribution. To do so, we would might decide that if we have a mean of 0.5, that is enough different from 0 that we consider it unlikely to have happened just by chance. In this case, if the null hypothesis were true, it turns out that 97.3% of those experiments would produce values less than 0.5, so this seems reasonable, and pretty cautious. About 5% of the time would be reject the null hypothesis accidentally, and maybe this is acceptable to us.

But what would happen if it did have a true difference? Suppose the true mean were +.5, and the variance of the distribution was the same. In this case, our observed means would straddle 0.5 instead of 0, and half of the time we'd fail to reject the null hypothesis even if it was true. Thus, although we make only 5% false alarms, we fail to detect a true effect 50% of the time. This is known as the power of the test, and this test has limited power, mostly because the sample size is very small. The smaller the sample size, the larger a difference that is required if we want to limit the probability of false alarms, and so the smaller the power of the test becomes.

So, under the NHST, we must identify a criterion on the null distribution that we can only know if we collect thousands of experiments. We'd like to avoid conducting thousands of experiments, because that is the point of doing an inferential test. So, we make this estimate based on the one experiment we did do, in which we calculated the mean and standard deviation statistics. It makes intuitive sense that our test should depend on three things: the observed difference between means (in this case $0.5 - 0 = 0.5$), the variability of the distribution (in this case, standard deviation = 1.0), and the number of observations (in this case, 10). That is, if we want to be confident about very small differences in mean relative to the standard deviation, we should be prepared to collect a lot of data. We showed above that what matters is the distribution of the mean, which was about .315 when $N = 10$ and stdev = 1.0. What happens to this relationship when we increase N?. Does it go down? in what fashion–linearly, or with some other relationship?

Exercise 8.3

What is the sd of the means for sampling 10, 20, 30, 50, and 100 per experiment?

As you can see from the exercise, the standard deviation of the mean does not decrease linearly, but the extent to which it decreases diminishes as N increases. In fact, it can be proven that the standard deviation of the mean descreases with the square root of N. So, for examples, $1/\sqrt{15} = .258$, roughly the same standard deviation we observed in the simulation. We'd predict that for 100 trials, the standard deviation of the means should be 0.1.

8.3.1 Estimating the variability of the mean

The previous simulation shows what should be intuitive—that the standard deviation of the random variable describing our estimate of the mean is related to how many trials were in the experiment. In fact, this variability has a name: the 'standard error', and it has a well-known relationship with the standard deviation of a single experiment. In a standard NHST, if we want to estimate the variability of the null hypothesis distribution, we use the standard error:

$$se = sd/\sqrt{N} \tag{8.2}$$

Not coincidentally, when plotting error bars, it is typical to plot standard error as the error bars. These error bars give a confidence interval in your mean, as opposed to the standard deviation, which gives a confidence interval of your data.

Exercise 8.3.1

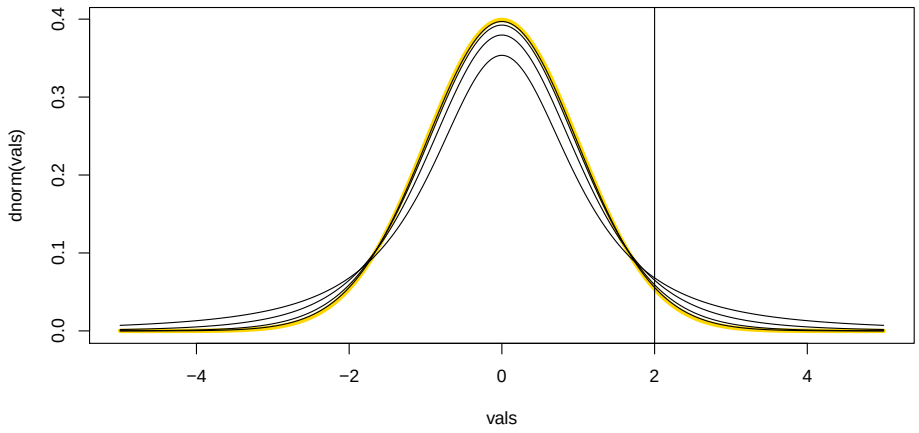
Compute the expected standard error for experiments of size 10, 20,30, 50, and 100 that have an observed standard deviation of 1.0, and compare this to the results of the previous exercise.

Now that we know how to estimate the standard deviation of our estimate of the mean, we can use this fact to do a simple test. We do this via a simple statistic: $t = mean/se$. If the mean were 0 and the observed standard deviation was the same as what we observed, the expected distribution (under the null hypothesis of no difference) is called a t distribution, and this distribution has a well-understood density and distribution function. In R, this distribution is available via several functions, including `dt` for the density and `pt` for the cumulative probability less than a particular value. Now, instead of an ad hoc criterion like +0.5, we can choose a criterion specifically to depend on the number of false alarm errors we want, and compute the power we would need to detect the difference.

So, to do an inferential t test about the mean difference, we go through the following steps:

- Step 0: Convince ourselves that the assumptions of the test are not violated
- Step 1: Compute the estimate of the mean
- Step 2: Estimate the variability of your estimate (standard error)
- Step 3: compute the t statistic for the sample.
- Step 4: look up the probability of getting your value, if there were really no difference (Null hypothesis).

Figure 8.3: The shape of the t distribution, as the number of observations increases from 2 to 50



Our simulation earlier was essentially a simulation of the t distribution. Instead of simulating it, we can look it up with R directly. The t distribution requires that we specify its degrees of freedom (df), which is $N - 1$ where N is the number of observations. As N increases, the distribution get sharper, and approximates the normal distribution more and more closely.

```
par(mfrow=c(1,1))
vals <- -500:500/100
plot(vals,dnorm(vals),type="l",lty=1,lwd=4,col="gold")

points(vals,dt(vals,2),type="l",main="t distribution",
       ylim=c(0,.5),xlab="Mean")
points(vals,dt(vals,5),type="l")
points(vals,dt(vals,15),type="l")
points(vals,dt(vals,50),type="l")
abline(v=2)
```

Notice that the shape only changes a little bit, and has its biggest effect in the tail. Although the shape changes a little bit as N increases, what changes a lot is the t statistic. For a fixed distribution and mean, the resulting t value will increase with the reciprocal of the square root of N. Suppose you observed a value of 2, for different experiments with different sample sizes. For a small experiment, it would be fairly likely to get a value of 2 or larger by chance if the null hypothesis were true. This chance can be computed with the *pt* function, which computes the area to the left of a value. We need to subtract from 1.0 to get the chance of a value as large as the observation.

```
 1- pt(2.0,2)  ## 2 df/3 observations
 1- pt(2.0,5)  ## 5 df/ 6 observations
 1- pt(2.0,15)  ## 15 df/ 16 observations
 1- pt(2.0,49)  ## 49 df/ 50 observations
> 1- pt(2.0,2)  ## 2 df/3 observations
[1] 0.09175171
```

```
> 1- pt(2.0,5)  ## 5 df/ 6 observations
[1] 0.05096974
> 1- pt(2.0,15)  ## 15 df/ 16 observations
[1] 0.0319725
> 1- pt(2.0,49)  ## 49 df/ 50 observations
[1] 0.02552957
```

Here, with just 3 observations, we have a 10% chance of seeing a value more extreme that 2.0 by chance. With just 6 observations, this dips to about 5%–meaning we have a relatively low false alarm rate. However, we'd probably suspect that the power of such a test is very low. Note that I subtracted from 1 to give us the probability to the right of a criterion. We could also have used use `pt(t,df,lower.tail=F)`, which directly gives the probability to the right of the value. If we are doing a 2-tailed test, we can compute the value as `(1-pt(t,df))/2`.

8.3.2 One-sample t

The simplest t-test is one where we just compare the difference of the mean of a sample to 0. This is called a one-sample t-test.

We will first do this by hand. Let's try this for 20 normal deviates whose mean differs slightly from 0.

```
set.seed(1000)
x0 <- rnorm(30,0)
x1 <- rnorm(20,.2)
x1a <- rnorm(200,.2)
x2 <- rnorm(20,.5)
x3 <- rnorm(20,mean=.2,sd=.2)
x4 <- exp(rnorm(100))-1

vioplot(x0,x1,x1a,x2,x3,x4,col="gold",
        names=c("x0","x1","x1a","x2","x3","x4"))
abline(0,0)
```

These are the steps to compute a t-test by hand.

Step 0.

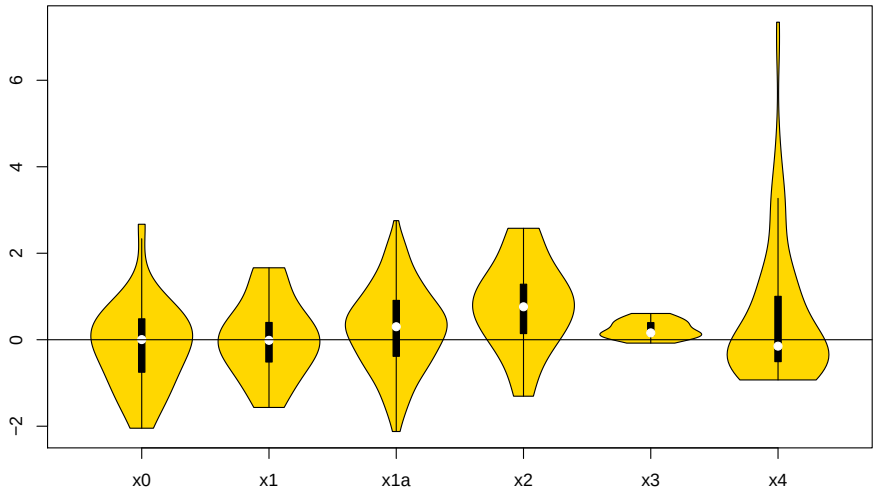
All but `x4` look reasonably normal, if we trust the vioplot visualization. It might be reasonable to make normal Q-Q plots as well.

Step 1.

Next, compute the mean of the distribution

```
mu <- mean(x1)
```

Figure 8.4: Example distributions, all but x0 were generated with a mean gerater than 0



Step 2.

Now, compute the standard error and the resulting t-value

```
sd <- sd(x1)
se <- sd/sqrt(length(x1))
t <- mu/se
t
```

The t value is just .15, which is very likely to have happened just by chance. Let's figure out what that value is:

Step 3.

Finally compute the area of the appropriate t distribution more extreme than the sampled value:

```
 pt(t,19,lower.tail=F)
[1]  0.4388957
```

If this is 'two-tailed', we just multiply by 2

```
pt(t,19,lower.tail=F)*2
[1] 0.8777914
```

This is all a bit tedious and error-prone. The *t.test*() function automates all of this, and provides additional information.

```
t.test(x1,alternative="greater")

  One Sample t-test

data:  x1
t = 0.15585, df = 19, p-value = 0.4389
alternative hypothesis: true mean is greater than 0
95 percent confidence interval:
 -0.3109046        Inf
sample estimates:
mean of x
0.0307994

> t.test(x1,alternative="two.sided")

  One Sample t-test

data:  x1
t = 0.15585, df = 19, p-value = 0.8778
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.3828153  0.4444141
sample estimates:
mean of x
0.0307994
```

Note that the p-values and t-values are the same as we calculated before.

Exercise 8.3.2

Compute a one-sample t-test for x0, x11 x1a, x2, and x3. For each one, determine the probability you would have seen the value if the null hypothesis were true.

8.3.3 One-sample non-parametric equivalent to the t test

There are many problems with this basic approach, which are sometimes looked at as features or benefits rather than limitations. These include:

- Reliance on knowing the form of the underlying distribution; typically that it is normally distributed
- Assuming the variability in the process is the variability we observed (in reality it is not likely to be true)
- Only trying to determine whether the null hypothesis is wrong. There are often situations where we'd like to also know whether it is right, or at least whether the balance of evidence favors one over another (or is ambivalent.).

The first concern is sometimes addressed using a non-parametric test. In non-parametric tests, we generally try to do the test based on order statistics, rather than the specific values. For example, if you want to know whether one condition of a test has a higher value than another, you can look at how many participants scored higher on it, and do a NHST using a test that no longer needs to make an assumption about normal distribution. However, the rest are still problematic, and many of them are dealt with using a Bayesian testing approach.

The non-parametric version of of a one-sample test is known as a sign test, because we'd expect about half of the samples to be above 0 and half below 0. The null distribution is just a binomial distribution, and a test is available with `binom.test`. This test just takes two values–the number greater than 0 and the total number being tested. The `psych` package has an alternative test called SIGN.test that allows you to use the entire data set.

```
> binom.test(sum(x0>0),length(x0),p=.5)

  Exact binomial test

data:  sum(x0 > 0) and length(x0)
number of successes = 15, number of trials = 30, p-value = 1
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.3129703 0.6870297
sample estimates:
probability of success
                   0.5
```

Here, we can see that because the number of successes is exactly half (15), the p=value is 1.0. We can test this against any particular probability (a fair or unfair), but more than likely we'd use this to compare to a particular criterion–maybe instead of 0, compare it to some known criterion to see if there is support that the observed data improved over some objective.

The easier to use "Sign test" is available within the BSDA library, among other places:

```
##BSDA library has a simpler to use version:
library(BSDA)
SIGN.test(x0)
  One-sample Sign-Test

data:  x0
s = 15, p-value = 1
alternative hypothesis: true median is not equal to 0
95 percent confidence interval:
 -0.5443371  0.2075894
sample estimates:
median of x
0.003213122

Achieved and Interpolated Confidence Intervals:

                  Conf.Level  L.E.pt U.E.pt
Lower Achieved CI     0.9013 -0.4759 0.1701
Interpolated CI       0.9500 -0.5443 0.2076
Upper Achieved CI     0.9572 -0.5545 0.2132
```

Exercise 8.3.3

Compute a one-sample non-parametric sign test for x01 x1a, x2, and x3. For each one, determine the probability you would have seen the value if the null hypothesis were true.

Bayes Factor	Decimal	Evidence
$< 1/150$	Smaller than .00667	Very strong evidence for Null
1/150 to 1/20	.00667 to .05	Strong evidence for Null
1/20 to 1/3	.05 to .333	Positive support for Null
1/3 to 3	.333 to 3.0	Not worth mentioning; ambivalent
3 to 20	3.0 to 20.0	Positive support for hypothesis
20 to 150	20.0 to 150.0	Strong support for hypothesis
150+	150.0+	Very strong evidence for hypothesis

8.3.4 Example: One-sample Bayes Factor t test

In the NHST approach, you can either reject the null hypotheses (no difference) or fail to reject, but you can never really accept the null. The result of our test is usually a p-value, which is the probability that the null would be incorrectly In contrast, the results of the Bayesian approach provide a posterior likelihood value of each hypothesis given the data. So, although strong evidence for the alternative usually produces similar results, a Bayesian test can differ from the others if there is not evidence. A standard NHST will only tell you that you failed to reject the null hypothesis; a Bayesian test can tell you whether there is strong or weak evidence for the Null hypothesis.

Bayesian tests take a number of different forms. We will be looking at Bayesian hypothesis testing using the Bayes Factor. Bayes Factor is a ratio of likelihood values–essentially weight of evidence in favor of the hypothesis versus an alternative. Here, instead of just rejecting a null hypothesis, we look for evidence for the current hypothesis, in contrast to the null. If they are equally likely, the balance of evidence is equal, with a ratio of 1:1. If the alternate hypothesis is ten times more likely than the null, the ratio would be 10:1 (or 1:10, depending on how the hypothesis was framed.). Bayes factors less than ten are considered no real weight in favor or against. Larger values start providing stronger weight, and values closer to 0 provide evidence in favor of the null (values smaller than 0.1). Thus, we can do more than just reject the null hypothesis–we can look at evidence in favor of each hypothesis, and we can also easily say that our experiment cannot determine whether either is likely to be true (any bayes factor between about 0.1 and 10).

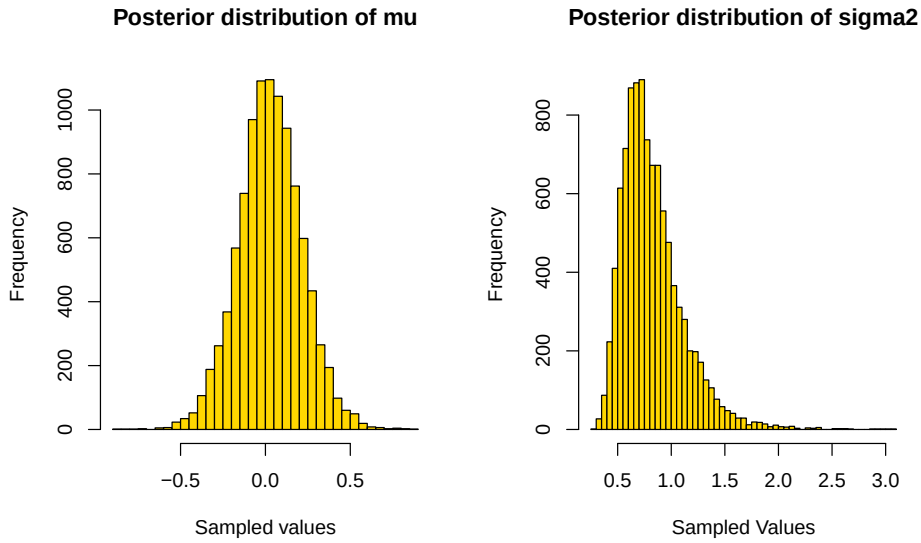
Some basic guidelines for Bayes factors (cf Jeffreys, 1961, Kass & Raftery)

Many bayesian tests are available within the BayesFactor package in R. The relevant test is `ttestBF`, and if you give it a single data set it will return a Bayes factor like in the table above:

```
library(BayesFactor)
ttestBF(x1)
Bayes factor analysis
--------------
  [1] Alt., r=0.707 : 1.064856 +/-0%

Against denominator:
Null, mu = 0
---
Bayes factor type: BFoneSample, JZS
```

Figure 8.5: Posterior estimates of the mean and variance of the distribution of X1.



Note that it does not provide a lot of detail. Importantly, the $r = .707$ is an assumption that impacts the prior distribution–the basic assumptions about how likely different alternative hypotheses are. The result of the test is a Bayes Factor of 1.06, which we can see from the table above is ambivalent support for the alternative. Interestingly, unlike the two NHST tests, our conclusions differ here. In those other tests, we merely said that we failed to reject the null hypothesis, which might imply that the mean is not different from 0. Here, our ambivalent result tells us that we don't have enough data to say whether or not the null is likely–and in fact there is a slight bias toward the alternative hypothesis.

Note that the Bayesian approach does not estimate a single value for the parameters (in this case, mean μ and variance σ^2. We know the true mean was .2 and true variance was 1.0. We can extract the posterior likelihood distribution of parameter estimates by turning samples to TRUE.

```
samples <- ttestBF(x1,iterations=10000,posterior=TRUE)
par(mfrow=c(1,2))
hist(samples[,1],breaks=100)
hist(samples[,2],breaks=100)
```

It is fairly common to use these distributions to plot likely values, or as the basis for error bars on a figure displaying the means. We can see that the likely value of the means range quite broadly, as do the likely values of the standard deviation. We just do not have a good idea what the true values might be in this case.

Exercise 8.3.4

Compute a one-sample Bayes factor test for x0, x1a, x2, and x3. For each one, examine the posterior distribution of the mean.

8.4 Paired Sample tests

The one-sample tests we just examined are formally identical to paired-sample tests. In a paired t-test, we have two observations from each person, participant, or observable category. We want to know whether the one group of observations differs from the other group of observations, but we want to take advantage of the fact that we measured each case twice. Thus if there is a wide variability in base level across people, we may be able to nevertheless detect small changes. For example, suppose you performed a pre-test on the number of push-ups completed, and then did a 7-day training intervention, and post-test. The training intervention may only increase average number of push-ups by one or two, but there may be a huge range (from none to hundreds). But if everyone improved by one or two, we still have a very consistent improvement. To do a paired test, you can just find the difference between each observation of the two groups (making sure pre and post are linked by person), and then compute a one-sample test. Or you can feed the two groups into various functions specifying you are doing a paired test. For our example, consider x0b:

```
x0b <- x0 + runif(length(x0),min=-.1,max=.2)
```

8.4.1 Paired t test

A paired t-test will compare the difference between two groups, by default ignoring order (higher or lower). Note that you have to be more careful if you choose to subtract the differences and do a one-sample t-test. In each case, we can also do one-sided or two-alternative tests. If you hypothesize the difference should be in one direction, use `alternative="greater"`, otherwise use the default (`"two-sided"`).

```
t.test(x0,x0b,paired=T)
t.test(x0,x0b,paired=T,alternative="less")
t.test(x0-x0b)  ##one-sample version
t.test(x0,x0b) ##this is wrong, and not a paired t-test
```

Here, the two-sided version is default, and we see a p-value close to .01, meaning that it is unlikely to have occurred if the null hypothesis were true. Doing a one-sample version by doing a t-test of the differences ends up giving the same results.

```
> t.test(x0,x0b,paired=T)

  Paired t-test

data:  x0 and x0b
t = -2.5012, df = 29, p-value = 0.01828
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.05761446 -0.00577806
sample estimates:
mean of the differences
             -0.03169626


> t.test(x0-x0b)  ##one-sample version

  One Sample t-test

```

```
data:  x0 - x0b
t = -2.5012, df = 29, p-value = 0.01828
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.05761446 -0.00577806
sample estimates:
  mean of x
-0.03169626
```

We can do a one-sided test, and the p-value gets smaller.

```
> t.test(x0,x0b,paired=T,alternative="less")

  Paired t-test

data:  x0 and x0b
t = -2.5012, df = 29, p-value = 0.009138
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
        -Inf -0.01016405
sample estimates:
mean of the differences
            -0.03169626
```

If you forget the paired option, it will report a two-sample test, and this is not nearly as powerful.

```
> t.test(x0,x0b) ##this is wrong, and not a paired t-test

  Welch Two Sample t-test

data:  x0 and x0b
t = -0.12368, df = 57.939, p-value = 0.902
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.5447144  0.4813219
sample estimates:
 mean of x  mean of y
-0.1515832 -0.1198869
```

8.4.2 Non-parametric Paired Comparisons

Again, you can use a binomial.test on comparisons to determine the difference between two groups in a non-parametric test. However, the `wilcox.test` (sometimes called the according to its corresponding "Wilcoxon" distribution) is a more direct replacement for the t test based on similar principles. Essentially, it will determine how many of the values are greater than some value, ignoring the actual values. In this case, the wilcox test seems to be more powerful, because it not only counts values absolute differences, but will penalize the smaller differences less than the greater differences.

```
##non-parametric paired tests:
diff <- x0-x0b
binom.test(sum(diff>0),length(diff))
```

```
wilcox.test(diff,mu=0)
wilcox.test(x0b,x0,paired=T)
```

Results show that the binomial version is not sensitive to the difference at $p < .05$. The one-sample wilcox test and the paired wilcox test are identical.

```
binom.test(sum(diff>0),length(diff))

  Exact binomial test

data:  sum(diff > 0) and length(diff)
number of successes = 11, number of trials = 30, p-value = 0.2005
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.1992986 0.5614402
sample estimates:
probability of success
             0.3666667

> wilcox.test(diff,mu=0)

  Wilcoxon signed rank test

data:  diff
V = 133, p-value = 0.04049
alternative hypothesis: true location is not equal to 0

> wilcox.test(x0b,x0,paired=T)

  Wilcoxon signed rank test

data:  x0b and x0
V = 332, p-value = 0.04049
alternative hypothesis: true location shift is not equal to 0
```

8.4.3 Bayes Factor Paired Comparisons

The `ttestBF` permits testing paired samples as well.

```
ttestBF(x0,x0b,paired=T)
ttestBF(x0-x0b)
> ttestBF(x0,x0b,paired=T)
Bayes factor analysis
--------------
[1] Alt., r=0.707 : 2.715257 +/-0%

Against denominator:
  Null, mu = 0
---
Bayes factor type: BFoneSample, JZS

> ttestBF(x0-x0b)
Bayes factor analysis
--------------
[1] Alt., r=0.707 : 2.715257 ?0%

Against denominator:
```

```
  Null, mu = 0
---
Bayes factor type: BFoneSample, JZS
```

Note that the results for the paired and the one-sample equivalent are again identical: 2.7. This is below the 3.0 criterion for positive support, and is identified as not worth mentioning. However, this is essentially a two-sided test. We can restrict the null hypothesis to just positive differences using the nullInterval–treating it as any values between 0 and 100 for example:

```
ttestBF(x0-x0b,nullInterval=c(0,100))
> ttestBF(x0-x0b,nullInterval=c(0,100))
Bayes factor analysis
--------------
[1] Alt., r=0.707 0<d<100    : 0.0624813 ?0%
[2] Alt., r=0.707 !(0<d<100) : 5.344257  ?NaN%

Against denominator:
  Null, mu = 0
---
Bayes factor type: BFoneSample, JZS One-sample Sign-Test

data:  x0
s = 15, p-value = 1
alternative hypothesis: true median is not equal to 0
95 percent confidence interval:
 -0.5443371  0.2075894
sample estimates:
median of x
0.003213122

Achieved and Interpolated Confidence Intervals:

                  Conf.Level  L.E.pt U.E.pt
Lower Achieved CI     0.9013 -0.4759 0.1701
Interpolated CI       0.9500 -0.5443 0.2076
Upper Achieved CI     0.9572 -0.5545 0.2132
```

Here, we have two Bayes factors. The evidence for the null hypothesis is very strong against, whereas the evidence for the alternative is very strong in favor of.

We can use nullInterval to specify other ranges we want to consider as the null. For example, maybe for x1, we want to know whether there is evidence of cheating, and we think a reasonable guessing strategy would net you between 15% and 25% correct. We can treat this as the null, and then look for evidence of cheating. If you get less than chance, you may be copying off the wrong sheet and so could get below chance. Looking at x1 and x1a the results show reasonably strong support in favor of the null, with weaker support for the alternative.

```
ttestBF(x1,nullInterval=c(.15,.25))
Bayes factor analysis
--------------
[1] Alt., r=0.707 0.15<d<0.25    : 3.385209  ?0%
[2] Alt., r=0.707 !(0.15<d<0.25) : 0.9640481 ?0%

Against denominator:
Null, mu = 0
```

```
---
Bayes factor type: BFoneSample, JZS

> ttestBF(x1a,nullInterval=c(.15,.25))
Bayes factor analysis
--------------
[1] Alt., r=0.707 0.15<d<0.25    : 10.83408  ?0%
[2] Alt., r=0.707 !(0.15<d<0.25) : 0.5729178 ?0%

Against denominator:
Null, mu = 0
---
Bayes factor type: BFoneSample, JZS
```

Exercise 8.4.3

Compute each paired-samples test (t-test, wilcox, and Bayes factor) for the following data, in comparison to the original values (e.g., compare x1.2 versus x1):

```
y1  <- x1  + rnorm(20,mean=.04)
y1a <- x1a + rnorm(200,mean=.04)
y2  <- x2  + rnorm(20,mean=0,sd=5)
y3  <- x3  + rnorm(20,mean=.2,sd=3)
y4  <- x4  + exp(rnorm(100))/10-1
```

8.5 Comparing two independent samples.

When we don't have paired comparisons or a single group and want to compare two independent groups, it becomes a bit trickier. We need to come up with an estimate of the standard deviation in order to estimate the standard error and t-value, but there are now two standard deviations.

8.5.1 Independent samples t-test

```
muX <- mean(x1)
muY <- mean(x1a)

sdX <- sd(x1)
sdY <- sd(x1a)
```

When we have two groups, we'd still like to be able to use the t-test logic. In this case, we have a difference in means, and we have an estimate of the variance of the differences. But since the variances are likely to be different, and might come from two different group sizes, we have to do sort of a weighted mean.

We will typically use Welch's formula for pooling variance across groups:

```
se.pooled <- function(x,y)
{
  varx <- var(x)
```

```
  vary <- var(y)
  nx <- length(x)
  ny <- length(y)
  sqrt(varx/nx + vary/ny)
}
```

Now, we can compute the t value based on this new s.d.

```

t <- (muX-muY)/se.pooled(x1,x1a)
t
pt(t,38)
2*(1-pt(t,38))

> t
[1] -1.060023


> pt(t,38,lower.tail=T)
[1] 0.1479127
```

This is what t.test does when given two sets:

```
> t.test(x1,x1a,alternative="less")

  Welch Two Sample t-test

data:  x1 and x1a
t = -1.06, df = 23.719, p-value = 0.1499
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
      -Inf 0.1361698
sample estimates:
mean of x mean of y
0.0307994 0.2522959
```

However, notice that the p-value is slightly different, and the degrees of freedom are also different. This is because the Welch's t-test applies a correction to the degrees of freedom to account for the different variances, and adjusts to 2.3719. We can see that this is exactly what the `pt` function produces if we applied the same adjustment:

```

> pt(t,38,lower.tail=T)
[1] 0.1479127
 pt(t,23.719,lower.tail=T)
[1] 0.1499048
```

However, you would need to compute the degree-of-freedom correction yourself, so it is easiest to ues the t-test.

8.5.2 Independent-samples non-parametric tests

When the Wilcox test is given two independent samples, this is sometimes called a "Mann-Whitney U" test.

```
wilcox.test(x1,x1a)
> wilcox.test(x1,x1a)

  Wilcoxon rank sum test with continuity correction

data:  x1 and x1a
W = 1695, p-value = 0.2619
alternative hypothesis: true location shift is not equal to 0
```

The Wilcox test computes its test statistic by first looking at each pairing of X and Y, and then counting for how many of them X is greater or equal to Y. We can do this with the following code:

```
  set.seed(100)
  x <- runif(20)
  y <- runif(20)+.1
  pairs <- outer(x,y,"<=")
  pairs
  sum(pairs)
[1] 293
```

We can see that 293 out of 400 pairs satisfied the test of being greater than or equal to. Just like the t statistic, the Wilcox test computes its own statistic (U), which is often referred to as the Mann-Whitney U. These U values have a distribution under the null hypothesis for a given set of data. In R, This null distribution is given by the `pwilcox` function. Here, we can plot the density and distribution functions of the null distribution we'd expect from sampling 20 values.

```
plot(pwilcox(0:400,20,20))
plot(dwilcox(0:400,20,20))
```

We can see that the likelihood of getting 293 or more is very low, and so this is likely to be highly significant.

We could look up our particular results using `pwilcox`:

```
1- pwilcox(293,20,20)
[1] 0.005157019
```

Thus, this result was very unlikely to happen under the null hypothesis. If we compare this to the result of the wilcox.test function, we get a slightly different number because wilcox.test uses a two-sided test. If we set it to the "less" option, we get exactly the same value. Note that W=107 is the same as W=293, because they add up to 400.

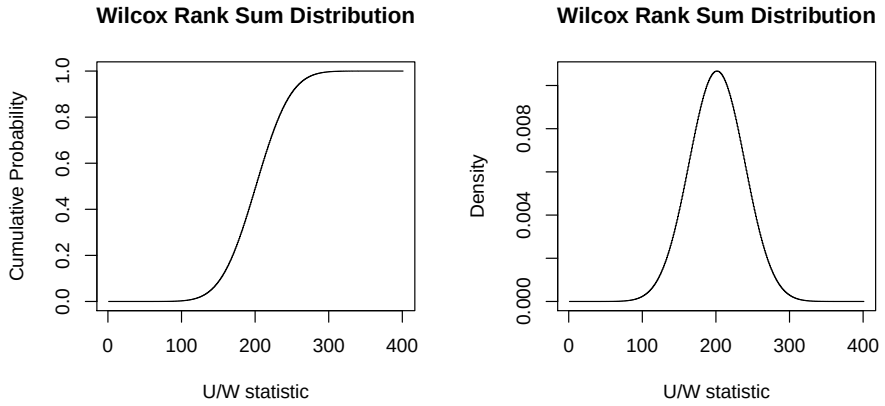
```
> wilcox.test(y,x)

  Wilcoxon rank sum test

data:  y and x
W = 293, p-value = 0.01121
alternative hypothesis: true location shift is not equal to 0

```

Figure 8.6: Cumulative distribution and density of the null distribution comparing sets of size 20.




```
> wilcox.test(x,y,alternative = "less")

Wilcoxon rank sum test

data:  x and y
W = 107, p-value = 0.005603
alternative hypothesis: true location shift is less than 0
```

If you have a factor defining groups, and the data are stored in long format (on variable is the independent variable; one is the dependent variable) you can run the test this way too, and it produces the same results:

```
groups <- rep(1:2,each=length(x))
values <- c(x,y)
w <- wilcox.test(values~groups)
```

8.5.3 Bayesian independent samples comparisons of group means

Of course, there is a corresponding Bayes Factor test for comparing means of independently sampled distributions.

```

ttestBF(x1,x1a)

> ttestBF(x1,x1a)
Bayes factor analysis
--------------
[1] Alt., r=0.707 : 0.3680828 0.01%

Against denominator:
  Null, mu1-mu2 = 0
---
Bayes factor type: BFindepSample, JZS
```

Here, the bayes factor is between .333 and 3.0, which is nothing to remark about, but biased toward the null hypothesis. All three tests seem to agree here.

8.6 Estimating Covariance and Correlation

The previous chapter showed how we estimate the variability of a set of data. Sample variance can be written like this:

```
set.seed(100)
data<- runif(1000)
var.sample <-  function(x){sum((x-mean(x))^2)/(length(x)-1)}
> var(data)
[1] 0.08254194
> var.sample(data)
[1] 0.08254194
```

We can easily factor the squared term out into two terms, to calculate the same thing:

```
var.sample2 <- function(x){sum((x-mean(x))*(x-mean(x)))/(length(x)-1)}
> var.sample2(data)
[1] 0.08254194
```

Or we can separate the first and second x values apart like this:

```
var.sample3 <-function(x,y){sum((x-mean(x))*(y-mean(y)))/(length(x)-1)}
> var.sample(data,data)
[1] 0.08254194
```

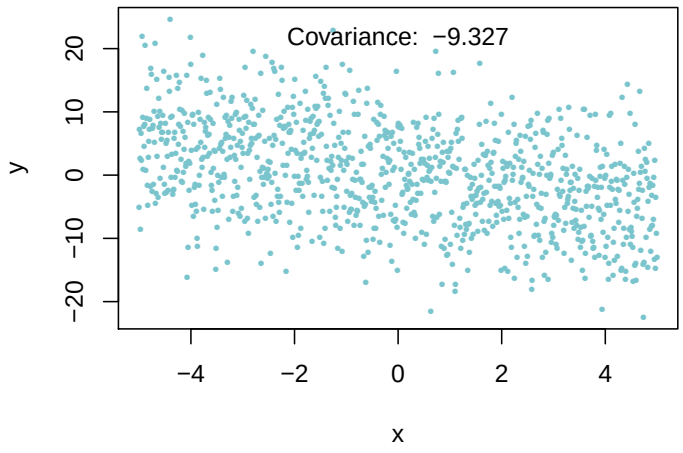
What would happen if we give this two different sets of data, where each entry of one set is related in some way to the corresponding entry in another set. Maybe it was measured on the same person, or the same trial, or the same day. If these are identical, it is equivalent to the variance. But when we give it two different data sets, this is called the *covariance*. To examine this, we will simulate a y value that depends on an x value as follows (data are shown in Figure 8.7

```
x<- -500:500/100
y <- rnorm(1001)*7-x
plot(x,y)
> var.sample(x,y)
[1] -9.327422

> var.sample(x,-y)
[1] 9.327422
> var.sample(-x,x)
[1] -8.35835
```

Here, y is related to $-x$, so the result is negative. The more two sets of data are related, the larger their covariance, but this also will depend on their overall scale. The R function `cov` will compute this directly:

Figure 8.7: Randomly generated data, where x and y are related via a noisy abstract relationship. This leads to a negative covariance between the values.




```
> cov(x,y)
[1] -9.327422
```

Covariance is useful to have, but hard to interpret because it is not scaled by the values:

```
cov(x,y)
[1] -9.327422
cov(10*x,y)
[1] -93.27422
```

Just by multiplying one of these by ten, we multiply the covariance by ten. It would be nice to just 'factor out' the scale when we compute it, so it is at a maximum of 1.0, and it will not depend on simple things like multiplying one variable by ten. If we think about the variance, dividing variance by itself would scale to 1.0, and this should be the maximum scaled covariance–perfect agreement. We generalize this by dividing covariance the standard deviations of both `x` and `y`:

```
cov.normalized <-function(x,y){
  cov <-  sum((x-mean(x))*(y-mean(y)))/(length(x)-1)
  var1 <- sum((x-mean(x))^2)/(length(x)-1)
  var2 <- sum((y-mean(y))^2)/(length(y)-1)
  cov/sqrt(var1)/sqrt(var2)
  }
```

You can verify that scaling y by a multiplicative factor has no impact:

```
> cov.normalized(x,y)
[1] -0.4105705
>
> cov.normalized(x,y*1000)
[1] -0.4105705
```

This is exactly how Pearson's correlation is computed, and is available in R as `cor(x,y)`. Correlation is a useful statistic to measure the association between two variables. Furthermore, its square, R^2, is bounded between 0 and 1 and measures of the proportion of variance accounted for in one set of data by another set. Typically, correlation can be viewed as an effect size directly, although R^2 is perhaps a bit more intuitive because it tells you directly how much of the variance is accounted for by a particular variable.

We can verify this 'proportion of variance' assertion by simulation. For example, consider the variable z, which is composed of two identically-distributed uniform random variables, which should each contribute half of the variance.

```
x <- runif(10000)
y <- runif(10000)
z <-  x + y

> cor(x,y)
[1] 0.003103427
> cor(x,z)
[1] 0.70823555
> cor(x,z)^2
[1] 0.5015975
> sqrt(.5)
[1] 0.7071068
```

Notice that the square of the correlation is about .5, and that `sqrt(.5)` is close to the correlation. We could create variables that are related to one another with other proportions by mixing the proportion of x and y, and the squared correlation will tend to recover the mixing proportion.

8.7 A statistical test for correlation

You can do a significance test on a correlation using the `cor.test` function. This test compares the observed value to the distribution that would be expected from a data set of equivalent size that are completely independent. The outcome is actually a t test:

```
set.seed(1000)
x<- -500:500/100
y <- rnorm(1001)*5-x
cor.test(x,y)


Pearson's product-moment correlation

data:  x and y
t = -18.953, df = 999, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
-0.5584397 -0.4671979
```

```
sample estimates:
cor
-0.5142725
```

The current x/y data set has 1001 points, and a correlation of around -.5. How sensitive is correlation to the range of our data? What if we had only sampled the smallest 100 x points, which go from -5 to -4, instead of the whole range from -5 to +5?

```
> sub<-1:100
> cor.test(x[sub],y[sub])

Pearson's product-moment correlation

data:  x[sub] and y[sub]
t = 1.4361, df = 98, p-value = 0.1542
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
-0.05438692  0.33065880
sample estimates:
cor
0.1435648
```

Here, the correlation is close to 0, because the range of the x values is so small (the red points in Figure 8.8. A rule of correlation is that if there is no variance, there can be no covariance (and thus no correlation). What if, instead, the 100 points were spread across the values of x?

```
sub2<-sample(1:1000,100)
cor.test(x[sub2],y[sub2])

> cor.test(x[sub2],y[sub2])

Pearson's product-moment correlation

data:  x[sub2] and y[sub2]
t = -5.1485, df = 98, p-value = 1.355e-06
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
-0.6031627 -0.2913998
sample estimates:
cor
-0.4614086
```

When we sampled 100 spread across the range, it was significant. What if we only sampled fewer? 20? Let's do this 10000 times and see what the distribution of correlations looks like:

```
cors <- rep(0,10000)
for(i in 1:10000)
 {
  sub<-sample(1:1000,20);
  cors[i] <- cor.test(x[sub],y[sub])$estimate
 }
```

Figure 8.8: Effects of sampling a subset of data on correlation. When a truncated range is sampled (red), no significant correlation is obtained, in contrast to when 100 points are sampled throughout the range.

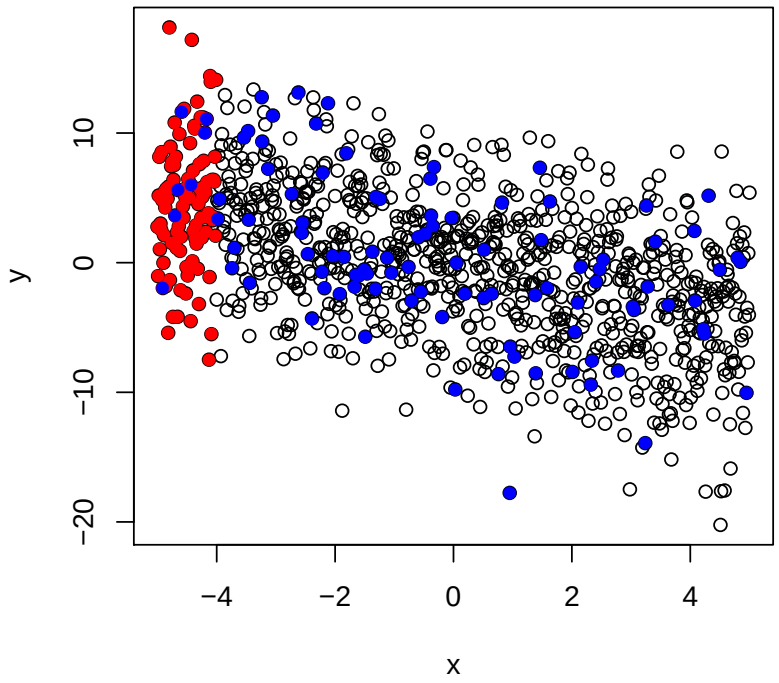
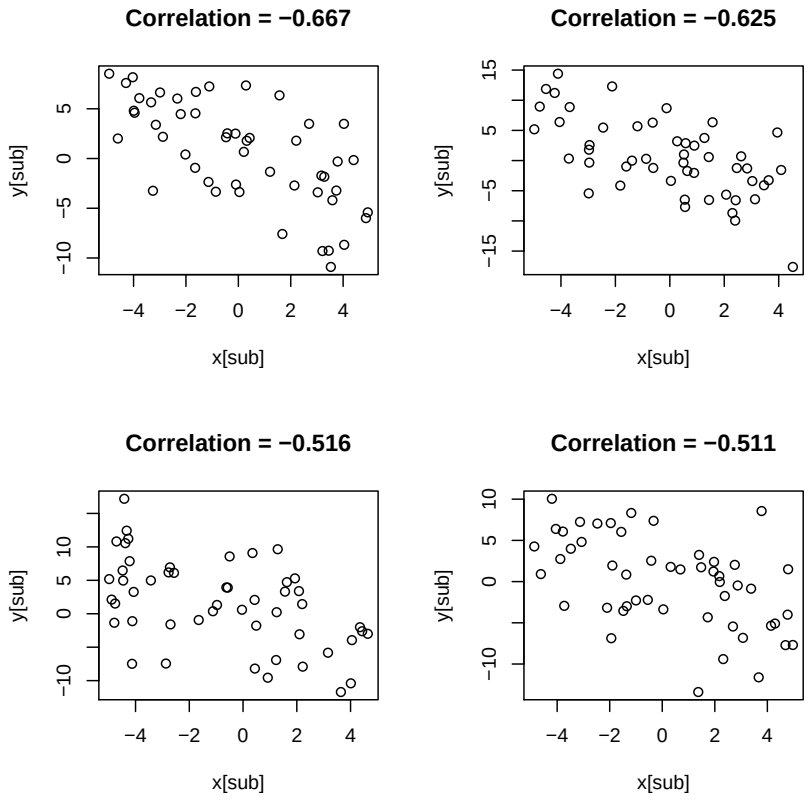
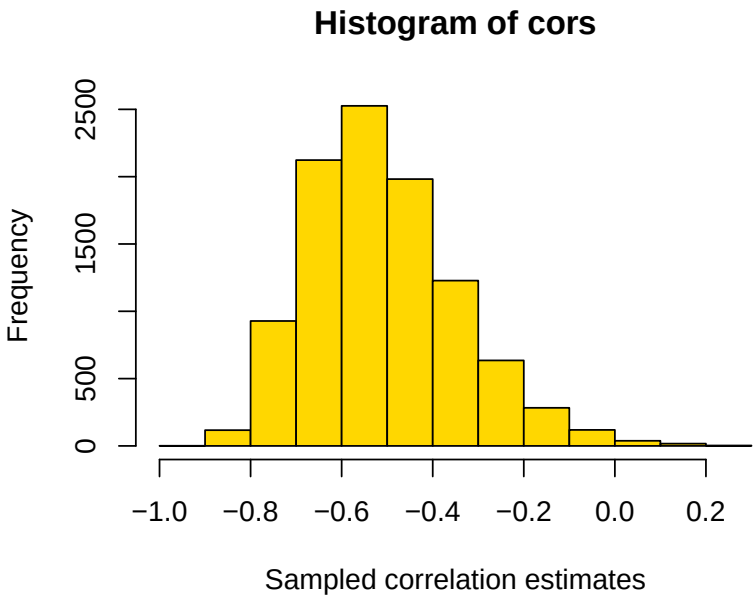


Figure 8.9: The effect on sampling small numbers of of two variables whose true relationship has a correlation around -.5.





Although the correlation is centered on -.3, it ranges quite a lot. This should give you some clue about how many observations you need to find a correlation in the .3 range–many more than 20!

Exercise 8.7

Suppose you have a true correlation of .3 between two variables, created by mixing one uniform with a second like this:

```
 x <- runif(10000)
 y <- runif(10000)
 z <- x + 3*y
> cor(x,z)
[1] 0.3096357
> cor(x,z)^2
[1] 0.09587427
```

Determine by monte carlo simulation roughly how many observations you would need to find a 5% increase 95% of the time? How about a 10% difference?

8.8 Robust non-parametric Correlation Estimates

What about if you have weird data, and want a non-parametric and more robust measure, like the wilcox test? Consider the following data set:

```
x <- runif(100)
y <- runif(100)
> cor.test(x,y)

  Pearson's product-moment correlation

data:  x and y
t = -0.646, df = 98, p-value = 0.5198
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.2582356  0.1329988
sample estimates:
        cor
-0.06512051
```

These are uncorrelated. But what if we change just one value on each:

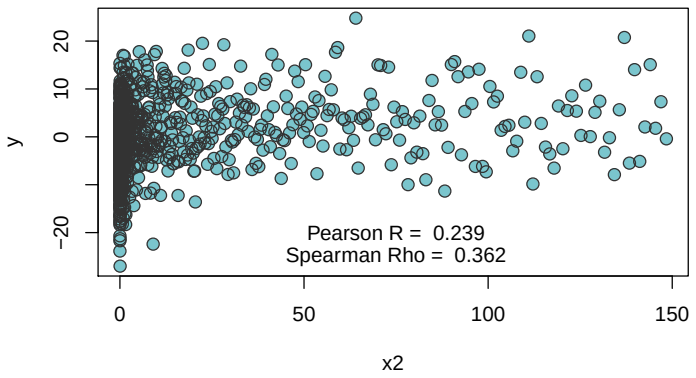
```
x[1] <- 500
y[1] <- 500
cor.test(x,y)
  Pearson's product-moment correlation

data:  x and y
t = 1135.746, df = 98, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9999434 0.9999745
sample estimates:
     cor
0.999962
```

The correlation is almost perfect! This might seem like the correlation notion is broken somehow, but remember how it is created: we divide the covariation by the variation, and in a data set like this, these two will be almost the same. This single outlier shows how sensitive correlation can be to individual points. This can cause trouble for skewed distributions as well:

```
> x<- -500:500/100
> y <- rnorm(1001)*7 +x
>
> x2 <- exp(x)
> plot(x2,y)
> cor(x,y)
[1] 0.3744526
> cor(x2,y)
[1] 0.2393363
> cor(x2,y,method="spearman")
[1] 0.361875
>
```

Figure 8.10: Here, x2 is highly skewed, and may distort the relationship between x2 and y. Now, the positive correlation is higher when using the spearman correlation coefficient



In this case, even though the original had a correlation of .37, the transformed data had a correlation considerably lower, and you might not detect it. It is somewhat troubling that the correlation can be dependent on simple transforms like this. What if we used some of the same tricks for the wilcox test–using ranks. This is known as the 'spearman' correlation, and can be given to `cor` or `cor.test` as an option. In fact, except for some decisions that may need to be made about how to handle ties, it is just the pearson correlation on the rank-order of the data:

```
cor.test(rank(x),rank(y))
 > cor.test(rank(x2),rank(y))

  Pearson's product-moment correlation

data:  rank(x2) and rank(y)
t = 12.269, df = 999, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.306792 0.414542
sample estimates:
      cor
0.361875


 cor.test(x2,y,method="spearman")

  Spearman's rank correlation rho

data:  x2 and y
S = 106670000, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.361875
```

In this case, because the x values were spaced constantly, and the y values are fairly contiguous, the Spearman's correlation is close to the Pearson's correlation. Generally, Pearson's correlation is referred to as Pearson's r, and Spearman's as ρ or rho.

8.9 Correlations among a set of variables

In R, you can compute the entire correlation matrix of a data frame using `cor`, and visualize these using `pairs`. A nice alternative is available in the `GGally` library with ggPairs.

```
data(iris)
pairs(iris)
pairs(iris,col=iris[,5],pch=16)

library(GGally)
library(ggplot)
ggpairs(iris,ggplot2::aes(colour=Species))

cor(iris[,1:4])
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

However, cor.test does not work this way. Instead, you will have to iterate over column pairs. Notice how you can iterate so that you get each pair exactly once.

```
 cor.test(iris)
Error in cor.test.default(iris) :
  argument "y" is missing, with no default

for(i in 1:3)
 {
    for (j in (i+1):4)
    {
        cat("Comparing",i,"to",j,"\n")
        print(cor.test(iris[,i],iris[,j]))

    }
 }
```

8.9.1 Bayes Factor test for correlation

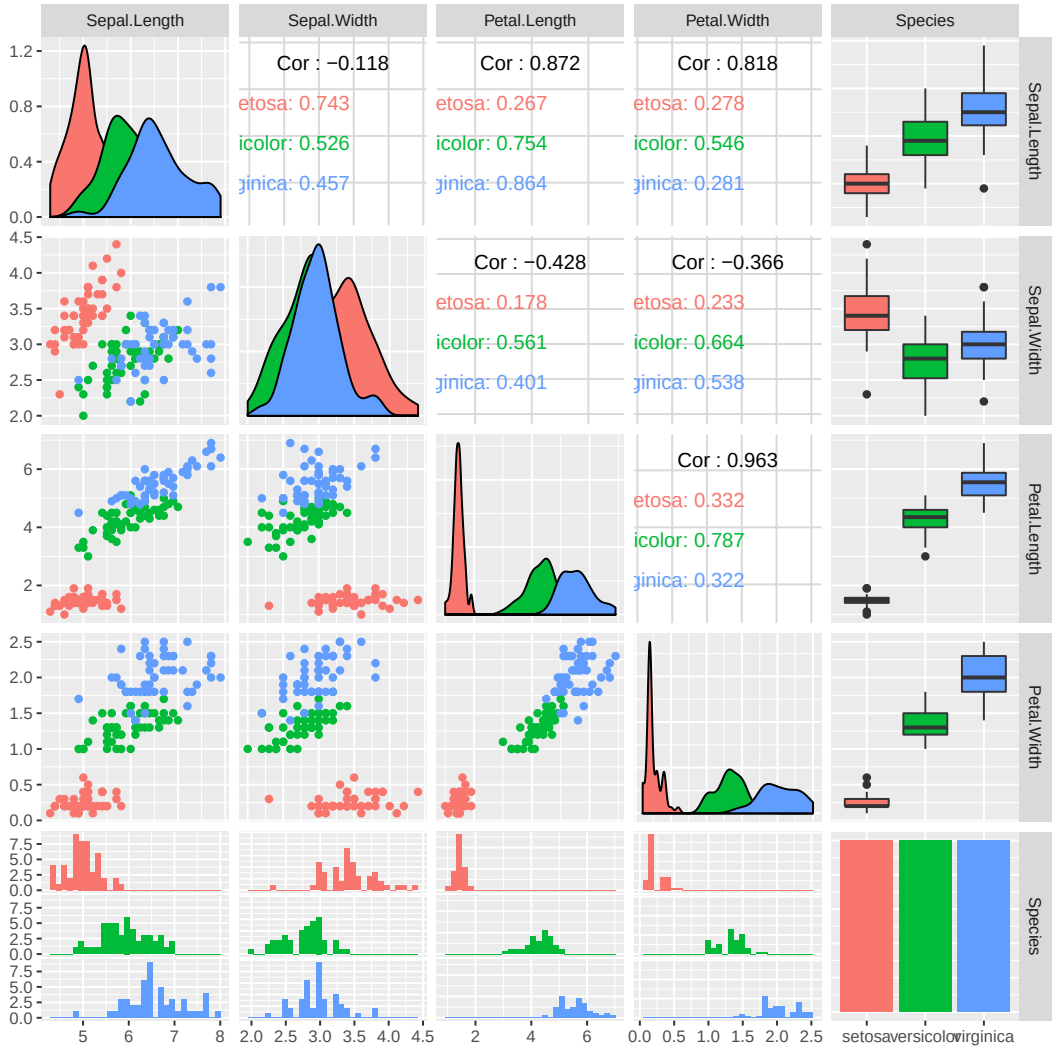
The BayesFactor library provides a BayesFactor test for correlation as well. If we compare just the first column of iris (Sepal.Length) to the others, we can see we get similar results:

```

 for (j in 2:4)
  {
    cat("Comparing",1,"to",j,"\n")
    print(correlationBF(iris[,1],iris[,j]))
  }

Comparing 1 to 2
```

Figure 8.11: Correlations among three subspecies of the iris data set. This figure is produced by ggPairs in the GGally library, but similar functionality is available with the pairs function.



```
Bayes factor analysis
--------------
[1] Alt., r=0.333 : 0.5090175 ?0%

Against denominator:
  Null, rho = 0
---
Bayes factor type: BFcorrelation, Jeffreys-beta*

Comparing 1 to 3
Bayes factor analysis
--------------
[1] Alt., r=0.333 : 2.136483e+43 ?0%

Against denominator:
  Null, rho = 0
---
Bayes factor type: BFcorrelation, Jeffreys-beta*

Comparing 1 to 4
Bayes factor analysis
--------------
[1] Alt., r=0.333 : 2.621977e+33 ?0%

Against denominator:
  Null, rho = 0
---
Bayes factor type: BFcorrelation, Jeffreys-beta*

```

8.10 Comparison of Categorical Variables

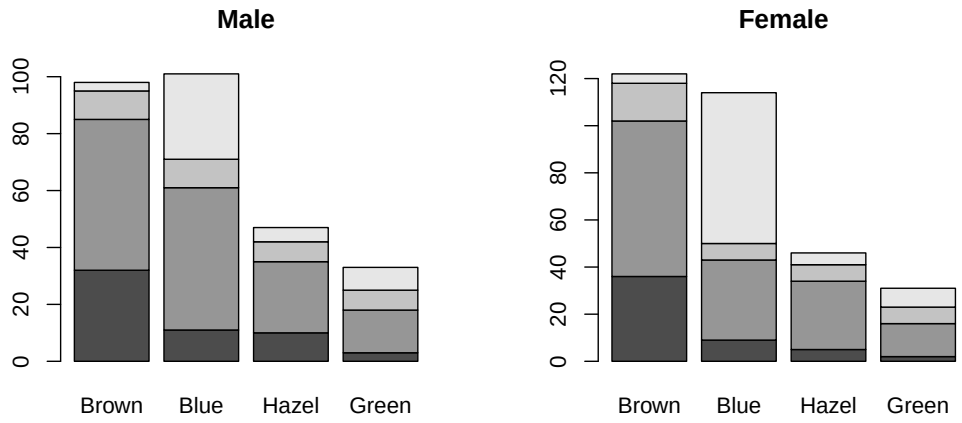
So far, we have looked at determining whether a continuous variable depends on a category (t-tests), or whether two continuous variable depend on one another (correlation). Many times you have two categorical variables and you want to determine whether one depends on another. For example, you might ask whether the type of car someone drives depends on gender, or whether hair color depends on eye color. In addition, you sometimes have continuous data that you categorize into groups (e.g., positive or negative opinion of something), and here you like to determine whether one measure depends on another. If you have a single category survey instruments create data that are categorical Responses will sometimes have an ordinal scale, but they may often be completely nominal. Consider the HairEyeColor data set:

```
print(HairEyeColor)
, , Sex = Male

       Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    53   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8

, , Sex = Female
```

Figure 8.12: Graphic showing the distribution of eye and hair color




```
       Eye
Hair     Brown Blue Hazel Green
  Black     36    9     5     2
  Brown     66   34    29    14
  Red       16    7     7     7
  Blond      4   64     5     8

par(mfrow=c(1,2))
barplot(HairEyeColor[,,1],main="Male")
barplot(HairEyeColor[,,2],main="Female")
```

We can do better:

```
barplot(t(HairEyeColor[,,1]),
          col=c("brown4","blue","khaki","darkgreen"),
          main="Male eye color")
barplot(t(HairEyeColor[,,2]),
          col=c("brown4","blue","khaki","darkgreen"),
          main="Female eye color")
```

The two genders look roughly the same (although there seem to be more blonde women than men, relatively-speaking), so maybe we can collapse over them:

```
hc2 <- t(HairEyeColor[,,1]+HairEyeColor[,,2])
par(mfrow=c(1,1))
barplot(hc2,col=c("brown4","blue","khaki","darkgreen"))
```

How can we know whether we are justified in combining gender? How can we know whether eye color is dependent on hair color? It looks like blond hair and blue eyes might tend to be related, which makes sense, but how can we test this?

Figure 8.13: Graphic showing the distribution of eye and hair color

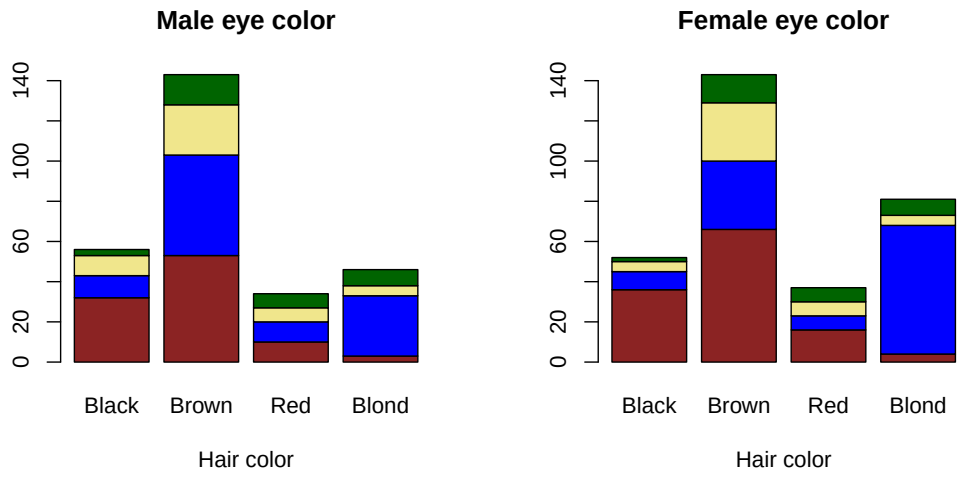
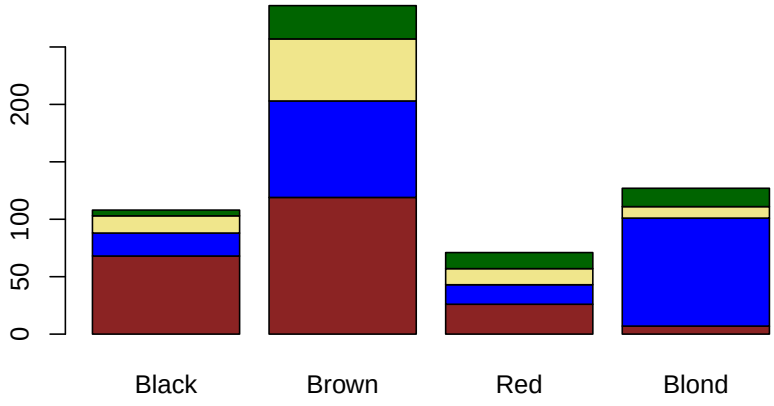


Figure 8.14: Graphic showing the distribution of eye and hair color



To simplify things, let's consider the question of whether eye color distribution differs between blond and brown-haired people. First, let's compute the expected frequencies, which is the marginal distribution of eye color for these two conditions

```
##select just the two hair colors
brownblond <- hc2[,c(2,4)]
brownblond
       Hair
Eye     Brown Blond
  Brown   119     7
  Blue     84    94
  Hazel    54    10
  Green    29    16

##Compute the marginal mean
expected.prop <- rowSums(brownblond)/sum(brownblond)
expected.prop
    Brown      Blue     Hazel     Green
0.3050847 0.4309927 0.1549637 0.1089588

expected <-  matrix(rep(colSums(brownblond),each=4),nrow=4)*
                 matrix(rep(expected.prop,2),nrow=4)

expected
          [,1]     [,2]
[1,]  87.25424 38.74576
[2,] 123.26392 54.73608
[3,]  44.31961 19.68039
[4,]  31.16223 13.83777
```

The `expected` matrix is our best estimate for what the observed distribution should be if the proportion was the same, i.e., independent. This is easiest to see of we plot it:

```
barplot(expected,beside=T,names=c("Brown","Blond"),
        col=c("brown4","blue","khaki","darkgreen"),
        ylab="Expected count",main="Expected Distribution")
```

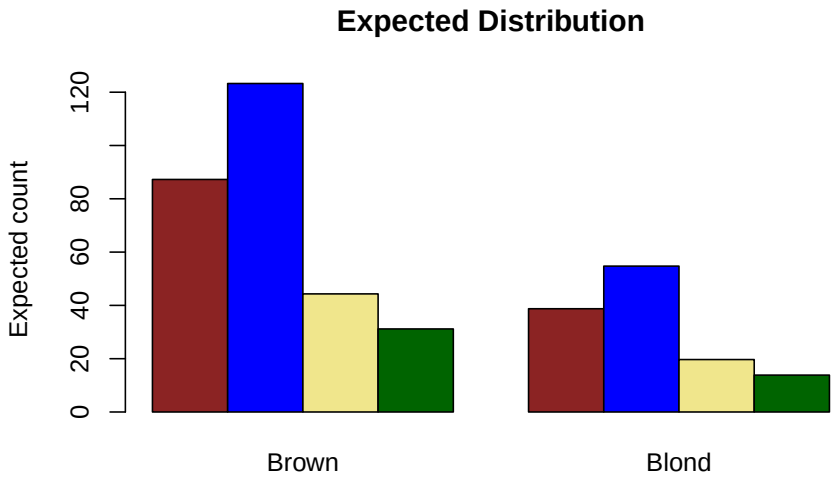
Now, we can compute the difference between this expected distribution and the observed one

```
sumdiff <- sum((brownblond - expected)^2/(expected))
[1] 85.59659
```

This statistic is referred to as the χ^2 (or Chi-squared) statistic. For data like this, samples that are truly independent are modeled with a χ^2 distribution having degrees of freedom equal to $(n-1)*(m-1)$. Thus, just as with all the other tests, we can compute the statistic for our data, and compare it to the expected distribution for the null hypothesis, and determine whether the result was likely to have happened by chance. In this case, because we have a 2×4 matrix, it has degrees of freedom $1 \times 3 = 3$. We can look this up using the `pchisq` function, or allow the function `chisq.test` to do the whole operation for us. Verify here that they do the same thing:

```
pchisq(85.5966,3)
[1] 1
>
```

Figure 8.15: Graphic showing the expected distribution of eye and hair color, if they were independent.




```
chisq.test(brownblond)

  Pearson's Chi-squared test

data:  brownblond
X-squared = 85.5966, df = 3, p-value < 2.2e-16
```

In R, you give `chisq.test` a matrix of counts, and it will compare the rows to the expected proportions based on the marginal values, and determine whether the rows and columns are independent. For example, we could do the entire data set:

```
  chisq.test(hc2)

  Pearson's Chi-squared test

data:  hc2
X-squared = 138.2898, df = 9, p-value < 2.2e-16

```

Just as with t-tests, we can also compute a Bayesian test:

```
contingencyTableBF(brownblond,sampleType = "indepMulti", fixedMargin = "cols"
   )
Bayes factor analysis
--------------
[1] Non-indep. (a=1) : 1.199192e+18

Against denominator:
```

```
7  Null, independence, a = 1
   ---
9  Bayes factor type: BFcontingencyTable, independent multinomial

11
```

Here, 1.1999e+18 is a VERY large number: 1,000,000,000,000,000,000. the a=1 value is a parameter controlling the prior hypotheses, in this case an equal probability of a independence and non-independence. Generally, this is reasonable, because you don't want your prior hypothesis to bias your results and overwhelm the observed data. But there could be cases where it makes sense; especially in more applied contexts. Suppose you have a lot of experience is survey/questionnaires, and know that two variables are typically NOT related–maybe you want to incorporate that knowledge into your test so that you don't false-alarm and end up down a rabbit hole. Finally, the test requires you to choose a fixed margin. Sometimes it is arbitrary–if one of your arguments is a quasi-experimental or experimental variable that is selected or assigned, it is reasonable to consider this fixed with the other variable dependent on it. The main impact for the Bayesian test is how it frames the underlying sampling distribution, which must be done iteratively. For example, if you were to ask each person two questions (e.g., maybe about trust and reliance) and wanted to know whether their response depended on question, you might form a table of question by response, and here question would be the fixed margin. Similarly, demographic variables that you are selecting for (gender, age, etc.) would be fixed. In cases where neither are fixed, you might have to make a reasonable decision or try both ways. For example, on a survey you might ask a preferred music genre as well as a favorite food type–since you didn't sample based on either of these (i.e., choosing 10 people from each of 3 favorite genres), neither is fixed but you still must choose for this test.

Here, both the χ^2 and Bayes factor tests agree.

You may want to compare it to a specific expected set of values specify the 'p' argument in that case. You can compare any observed distribution to any particular distribution. For example, you can test whether the the overall hair color distribution differs from some specific distribution, such as a uniform one.

```
colSums(hc2)
2 Black Brown   Red Blond
    108   286    71   127
4
  chisq.test(colSums(hc2),p=c(.25,.25,.25,.25))
6
    Chi-squared test for given probabilities
8
  data:  colSums(hc2)
10 X-squared = 182.527, df = 3, p-value < 2.2e-16
```

This indicates that the observed distribution is not from the specified (uniform) distribution, because it would have been very unlikely for the data to have been observed from it. Let's consider a few other tests we could do by specifying the null distribution:

Is the eye color distribution for black hair the same as the observed distribution of brown? Here, we use `rescale.p=T` to indicate that probability distribution should come from that argument but it isn't normalized yet.

```
chisq.test(hc2[,1],p=hc2[,2],rescale.p=T)
2
```

```
  Chi-squared test for given probabilities

data:  hc2[, 1]
X-squared = 20.8265, df = 3, p-value = 0.0001144
```

Is the eye color distribution for brown the same as the observed distribution of blonde?

```
chisq.test(hc2[,2],p=hc2[,4],rescale.p=T)

  Chi-squared test for given probabilities

data:  hc2[, 2]
X-squared = 798.4851, df = 3, p-value < 2.2e-16

>
```

Are blonde and brown the same as their average?

```
chisq.test(hc2[,c(2,4)])


  Pearson's Chi-squared test

data:  hc2[, c(2, 4)]
X-squared = 85.5966, df = 3, p-value < 2.2e-16
```

It might seem strange that these last two are different. By default, the test compares each observed to the expected produced by their average. By specifying `p`, you can test it against a particular distribution, sort of like a one-sample t-test. The most natural baseline to compare it to would be a uniform distribution across all levels–it would test whether your different cases are random. However, you may have some other values you chose for particular reasons.

8.10.1 Exercise

Suppose you counted the number of people that used each of four adjacent doors to a building, and found that from left to right, there were 131, 151, 140, and 124 users. Use a chi-squared test to determine whether people used the doors equally.

8.10.2 Technical issues

If instead of a table, you have a vector of categories (i.e, responses), you can compare that vector to another vector directly using a chi-squared test to see if the two categories are independent. This is sort of using it like you would use corr().

```
library(BayesFactor)

set.seed(100)
sample1 <- sample(LETTERS[1:6],1000,prob=c(1,1,1,1,1,.2),replace=T)
sample2 <- sample(LETTERS[1:6],1000,replace=T)
sample3 <- sample(c("yes","no","maybe"), 1000,replace=T)

```

```
##do the pairings of sample 1 and sample 2 depend on eachother?
chisq.test(sample1,sample2)

Pearson's Chi-squared test

data:  sample1 and sample2
X-squared = 25.954, df = 25, p-value = 0.4101

Warning message:
In chisq.test(sample1, sample2) :
Chi-squared approximation may be incorrect
> chisq.test(sample1,sample3)

Pearson's Chi-squared test

data:  sample1 and sample3
X-squared = 11.239, df = 10, p-value = 0.3392

>
```

Before we look at the results of the test, see that it gives a warning "Chi-squared approximation may be incorrect". This is caused because some of the cell counts are small–in this case a couple 3s and and 5 and some 6s. There is not anything you can do about this except collect more data, and you just might want to be wary about the test results because of the small counts.

Notice that this determines whether the categories of sample1 depend on the categories of sample2. This is testing the cross-tab table of these two variables, and whether the distribution of one variable depends on the distribution of the second. So, the two variables do not have to have the same levels:

```
table(sample1,sample2)
       sample2
sample1  A  B  C  D  E  F
      A 31 28 49 34 29 29
      B 38 38 26 35 26 32
      C 34 28 38 23 36 28
      D 33 34 34 36 39 21
      E 32 28 34 43 27 27
      F  6  3  7  3  6  5
> table(sample1,sample3)
       sample3
sample1 maybe no yes
      A    53 76  71
      B    67 60  68
      C    67 63  57
      D    72 70  55
      E    69 63  59
      F     8  8  14
```

Thus, this is the same as the test on the table:

```
chisq.test(table(sample1,sample2))  ##this is the same.
contingencyTableBF(table(sample1,sample2),
         sampleType = "indepMulti", fixedMargin = "cols")
```

But since sample1 and sample2 are the same outcomes, maybe we want to know whether they differ. We would need to make a 2x5 table to test whether these marginal distributions are the same. There are several ways to do this, but here is one:

```
##But are the proportions different in two independent conditions?
table <- tapply(rep(1,2000),list(c(sample1,sample2),
                  c(rep(1:2,each=1000))),length)
table[is.na(table)]<- 0
table
> table
1   2
A 200 174
B 195 159
C 187 188
D 197 174
E 191 163
F  30 142
chisq.test(table)

Pearson's Chi-squared test

data:  table
X-squared = 82.042, df = 5, p-value = 3.137e-16

> contingencyTableBF(table,
              sampleType = "indepMulti",
              fixedMargin = "cols")
Bayes factor analysis
--------------
[1] Non-indep. (a=1) : 1.005753e+14 ?0%

Against denominator:
Null, independence, a = 1
---
Bayes factor type: BFcontingencyTable, independent multinomial

>
```

Here, the p-value shows the Null hypothesis would be very unlikely to produce the data we saw, and similarly the Bayes factor show strong evidence against the null hypothesis.

8.11 Summarizing the different tests

So far, we have looked at three approaches to testing different kinds of relationships and comparisons between variables. The following table summarizes these:

8.12 Special considerations for comparing group means

There are a number of things to consider when doing the simple mean comparisons we have here. These include: What happens when you have non-normal data? What if you have different sample sizes in two groups? What if the variance differs substantially between groups? Finally, how do you compute an effect size for mean comparisons?

Table 8.1: Different comparisons and testing approaches for simple tests

Comparison	Traditional	Non-parametric	Bayes Factor
Group's mean vs. value	Student's t-test `t.test(x)`	Binomial or wilcox `binom.test(x)`	Bayes factor t-test `ttestBF(x)`
Two group means-paired	Paired t test `t.test(x,y, paired=T)`	Wilcox(on) test `wilcox.test(x,y, paired=T)`	Bayes factor t-test `ttestBF(x,y, paired=T)`

Two group means–independent	T-test or Welch's t-test
	`t.test(x,y)`
	`t.test(x∼group)`
		Mann-Whitney U
		`wilcox.test(x,y)`
		`wilcox.test(x∼group)`
			Bayes factor t-test
			`ttestBF(x,y)`

Association: continuous variables	Pearson correlation `cor(x,y)`	Spearman correlation `cor(x,y, type="spearman")`	Bayes factor correlation `correlationBF(x,y)`
Association: categorical variables	Chi-squared test `chisq.test(tab)`	Chi-squared test `chisq.test(tab)`	Bayes `contingencyTableBF(tab,...)`

8.12.1 Non-normal and skewed data

Generally, we consider a random variable to introduce noise above and beyond the contribution of the mean. So, the noise is just a nuisance, and we try to see if the central tendency is likely to differ in the population. But the mean is only the central tendency if the noise is unskewed. If it is not, then it is not as clear. Sometimes, if we have a good model of the noise, we might use a different test that estimates the parameters of the distribution itself. One example comes when trying to understand response times in human choice. Response time is highly skewed, but reasonably good models exist to predict the shape of the distributions. The estimation process can be tricky, but we might estimate parameters from a complex model, and then use the estimated parameters for each participant/condition as the data we test.

In other cases, we may just want to know that the data distribution does not violate our test's assumptions. Many ad hoc and principled transforms are used, depending on the shape of the data distribution. These include `log`, `exp`, `sqrt`, and others. For accuracy/probability data, people sometimes use log, log-odds, trigonometric functions, and and others. If we can do a transform and get rid of a long tail, this will help us conduct a test that is more reasonable. We will discuss transforms later in the context of regression models.

Overall, remember that non-normality matters most for small data sizes. As N gets large, the hypothetical distribution of the means will approach normality. Finally, a non-parametric test is always an alternative.

8.12.2 Different sample sizes between independent groups

We get the best power if the two groups have equal sample sizes, assuming you have the chance to assign to treatment groups or select among cases to include. If you have unequal sample sizes, there is generally no reason to discard data from the larger sample to make the sample size equal–independent samples tests all account for sample size differences fairly easily. Note that for independent samples tests, the standard deviation gets pooled across groups. But this can also be problematic, especially if you have very different sample sizes. For example, if you have 200 observations, but there are only 10 in one group and 190 in the second, it may be misleading, because the pooling of standard deviation will more heavily weigh the larger group, and you may be more likely to mistakenly detect a difference that does not exist. The unequal can be handled, but the small sample size can still get washed

out when the standard error is computed.

For example, suppose we have a chance of increasing the number of observations of the control group, but not the treatment group?

```
t.test(x1,y1)
t.test(c(x1,rnorm(20,.2)),y1)


> t.test(x1,y1)

Welch Two Sample t-test

data:  x1 and y1
t = -1.4248, df = 37.967, p-value = 0.1624
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-0.7956115  0.1383187
sample estimates:
mean of x mean of y
0.3078671 0.6365136

> t.test(c(x1,rnorm(20,.2)),y1)

Welch Two Sample t-test

data:  c(x1, rnorm(20, 0.2)) and y1
t = -0.78246, df = 42.855, p-value = 0.4382
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-0.5948305  0.2623037
sample estimates:
mean of x mean of y
0.4702502 0.6365136
```

Here, the t value actually got smaller–but this is not a general rule; it only happened because of randomness. But our confidence region got smaller because of additional observations–from a band of about 0.95 to one of 0.75.

8.12.3 Between versus within studies

In general, you get more statistical power from within-participant experiments. This is because you get to ignore any systematicity within each person, and focus only on the differences:

```
t.test(x1-y1)
t.test(x1,y1,paired=T)
t.test(x1,sample(y1),paired=T)  ##re-ordered so no linkage between groups
t.test(x1a,y1a,paired=T)  ##200 values
t.test(x3,y3,paired=T)  ##smaller variance
t.test(x4,y4,paired=T)  ##same number


wilcox.test(x1,y1,paired=T)  ##same number as above
wilcox.test(x1a,y1a,paired=T)  ##200 values
wilcox.test(x3,y3,paired=T)  ##smaller variance
wilcox.test(x4,y4,paired=T)  ##


ttestBF(x1,y1,paired=T)
```

```
ttestBF(x1a,y1a,paired=T)
ttestBF(x3,y3,paired=T)
ttestBF(x3,sample(y3),paired=T)
ttestBF(x4,y4,paired=T)
ttestBF(x4,sample(y4),paired=T)
ttestBF(x4,sample(y4))
```

In some of the examples above, I reshuffled the second pairing so it was no longer yoked to the first using the sample() command. Now, the results are going to be more like the unpaired test. We get a lot of additional power by doing paired t-tests, as long as the measures we are taking have some of their total variability tied to the person, not the measurement.

In general, this shows why doing a within-subject experiment can be so powerful. If every participant shows an effect, you can establish reliable estimates of the population with just a handful of subjects, and in perception research, it is typical for studies to have just a few subjects (each of whom might be studied for thousands or tens of thousands of trials).

What happens if we have greater within-subject variability? That is, we have an unreliable measure. Here, the y values is on average 1.3 greater than the x value, but y2 has normally-distributed noise having an s.d. of 5, instead of the 0.3 of y1:

```
set.seed(1001)
x <- runif(20)
y1 <- x + 1.3 + rnorm(20)*.3
y2 <- x + 1.3 + rnorm(20)*5
par(mfrow=c(1,2))
plot(x,y2)
matplot(cbind(x,y2))
delta <- y2-x
t <- mean(delta)/(sd(delta)/sqrt(20))
pt(t,19)
t.test(x,y2,paired=T)

>data:  x and y1
t = -18.387, df = 19, p-value = 1.461e-13
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-1.479736 -1.177280
sample estimates:
mean of the differences
-1.328508

> t.test(x,y2,paired=T)

Paired t-test

data:  x and y2
t = -1.6731, df = 19, p-value = 0.1107
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-5.856208  0.652935
sample estimates:
mean of the differences
-2.601637
```

With small within-subject variability, the paired t-test is highly significant; when this increases it no longer is helpful.

8.12.4 Effect sizes for t tests

Many journals now recommend reporting 'effect size' which go beyond 'statistical significance' and hope to give a guideline of how much of the variability is explained. An effect size can be a really useful statistic to help judge how important an effect is. You might have a very small difference between two groups that have a large overlap in their distributions, but if you collect enough data, you could conclude they are different. However, the effect size will help you judge that, out of all the ways that people differ in the task, your manipulation is only explaining a small amount. The most common effect size for a simple comparison is called *Cohen's d*.

In a two-sample t-test, the effect size is calculated as the difference between means divided by the pooled standard deviation. Note that to pool sd, subtract the mean from each group. There are other schemes for pooling variance of two groups, but the result should be roughly the same:

```
 (mean(x)-mean(y1)) / sd(c(x-mean(x),y1-mean(y1)))
[1] -3.436513

 (mean(x)-mean(y2)) / sd(c(x-mean(x),y2-mean(y2)))
[1] -0.5291243
```

However, we did treat x and y1/y2 as paired samples. Shouldn't the effect size be larger in this case? Here, we can simply find the relative size of the mean of the difference, in comparison to the standard deviation of the difference:

```
delta1 <- x - y1
delta2 <- x - y2

> mean(delta1)/sd(delta1)
[1] -4.111412
> mean(delta2)/sd(delta2)
[1] -0.374121
```

In this case, the effect size is larger in the case of y1; in the case of y2 it is actually a bit smaller.

Exercise 8.12.4

- Generate a set of 150 random numbers, and compare them to their sorted values with the Wilcox test. What value do you get for W? Why?
- Generate another set of 150 normal random numbers whose means differ by .2 standard deviation units, but that are unrelated to the first set. Run a standard t-test comparing these, and then a wilcox test. How do the p values differ?
- For the data you created in the previous exercise, exponentiate using the `exp` function, and plot the distributions using `hist`. Then do a wilcox test on the exponentiated values. How do this test compare to the previous test?
- For the same data, run a paired t-test, as well as a paired wilcox test, even though the data were not paired. How do these compare to the unpaired versions of the tests?

8.12.5 Effect sizes for Wilcox test

Cowan's d is not tied directly to the statistical test, and so it might be reasonable to use a standard computation of effect size on your data, even if you are not using a t test. But you might like the equivalent that is robust to the same issues the wilcox test is. Rand Wilcox has published several papers on effect sizes[1], and has produced an R library that includes some effect size measures that are probably more relevant[2].

One simple approach to reporting an effect size for the Wilcox test is to compute the average proportion of the lower group that each member of the higher group was greater than. The U or W statistic reported by wilcox.test reports the total number, and if you divide that by the total number of comparisons, this gives a value. So, if you have 20 observations in each group and a W of 300, you divide 300 by (20*20), to get a proportion of .75. The farther the value is from 0.5, the greater the difference between groups. The value of .75 corresponds closely to the Area under the ROC Curve (AUC), which is a commonly-used effect size measure in human and machine classification.

8.12.6 The *effectsize* Library

The `effectsize` library in R provides dozens of functions that help calculate effect size. It even has an all-in-one function that takes the results of any particular analysis and tries to compute the effect size. Here are three examples using t tests calculated in the previous section:

```
> effectsize(t.test(x,y1,paired=T))
Cohen's d |          95% CI
--------------------------
-4.11     | [-5.47, -2.74]


> effectsize(t.test(x,y2,paired=T))
Cohen's d |         95% CI
-------------------------
-0.37     | [-0.82, 0.08]


> effectsize(t.test(x,y2))
Cohen's d |         95% CI
-------------------------
-0.52     | [-1.16, 0.13]

- Estimated using un-pooled SD.
```

The library will produce reasonable of standard effect sizes for many tests, either by default using `effectsize` or via special-purpose functions. It includes a number of effect size calculations, including:

- For categorical variables, phi, Cramer's V, Tschuprow's T, Cohen's W, Fei, or pearson's contingency coefficient. Choice of these depends on size of table and whether the table is square.

[1] See Wilcox, R. R. (2011). Inferences about a Probabilistic Measure of Effect Size When Dealing with More Than Two Groups. *Journal of Data Science, 9*, 471-486; Wilcox, R. R. & Tian, S. (2011). Measuring effect size: a robust heteroscedastic approach for two or more groups, Journal of Applied Statistics, 38:7, 1359-1368.

[2] http://r-forge.r-project.org/projects/wrs/

- For differences in means, Cohen's d `cohens_d`
- For ANOVA/regression models, Cohen's F or η^2, ω^2.
- numerous functions to convert statistics to other effect sizes
- Functions to convert one effect size to another.

Many of the functions in effect size library are useful for meta-analyses, in which you normally convert values to effect sizes across experiments. If a researcher does not include effect sizes, you usually can convert based on the statistic, degrees of freedom, and obtained p-value.

Here is an example of an effect size for a contingency table:

```
> chisq.test(table)

  Pearson's Chi-squared test

  data:  table
  X-squared = 107.43, df = 5, p-value < 2.2e-16

  > effectsize(chisq.test(table))
  Cramer's V (adj.) |        95% CI
  --------------------------------
  0.23              | [0.18, 1.00]

> cramers_v(table)

Cramer's V (adj.) |       95% CI
--------------------------------
0.23              | [0.18, 1.00]
```

The functions sometimes will take the input of a test, and other times take the output of a test, so you need to be careful about reading the instructions to be sure you are giving it the right information.

The `effectsize` library includes many conversion functions, including converting from inferential stats (like t) to an effect size stat (like Cohen's d). This can be helpful in a meta-analysis, to allow you to calculate a common effect size across many experiments–even those that did not report them directly.

8.13 Exercise Solutions

Exercise 8.3.2 Solution

Compute a one-sample t-test for x01 x1a, x2, and x3. For each one, determine the probability you would have seen the value if the null hypothesis were true.

```
t.test(x0,alternative="greater")
t.test(x1a,alternative="greater")
t.test(x2,alternative="greater")
t.test(x3,alternative="greater")
```

Exercise 8.3.4 Solution

Compute a one-sample Bayes factor test for x0, x1a, x2, and x3. For each one, determine the probability you would have seen the value if the null hypothesis were true.

```
> ttestBF(x1a)
Bayes factor analysis
--------------
[1] Alt., r=0.707 : 1.000153 ?0%

Against denominator:
Null, mu = 0
---
Bayes factor type: BFoneSample, JZS

> ttestBF(x2)
Bayes factor analysis
--------------
[1] Alt., r=0.707 : 0.6329924 ‘0.01’

Against denominator:
Null, mu = 0
---
Bayes factor type: BFoneSample, JZS

> ttestBF(x3)  #very large
Bayes factor analysis
--------------
[1] Alt., r=0.707 : 495.6574 ‘0’

Against denominator:
Null, mu = 0
---
Bayes factor type: BFoneSample, JZS

> ttestBF(x4)
Bayes factor analysis
--------------
[1] Alt., r=0.707 : 16.72785 ‘0’

Against denominator:
Null, mu = 0
---
Bayes factor type: BFoneSample, JZS
```

Exercise 8.4.3 Solution

Compute each paired-samples test (t-test, wilcox, and Bayes factor) for the following data, in comparison to the original values (e.g., compare x1.2 versus x1):

```
y1  <- x1 + rnorm(20,mean=.25)
y1a <- x1a + rnorm(200,mean=.25)
y2  <- x2 + rnorm(20,mean=0,sd=5)  #no differences
y3  <- x3 + rnorm(20,mean=.2,sd=3)
y4  <- x4 + exp(rnorm(100))-1
```

Chapter 9

Introduction to Linear Regression

Libraries used in this chapter: rgl (if available), quantreg scatterplot3d, manipulate, plotly

9.1 Linear Regression: The Eyeball Method

Linear regression models are the basis for much of applied statistics, including factorial ANOVA models used frequently in psychology. This relationship is often hidden from the researcher, who is led to believe that linear regression is a completely different procedure, because it appears in a different menu or one has to purchase a different license to use it, but a clearer understanding can be obtained by learning the ANOVA procedure after learning regression.

At its core, linear regression is a slight elaboration to correlation. You try to explain the variability in one measured variable based on the variability in another measured variable. But instead of just measuring the strength of the relationship, we assume the relationship is linear, and use the best line we can find (described by an intercept and a slope) to make inferences and make predictions.

9.2 Least-squares fitting with one variable

We will start with a simple demonstration at the heart, using the `manipulate` library, which is already part of RStudio. This will allow us to make little controllers for our data. Here is a sample randomly generated data set, where y depends on x in a linear fashion, but with some extra noise thrown in.

```
set.seed(500)
x <- runif(100)*10
y <- x*5 + 15 + rnorm(100)*10
```

We'd like to infer a line to represent the data. A line is represented mathematically using a slope and a height. The slope indicates how many units of the y value change for each unit of the x value. If the slope is set, the height of the line is most intuitively measured at the center-point of the data. However, because of mathematical simplicity and tradition,

we usually represent the height by the point at which the line intercepts the y-axis–where x=0. This is thus referred to as the intercept of the line, and is generally not interpretable directly, unless the value of the predictor x=0 is really meaningful.

In this example, we'd like to find the parameters of a linear model that represent the data–ideally a model with intercept=15, slope=5, and variance=10, matching the process that generated the data. To begin, I'll create a little function that will plot the data and let us choose the slope and intercept by hand, allowing us to find the best fitting line by hand. Try a few examples with different parameters close to the slope and intercept of the data.

```
plotme <- function(x,y,a=0,b=1)
{
   plot(x,y,ylim=c(-20,100),col="gold",pch=16,
        main=paste("intercept:",a,"slope:",b))
   points(x,y)
     abline(a,b)

   pred <- a + b*x
   points(x,pred,pch=1,cex=.5)

   rmse <- sqrt(mean((y-pred)^2))
   text(4,-10,paste("RMSE=",round(rmse,3)))

   }

plotme(x,y,10,7)
```

Some examples are shown in Figure9.1. You can play with plotme to find values that seem to work. In the figure shown here, we see that, at least in terms of deviation from the original, an incorrect model fits slightly better than the correct one.

The manipulate function let's you create a slider widget that makes this easier.

```
library(manipulate)
manipulate(plotme(x,y,a,b),
              a=slider(0,100,step=.1),
              b=slider(0,20,step=.1))
```

When considering a linear model as a model of the world, it is often written as follows:

$$\hat{y} = \beta_0 + \beta_1 \hat{x} + \epsilon \tag{9.1}$$

Here, $\hat{y}$ and $\hat{x}$ represent the entire set of observed x and y values, and you might refer to pairs of them individually as x_i and y_i. β represent fixed parameter values, and in this model there are two: β_0 is known as the intercept term, and β_1 the slope. These correspond to the a and b parameters in the plotme function. Finally, ϵ represents a random variable, typically arising from a normal random variable, so $\epsilon \propto N(0, \sigma^2)$. We assume that any deviation from the linear model comes from this noise source, and that the noise is not dependent on values of x, and that the noise is independent.

Just as when we learned about estimating parameters of a normal distribution in Chapter 7, we need to come up with an approach to deciding what the parameters should be, including the the variance of the normal distribution describing the deviations from the model. This error model is important because it is used for making inferences about the population parameters.

Figure 9.1: Four example plots created by plotme, with slope and intercept parameters chosen by hand

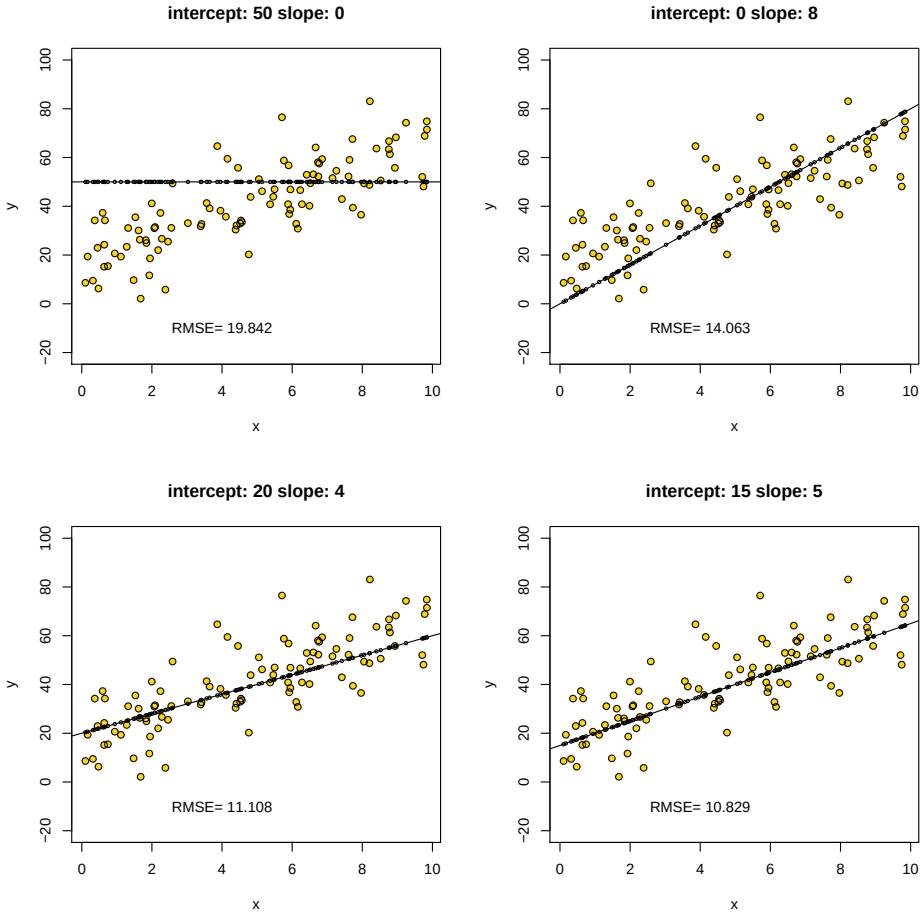
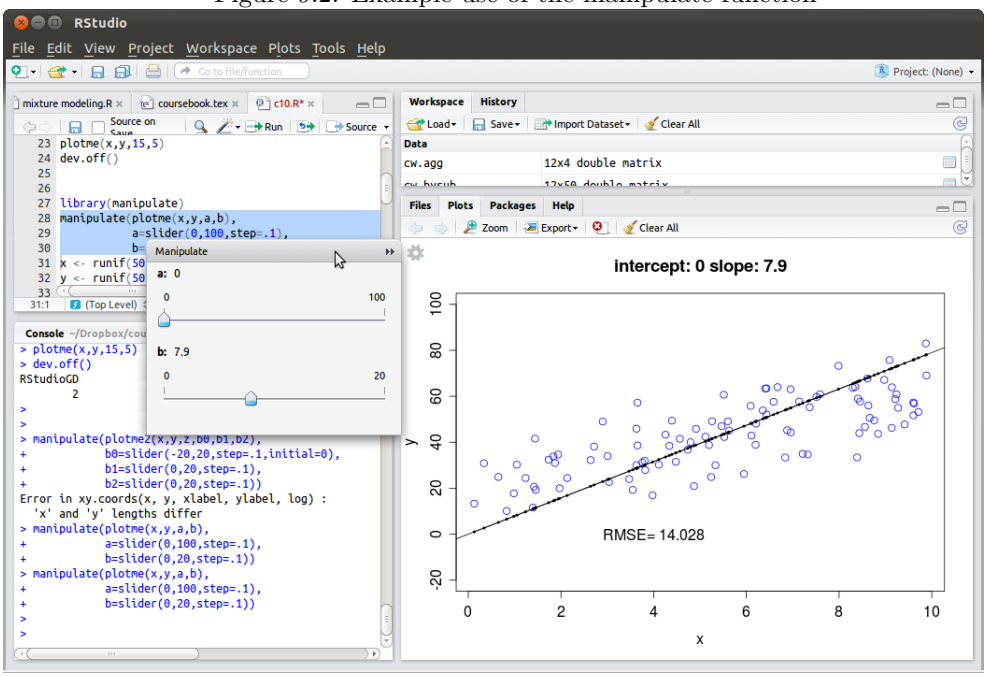


Figure 9.2: Example use of the manipulate function



There are several approaches commonly used for doing this estimation, and others used less often. The most traditional approach is to find a line that minimizes a least-squares criterion–the smallest root-mean square error, and then taking the residuals and estimating the variance using the methods described in Chapter 7. An alternative approach is to estimate the values based on maximum likelihood–these produce the same estimates for β, and a similar estimate for σ. There are some non-parametric approaches that use other criteria, like attempting to fit the quantile (available in the `quantreg` library in R). Another approach is based on Siegel's (1982) method, and available in the `mblm` library in R, but this approach only permits single linear predictors. Finally, Bayesian approaches use a likelihood-based approach, and estimate a distribution of probable parameter estimates that might explain the observed data. The `BayesFactor` package includes `lmBF` and `regressionBF` that use the standard linear model for estimating β values, but uses a Bayes Factor criterion for testing between models, which we will discuss in the next chapter.

Exercise 9.2

Exercise 1 Improve the plotme function to make it better, in one of the following ways, or in a way you choose yourself.

- compute expected values of the line
- coloring/shading of points based on under/over
- Compute R^2 or other statistic
- Count how many points are over/under

Exercise 2 Create a new set of x and y values and try to use the plotme/manipulate to find the best-fitting line.

So, we'd like to be able to estimate these values–especially the β weights in a way that doesn't require eyeballing. It turns out that there are computationally-efficient ways of doing this, which involve some matrix algebra and matrix inversion, which can be solved numerically. We can't just estimate slopes by computing means like we estimated the parameters of the normal random variable, but we can still estimate them with a closed-form formula. Without worrying about the details too much, this set of formulas work for the intercept+slope model:

```
n <- 100
beta1 <- (sum(x*y) - sum(x) * sum(y)/n)  /(sum(x^2) - sum(x)^2/n)
beta2 <- mean(y) - beta1 * mean(x)
> beta1
[1] 5.147532
> beta2
[1] 14.05491
```

The above formulas are shortcuts for a slope-intercept model. In reality they are special cases for a general formula involving transforming x and y values. The linear model can be written more generally as follows:

$$\vec{X}\beta = \vec{Y} \tag{9.2}$$

This just means the best guess for each Y value will be found by a weighted sum of the corresponding X and β values. In this case, X has two values per Y; the observed X and a stand-in of 1–a constant that lets us estimate the intercept. We want to find the β values that minimize sum squared error, which is:

$$SSE = \sum_i (Y_i - \vec{X}_i\beta)^2 \tag{9.3}$$

Without doing the calculus, this is minimized when

$$(\vec{X}'\vec{X})\hat{\beta} - \vec{X}Y = 0 \tag{9.4}$$

and β values can be obtained after some linear algebra:

$$\hat{\beta} = (\vec{X}'\vec{X})^{-1}\vec{X}'Y \tag{9.5}$$

The above formulas split this up for a two-beta model, and the following code computes it for two predictors (1 and x), but could be extended to any number of predictors:

```
xs <- cbind(1,x)
solve(t(xs) %*% xs) %*% t(xs) %*% y
       [,1]
  14.054913
x  5.147532
```

The insight this should give you is that the β coefficients require a mathematical function of data to estimate them–just as with the normal distribution, where we used the mean of the data to estimate the mean of the distribution. However, now it is no longer as straightforward. The estimation here is exact though–there is no need for searching through parameter values to find the best set.

The `lm` function in R will fit this automatically and report the results in an easier to view format:

```
model <- lm(y~x)
> model

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     14.055        5.148
```

We can use the coefficients of the lm to draw the best-fit line on a scatterplot in Figure 9.3

```
plot(x,y,pch=16,cex=1.5,col="gold",
     main=paste("Best-fitting line\n",
                "y = ",round(model$coef[1] ,2) ," + ", round(model$coef[2],3) , " * x",sep=""))

points(x,y,pch=1,cex=1.5,col="grey20")
abline(model$coef,lwd=2)
```

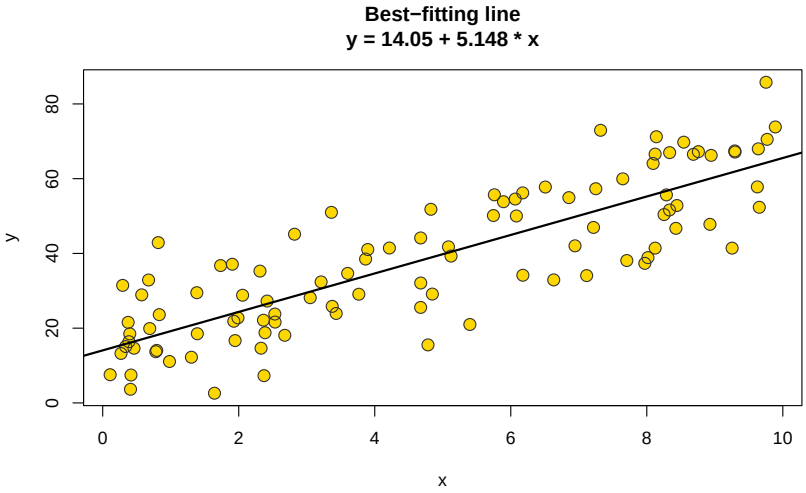
Because we have a best-fit model, we can use it for prediction and identifying the best-fit values for the observed data.

For the latter, the best-fit values are stored in `model$fitted.values`. Or we could use the coefficients to make these by hand:

```
#we can use the model for prediction:
predictedy <- model$coef[1] + model$coef[2]* x
plot(y,predictedy,cex=.5,col="grey20",pch=16)
abline(0,1)
cor(y,predictedy)^2
[1] 0.7005609


cor(predictedy,model$fitted.values)
[1] 1
cor(predictedy,predict(model,newdata=data.frame(x=x)))
[1] 1
```

Figure 9.3: The relationship between x and y, and the line describing that relationship



Here, if we square our correlation between y and predicted, it gives us a goodness-of-fit statistic that tells us how much of the variability in the outcome we have accounted for. In this case, it is .7, or about 70%. At the end, you can see that there are at least two other ways to get these predicted values.

We can use `predict` to get predictions for x values we did not see. Here, we need to give it a data frame of data values having the same names as our predictor variables. In this case, there was only one (x).

```
 predict(model,newdata= data.frame(x=c(-100,0,10,30,5000)))
2          1           2           3           4           5
 -500.69829    14.05491    65.53023   168.48087 25751.71500
```

Notice that when making these predictions, they are almost all outside the range of data we actually observed. We are extrapolating here, and the model has no way of judging whether this is a good idea.

9.2.1 Estimating a slope-only model

There are many cases where we are confident that there should be no intercept, or we want to interpret the relationship on the assumption that the intercept is 0. For example, you might want a way to convert prices from the 1950s to prices in prices for the same products in the 1990s. You know that cost 0 should be the same in both cases. Taking real prices that are published on the internet for five grocery products, we see that the best-fit line has a slope of 2.07 and an intercept of .54.

```
#### slope-only
2 costs <- data.frame(
    product=c("apples", "bananas", "eggs", "steak", "potatoes"),
4   cost1950=c(0.39,0.14,0.79,0.59,0.07),
    cost1990=c(1.98,   0.48,    1.05, 2.99,  0.31)
6 )
```

```
> lm(cost1990~cost1950,data=costs)

Call:
lm(formula = cost1990 ~ cost1950, data = costs)

Coefficients:
(Intercept)     cost1950
     0.5391       2.0781
```

This means that if you wanted to predict the cost of another product, you would take the 1950 price, multiply by 2.078, and add \$0.54. But since we expect this to be a ratio, we'd like to get the slope if the line passed through 0.

```
lm(cost1990~cost1950+0,data=costs)

Call:
lm(formula = cost1990 ~ cost1950 + 0, data = costs)

Coefficients:
cost1950
   3.007

##Plot the relationship here:
plot(costs$cost1950,costs$cost1990,xlim=c(0,1),cex=.3,ylim=c(0,4))
text(costs$cost1950,costs$cost1990,costs$product)
abline(lm(cost1990~cost1950,data=costs)$coef, col="red")
abline(0,lm(cost1990~cost1950+0,data=costs)$coef,col="black")
```

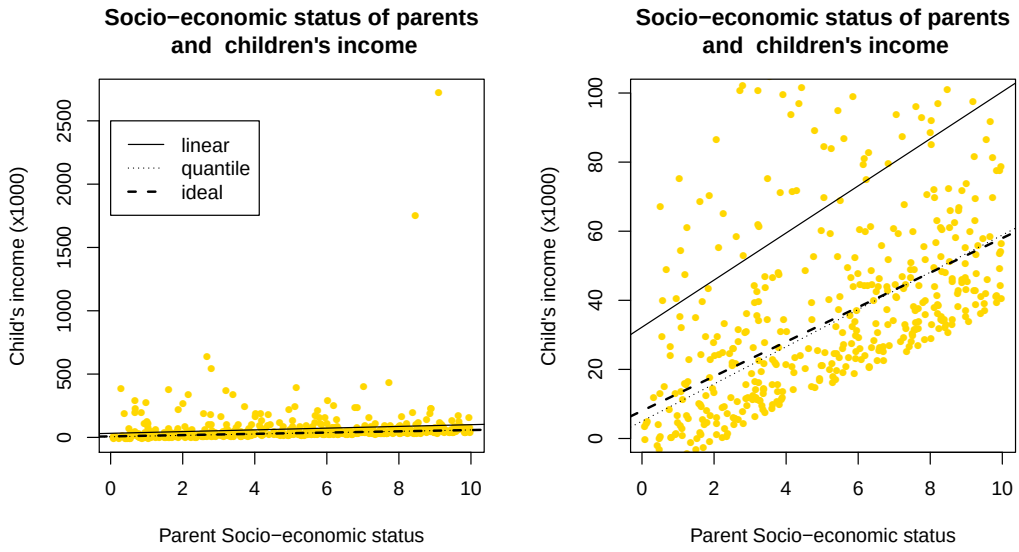
Here, the ratio is just about exactly 3. The slope is substantially different, and this might really be a better estimate, because costs scale with quantity, and the quantity is somewhat arbitrary.

9.3 Estimating parameters with quantile regression

Most estimation and inference methods for regression produce the same estimates as least-squares. But if you have a few outliers or something that is skewed, you can explicitly fit the median values (or any other quantile) using the quantreg library.

In the example below (with completely fake data), suppose we want to model the relationship between socio-economic status of the parents (on a 1 to 10 scale) and the income of the children. As we can see in the left panel of Figure 9.4, income is highly skewed. Any linear model fit using least-squares criteria will be highly influenced by outliers. Because we created the data, we can estimate that the median value of the noise distribution is about 20. That is, although it is skewed, the best discription of the central tendency is about 20. A linear model will be highly influenced by the skewness, but a good model might try to fit a line thruogh the median, but with a slope of 5/ses unit.The thick dashed line in each figure shows this ideal relationship, generated from the true values we created the data with.

Figure 9.4:



On the right panel, we can see that the solid line of the linear model over-estimates this midpoint, and also misses the slope. In contrast, the quantile regression hits the ideal best line on the nose.

```
library(quantreg)

set.seed(3)
n <- 500
ses <- runif(n)*10
##one-predictor model
y <-  -12 +  5*ses + (exp(rnorm(n)*1.4 -.25))* 25

mod1.lm<-lm(y~ses)$coef
mod1.rq <- rq(y~ses)$coef

par(mfrow=c(1,2))
plot(ses,y,col="gold",pch=16,cex=.8,
     main="Socio-economic status of parents\n and  children's income",
     xlab="Parent Socio-economic status",ylab="Child's income (x1000)")
abline(mod1.lm,lty=1)
abline(mod1.rq,lty=3)
abline(-12+20,5,lwd=2,lty=2)
legend(0,2500,c("linear","quantile","ideal"),lty=c(1,3,2),lwd=c(1,1,2))

plot(ses,y,col="gold",pch=16,cex=.8,
     main="Socio-economic status of parents\n and  children's income",
     xlab="Parent Socio-economic status",ylab="Child's income (x1000)",
     ylim=c(0,100))

abline(mod1.lm,lty=1)
abline(mod1.rq,lty=3)
abline(-12+20,5,lwd=2,lty=2)
```

9.4 Least-squares fitting with two variables

Suppose you have a DV that is related to two IVs? Now, a simple line no longer can describe the relationship. We could visualize this in a 3-dimensional space though, with the two predictors along the two horizontal dimensions, and the dv on the third dimension.

A corresponding relationship, and improved plotme function, are:

```
set.seed(1)
x <- runif(50)*10
y <- runif(50)*10
z <- 5*x +  15*y  -12 + rnorm(50)*10

plotme2 <- function(x,y,z,b0,b1,b2)
{

  pred <- b0 + b1 * x + b2 * y
  cols <- c(rgb(.9,.2,.2,.7), rgb(.2,.8,.4,.7))[1+(z<pred)]

  par(mfrow=c(1,3))
   plot(x,z,pch=16)
   points(x,pred,cex=1.4,col=cols,pch=16)
   title(main=paste(round(b0,3), "+",round(b1,3), "*x +" ,round(b2,3), "*y"))

  ## draw segments between each given and preidcted value,
  ## with a color that tells up whether it is an under or
  ## over-estimate


  segments(x,z,x,pred,lty=1,col=cols)
  abline(b0+b2*5,b1)

   plot(y,z,pch=16)
   points(y,pred,pch=16,cex=1.4,col=cols)
   abline(b0+b1*5,b2)
   segments(y,z,y,pred,lty=3,col=cols)

  rmse <- mean(sum((z-pred)^2))

  plot(z,pred,main=paste(round(rmse),round(cor(z,pred)^2,3)),
       xlim=range(z),ylim=range(z),col=cols,pch=16)

  abline(0,1)
}
```

Now, use the new plotme, adjust the three new values to find a best-fitting prediction line.

```
manipulate(plotme2(x,y,z,b0,b1,b2),
           b0=slider(-20,20,step=.1,initial=0),
           b1=slider(0,20,step=.1),
           b2=slider(0,20,step=.1))
```

Finding the best prediction is more difficult when you have multiple predictor variables. For a certain class of relationships between variables, we can compute the parameters directly from the data. Other sources provide the mathematical basis for this. For other models, we can't estimate these parameters directly as a function of the data, but we can usually devise a search method that systematically explores the parameter space in order to find a best-fitting model.

The formulas we used before won't quite work, because we have two different slopes now, plus an intercept. So really we have three predictors in x: a set of 1s (indicating the intercept term) and a set of the observed x values and a set of y values. Using the same formula we did before, we can compute the least-squares coefficients like this (now z is the predicted variable).

```
xs <- cbind(1,x,y)
betas <- solve(t(xs) %*% xs) %*% t(xs) %*% z
betas
         [,1]
  -16.477098
x   5.854821
y  15.218176
```

The process of fitting a linear model to sets of data such is this is known as linear regression, and one can do it in R using the `lm` function.

```
model1 <- lm(z ~ x+y)
print( model1)

Call:
lm(formula = z ~ x + y)

Coefficients:
(Intercept)            x            y
    -16.477        5.855       15.218
```

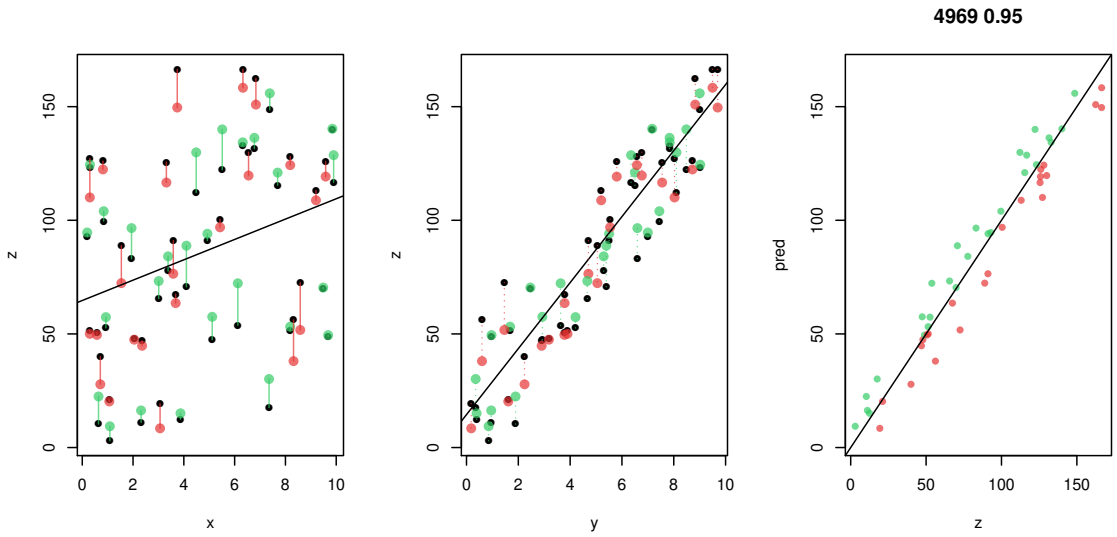
The lm function finds the best-fitting line that minimizes the squared deviations from the model, using the exact same method we did above. Again, you specify the parts of the model using a formula, with the DV on the left side of the ~ and the set of IVs on the right side. The `lm` function will produce an object you can save to a variable name and later get information about or extract different values from.

The lm function computes statistics on the data that estimate a set of linear slopes, typically called coefficients or 'beta' weights, where $beta_0$=intercept, $beta_1$ = first slope $beta_2$ = second slope, etc. Along with `model$coef`, These can be obtained using `coef(model)`. Each slope is associated with a different predictor variable you provide. We can now provide the plotme function with the best-fitting coefficients, as shown in Figure 9.5

```
betas <- lm(z ~ x+y)$coef
plotme2(x,y,z,betas[1],betas[2],betas[3])
```

For linear combinations of variables (e.g., $b_1x_1 + b_2x_2 + \ldots$), and assuming a linear relationship between each predictor and outcome, the lm() function tries to infer what the relationships are. Let's try one with no noise:

Figure 9.5: Output of the plotme function, with the three coefficients estimated from the linear regression model.




```
x <- 1:100
y <- x*3 + 55

plot(x,y,main="Data and predicted model with no noise",col="cadetblue3",pch
    =16)
g1 <- lm(y~x)
abline(g1$coef)
print(g1)
print(g1$coef)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
         55            3
```

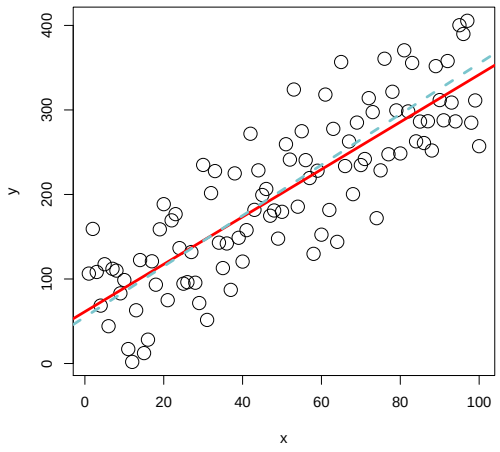
With no noise, we can estimate the original relationship with exact precision. But there is usually noise in the measurement. So, let's try one with noise:

```
x <- 1:100
y <- x *3 + 55 + rnorm(100)*50

g2 <- lm(y~x)
plot(x,y,main="Data and predicted model with some noise",col="cadetblue3",pch
    =16)
abline(g2$coef)
print(g2)


Call:
lm(formula = y ~ x)
```

Figure 9.6: Example best-fit line (solid red), along with a line describing the actual linear model that generated the data (dashed).



.

```
12
Coefficients:
14 (Intercept)            x
     47.402        3.106
```

Here, the parameters we estimated are somewhat off–an intercept of 47.4 instead of 55, and a slope of 3.1 instead of 3. But that model fit the data better than the one that actually generated it. We can be confident of this because the least-squares model is guaranteed to fit the model the best of all linear models.

We can get the coefficients directly, and plot the line inferred using `abline`, which is shown in Figure 9.6

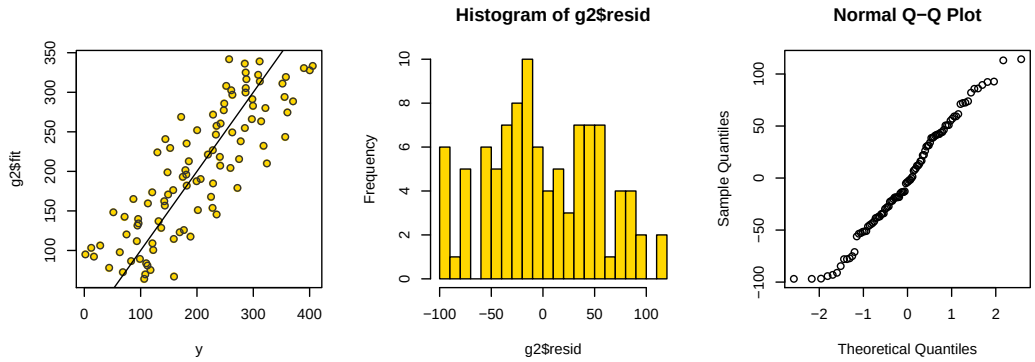
```
plot(x,y,cex=2)
2 g2 <- lm(y~x)
print(g2)
4 g2$coeff      ##<<- get coefficients directly

6 abline(g2$coeff,lwd=3,col="red")
abline(55,3,col="blue",lty=2,lwd=3)
```

We can also look at the observed versus predicted values, shown on the left panel of Figure 9.7, and a normal qqplot of the residual (right panel).

```
1 plot(x,y,cex=2,col="gold",pch=16)
points(x,y,cex=2,col="grey20")
3 abline(g2$coeff,lwd=3,col="red")      ##the best model
abline(55,3,col="grey20",lty=2,lwd=3) ##the true model
```

Figure 9.7: Plots showing the fit of our regression model. Left panel shows the actual versus fitted values. Center panel shows histogram of residuals. Right panel shows normal q-q plot.



The `residuals`, equivalent to `(y-g2$fit)`, can be accessed via `model$residuals`. The residuals are one of the first things you should look at to assess whether a linear model is appropriate, and to determine whether the model is fitting well. Also, if you want to do inferential statistics, the residuals are usually assumed to be normally distributed, and so this will help test whether the assumptions of a statistical test are met. The histogram and qqnorm plots are shown in the center and right panels of Figure 9.7.

```
par(mfrow=c(1,3))
plot(y,g2$fit,col="gold",pch=16,cex=1.2)
points(y,g2$fit,col="grey20")
abline(0,1)
hist(g2$resid,breaks=20,col="gold")
qqnorm(sort(g2$resid))
```

Exercise 9.2

Exercise: A problematic linear model
Consider the following two variables, with a known relationship between them:

```
x <- 1:100
y <- -.4*x - 150 - log(runif(500)*x*.3)*10
```

1. Look at the relationship visually
2. Compute the linear model for y versus x
3. Determine you best estimate for y based only on x
4. Plot actual versus predicted.
5. Examine the residuals.

When you fit the model, how close is it? What do the residuals look like?

9.5 Examining Models with multiple predictors

Here is an example with multiple predictors, which we will call `x` and `y`, and `z`, where `z` depends on `x` and `y` with some noise.

```
x <- runif(100)*5
y <- 1:100
z <- 100 + 3*x -y*.2 + rnorm(100)*5
```

We can fit a regression model easily by including both x and y as predictors in the formula:

```
g3 <- lm(z~x+y)
g3
> g3

Call:
lm(formula = z ~ x + y)

Coefficients:
(Intercept)            x            y
   100.3945       2.9313      -0.2031
```

For the most part, the regressions work the same, but have multiple predictors. To make a new prediction, it becomes more difficult to use the raw coefficients. The prediction of z in this case becomes 100.39 plus 2.93 times x plus -.203 times y. That would not be too difficult, but if we had ten predictors, it would get more challenging. We can use predict() to do this, giving it a data frame with all the values:

```
> predict(g3,data.frame(x=10,y=-15))
       1
132.7537
```

An number of additional predictors can be added using this syntax (as long as we don't have more predictors than we have complexity in our data). But this gets harder to visualize individual effects. One approach is to look at each predictor variable one at a time.

```
par(mfrow=c(1,2))
plot(x,z)
plot(y,z)
```

This can be useful, but another approach is to use a library that allows us to look at things in 3D. Here are 9 panes of a 3d scatterplot, each from a different angle between 0 and 90 (see Figure 9.8:

```
library(scatterplot3d)
for(angle in (0:8)*90/8)
{
  s3d <-scatterplot3d(x,y,z ,pch=16, highlight.3d=TRUE,
                      angle=angle,
                      type="h", main="3D Scatterplot")
 s3d$plane3d(g3,lty.box="solid")

}
```

For two predictors, the best fitting model is no longer a line; it is a plane. For more predictors, the best fitting model becomes a hyperplane in a high dimensional space. If you add more complexities (such as interactions, power terms, etc.) the model might become some curved surface within a higher-dimensional space.

Another library let's you rotate 3-D data directly:

```
library(rgl)
plot3d(z,x,y,col="red", size=3)
```

Finally, a third (more modern) library that supports a 3d scatterplot is plotly. Its syntax is unlike what we have used so far, but it creates nice web-enabled visualizations that can be interacted with. A screenshot of the visualization is shown in Figure 9.9.

```

library(plotly)
##This is hard to visualize

data <- data.frame(x,y,z)
plot_ly(data, x = ~x, y = ~y, z = ~z) %>% add_markers()
```

For more than one predictor, you can visualize observe versus predicted, or predicted vs. single variable, possibly including predicted. (see plotme2 example above)

9.6 Solutions to Exercises

Exercise 9.2 Solution

```
plot(x,y)
lm2 <- lm(y~x)
abline(lm2$coef)

pred <- lm2$coef[1] + lm2$coef[2]*x
pred2 <- lm2$fit
plot(pred,pred2)

plot(y,pred)
abline(0,1)

hist(lm2$resid,breaks=20)
qqnorm(lm2$resid)
```

Figure 9.8: Nine panels visualizing the data at different angles, using the scatterplot3d function

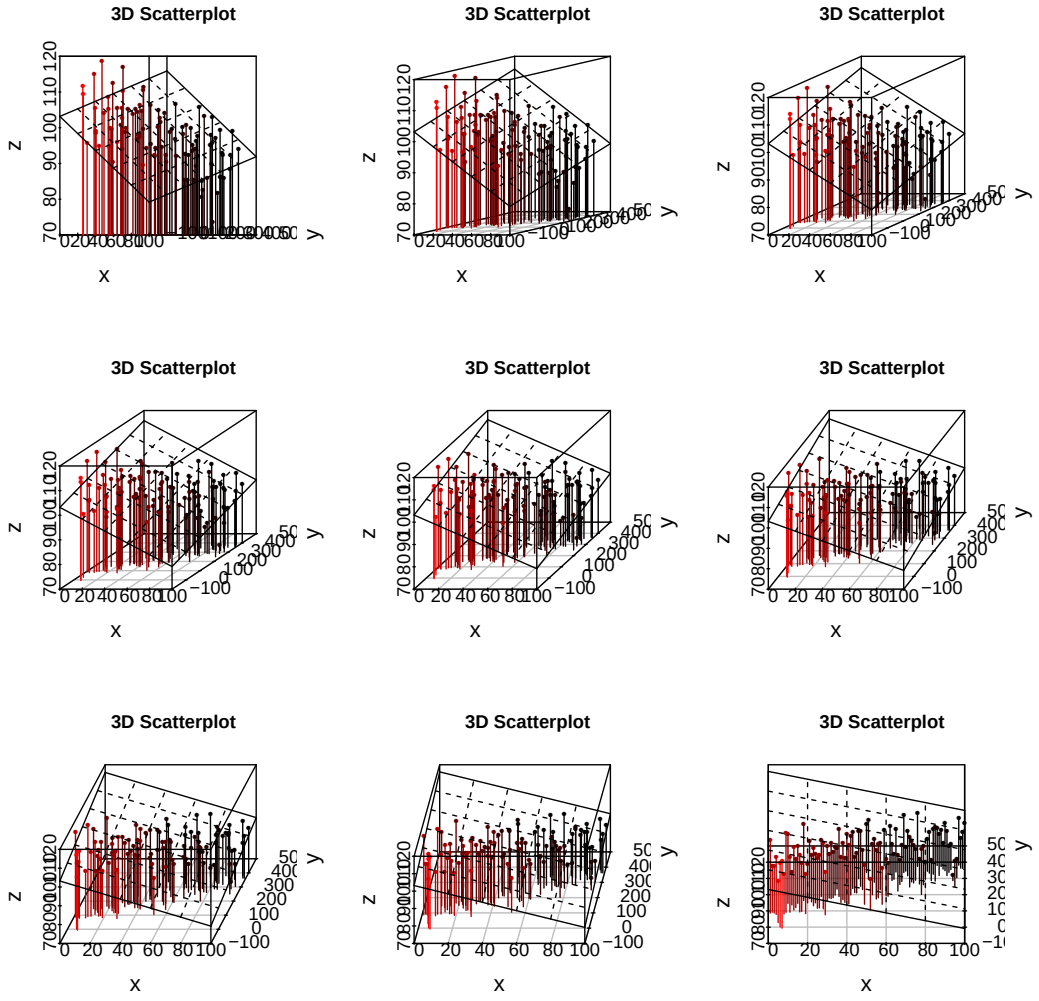
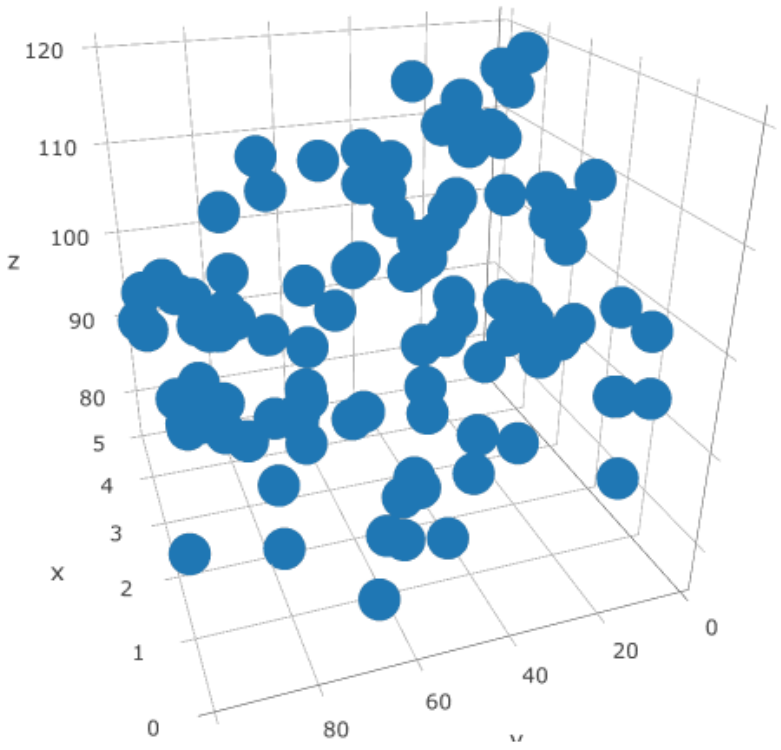


Figure 9.9: A 3D scatterplot created using the plotly package, that produces interactive web-enabled graphics.



.

Chapter 10

Testing the Linear Model

Libraries used in this chapter: quantreg, faraway

10.1 Estimating the variability of the linear regression model

In the previous chapter, we looked at how we can use a linear regression model for estimating how a combination of predictor variables can reproduce an outcome variable. The relative change of each variable per unit of the outcome is called a beta weight or coefficient, and can be interpreted as a slope in a geometric space.

This process is the equivalent to estimating the mean of a normal distribution when comparing the means of two groups. But to conduct an inferential test, such as the t-test, we need to look at the variability around the mean. We have the same situation in regression modeling. In the t-test, the standard deviation is used, which is really the residual variation that cannot be accounted for by the mean-only model. Now, just like in the t-test, we will look at the residual variation that cannot be accounted for by the model, but we can expand the model to include additional predictors.

If you consider a linear model with only an intercept prediction, this is identical to a one-sample t-test. Similarly, if you have a single predictor with two levels, this is identical to a two-sample t-test.

10.1.1 Analogy to simpler tests

Just like the earlier hypothesis tests, we can either take the approach of a null-hypothesis test, or a Bayes Factor test. In fact, suppose we had a single predictor that had exactly two levels, and we want to test whether the predictor impacts the dependent variable. This is sort of like doing a standard t-test:

```
set.seed(20)
predictor <- c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
               1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)

outcome <- predictor + rnorm(40)
aggregate(outcome,list(predictor),mean)


> aggregate(outcome,list(predictor),mean)
  Group.1          x
```

```
1       0 -0.1877639
2       1  0.8281528

model <- lm(outcome~predictor)
model

Call:
lm(formula = outcome ~ predictor)

Coefficients:
(Intercept)    predictor
    -0.1878       1.0159

plot(predictor,outcome,col="cadetblue",pch=16,
     main="Fitting difference in mean\nwith linear regression")
abline(lm(outcome~predictor)$coef)
```

We can see here that the means of predictor value 0 and 1 are -0.1878 and .82. The lm model handles this a bit differently. In this case, the intercept is the same as value 0: -.1878. But the slope now becomes the difference between .82 and -.1878: 1.0159. Of course, the t test wants to know if the means of these two groups differ, and the slope computes that same difference. For a null-hypothesis test of whether the slope differs, we just have to ask whether a slope of 1.0159 could have happened by chance if the null hypothesis were true. For the t test, this probability is .0058, as we can see below. We can get an equivalent statistical test of the regression model by using `summary`:

```
t.test(outcome~predictor)

  Welch Two Sample t-test

data:  outcome by predictor
t = -2.9235, df = 37.919, p-value = 0.005809
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.7194373 -0.3123962
sample estimates:
mean in group 0 mean in group 1
     -0.1877639       0.8281528

> summary(model)

Call:
lm(formula = outcome ~ predictor)

Residuals:
     Min       1Q   Median       3Q      Max
-2.70195 -0.50156 -0.00853  0.63584  1.97323

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.1878     0.2457  -0.764   0.4495
predictor     1.0159     0.3475   2.924   0.0058 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 1.099 on 38 degrees of freedom
Multiple R-squared:  0.1836,  Adjusted R-squared:  0.1621
F-statistic: 8.547 on 1 and 38 DF,  p-value: 0.005804
```

Now, each predictor has a null hypothesis test, which are all actually one-sample t-tests–comparing whether the value is different from 0. Here, the intercept test has a low t value and a high p, and so we cannot conclude that the value of is different from 0. But the t value for the predictor (the slope) is identical to the t test for the difference: 2.9.

The summary function provides other important information about the regression model, including the R^2 values. In addition, it computes an `F` test at the bottom. This tests whether the entire model is better than a model with a single mean (the intercept-only model). In other words, is the model with a slope better than the model without a slope. Although this is an F test, the p value here is identical to the p value for our slope alone! This is not a coincidence, because in this model the slope is the only difference between the intercept-only model and the full model. Mathematically, the t and F distributions are closely linked, so that they can produce the same p-values.

The t-test of the parameter estimate in the regression is based on the variability of the residuals. This variability is reported in the model under "residual standard error", which is 1.099. The variability of each estimate is based on this, in relationship to how many of the observed points helped make the estimate. For the intercept, the std. error is $RSE/\sqrt{20}$, and for the slope, the std. error is $RSE/\sqrt{10}$, so you can see that there is some relationship between these. The standard error of each term is calculated based on an estimated covariance matrix of the parameter estimates, and the RSE is based on the residual error once all predictors are accounted for, so you can see there is a close relationship between these.

10.2 Inferential statistics about parameter estimates

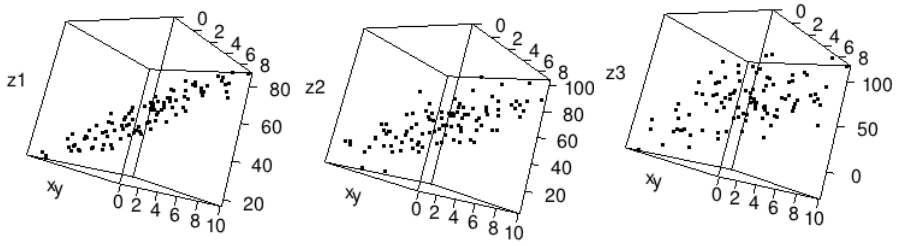
Just as with the inferential t-tests, we need to be able to test whether a slope coefficient is useful in a regression model. We'd like to know if the slope we observed was likely to have arisen even if there was not a relationship with that variable in "the population"–the null hypothesis. But if it were 0, we wouldn't expect to estimate exactly a slope of 0. We can try to determine the chance that we would have observed the outcome we did if this were true. If this probability is low, we reject the null hypothesis.

Although the most common case is we want to know whether a parameter estimate differs from 0, we might also want some other related tests, such as which of two slopes is larger, whether a slope is negative, and if two two slopes differ from one another. There are a number of ways of running these tests and framing the models to make these tests easier to run.

So far, we've estimated the linear coefficients of the function, and shown how a simple regression with two levels of a predictor map onto the same test we do for a t-test. Let's try this again for a more complex model with two predictors that are both continuous. We will create three DVs, each of which have the same relationship with the IVs, but with different levels of random noise, to see how our estimates of residual standard error are impacted.

```
set.seed(1000)
x <- runif(100)*10
y <- runif(100)*10
z1 <- 10 +3*x + 5*y + rnorm(100)*5
z2 <- 10 +3*x + 5*y + rnorm(100)*10
z3 <- 10 +3*x + 5*y + rnorm(100)*25
```

Figure 10.1: Three 3D scatterplots, each from the same perspective, of z1, z2, and z3.



These are sort of difficult to visualize, so we will use the `plot3d` function from the `rgl` library

```
library(rgl)
plot3d(x,y,z1)
plot3d(x,y,z2)
plot3d(x,y,z3)
```

Or use plotly:

```
library(plotly)
df <- data.frame(x,y,z1,z2,z3)
plot_ly(df, x = ~x, y = ~y, z = ~z1) %>% add_markers()
plot_ly(df, x = ~x, y = ~y, z = ~z2) %>% add_markers()
plot_ly(df, x = ~x, y = ~y, z = ~z3) %>% add_markers()
```

Next, we'll make three models, one for each DV:

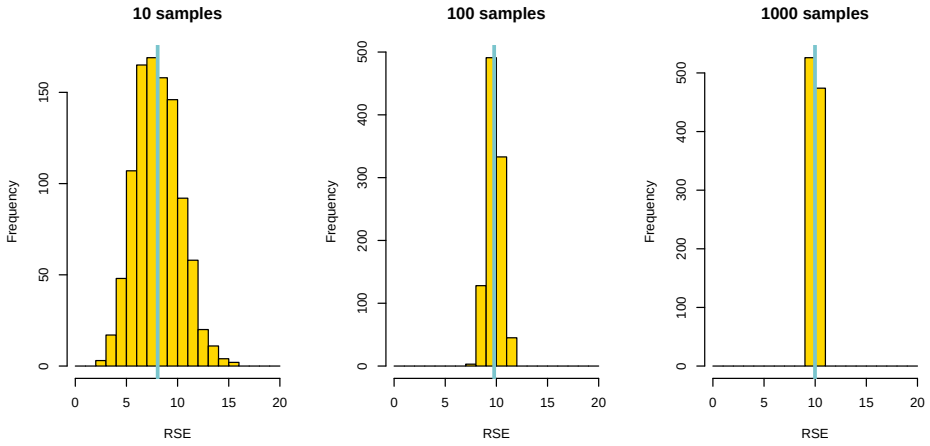
```
lm1 <- lm(z1~x+y)
lm2 <- lm(z2~x+y)
lm3 <- lm(z3~x+y)
```

Let's compute the root mean square of the residuals, as a measure of the remaining variability of the model:

```
rmse1 <- sqrt(mean(lm1$resid^2))
rmse2 <- sqrt(mean(lm2$resid^2))
rmse3 <- sqrt(mean(lm3$resid^2))

> rmse1
[1] 4.394906
> rmse2
[1] 9.639705
> rmse3
[1] 24.49746
>
```

Figure 10.2: Histogram of residual standard errors of linear models.



These values approximate the generating values of std. dev of the error term, but how accurate are these values? Are they biased? Let's find out via a simulation

The following will generate data then re-estimate the parameters of the model, including the slopes and standard error.

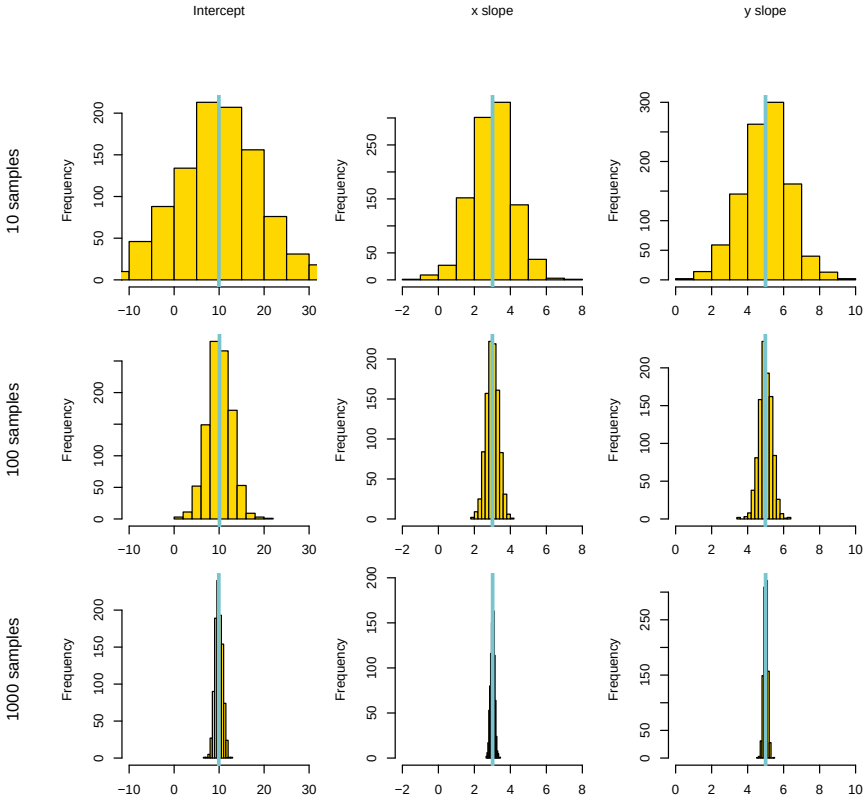
```
test  <- function(sd,n=10)
{
  x <- 1:n/n*10
  y <- runif(n)*10
  z <- 10 +3*x + 5*y + rnorm(n)*sd
  lm1 <- lm(z~x+y)
  c(lm1$coef,  sqrt(mean(lm1$resid^2)))
}

out <- matrix(0,ncol=12,nrow=1000)
for(i in 1:1000)
{
  out[i,1:4] <- test(10,10)
  out[i,5:8] <- test(10,100)
  out[i,9:12] <- test(10,1000)

  if(i%%100==0)   print(i)
  cat(".")
}
```

First, we can see how as the number of observations increase, the residual standard error decreases–just as with the standard error of a simple model. This also appears biased–with fewer data points, we tend to underestimate the variability (by a lot!!!). Like with estimating the s.d. in a sample, there is an adjustment: divide sum by n-p, where n is the number of observations and p is the number of parameters being fitted (adding an intercept term if used). But the residual standard error doesn't say much about the parameters of the model. It says the mean of any given set of independent values should be within +/- 1 value about 2/3 of the time. What we are more interested in is the possible estimated values of the slopes. Here, when we know the true slope, and we observe that the actual slopes vary greatly, just as the RSE varies. We can look at the estimates of the intercept and the two

Figure 10.3: Histogram of estimates of parameters as sample size increases.



slope parameters in Figure 10.3.

Under a NHST, we'd like to be able to estimate the variability of means if we had a model without a slope, but observed the same residual error we did in this model. In the next section, we will see how we can generate these estimates and use them as part of a null-hypothesis statistical test. In each case, the test ends up being identical to a one-sample t-test: we determine whether the parameter value we observed (i.e., a slope or intercept) would have been probable to have occurred by chance if there really were no relationship.

Exercise 10.2

Load a data set from the faraway library.

```
install.packages("faraway")
library(faraway)
data(stat500)
summary(stat500)
```

Follow the basic steps below:

1. Look at the relationships and correlations between the four variables.
2. Compute a linear model predicting the final based on the midterm.
3. Look at the parameter estimates and residuals
4. Compute a linear model predicting final based on midterm + homework
5. Look at the parameter estimates and the residuals. Look especially at the standard deviation of the residuals. How much of an improvement do you get? Do the estimates change?

10.3 The estimate of sigma provided by `lm`

Usually we won't have to estimate sigma by ourselves, because lm will tell us what it is. Here is a linear model. Notice that if we sum the squared residuals and divide by $n-3$ (instead of n, which would be biased), we get the exact same value that summary reports.

```
set.seed(1000)
sd <- 10
n <- 50
x <- 1:n/n*10
y <- runif(n)*10
y2 <- runif(n) * 3
z <- 10 +3*x + 5*y + rnorm(n)*sd
lm1 <- lm(z~x+y+y2)

lm1

Call:
lm(formula = z ~ x + y + y2)

Coefficients:
(Intercept)            x            y           y2
      7.803        3.547        4.680        2.011
```

Let's compute the unbiased MSE of our residuals–the sum of the squared errors, divided by $n-p$, (where $p=4$):

```
res1 <- sqrt(sum(lm1$resid^2)/(n-4))
print(res1)
[1] 10.68216
```

The summary function will compute various inferential statistics and hypothesis tests on the linear model. First, look at summary(lm1) and examine what it reports for the 'Residual Standard Error' value.

```
summary(lm1)

Call:
lm(formula = z ~ x + y + y2)

Residuals:
    Min      1Q  Median      3Q     Max
-24.886  -5.701   0.414   6.042  22.081

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   7.8033     4.7711   1.636    0.109
x             3.5475     0.5411   6.556 4.23e-08 ***
y             4.6796     0.5631   8.311 1.03e-10 ***
y2            2.0106     1.7924   1.122    0.268
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 10.68 on 46 degrees of freedom
Multiple R-squared:  0.743, Adjusted R-squared:  0.7263
F-statistic: 44.34 on 3 and 46 DF,  p-value: 1.277e-13
```

You can see that these values are identical. There RSE gives a measure of the overall standard error–the combined noise of all the estimated predictor together. The `summary` also provides more information though.

10.4 Summarized results from a linear model

The `summary()` function does more than just report the residual errors; it will conduct a t-test of each parameter against 0, compute RSE, R^2, and an overall F test of the model. We will go over each of these in turn.

10.4.1 What is the std. error and what does the t-test for a coefficient compute?

The formula for computing std. error of a coefficient is not incredibly intuitive. But the outcome is a value that tells you the estimated variability of the estimate, which let's you do a null-hypothesis significance test using the t distribution.

```

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   7.8033     4.7711   1.636    0.109
x             3.5475     0.5411   6.556 4.23e-08 ***
y             4.6796     0.5631   8.311 1.03e-10 ***
y2            2.0106     1.7924   1.122    0.268
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

The summary computes a t-test for each predictor, including the intercept. The t-test for an individual parameter estimate can be interpreted several ways.

First, it is just like a t-test studied in earlier chapters–it determines whether the data can be explained by the null hypothesis (that the parameter value is equal to 0). If the t value is big and the p value is small, the null model is rejected. For the intercept, this just asks whether the model predicts the value should be different from zero if all other input values are also zero. This usually does not make much sense, and so a significant intercept term is rarely very interesting unless you have a very specific hypothesis you are testing. But the slope values are more interesting. A slope of 0 indicates that the outcome is not related to the predictor, and so if the observed beta value was unlikely to have been produced in a data set where there was no relationship, we say that the parameter was statistically different from 0. In lm1, we can see that the intercept was not significant, which has no real importance. Then, x and y were both significant, which means there each predicted the outcome independently. But y2 did not. It is important to note that the least-squares criterion estimates all predictors simultaneously. When two predictors are correlated, you will get estimates, but may not be able to interpret the estimates properly, and perhaps you can only make sense of the model by comparing sets of models.

This leads to the alternative way of interpreting slope coefficients. Asking whether a beta is different from 0 is equivalent to asking whether the model improves over the model not containing the parameter.

To demonstrate this, let's compare the residuals of lm1 to a model without y:

```
lm2 <- lm(z~x+y)
res2 <- sum(lm2$resid^2)
summary(lm2)
```

Then compute the model without y:

```

lm3 <- lm(z~x)
res3 <-(sum(lm3$resid^2))
summary(lm3)

tval <-  sqrt((res3-res2)/(res2/(50-3)) )
```

Here, we computed a t value directly by comparing the residuals from two nested models. The t value computed here is the square root of an statistic known to follow an F distribution, which we use to compare models. Consequently, they are testing exactly the same thing. Look at how the 8.486 we calculated by hand appears in the regression as a t-test on the missing predictor.

```
> tval
[1] 8.485574
> summary(lm2)

Call:
lm(formula = z ~ x + y)

Residuals:
     Min       1Q   Median       3Q      Max
-25.4453  -5.7720   0.2618   5.5046  24.3513

Coefficients:
```

```
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.9649     3.8601   2.841  0.00664 **
x             3.4168     0.5298   6.449 5.62e-08 ***
y             4.7560     0.5605   8.486 4.85e-11 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 10.71 on 47 degrees of freedom
Multiple R-squared:  0.736, Adjusted R-squared:  0.7248
F-statistic: 65.52 on 2 and 47 DF,  p-value: 2.557e-14
```

This second interpretation seems a bit more complicated, but it is important to keep in mind, especially as we look at Bayes Factor version of these tests.

10.4.2 What is Multiple R^2 and Adjusted R^2

Multiple R^2 is the square of the correlation coefficient of observed vs. predicted:

```
cor(lm1$fit,z)^2
[1] 0.7430299

summary(lm1)

Call:
lm(formula = z ~ x + y + y2)

Residuals:
     Min       1Q   Median       3Q      Max
-24.886   -5.701    0.414    6.042   22.081

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   7.8033     4.7711   1.636    0.109
x             3.5475     0.5411   6.556 4.23e-08 ***
y             4.6796     0.5631   8.311 1.03e-10 ***
y2            2.0106     1.7924   1.122    0.268
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 10.68 on 46 degrees of freedom
Multiple R-squared:  0.743, Adjusted R-squared:  0.7263
F-statistic: 44.34 on 3 and 46 DF,  p-value: 1.277e-13
```

R^2 is useful because it tells you the proportion of the variance your model accounts for.

Multiple R^2 is sort of unfair, because you have multiple predictors, even if they are random, they will by chance increase the goodness of fit. Adjusted R^2 is an attempt to counteract the improvement you get by chance alone.

Whereas R^2 is computed as:

$$R^2 = 1 - SS_{resid}/SS_{total} \tag{10.1}$$

Adjusted R^2 is an attempt to counteract the improvement you get by chance alone:

$$R^2_{adj} = 1 - (1 - R^2) \times \frac{n-1}{n-k-1} \tag{10.2}$$

Where n is the number of observations and k is the number of predictors. As a consequence, adjusted R^2 tries to 'punish' models for having more predictors, accounting for the potential additional spurious correlations.

For example, look what happens when we add additional random predictors

```
q1 <- runif(50)
q2 <- runif(50)
q3 <- runif(50)
q4 <- runif(50)
q5 <- runif(50)
q6 <- runif(50)
q7 <- runif(50)
q8 <- runif(50)
q9 <- runif(50)
q10 <- runif(50)

summary(lm(z~q1))$r.squared
[1] 0.001243165
summary(lm(z~q1))$adj.r.squared
[1] -0.01956427

summary(lm(z~q1+q2))$r.squared
[1] 0.001615148

summary(lm(z~q1+q2))$adj.r.squared
[1] -0.04086931

summary(lm(z~q1+q2+q3))$r.squared
[1] 0.006983906

summary(lm(z~q1+q2+q3))$adj.r.squared
[1] -0.05777801

summary(lm(z~q1+q2+q3+q4))$r.squared
[1] 0.00761476

summary(lm(z~q1+q2+q3+q4))$adj.r.squared
[1] -0.08059726

summary(lm(z~q1+q2+q3+q4+q5))$r.squared
[1] 0.02241639

summary(lm(z~q1+q2+q3+q4+q5))$adj.r.squared
[1] -0.08867266

summary(lm(z~q1+q2+q3+q4+q5+q6+q7+q8+q9+q10))$r.squared
[1] 0.2007088

summary(lm(z~q1+q2+q3+q4+q5+q6+q7+q8+q9+q10))$adj.r.squared
[1] -0.004237655
```

Notice how R^2 always goes up when you add more predictors, but adjusted R^2 stays around 0–where it probably should, given that these predictors are randomly generated and not intentionally related to the outcome variable.

Exercise 10.4.2

Create four correlated data sets, and another 4 independent ones. Compute the sum of four of these, and create a regression model predicting that sum by all eight variables. Examine what happens to R^2 and adjusted R^2 as you remove and add predictors from the model.

10.4.3 How do you interpret the F-statistic?

In regression, the F statistic works like the χ^2 statistic we computed in the previous chapters. In fact, the χ^2 is a special case of the F distribution. It is a comparison of two variances: the variance associated with the model, in comparison to the variance associated with the intercept-only model. To the extent that the model explains enough variance given the degrees of freedom it uses, this test will be significant, and tells you that the results would be unlikely to have occurred by chance. In later chapters, we will use this F test extensively.

Be careful when reporting the F statistic of a regression. It does not really mean "the model is significant"; it compares the model you looked at versus the best-fitting 1-parameter intercept model, and asks if your model is better. So, your F statistic may be significant even if it contains many predictors that are not significant. It is usually better to compare pairs of models that differ by just a single predictor, which is what the t-test does, and this t-test turn out to be equivalent to an F-test comparing that pairing of models.

10.5 Bayes Factor Regression Model

Instead of computing a NHST test on the entire model (i.e., the F test reported) or on individual predictors (i.e., the t-tests), you might use a Bayes Factor test to look at support for alternate hypotheses. The Bayes Factor regression is available in the functions `lmBF`, which computes the Bayes factor for a single model, and `regressionBF`, which computes Bayes factors for comparing all submodels–all combinations of predictors. In both cases, the Bayes Factor test is about inference, and not estimation of coefficients–if you need those coefficients, residuals, predicted values, and other aspects of a regression, you will have to run the traditional `lm()` function.

In these tests, we don't test individual coefficients differently from the entire model. The Bayes Factor package computes a Bayes Factor for the model versus the null hypothesis model of intercept-only–this is analogous to the F-test reported by the summary function of lm. Furthermore, the Bayes Factor approach tests all possible nested models and provides a Bayes factor that can be examined between any pairing of models. Thus, if you have three predictors (x, y, and z), and want to know if z is useful, traditionally you would compare intercept + x+y+z against intercept x+y in an F test or the equivalent t-test for significant slope of the β_z. But the Bayes factor model also compares the intercept+z to the intercept-only model; the intercept+x+z to the intercept + x model, and so on.

The BayesFactor regression requires a data frame argument, so we will need to combine our predictors into a single data frame. We will re-use the data set we created earlier:

```
1 library(BayesFactor)
```

```
set.seed(1000)
sd <- 10
n <- 50
x <- 1:n/n*10
y <- runif(n)*10
y2 <- runif(n) * 3

##z is made from x and y, but not y2:
z <- 10 +3*x + 5*y + rnorm(n)*sd

df=data.frame(x,y,y2,z)
lmbayes <- lmBF(z~x+y+y2,data=df)
summary(lmbayes)

> summary(lmbayes)
Bayes factor analysis
--------------
[1] x + y + y2 : 61257302941 ?0%

Against denominator:
Intercept only
---
Bayes factor type: BFlinearModel, JZS
```

Here, the Bayes Factor test shows strong evidence for the set of predictors, in comparison to the intercept-only null hypothesis. However, in our example, y2 was generated independently. We'd like to be able to test individual submodels. The regressionBF lets us test all submodels at once.

```
lmbayes2 <- regressionBF(z~x+y+y2,data=df)

lmbayes2
Bayes factor analysis
--------------
[1] x           : 1393.108     0%
[2] y           : 1038588      0.01%
[3] y2          : 0.2925324    0%
[4] x + y       : 240747175456 0%
[5] x + y2      : 750.9676     0%
[6] y + y2      : 187407.8     0%
[7] x + y + y2 : 61257302941  0%
```

This compares a set of all possible submodels to the intercept-only model. This set of models forms a lattice, and if we'd like to know whether any predictor improves on another model, we can just compute a relative Bayes Factor as the ratio of the two factors in the table above. Each Bayes factor is in relationship to the intercept-only Bayes factor, and you can compare any two models by forming their ratio. So, in comparison to the intercept-only model, the x predictor has a Bayes factor of about 1393–strong support for the hypothesis that x matters. But we can also look at how x does when added to other models. The ratio of the x+y Bayes Factor (model 4) to the y Bayes factor (model 2) is the relative factor between these models, and we can actually get this test by dividing one submodel against the other:

```
> lmbayes2[4]/lmbayes2[2]
Bayes factor analysis
--------------
```

```
[1] x + y : 231802.4 0.01%

Against denominator:
  z ~ y
---
Bayes factor type: BFlinearModel, JZS
```

In the context of y, the x Bayes Factor is more than 230,000! Similarly, there is ambivalent evidence for y2 in comparison to the intercept-alone (BF=.29), actually with fairly strong evidence for the null hypothesis. For any pair of models you might compare where one involves y2 and the other does not, the Bayes Factor is similarly small.

```
> lmbayes2[5]/lmbayes2[1]
Bayes factor analysis
--------------
[1] x + y2 : 0.5390591 0%

Against denominator:
  z ~ x
---
Bayes factor type: BFlinearModel, JZS


> lmbayes2[6]/lmbayes2[2]
Bayes factor analysis
--------------
[1] y + y2 : 0.1804448 0.01%

Against denominator:
  z ~ y
---
Bayes factor type: BFlinearModel, JZS
```

In these cases, we would tend to prefer the smaller model, but there is not extremely strong evidence against y2 being involved.

10.6 Categorical Predictors

When people think of regression analysis, they often think that it only is valid for sets of continuous predictors, and not categorical predictors. When we have categorical predictors, there are several approaches people take. If your categorical predictor is binary, then it fits into the regression model very easily. The coefficient simply codes the extent to which the two levels differ. If you have multiple levels, the regression model needs to do some underlying coding scheme to represent those levels as a combination of binary predictors. If you give `lm` a categorical predictor, it will code it for each of its levels with respect to the first level of a factor, so that the first level is equivalent to the intercept-only model.

For example, suppose we want to predict longevity of a fruitfly based on its size (measured by its thorax), and a behavior category of activity level. Start with a simple model of the thorax alone:

```
library(faraway)
data(fruitfly)
summary(fruitfly)
ff.lm <- lm(longevity~thorax,data=fruitfly)
```

```
5
7  summary(ff.lm)
9  Call:
   lm(formula = longevity ~ thorax, data = fruitfly)
11
   Residuals:
13     Min      1Q  Median      3Q     Max
   -28.364  -9.986   1.258   9.264  36.825
15
   Coefficients:
17             Estimate Std. Error t value Pr(>|t|)
   (Intercept)   -61.86      13.37  -4.625 9.39e-06 ***
19 thorax         145.28      16.19   8.971 4.27e-15 ***
   ---
21 Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

23 Residual standard error: 13.65 on 122 degrees of freedom
   Multiple R-squared:  0.3975,  Adjusted R-squared:  0.3926
25 F-statistic: 80.49 on 1 and 122 DF,  p-value: 4.275e-15
```

Here, thorax is a continuous predictor of measured thorax size. The intercept of -61.86 is the baseline intercept, meaning the value if thorax size was 0. Your best prediction is -61.86 + 145.28 times thorax-size.

Now, consider the categorical predictor, which has five categories: isolated, one, low, many, and high.

```
1  ff.lm2 <- lm(longevity~thorax+activity,data=fruitfly)
   summary(ff.lm2)
3
5  Call:
   lm(formula = longevity ~ thorax + activity, data = fruitfly)
7
   Residuals:
9      Min      1Q  Median      3Q     Max
   -26.108  -7.014  -1.101   6.234  30.265
11
   Coefficients:
13             Estimate Std. Error t value Pr(>|t|)
   (Intercept)   -48.749     10.850  -4.493 1.65e-05 ***
15 thorax        134.341     12.731  10.552  < 2e-16 ***
   activityone     2.637      2.984   0.884   0.3786
17 activitylow    -7.015      2.981  -2.353   0.0203 *
   activitymany    4.139      3.027   1.367   0.1741
19 activityhigh  -20.004      3.016  -6.632 1.05e-09 ***
   ---
21 Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

23 Residual standard error: 10.54 on 118 degrees of freedom
   Multiple R-squared:  0.6527,  Adjusted R-squared:  0.638
25 F-statistic: 44.36 on 5 and 118 DF,  p-value: < 2.2e-16
```

Here, the coefficient for thorax stays about the same, but we have four values for different levels of activity. Notice that the output is missing an 'activityisolated' level, and the model includes an (Intercept) value, which differs from the simpler model intercept. The way to

interpret this table is that the first level of activity has a value of 0, so adds nothing to the baseline intercept-only model: -48.75 + 0 + 135 * thorax. To predict activity level 'one', we would add the appropriate coefficient to this model: -48.75 +2.6 + 135* thorax. Each level of activity is essentially scored relative to the baseline model.

This might not always make sense, and so an alternative is to force the baseline-only model to be 0. Then, each level of your categorical predictor will be scored with respect to 0. You can do this by adding a 0 predictor to the model:

```
ff.lm3 <- lm(longevity~0+thorax+activity,data=fruitfly)


Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
thorax              134.34      12.73  10.552  < 2e-16 ***
activityisolated   -48.75       10.85  -4.493 1.65e-05 ***
activityone        -46.11       10.72  -4.301 3.52e-05 ***
activitylow        -55.76       10.87  -5.130 1.15e-06 ***
activitymany       -44.61       10.57  -4.222 4.78e-05 ***
activityhigh       -68.75       10.40  -6.610 1.17e-09 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 10.54 on 118 degrees of freedom
Multiple R-squared:  0.9708,  Adjusted R-squared:  0.9694
F-statistic: 654.7 on 6 and 118 DF,  p-value: < 2.2e-16
```

This may be easier to interpret (because all levels of the IV are there as essentially distinct intercepts). If you do the math, the model is identical to the one above, but now we can see that the predictive model for isolated is -48.75+134.34. However, the t-tests in this new model are a bit misleading. In this case, they test whether the predictor is different from 0, which is probably not interesting. What we might really care about is whether they are larger than a control condition–in this case maybe the first level of our predictor. So the first model may provide us a more interesting default test, because it shows that activity level on and many are not different from the isolated condition.

As we develop the linear model for categorical predictors more in the form of the factorial ANOVA models, we will delve into how categorical factors can be represented in the model, and how the values can be interpreted.

10.6.1 Caveats and Warnings

You need to be careful specifying categorical predictors in a linear regression model. If your category is numerically-labeled (e.g., participant code), you need to be sure it is a factor before using it as a predictor. If you do not, it will treat the predictor as a numerical predictor, and estimate a slope for that predictor, and this is not what you meant to do. Read output carefully to be sure all the levels you expect to be there have their own coefficients.

You need to be careful and precise when interpreting the t-test and F-tests reported by a regression. The fact that a test is 'significant' does not give a stamp of approval. It is testing a very specific hypothesis, and that hypothesis is not often what you are interested in. You may have to craft a different test to address the thing you are truly interested in. For example, the t-tests on coefficients each test whether a value is likely to have been obtained if the true coefficient were 0. As we can see above in the fruitfly example, this may not be what we are interested in if we have forced the model intercept to 0, and it may not be what we are interested unless the first level of our factor happens to be the control condition. You

can reorder levels of a factor to be in a convenient order (look at the documentation for factor()), which can be helpful, but even then you may want to test other hypotheses about the slope coefficients.

Similarly, the main F test for the model simply compares the entire model to the intercept-only model. It uses the intercept-only model as a null hypothesis, and asks whether a you might have observed what you did by chance, if you had assigned predictors randomly. This does not mean that everything within the model is useful, and although it usually means that one or more things are useful (statistically significant), you would need to look at other tests to isolate that.

Regression models with categorical predictors form the basis for what is frequently referred to as "ANOVA" models, which stands for "Analysis of Variance". This is somewhat of a misnomer. The regression model is used to estimate the effects, and the ANOVA is the inferential statistical test used to compare related models to one another to judge whether individual sets of predictors are useful. However, many people fail to recognize the intrinsic link between regression and ANOVA. In fact, they are the same thing, and ANOVA is a specific way of testing the difference between related submodels with categorical predictors.

10.6.2 Category by slope interactions

We might want to determine whether there is a different coefficient for thorax for each level of activity. When the effect of one variable depends on the level of another variable, this is called an interaction–just like we refer to drug interactions as when the effect of a drug depends on whether another drug is being taken.

It makes logical sense that the thorax coefficient may depend on activity–an activity level could impact not only mean longevity, but how strongly longevity is associated with body size. We can do this using an interaction term, which can be specified using the ':' operator instead of the '+'. The following model tests whether the slopes differ, assuming the same intercept.

```
> ff.lm4a <- lm(longevity~thorax:activity,data=fruitfly)
> summary(ff.lm4a)

Call:
lm(formula = longevity ~ thorax:activity, data = fruitfly)

Residuals:
Min       1Q   Median       3Q      Max
-26.5039  -6.1796  -0.9561   7.0033  30.2081

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)               -53.16      10.51  -5.057 1.58e-06 ***
thorax:activityisolated   139.59      12.70  10.988  < 2e-16 ***
thorax:activityone        142.80      12.90  11.070  < 2e-16 ***
thorax:activitylow        131.27      12.72  10.324  < 2e-16 ***
thorax:activitymany       144.85      13.10  11.057  < 2e-16 ***
thorax:activityhigh       114.94      13.28   8.654 2.97e-14 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 10.54 on 118 degrees of freedom
Multiple R-squared:  0.6524,  Adjusted R-squared:  0.6377
F-statistic:  44.3 on 5 and 118 DF,  p-value: < 2.2e-16
```

What do the different coefficients tell us? Although we can now see a different slope for each group, each test just tells us whether that slope differs from 0. This on its own is not very interesting; we might prefer to understand whether the isolated slope (139.6) differs from the high-activity slope (114.9). That can't be done directly in this particular framing of the model.

We could also look at estimating a separate intercept and slope for each group. This is like doing five completely separate regressions, but puts them all into the same model, which could be useful if the first level is a control group. Looking at this test, try to determine whether any specific group has a distinct slope and intercept from the first 'isolated' group. Comparing this to the ff.lm2 model, where we found significant differences in the relative intercept for isolated vs. high, this is no longer apparent.

```
> summary(ff.lm5)

Call:
lm(formula = longevity ~ thorax * activity, data = fruitfly)

Residuals:
Min       1Q   Median       3Q      Max
-25.9509  -6.7296  -0.9103   6.1854  30.3071

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          -50.2420    21.8012  -2.305    0.023 *
thorax               136.1268    25.9517   5.245 7.27e-07 ***
activityone            6.5172    33.8708   0.192    0.848
activitylow           -7.7501    33.9690  -0.228    0.820
activitymany          -1.1394    32.5298  -0.035    0.972
activityhigh         -11.0380    31.2866  -0.353    0.725
thorax:activityone    -4.6771    40.6518  -0.115    0.909
thorax:activitylow     0.8743    40.4253   0.022    0.983
thorax:activitymany    6.5478    39.3600   0.166    0.868
thorax:activityhigh  -11.1268    38.1200  -0.292    0.771
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 10.71 on 114 degrees of freedom
Multiple R-squared:  0.6534,  Adjusted R-squared:  0.626
F-statistic: 23.88 on 9 and 114 DF,  p-value: < 2.2e-16
```

Now, the predictors become a bit more confusing. We have an intercept term, which is the baseline value for activity-isolated. Each other baseline condition would add or subtract from that value, and each t-test on those parameters are really testing whether they differ from the isolated condition. Next, we have a thorax slope, which is the estimated slope for the isolated condition (NOT the average slope for all conditions). Each interaction term is how much higher or lower the slope is for the different conditions. So the t-tests here are mostly comparisons to the isolated condition, and not between conditions or in comparison to an 'average'.

We might also notice that almost nothing is significant anymore. This is merely because all the significant slopes were tested against 0-slope in the previous model, but now they are all tested against the baseline condition.

This type of model forms the baseline for what is generally referred to as 'ANCOVA' models–analysis of covariance. They are used frequently in social and personality research, and are typically used to examine how the relationship between two continuous variables (i.e, personality and an outcome) depend on group membership (gender, age, etc.).

Exercise 10.6.2

Use the galapagos (`data(gala)`) data set, try to predict the number of plant species found on each island based on the other factors. At each step of your model, try to look at the observed vs. predicted, and the residuals to determine whether they are appropriate for the statistical test. Conduct a Bayes Factor test of the model space. Try to identify which predictors are important, and which are not, and for those that are, interpret what the model is telling you.

10.6.3 Variable Selection

So far, we've seen how adding predictors will always make our R^2 value better (or at least not worse). Sometimes, the amount they make the fit better is not worthwhile, and so we would favor using a model without a predictor–even if it makes our fit better. Your intuition for this should be that if a predictor is not very useful, it will still tend to make the current model a little better, but it will probably make future prediction worse, because it is likely to be explaining the non-systematic noise more than the systematic pattern. So, we are generally interested in finding the smallest useful model that predicts our data. This is a typical and normal phase of regression analysis–trying to select the parameters that help enough to justify their use. In more general situations such as classification and machine learning, a lot of research involves trying to find schemes for selecting which predictors (often referred to as features) are useful, and this is the exact same problem faced in regression analysis.

10.7 Solutions to exercises

10.7.1 Stat500 data

Begin by loading the data set and looking at correlations between the data

```
>library(faraway)
> data(stat500)
>summary(stat500)

stat500$total2 <- stat500$midterm*.25 + stat500$final*.50 + stat500$hw*.25
> print(cor(stat500),3)
        midterm  final     hw total
midterm   1.000 0.5452 0.2721 0.844
final     0.545 1.0000 0.0873 0.779
hw        0.272 0.0873 1.0000 0.564
total     0.844 0.7789 0.5644 1.000


pairs(stat500)
```

It looks like there is a high correlation between midterm and final, as well as the homework and final. Let's try to predict how somebody should do on the final based on their homework and midterm. To do this, we'll start by making two models.

```
s500.1 <- lm(final~midterm,data=stat500)
s500.2 <- lm(final ~ midterm + hw,data=stat500)

> summary(s500.1)

Call:
lm(formula = final ~ midterm, data = stat500)

Residuals:
   Min     1Q Median     3Q    Max
-9.932 -2.657  0.527  2.984  9.286

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  15.0462     2.4822   6.062 1.44e-07 ***
midterm       0.5633     0.1190   4.735 1.67e-05 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 4.192 on 53 degrees of freedom
Multiple R-squared: 0.2973, Adjusted R-squared: 0.284
F-statistic: 22.42 on 1 and 53 DF,  p-value: 1.675e-05

> summary(s500.2)

Call:
lm(formula = final ~ midterm + hw, data = stat500)

Residuals:
     Min       1Q   Median       3Q      Max
-10.0388  -2.5964   0.3714   3.0063   9.3497

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 16.81061    4.08112   4.119 0.000137 ***
midterm      0.58179    0.12445   4.675 2.12e-05 ***
hw          -0.08157    0.14916  -0.547 0.586836
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

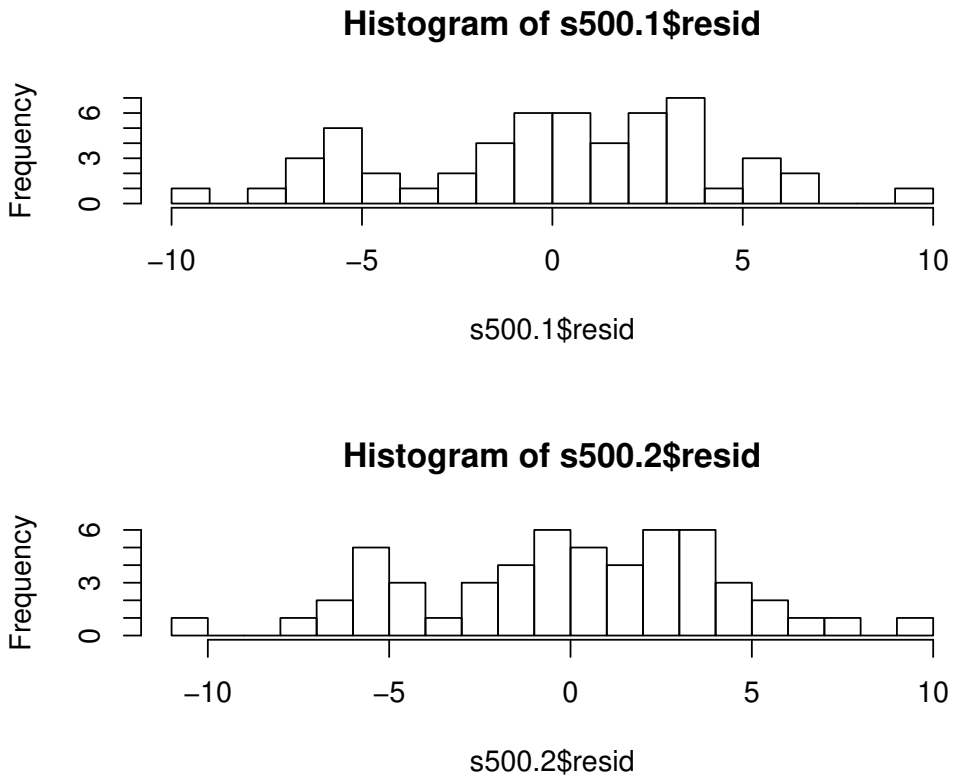
Residual standard error: 4.22 on 52 degrees of freedom
Multiple R-squared: 0.3013, Adjusted R-squared: 0.2744
F-statistic: 11.21 on 2 and 52 DF,  p-value: 8.948e-05


plot(stat500$final,s500.2$fit)
cor(stat500$final,s5002$fit)


##compare the residuals of the two models.  Is one better
plot(rep(1,length(s500.1$resid)),s500.1$resid,xlim=c(0,3))
points(rep(2,length(s500.2$resid)),s500.2$resid,xlim=c(0,3),add=T)
library(vioplot)
vioplot(s500.1$resid,s500.2$resid)
```

Notice that the hw variable is not predictive of the final when combined with the midterm. The fit of the model does not change much when hw is added. Also, notice that the coefficient of hw is negative, which indicates that the better somebody did on the hw the worse they did on the final (although this was not significant).

Figure 10.4: Histograms of the residual (errors) of the model,either with or without the `hw` predictor.



Looking at the residuals (in Figure 10.4, we can see that the error distribution does not change much. All signs point to the conclusion that the homework is not predictive of final test performance.

```
g <- lm(Species~.,data=gala)
g2 <- lm(Species~Endemics+Area+Elevation+Nearest+Adjacent,data=gala)
g3 <- lm(Species~Endemics+Area+Elevation+Nearest,data=gala)
g4 <- lm(Species~Endemics,data=gala)
```

Chapter 11

Comparing Regression Models, Variable Selection, Prediction

11.1 Comparing (nested) Regression Models

A regression model involves a single dependent variable, a set of predictor variables, and beta weights or coefficients that constitute the combination of slopes of each predictor that fit the data the best. As was discussed in the previous chapter, each parameter can be evaluated with a t-test, which can be interpreted EITHER as a test of whether that parameter could have arisen from the null model (where that dimension was unrelated to the outcome), or as a comparison of models with and without that predictor. If a predictor is not significant, it is typical to remove it from the model and use the smaller, simpler model. This has two benefits. First, unless you have strong control over the predictors, they are likely to be somewhat correlated, and so removing a predictor is likely to allow other predictors to show their effects without complications. And second, a model with too many predictors risks over-predicting the data–fitting noise as well as the actual effects. By removing predictors, the model is less likely to over-predict, and more likely to reveal the truth.

11.1.1 Parameter selection versus model testing

The process of building linear regression models is often one of variable selection. Moreover, variable selection is really model comparison. To determine whether you want to use a variable or set of variables, you compare the model with the variable to the model without the variable or set of variables. When you build a regression model, the `summary()` gives a statistical test at the end, reported as an F test. This test compares the smallest reasonable model to your model, and asks "is the goodness of fit I observed likely to have occurred if the smallest model were the truth?". The smallest model is the model that assumes all of the data come from a single distribution with a single mean. So the F test asks whether the goodness-of-fit of your model would have been produced in a situation where all of the predictors were not predictive.

We can examine this using a data set that looks at a survey of faculty, students, and industry representatives. In this survey, respondents were asked to rate the most important skills among the Michigan Tech grads they hired, including academic performance, communication skill, leadership experience, hands-on experience, teaming, prior work experience, multi-cultural experience, and creativity. We'd like to know if academic performance is

viewed as a proxy for some combination of these, so we'd like to make a linear regression predicting the academic rating by the other ratings. First, load the data and look at some basic statistics.

```
dat <- read.csv("survey.csv")

summary(dat)
      group        academic      communication     leadership
 Faculty : 65   Min.   :1.000   Min.   :1.000   Min.   :1.00
 Industry: 99   1st Qu.:3.000   1st Qu.:4.000   1st Qu.:3.00
 Student :119   Median :4.000   Median :5.000   Median :4.00
                Mean   :3.496   Mean   :4.442   Mean   :3.91
                3rd Qu.:4.000   3rd Qu.:5.000   3rd Qu.:4.75
                Max.   :5.000   Max.   :5.000   Max.   :5.00

       handson          teaming         priorwork
    Min.   :2.000   Min.   :1.000   Min.   :1.000
    1st Qu.:4.000   1st Qu.:4.000   1st Qu.:3.000
    Median :4.000   Median :4.000   Median :4.000
    Mean   :4.134   Mean   :4.334   Mean   :3.933
    3rd Qu.:5.000   3rd Qu.:5.000   3rd Qu.:5.000
    Max.   :5.000   Max.   :5.000   Max.   :5.000

 multicultural      creative        ethical
 Min.   :1.000   Min.   :1.000   Min.   :1.000
 1st Qu.:2.000   1st Qu.:3.000   1st Qu.:3.000
 Median :3.000   Median :3.000   Median :4.000
 Mean   :2.599   Mean   :3.325   Mean   :3.633
 3rd Qu.:3.000   3rd Qu.:4.000   3rd Qu.:5.000
 Max.   :5.000   Max.   :5.000   Max.   :5.000

 round(cor(dat[,2:9]),3)
              academic commun. leadership handson teaming priorwork multicult creative
academic         1.000    0.172      0.064   0.030   0.164     0.166   -0.056   -0.104
communication    0.172    1.000      0.264   0.105   0.411     0.085    0.131    0.239
leadership       0.064    0.264      1.000   0.167   0.386     0.132    0.147    0.106
handson          0.030    0.105      0.167   1.000   0.167     0.142   -0.051    0.094
teaming          0.164    0.411      0.386   0.167   1.000     0.082    0.181    0.205
priorwork        0.166    0.085      0.132   0.142   0.082     1.000    0.034   -0.144
multicultural   -0.056    0.131      0.147  -0.051   0.181     0.034    1.000    0.318
creative        -0.104    0.239      0.106   0.094   0.205    -0.144    0.318    1.000
```

Because these are all on 5-point scales, we are not too worried about outliers, and the looking at boxplots of the data they do not seem to violate normality substantially. There are some moderate correlations between different predictors, but if we focus on academic performance, the best correlation is about .17, with some actually negatively correlated. Let's create an initial regression model predicting academic by everything else:

```
lm1 <- lm(academic~communication+leadership+handson+teaming+priorwork+
    multicultural+creative+group,data=dat)
summary(lm1)

> summary(lm1)

Call:
lm(formula = academic ~ communication + leadership + handson +
    teaming + priorwork + multicultural + creative + group, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.44275 -0.50211  0.07341  0.53071  2.02186

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    2.39548    0.44209   5.419 1.32e-07 ***
communication  0.15613    0.07769   2.010  0.04545 *
leadership    -0.02104    0.06484  -0.325  0.74580
handson       -0.04730    0.05628  -0.841  0.40133
teaming        0.15394    0.07787   1.977  0.04905 *
priorwork      0.14145    0.05196   2.722  0.00690 **
multicultural -0.06659    0.05144  -1.294  0.19662
creative      -0.06472    0.04873  -1.328  0.18525
```

```
groupIndustry -0.01054    0.13292  -0.079  0.93687
groupStudent  -0.34770    0.13144  -2.645  0.00863 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.8193 on 273 degrees of freedom
Multiple R-squared:  0.121, Adjusted R-squared:  0.092
F-statistic: 4.175 on 9 and 273 DF,  p-value: 4.58e-05
```

The first thing to recognize is that there are significant predictors, although the adjusted R^2 is just .09, which means we are not explaining much at all. Looking at it the t-values, we'd expect leadership, hands-on, multicultural, and creative to not be significant predictors of rated importance of academic ability. The best practice would be to remove each one of these one at a time and see how it impacts the model. At each point, we could evaluate multiple R^2 or adjusted R^2 to determine whether it produces a reasonable fit. These are all common ad hoc approaches, and they are governed as much by intuition about the predictors as evidence. For example, you might typically use some measure of socio-economic status as a predictor. You might find that for a particular outcome, it is not significant, but decide to retain the predictor anyway so you can interpret parallel models more easily.

11.2 Parameter selection/Model testing using F Tests and the Analysis of Variance procedure

To be more specific, let's define a few terms:

- RSS = Residual Sum Squared
- RSS_1 = RSS of larger model
- RSS_2 = RSS of smaller model
- $RSS_{(1-2)}$ = improvement in RSS by moving to the larger model.

The basic test we want to perform looks at $RSS_{(1-2)}$ in comparison to RSS_1. If the reduction in fit was big enough to matter, you should choose the more complex model. If it is not big enough, choose the smaller model. Here, we can interpret "big enough to matter" as "unlikely to have occurred if there additional parts of the larger model did not have an effect".

It can be shown that the ratio of these variances have an F distribution under the Null hypothesis. That is, if the additional parameters really were 0, you'd still find some difference, and this difference has an F distribution if we were to run the experiment repeatedly. If we find a difference that is enough larger than the null, we would reject the null and decide that the parameter is useful.

Here is the logic of an inferential test we can use:

- Ratios of independent variances have an F distribution when they are identical (Null hypothesis).
- Using this type of test of ratios of variances is known as the Analysis of Variance, or ANOVA procedure.
- The test you may know as the ANOVA is really about comparing variances you can explain to ones you cannot explain (a ratio of two independent variances).

Under the null hypothesis that there is no difference between the two models (and so should prefer the smaller), these variances have an F distribution (assuming normal and independent error). That is, if the additional parameters really did nothing to predict the data, you'd still find some difference just by chance. If we find a difference that is enough larger than the null, we would reject the null and decide that the parameter is useful.

The test commonly labeled by software tools as the "Univariate ANOVA" or "Factorial ANOVA" is just a systematic way to compare a set of nested models, by including and excluding sets of predictors related to factors. We typically don't think of the ANOVA as a test of two specific linear models, but that is exactly what it is doing. In R, the general version of the test–used to compare models, can be invoked using the `anova()` function. So, by giving `anova` two models, it will compute the F test comparing them–provided they are nested–and determine the probability of the observed outcome given the null hypothesis (the smaller/first model).

Let's start by choosing a new model with fewer parameters, but how can we do that? A reasonable approach would be to look at the predictor with the smallest t-value or p-value, but really what we want to know is which resulting smaller model loses us the least amount of predictability. Thus, we can compare each model to a neighboring model with one fewer predictors, using an F-test/ANOVA. The easy way to do all of these models is using the `drop1` function, with the `test="F"` argument:

```
> drop1(lm1,test="F")
Single term deletions

Model:
academic ~ communication + leadership + handson + teaming + priorwork +
    multicultural + creative + group
              Df Sum of Sq    RSS      AIC F value   Pr(>F)
<none>                     183.27 -102.955
communication  1    2.7114 185.99 -100.799  4.0389 0.045446 *
leadership     1    0.0707 183.34 -104.846  0.1053 0.745796
handson        1    0.4743 183.75 -104.223  0.7065 0.401335
teaming        1    2.6238 185.90 -100.932  3.9084 0.049051 *
priorwork      1    4.9749 188.25  -97.375  7.4105 0.006902 **
multicultural  1    1.1248 184.40 -103.223  1.6755 0.196618
creative       1    1.1841 184.46 -103.132  1.7639 0.185250
group          2    7.4194 190.69  -95.724  5.5259 0.004441 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
>
```

Here, we can see the relative change in sum-of-squares, along with the absolute RSS for each adjacent model. The next column shows a related statistic call the AIC (Akaike Information Criterion)–which balances goodness-of-fit and number of parameters, which we will talk about in the next section. Finally, it shows an F value and the probability of the F test showing a result like we saw if the variable had no effect. Based on this, we might consider dropping leadership and handson and repeating the process:

```
lm2 <- lm(academic~communication+teaming+priorwork+multicultural+creative+
    group,data=dat)

summary(lm2)

Call:
lm(formula = academic ~ communication + teaming + priorwork +
    multicultural + creative + group, data = dat)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-2.4230 -0.5262  0.1163  0.5299  1.9868

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    2.23799    0.40793   5.486 9.32e-08 ***
communication  0.15309    0.07704   1.987  0.04790 *
teaming        0.13869    0.07390   1.877  0.06162 .
priorwork      0.13233    0.05093   2.598  0.00987 **
multicultural -0.06243    0.05076  -1.230  0.21975
creative      -0.07043    0.04817  -1.462  0.14484
groupIndustry -0.01583    0.13253  -0.119  0.90500
groupStudent  -0.33539    0.13034  -2.573  0.01060 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.8177 on 275 degrees of freedom
Multiple R-squared:  0.1182,  Adjusted R-squared:  0.09571
F-statistic: 5.264 on 7 and 275 DF,  p-value: 1.174e-05


> drop1(lm2,test="F")
Single term deletions

Model:
academic ~ communication + teaming + priorwork + multicultural +
    creative + group
              Df Sum of Sq    RSS      AIC F value   Pr(>F)
<none>                     183.86 -106.047
communication  1    2.6399 186.50 -104.012  3.9485 0.047904 *
teaming        1    2.3548 186.22 -104.445  3.5220 0.061619 .
priorwork      1    4.5140 188.38 -101.183  6.7515 0.009872 **
multicultural  1    1.0115 184.87 -106.494  1.5129 0.219747
creative       1    1.4294 185.29 -105.855  2.1379 0.144836
group          2    6.9734 190.84  -99.512  5.2150 0.005985 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
>
```

Notice that the model coefficients change a bit between the two models. This will usually happen when you add or remove predictors from a model, unless the predictors are specifically designed to be independent. Now, compare the goodness of fit of these models (R^2 and RSE). How do they differ? They are very close to one another. The first model has all the same predictors as the smaller model, with two additional predictors, and so must have a larger R^2, but maybe it isn't much better. In our case, the improvement is so small it barely changes register on the output. Logically, if two models provide the same fit, but one is more complicated, we should usually prefer the smaller model. But the larger model will usually be a little better. But it still looks like some of the predictors are not important, and drop1 suggests that dropping multicultural and probably creative is justifiable.

Also, notice the the F test p-values of drop1 are exactly the same as the t-test values in the summary of the model. We are really asking the same question in two different ways here. Let's remove multicultural:

```
> lm3 <- lm(academic~communication+teaming+priorwork+creative+group,data=dat)
> summary(lm3)

```

```
Call:
lm(formula = academic ~ communication + teaming + priorwork +
    creative + group, data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4702 -0.5165  0.1198  0.5041  2.0287

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    2.18141    0.40570   5.377 1.62e-07 ***
communication  0.15348    0.07711   1.990   0.0475 *
teaming        0.12913    0.07356   1.755   0.0803 .
priorwork      0.12669    0.05077   2.496   0.0132 *
creative      -0.08869    0.04587  -1.934   0.0542 .
groupIndustry  0.00705    0.13134   0.054   0.9572
groupStudent  -0.31426    0.12932  -2.430   0.0157 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.8184 on 276 degrees of freedom
Multiple R-squared:  0.1133,  Adjusted R-squared:  0.09402
F-statistic: 5.878 on 6 and 276 DF,  p-value: 8.679e-06

> drop1(lm3,test="F")

Single term deletions

Model:
academic ~ communication + teaming + priorwork + creative + group
              Df Sum of Sq    RSS     AIC F value   Pr(>F)
<none>                     184.87 -106.49
communication  1    2.6534 187.53 -104.46  3.9614 0.047543 *
teaming        1    2.0642 186.94 -105.35  3.0816 0.080291 .
priorwork      1    4.1714 189.04 -102.18  6.2275 0.013162 *
creative       1    2.5042 187.38 -104.69  3.7385 0.054195 .
group          2    6.6938 191.57 -100.43  4.9966 0.007385 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
>
```

Now, remember that drop1 is really comparing the given model to every neighboring model with one fewer predictor, and we see that each one is significant. Here, the null hypothesis that the two models are the same and the additional predictors really don't matter. The fact that all comparisons to smaller models are significant tells us that any smaller model produces more error than we'd expect just by chance, so using an ANOVA/t-test criterion, we'd prefer the larger model 3.

Because we only examined model3 with respect to model2, We'd like to be sure that model doesn't lose anything over the most complex model. We can do an ANOVA test between any pairing of models:

```
anova(lm1,lm3)
Analysis of Variance Table

Model 1: academic ~ communication + leadership + handson + teaming + priorwork
    +
    multicultural + creative + group
Model 2: academic ~ communication + teaming + priorwork + creative + group
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
```

```
1     273 183.27
2     276 184.87 -3    -1.6004 0.7947 0.4977
```

Here, we see that the smaller model3 is not significantly different from model1, and so we would still prefer the smaller model3.

We can give `anova()` a series of models, and as long as they are nested, it will compare each consecutive pair. How would you interpret the following:

```
lm4 <- lm(academic~communication+priorwork+creative+group,data=dat)
lm5 <- lm(academic~communication+priorwork+group,data=dat)
lm6 <- lm(academic~priorwork+group,data=dat)
lm7 <- lm(academic~group,data=dat)
lm8 <- lm(academic~1,data=dat)

anova(lm1,lm2,lm3,lm4,lm5,lm6,lm7,lm8)

Analysis of Variance Table

Model 1: academic ~ communication + leadership + handson + teaming + priorwork
     +
    multicultural + creative + group
Model 2: academic ~ communication + teaming + priorwork + multicultural +
    creative + group
Model 3: academic ~ communication + teaming + priorwork + creative + group
Model 4: academic ~ communication + priorwork + creative + group
Model 5: academic ~ communication + priorwork + group
Model 6: academic ~ priorwork + group
Model 7: academic ~ group
Model 8: academic ~ 1

  Res.Df    RSS Df Sum of Sq       F   Pr(>F)
1    273 183.27
2    275 183.86 -2   -0.5889  0.4386 0.645372
3    276 184.87 -1   -1.0115  1.5067 0.220696
4    277 186.94 -1   -2.0642  3.0747 0.080640 .
5    278 188.89 -1   -1.9497  2.9042 0.089487 .
6    279 192.90 -1   -4.0136  5.9786 0.015113 *
7    280 200.07 -1   -7.1706 10.6811 0.001221 **
8    282 208.50 -2   -8.4248  6.2747 0.002166 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1


anova(lm1,lm4,lm8)

Analysis of Variance Table

Model 1: academic ~ communication + leadership + handson + teaming + priorwork
     +
    multicultural + creative + group
Model 2: academic ~ communication + priorwork + creative + group
Model 3: academic ~ 1
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1    273 183.27
2    277 186.94 -4   -3.6646 1.3647    0.2464
3    282 208.50 -5  -21.5587 6.4227 1.162e-05 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
>
```

Each row compares the model to the adjacent model. As we remove predictors, at some point additional predictors become too costly, in that the smaller model is significantly worse than the larger model. This just shows again that we can't really justify the additional complexity of model4. This should make some sense–we are explaining only about 10% of the variance, and so any one predictor is really pretty small, and we probably won't lose much by ignoring it.

This ANOVA/drop1 method requires the to have different numbers of predictors. If we look at at two models that are not nested, it does not really know what to do:

```
lm9 <- lm(academic~communication,data=dat)
lm10 <- lm(academic~creative,data=dat)
anova(lm9,lm10)

Analysis of Variance Table

Model 1: academic ~ communication
Model 2: academic ~ creative
  Res.Df    RSS Df Sum of Sq F Pr(>F)
1    281 202.30
2    281 206.24  0   -3.9375
```

However, in this case, because the number of predictors are the same and one model predicts better, you might prefer that model (in this case, lm9), but we aren't doing an F test to determine this.

Also, we typically only use anova tests for nested models. Consider comparing lm10 to lm5:

```
> anova(lm5,lm10)
Analysis of Variance Table

Model 1: academic ~ communication + priorwork + group
Model 2: academic ~ creative
Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1    278 188.89
2    281 206.24 -3   -17.351 8.5121 1.989e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can still perform the F test, but the basic logic no longer completely makes sense. Here, we see a significant difference, suggesting we'd prefer the more complex model 5 versus the simpler model 10–but that doesn't tell us which predictors of model 5 are better, or whether the predictor of model 10 is not any good.

Exercise 11.2

Load the galapogos data set from the faraway library (its name is gala). Make a model predicting the number of species (Species) on each island based on all the other variables. Remove variables one at a time, until you have the smallest justifiable model.

11.2.1 Parameter selection/Model comparison using AIC and BIC

The drop1 also gave us a statistic called AIC (Akaike Information Criterion). This criterion asks whether adding or removing a parameter is worthwhile with respect to the increased or

decreased goodness of fit. The way the AIC is framed here, the more negative the better. So we can see if we remove group, the fit gets so much worse that it the AIC says it is worth keeping that variable. The SS, RSS, and AIC indicate that leadership and hands-on may not really matter, so let's drop those and redo the model.

The AIC is a common way of deciding whether to remove a variable. There are other information criteria as well–the most popular is the Bayesian Information Criterion (BIC; Schwarz et al. 1977). The BIC is related to AIC but incorporates the number of observations you have, and so is generally a bit more conservative about adding additional parameters, as each additional parameter essentially requires a larger sample size to justify. The AIC score is sometimes reported positive and sometimes negative depending on the software, so you have to be careful to interpret it correctly. The absolute value is hardly ever interpretable, instead we look for relative values between two models. The drop1 gives an AIC score where the more negative (less positive) the score, the better the model.

If we revisit he AIC values for consecutive lm1, we see:

```
> drop1(lm1,test="F")
Single term deletions

Model:
academic ~ communication + leadership + handson + teaming + priorwork +
    multicultural + creative + group
              Df Sum of Sq    RSS      AIC F value   Pr(>F)
<none>                     183.27 -102.955
communication  1    2.7114 185.99 -100.799  4.0389 0.045446 *
leadership     1    0.0707 183.34 -104.846  0.1053 0.745796
handson        1    0.4743 183.75 -104.223  0.7065 0.401335
teaming        1    2.6238 185.90 -100.932  3.9084 0.049051 *
priorwork      1    4.9749 188.25  -97.375  7.4105 0.006902 **
multicultural  1    1.1248 184.40 -103.223  1.6755 0.196618
creative       1    1.1841 184.46 -103.132  1.7639 0.185250
group          2    7.4194 190.69  -95.724  5.5259 0.004441 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
>
```

Here, the AIC of the standard model is -102.955. Any time we drop a parameter, the two components of the AIC change. It gets a little better (more negative) because we are using fewer parameters, but gets a little worse (less negative) because the likelihood of the model goes down. These together result in the AIC for each new model In this case, removing leadership, handson, multicultural, and creative each produce AIC values that are a little better (more negative), so we'd like prefer models without these. We can follow the same process as with F tests, but use the AIC at each point. Rather than using drop, we can get the AIC value using extractAIC from each model. You can obtain the AIC for any model like this:

```

data.frame(model=paste("lm",1:10,sep=""),
           rbind(extractAIC(lm1),
extractAIC(lm2),
extractAIC(lm3),
extractAIC(lm4),
extractAIC(lm5),
extractAIC(lm6),
extractAIC(lm7),
extractAIC(lm8),
extractAIC(lm9),
```

```
extractAIC(lm10)))
13
 model X1          X2
15 1     lm1 10 -102.95477
   2     lm2  8 -106.04683
17 3     lm3  7 -106.49417
   4     lm4  6 -105.35191
19 5     lm5  5 -104.41563
   6     lm6  4 -100.46527
21 7     lm7  3  -92.13634
   8     lm8  1  -84.46352
23 9     lm9  2  -91.00088
   10   lm10  2  -85.54560
```

Note that doing this produces the same AIC values that drop1 did. Looking across the AIC values, the models got better (more negative) through lm3, but then started getting worse. This is a similar conclusion to what we found with F-tests.

To compute the value, the AIC (and BIC) measures start with a value known as log-likelihood. Technically, for b predictors,

$$AIC = 2b - 2ln(Likelihood) \tag{11.1}$$

To compute the likelihood of a model, you take the product of the likelihood of each observed data point. The likelihood of an observed data point is the height of the density function of the error distribution at that point, with an assumption about the particular form of the error distribution. For normal linear regression we would typically assume a normal distribution. So, this is the density of the model's prediction for each observed point, and the likelihood of the model is the product of the likelihoods of each data point. We use log-likelihood because the log of a product is the sum of the logs, which is easier to calculate. Models that are better will have a larger (more positive and or less negative) log-likelihood, but because AIC subtracts this from the total, the more negative the better. The AIC takes $2b$ as a baseline, where b is the number of predictors. Consequently, AIC essentially punishes likelihood for each parameter used.

In contrast, BIC also considers the size of the experiment, and punishes for the total number of parameters b multiplied by the logarithm of the number of observed data points n.

$$BIC = bln(n) - 2ln(Likelihood) \tag{11.2}$$

BIC punishes each additional parameter more than AIC does (as long as n is greater than 7). Consequently, BIC is usually more conservative than AIC and will tend to select fewer parameters, and a smaller overall model. Note that the formula for BIC gets larger (worse) as n increases. This is a bit counterintuitive because it suggests that BIC will get worse by conducting a larger experiment, but AIC and BIC are only useful for a fixed data set in which n does not change. Furthermore, log-likelihood will also get higher as more data is added to a data set, so the $ln(n)$ balances this out.

BIC can be computed from `extractAIC` and other R functions by feeding it a `k` argument of $ln(N)$ (which in R is `log(n)`). Note that in extractAIC, k refers to the value that the number of predictors gets multiplied by.

```
2 extractBIC <- function(model)
  {
```

```
  extractAIC(model,k=log(length(model$residuals)))
}


data.frame(model=paste("lm",1:10,sep=""),
           rbind(extractBIC(lm1),
                 extractBIC(lm2),
                 extractBIC(lm3),
                 extractBIC(lm4),
                 extractBIC(lm5),
                 extractBIC(lm6),
                 extractBIC(lm7),
                 extractBIC(lm8),
                 extractBIC(lm9),
                 extractBIC(lm10)))



  model X1        X2
1    lm1 10 -66.50030
2    lm2  8 -76.88325
3    lm3  7 -80.97604
4    lm4  6 -83.47923
5    lm5  5 -86.18840
6    lm6  4 -85.88349
7    lm7  3 -81.20000
8    lm8  1 -80.81807
9    lm9  2 -83.70999
10  lm10  2 -78.25470

lm5

Call:
lm(formula = academic ~ communication + priorwork + group, data = dat)

Coefficients:
  (Intercept)  communication      priorwork  groupIndustry   groupStudent
     2.305585       0.169360       0.150266      -0.004061      -0.358903
```

In this case, the best BIC criteria suggests model 5, which has just communication, prior work, and group as predictors. This turned out to be the smallest model we have selected so far.

11.2.2 Stepwise variable selection

When you have a large model, you have a large number of submodels to possible compare. This forms a lattice of models, and with enough parameters you won't be able to check them all. A common strategy for searching among these models is to use a stepwise procedure, which will often use an AIC or BIC criterion to compare models. In this procedure, you pick a model starting point, and then compare to all models with one more and one fewer parameter. One of these models may provide a better AIC value than your current model. If so, you move to that model; otherwise you choose the model you last tested. This can sometimes get caught finding a local optimum, so you may want to repeat this process from different starting models.

The '`step`' function (and the `stepAIC` function in the `MASS` package) will use AIC to selectively add, prune, or move both ways within the model space to find the 'best' model according to that criterion. Note that with N parameters, the total number of models is 2^N,

and it may quickly be impossible to test all possible alternatives, so you are not guaranteed to find the truly best model. The function `step()` lets you specify a direction, which can be "both", "backward", or "forward". You should typically try "both", but backward and forward selection let you either move only to smaller or larger models from a starting point. As with `extractAIC`, by specifying the k parameter as `log(n)`, you can get step to use BIC.

To use step, give it a complex model and watch it iterate until it stops. Giving it a k value that computes BIC, this will produce the same results as the BIC analysis we already did, but it will do so automatically.

```
 gsmall <- step(g1,direction="both", k=log(nrow(dat)))

> summary(gsmall)

Call:
lm(formula = academic ~ communication + priorwork + group, data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-2.5609 -0.5448  0.1003  0.5587  2.0562

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    2.305585   0.364619   6.323 1.02e-09 ***
communication  0.169360   0.069682   2.430  0.01571 *
priorwork      0.150266   0.050037   3.003  0.00292 **
groupIndustry -0.004061   0.131958  -0.031  0.97547
groupStudent  -0.358903   0.128505  -2.793  0.00559 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.8243 on 278 degrees of freedom
Multiple R-squared:  0.09405, Adjusted R-squared:  0.08101
F-statistic: 7.215 on 4 and 278 DF,  p-value: 1.534e-05
```

11.3 Using Bayes Factor for Model Selection

The logic of the Bayes Factor parameter selection is similar to the BIC, with fairly similar motivation, but overall it is often somewhat simpler to manage. Because each model will have a Bayes Factor associated with it that compares it to the intercept-only model, comparing nested models is simply comparing the ratio of these factors. We need to run each model using a Bayes factor test first:

```
library(BayesFactor)
b1 <- lmBF(academic~communication+leadership+handson+teaming+priorwork+
    multicultural+creative+group,data=dat)
plot(b1)
b3 <- lmBF(academic~communication+teaming+priorwork+creative+group,data=dat)
b3/b1

b3/b1
Bayes factor analysis
--------------
[1] communication + teaming + priorwork + creative + group : 14.20427 1.57%

Against denominator:
```

```
  academic ~ communication + leadership + handson + teaming + priorwork +
    multicultural + creative + group
---
Bayes factor type: BFlinearModel, JZS
```

Here, the test prefers the smaller model by a factor of 14. We can compute pairwise comparisons, but a slightly easier way to do this is to compare all sub-models using regressionBF. However, this will not accept factor variables, so we will have to get rid of group. There are 127 remaining models and submodels, and so we will just look at the best few:

```
bmodel <- regressionBF(academic~communication+leadership+handson+teaming+
    priorwork+multicultural+creative,data=dat)

plot(head(bmodel))
head(bmodel)
Bayes factor analysis
--------------
[1] communication + teaming + priorwork + creative                  : 92.49299
    0%
[2] communication + priorwork + creative                            : 65.31011
    0.01%
[3] communication + teaming + creative                              : 51.61002
    0.01%
[4] communication + teaming + priorwork + multicultural + creative : 39.729
    0%
[5] teaming + priorwork + creative                                  : 35.22326
    0.01%
[6] communication + priorwork                                       : 33.54419
    0%

Against denominator:
  Intercept only
---
Bayes factor type: BFlinearModel, JZS
```

Here, we would prefer the model with communication, teaming, priorwork, and creative, which is the same as model3 (but without including group).

11.4 Parameter Selection when using Regression for Prediction

When we fit a model using `lm`, the predicted values for our observed cases are saved in the $fitted.values slot of the model. However, we can also make predictions about new cases that we did not observe. You need to be careful–there is no guarantee you will be as accurate for new data. You should be especially cautious if you are extrapolating to a new region of the parameter space. For example, maybe you found a relationship between age and working memory for a group of school-age adolescents, such that the score on a 10-point test goes up .5 points per year. It would be a mistake to assume that the score of a 30-year-old would by 10 points higher than a 10-year-old.

To examine prediction, suppose the truth involved a simple linear relationship between age, income, and digit span.

```
##normal prediction
```

```
set.seed(100)
age <- 20+runif(50) * 80
income <- 20000+ runif(50)*50000
digitspan <-  ( age * .5 + income * .003)/20 + rnorm(50)
data=data.frame(age,income,digitspan)
model <- lm(digitspan~age+income,data=data)
summary(model)


Call:
lm(formula = digitspan ~ age + income, data = data)

Residuals:
     Min       1Q   Median       3Q      Max
-2.14574 -0.90181  0.02403  0.69711  2.75379

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.2071955  0.7907307   0.262  0.79444
age         0.0266162  0.0084632   3.145  0.00288 **
income      0.0001414  0.0000129  10.962 1.52e-14 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 1.211 on 47 degrees of freedom
Multiple R-squared:  0.7393,  Adjusted R-squared:  0.7282
F-statistic: 66.65 on 2 and 47 DF,  p-value: 1.9e-14
```

Note that we can get the fitted values with `model$fit`, which is the same as what the predict function will produce if given the model and our original data set:

```
model$fit
predict(model,data)
cor(model$fit,predict(model,data))
1.0
```

But what if we wanted to predict the digit span for someone who was 30 years old and had an income of 10,000/year? To do this, create a custom data frame or list that has named values with each of the predictor names. You can give it an entire data set to make predictions about too–maybe for validation or other means.

```
predict(model,list(age=30,income=10000))
2.260161
```

It will do this, but be careful, because nobody in our sample had income this low.

```
predict(model,list(age=30,income=40000))
6.795762
```

This is our best prediction about the value we would have observed. Is it correct? It will only be approximately correct considering the entire range of the data we observed.

It is hardly ever the case in which we know all the right things to measure, and none of the wrong things, in order to best predict our outcome. Let's consider what would happen if we have measured a lot of uninteresting values that don't really predict the outcome.

```
#WHAT IF WE HAD OTHER VARIABLES TOO?
set.seed(1001)
n <- 50
xcat <- as.factor(sample(c("A","B","C"), n,replace=T))
x2 <- runif(n)
x3 <- runif(n)
x4 <- 1/runif(n)
x5 <- runif(n)
x6 <- runif(n)
x7 <- runif(n)

y <- as.numeric(xcat)*3 + 5*x2 - 10*x3 + rnorm(n)*2.5

#optional: add additional large noise to a handful of points
#y[sample(1:n,5)] <- rnorm(5,mean(y),sd(y))

dat <- data.frame(xcat,x2,x3,x4,x5,x6,x7,y)
model <- lm(y~0+xcat+x2+x3+x4+x5+x6+x7,data=dat)


> summary(model)

Call:
lm(formula = y ~ 0 + xcat + x2 + x3 + x4 + x5 + x6 + x7, data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-4.431 -1.468   0.010   1.415   5.946

Coefficients:
        Estimate Std. Error t value Pr(>|t|)
xcatA    0.759131   1.452895   0.522   0.6041
xcatB    3.272940   1.529143   2.140   0.0383 *
xcatC    7.368993   1.512420   4.872 1.69e-05 ***
x2       6.753986   1.180098   5.723 1.07e-06 ***
x3     -10.937376   1.251518  -8.739 6.56e-11 ***
x4       0.008137   0.022416   0.363   0.7185
x5       1.124921   1.210535   0.929   0.3582
x6       0.074951   1.273840   0.059   0.9534
x7       3.046021   1.303627   2.337   0.0244 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 2.355 on 41 degrees of freedom
Multiple R-squared:  0.9021,  Adjusted R-squared:  0.8807
F-statistic:    42 on 9 and 41 DF,  p-value: < 2.2e-16
```

We could predict a specific value from the data

```
predict(model,dat[1,])
dat[1,]
```

Or a new value. what if it had been a "B" but otherwise the same:

```
tmp <- dat[1,]
tmp$xcat="B"

predict(model,tmp)
```

Or this could work for a completely new value, using a list or data that contains named variables with all predictor variables:

```
predict(model,list(xcat="A",x2=.8,x3=.01,x4=1.5,x5=3,
                   x6=1.5,x7=1.3))
```

Because we created the data, we know that x4..x7 shouldn't matter. But the model is using these extra predictors, and so by using them, it may harm its ability to predict data that were not in the data set. Ideally, we would have removed variables that did not matter, so this might not be a case. But of course, had we removed those variables, we might have removed some variables that did matter as well.

In addition to all the model selection methods we used earlier, you can also use cross-validation. In this scheme, you fit the model on subset of your data, and then predict the other subset. Different schemes might do this repeatedly with different subsets, or use a fitting subset of N-1 points, predicting the leftover point each time. In the example below, we know that modelc is the correct model because we created the data.

```

set.seed(8)
reorder <- (1:n)[order(runif(n))]
half<- floor(n/2)
fitset <- reorder[1:half]
extra <- reorder[(half+1):n]

##fit the model on a random half of the data:
modelb <- lm(y~0+xcat+x2+x3+x4+x5+x6+x7,data=dat[fitset,])
modelc <- lm(y~0+xcat+x2+x3,data=dat[fitset,])

summary(model)
summary(modelb)
summary(modelc)

> summary(model)$r.squared
[1] 0.8841388
> summary(modelb)$r.squared
[1] 0.9164083
> summary(modelc)$r.squared
[1] 0.8527432
```

The exact results depend on the random data set, but here we see that the R^2 for model b is .92 and for modelc is .85. As you might expect, the over–specified model b fits better. Are the actual parameters any different?

```

params <- rbind(model$coef,
   modelb$coef,
   c(modelc$coef,0,0,0,0),
   c(3,6,9,5,-10,0,0,0,0))

>  round(params,3)
      xcatA xcatB xcatC    x2      x3    x4    x5    x6    x7
[1,]  0.759 3.273 7.369 6.754 -10.937 0.008 1.125 0.075 3.046
```

```
[2,] -2.654 1.072 5.581 8.476 -11.256 0.130 1.194 0.899 4.490
[3,]  1.874 4.766 9.485 7.527 -11.319 0.000 0.000 0.000 0.000
[4,]  3.000 6.000 9.000 5.000 -10.000 0.000 0.000 0.000 0.000
```

The last row is the true values, but overall different models give different estimates. So, to be fair, let's see what happens if we predict the left-out subset based on the original model (which used these data to produce the fit):

```
>  cor(dat$y[fitset],predict(model,dat[fitset,]))^2
[1] 0.7984062
>  cor(dat$y[extra],predict(model,dat[extra,]))^2
[1] 0.8421875
```

Here, the fit was pretty good for each half. The model was fitted to the entire data set, and so if we look at half of the data, we'd expect each to fit about the same; here the second half is a bit better than the first half, and we can use these values as a baseline.

Now, if we do the same for our complex over-fitted model:

```

> cor(dat$y[fitset],predict(modelb,dat[fitset,]))^2
[1] 0.8607574
> cor(dat$y[extra],predict(modelb,dat[extra,]))^2
[1] 0.6170834
```

Now, the fitted value for the fitting set is a bit better than the first model, but the the fit on the cross-validation is worse. Let's look at the smaller model:

```
> cor(dat$y[fitset],predict(modelc,dat[fitset,]))^2
[1] 0.7547074
> cor(dat$y[extra],predict(modelc,dat[extra,]))^2
[1] 0.7752517
```

Here, it fits both halves of the data about as well as the model fitted to the entire data set. So, by removing predictors, although the fit to the data we see goes down a bit, the fit to data we don't see should improve. Of course, this assumes that the smaller model you select is really a better model. In the following exercise, use variable selection methods to pick a smaller model, and see if you come up with one identical to model c.

Exercise 11.4

Create a model predicting y using all the x values. Use one of the following model selection approaches to find the smallest model that is predictive: F-tests; AIC, BIC, Bayes Factor, or cross-validation.

11.4.1 Predicting Categorical Variables

Although the linear model is not typically used to predict categories, it can sometimes be used in this way. There are better methods of course, including logistic regression and discriminant analysis, but these are essentially versions of regression with certain transformations applied.

If we have a two-level outcome variable such as gender, or any other binary outcome, we can code these, for example, as 1 and 2. When we build a model, the outcome predicted value will be a continuous number, and we pick a criterion between the two (maybe 1.5) to make a decision about our best guess. In fact, there are special versions of the regression model that might be better (logistic regression), but a normal regression can still work.

```
gender <- sample(as.factor(c("F","M")),50,replace=T)
height <- 30 + as.numeric(gender) * 10 + runif(50)*30  ##height depends on
    gender
weight <- 100 + as.numeric(gender) * 20 + runif(50)*50  ##weight depends on
    gender too

plot(height,weight,col=gender,pch=16)
```

Notice that gender is related to both height and weight, although neither one alone is perfect at discriminating the two. If we build a model by turning gender into a numeric, it will code female=1, male =2 (alphabetical order).

```
model <- lm(as.numeric(gender)~height+weight)
model
plot(model$fit~gender,ylab="Gender coefficient")
points(as.numeric(gender),model$fit)
abline(1.5,0,lwd=3)
```

Our prediction about gender is a real-valued number that in this case ranges between about 0.8 and 2.2. If we use 1.5 as the cutoff, we can use a table to see how well we did (since we know what the true outcome is).

```
predictedgender <- model$fit > 1.5
table(gender,c("F","M")[(predictedgender+1)])

gender  F  M
  F 20  5
  M  4 21
```

In this case, we got 41/50 correct–82%.

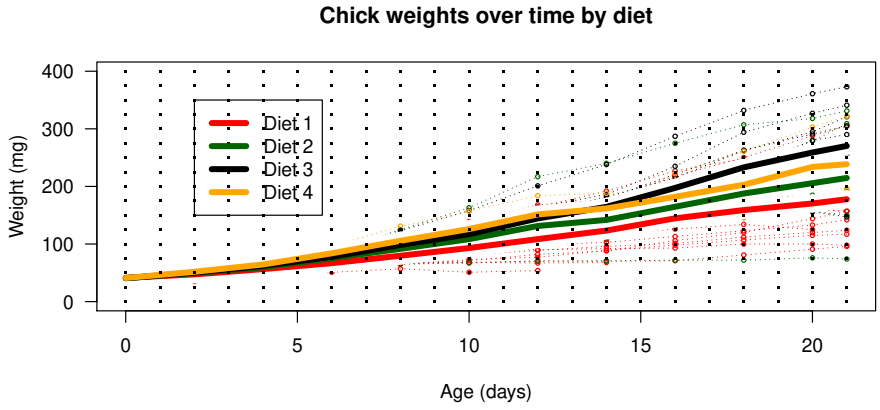
11.5 Worked Example: Categorical and linear predictors

Remember back to the chickweight data set, which we only looked at graphically. It tracked the growth of about 50 chicks from birth to 21 days, with four different feeds. The growth curves would be a good target for linear regression. But how do we incorporate the feed? They could be labeled 1 through 4, but they really have no order, and are categorical predictors.

The plotchicks function incorporates the plotting we defined in Chapter 3.

```
summary(ChickWeight)
plotchicks <- function()
```

Figure 11.1: Depiction of the chick weight data set, for four different diets across the first 21 days of life.




```
  {
  cscheme <- c("red","darkgreen","black","orange")
  ##write down the times of measurement--they are not regular
  times <- (c(0,2,4,6,8,10,12,14,16,18,20,21))

  ##Aggregate growth over the diets and times

  cw.agg <- tapply((ChickWeight$weight), list(time=ChickWeight$Time,
  diet=ChickWeight$Diet),mean)
  cw.bysub<- tapply((ChickWeight$weight),
  list(time=ChickWeight$Time,
  chick=ChickWeight$Chick),mean)

  diets <- aggregate(as.numeric(as.character(ChickWeight$Diet)),
  list(chick=ChickWeight$Chick),median)

  #Replot the growth curves, but and make reasonable headers
  matplot(times,cw.bysub,col=cscheme[diets$x],type="l",lty=3,
  ylab="Weight (mg)",xlab="Age (days)",ylim=c(0,400),las=1)
  points(ChickWeight$Time,ChickWeight$weight,cex=.5,
  pch=1, col=cscheme[ChickWeight$Diet])

  #Underlay a white line
  matplot(times,cw.agg,add=T,type="l",lwd=15, col="white",lty=1)
  matplot(times,cw.agg,add=T,type="l",lwd=5, col=cscheme,lty=1)
  ##Whoops, no title:
  title("Chick weights over time by diet")
  ##Lets add some gridpoints
  points(rep(-10:25,each=17), rep(0:16*25,36),pch=".",cex=2)

  ##Make a legend here.
  legend(2,350,paste("Diet",c(1:4)),col=cscheme,lty=1,lwd=5)
}

plotchicks()
summary(ChickWeight)
```

To start with, we might be able to answer a lot of questions we have without a linear model. We might want to know a few things about the chicks:

1. On average, are chicks from one diet heavier than another?
2. Do the diets give a higher end-weight?
3. Do the diets give chicks different growth rates?

The first question is easy (or is it?).

```
cwd <- ChickWeight$Diet
t.test(ChickWeight$weight[cwd==1],ChickWeight$weight[cwd==2])
t.test(ChickWeight$weight[cwd==1],ChickWeight$weight[cwd==3])
t.test(ChickWeight$weight[cwd==1],ChickWeight$weight[cwd==4])
t.test(ChickWeight$weight[cwd==2],ChickWeight$weight[cwd==3])
t.test(ChickWeight$weight[cwd==2],ChickWeight$weight[cwd==4])
t.test(ChickWeight$weight[cwd==3],ChickWeight$weight[cwd==4])

```

But there are problems with this approach: a. Is this really what we want? b. Multiple tests start reducing the validity of a .05 probability; and c. d.f. are around 230–Is this right?; and d. we aren't factoring in or factoring out the growth curve, and the repeated measurements of each chick. We shouldn't get credit for measuring each chick 12 times–had we measured every day instead of every other day, we shouldn't really get the same benefit as if we observed twice as many chicks. Let's ignore this issue for the moment though, and look at the results of the pairwise t tests.

According to the results, we get reliable differences 4/6 times:

```
   2    3    4
1  *    *    *
2       *    0
3            0

```

Because diet is coded as a factor (rather than an integer), let's try a regression model that essentially corresponds to these t-tests. To do this, we will just give Diet and a predictor:

```
lm1 <- lm(weight~Diet,data=ChickWeight)
summary(lm1)

```

```
plotchicks()
abline(102,0,col="red",lwd=3)
abline(102+19.97,0,col="darkgreen",lwd=3)
abline(102+40.3,0,col="black",lwd=3)
abline(102+32.62,0,col="orange",lwd=3)

```

This is basically like a one-way factorial ANOVA, with each effect of Diet coded relative to Diet 1. Typically, an ANOVA procedure will hide these parameter estimates from you, and packages like SPSS and SAS may code each level with respect to the last level instead of the first.

Without going into too much detail, we will probably find close to the same reliable outcomes, but now we can see that our R^2 is abysmally low–.048. There is so much more variability in the data we aren't accounting for that it silly to compare means.

11.5.1 Compare end weight or weight gain or end/start ratio

As discussed earlier, we need to find a single value per chick, not their whole history:

```
##Pick out the diet of each chick:
diets <- tapply(ChickWeight$Diet,list(ChickWeight$Chick),function(x)x[[1]])

minweights <- tapply(ChickWeight$weight,list(ChickWeight$Chick),min)
maxweights <- tapply(ChickWeight$weight,list(ChickWeight$Chick),max)

##Let's recategorize by weight gains
ratio <- maxweights/minweights
gain <- maxweights-minweights

hist(ratio,breaks=20)
hist(gain,breaks=20)
plot(ratio,gain)

##It hardly matters
t.test(gain[diets==1],gain[diets==2])
t.test(gain[diets==1],gain[diets==3])
t.test(gain[diets==1],gain[diets==4])
t.test(gain[diets==2],gain[diets==3])
t.test(gain[diets==2],gain[diets==4])
t.test(gain[diets==3],gain[diets==4])

plotchicks()

end <- tapply(gain,list(diets),mean)

abline(end[1],0,col="red",lwd=3 )
abline(end[2],0,col="darkgreen",lwd=3 )
abline(end[3],0,col="black",lwd=3 )
abline(end[4],0,col="orange",lwd=3 )
```

Why don't these line up with the end weights? This factored out starting weights.

```
plotchicks()
start <- tapply(minweights,list(diets),mean)
end <- tapply(gain,list(diets),mean)

abline(end[1]+start[1],0,col="red",lwd=3 )
abline(end[2]+start[2],0,col="darkgreen",lwd=3 )
abline(end[3]+start[3],0,col="black",lwd=3 )
abline(end[4]+start[4],0,col="orange",lwd=3 )
```

According to these new results, we get fewer reliable comparisons:

```
        2    3    4
1   .    *    *
2        .    0
3             0
4
5
```

This is partially because the d.f. are correct, and starting weights were subtracted out. There were dozens of chicks, not hundreds, and you are trying to do inference about the population of chicks, not the population of measurements. Importantly, we can assume the error in measurement is very small, but the variability in chick weight stemming from genetic variability across the population is larger.

So let's put this in a linear model, and while we are at it, incorporate a 'growth' factor:

```
lm2 <- lm(weight~ Time+Diet,data=ChickWeight)
summary(lm2)

plotchicks()
abline(lm2$coef[1],lm2$coef[2],col="red",lwd=3)
abline(lm2$coef[1]+lm2$coef[3],lm2$coef[2],col="darkgreen",lwd=3)
abline(lm2$coef[1]+lm2$coef[4],lm2$coef[2],col="black",lwd=3)
abline(lm2$coef[1]+lm2$coef[5],lm2$coef[2],col="orange",lwd=3)

```

But, this is not really appropriate, because it essentially assumes that the chicks started out at different weights, then gained the same amount regardless of diet, and only lets the starting weights change.

Here, we need to make an 'interaction' predictor: time x diet. This will force a single intercept across all groups, and a different slope for each:

```
lm3 <- lm(weight~Time:Diet,data=ChickWeight)
summary(lm3)
```

```
plotchicks()
abline(lm3$coef[1],lm3$coef[2],col="red",lwd=5,lty=4)
abline(lm3$coef[1],lm3$coef[3],col="darkgreen",lwd=5,lty=4)
abline(lm3$coef[1],lm3$coef[4],col="black",lwd=5,lty=4)
abline(lm3$coef[1],lm3$coef[5],col="orange",lwd=5,lty=4)

```

We can include all intercepts by using the * instead of the, which tells it to use both main effects and the interaction:

```
lm4 <- lm(weight~Time*Diet,data=ChickWeight)
summary(lm4)

```

11.5.2 Exercise

Plot the predicted lines for lm4.

11.5.3 Categorical outcome variables

Suppose a chicken farmer gets more money for 'jumbo' chicks; those above 300g; but cannot sell 'runts', those below 150 g. She may not care about any of this modeling, and may only be concerned with whether the feed impacts her bottom line. We can transform the DV into those categories, ignore the models, and just do a simple chi-squared test to see the answer.

```
weightgain <- ifelse(maxweights>=300,"jumbo",
ifelse(maxweights<=150,"runt","normal"))
chickTable <- table(diets,weightgain)
## Does the type of chick depend of feed?
chisq.test(chickTable)
```

Suppose diet 2 cost half as much as diet3. Is there a reliable difference between these two? Is it enough to make the more expensive feed worthwhile?

Exercise Solution 11.2

We can see this by examining the galapagos data set, in the Faraway library:

```
library(faraway)
```

First, let's look at the full model, as well as one without the Scruz predictor:

```
g1 <- lm(Species~.,data=gala)
g2 <- lm(Species~Endemics+Area+Elevation+Nearest+Adjacent,data=gala)

> summary(g1)

Call:
lm(formula = Species ~ ., data = gala)

Residuals:
    Min      1Q  Median      3Q     Max
-68.219 -10.225   1.830   9.557  71.090

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -15.337942   9.423550  -1.628    0.117
Endemics      4.393654   0.481203   9.131 4.13e-09 ***
Area          0.013258   0.011403   1.163    0.257
Elevation    -0.047537   0.047596  -0.999    0.328
Nearest      -0.101460   0.500871  -0.203    0.841
Scruz         0.008256   0.105884   0.078    0.939
Adjacent      0.001811   0.011879   0.152    0.880
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 28.96 on 23 degrees of freedom
Multiple R-squared: 0.9494, Adjusted R-squared: 0.9362
F-statistic: 71.88 on 6 and 23 DF,  p-value: 9.674e-14
```

Exercise Solution 11.2 Continued

Here, the F-statistic is 71, $p < .001$. This indicates that the goodness of fit we observed was very unlikely to have happened by chance. Because of this, the larger model is better than the small (mean-only) model, and we would prefer using the larger model, even though it is more complex.
What if we removed scruz:

```
> summary(g2)
Call:
lm(formula = Species ~ Endemics + Area + Elevation + Nearest +
    Adjacent, data = gala)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -15.056413   8.522230  -1.767    0.090 .
Endemics      4.383999   0.455268   9.629 1.02e-09 ***
Area          0.013207   0.011146   1.185    0.248
Elevation    -0.046921   0.045953  -1.021    0.317
Nearest      -0.077612   0.388341  -0.200    0.843
Adjacent      0.001763   0.011615   0.152    0.881
---

Residual standard error: 28.36 on 24 degrees of freedom
Multiple R-squared: 0.9494, Adjusted R-squared: 0.9388
F-statistic: 89.98 on 5 and 24 DF,  p-value: 9.61e-15
```

Chapter 12

Identifiability, Orthogonality, linear independence, and Multi-colinearity in Regression Models

Supplemental reading: Faraway's chapters 3.6, 3.7, and 3.9

12.0.1 Terminology

The concepts of identifiability, orthogonality, linear independence, and correlated/uncorrelated predictors are all related. In addition, predictors that are highly correlated are sometimes referred to as "multi-colinear" (or more commonly multicollinear, for which nobody can give a sensible explanation of the spelling). As explained by Rodgers et al (1984)[1], linear independence is the most general term, and both uncorrelated predictors and orthogonal predictors are linearly independent. When forming a regression model, if you do not have linearly-independent predictors, you do not have identifiability, but even if you have linear independence, your predictors may still be correlated and non-orthogonal. Regression models work best when predictors are orthogonal and uncorrelated.

Although these are all concepts related to independence, none of these are the same as "stochastic independence", which is usually what is meant when one says two (random) variables are independent. Stochastic independence is related to the distribution of random variables (if the joint density is equal to the product of the marginal distributions). Each of the concepts in this chapter relates specifically to predictor variables, which may be data, and may arise by sampling from a random variable, but might be selected or designed as well. I'll try to avoid using the term "independence" because of its many connotations to avoid the possibility of confusion.

[1] J. L. Rodgers, W. A. Nicewander, & L. Toothaker (1984). Linearly independent, orthogonal, and uncorrelated variables. The American Statistician, 38, 133-134.

12.1 Orthogonality of Predictors

The term orthogonal is a geometric generalization of the notion of perpendicularity. In two-dimensional space, two vectors with a 90◦ angle between them are orthogonal. Consider a pair of vectors that each start at $(0, 0)$, where $\vec{a}$ ends at $(1, 0)$ and $\vec{b}$ ends at $(0, 1)$. Figure 12.1 shows these in red. Notice that if you multiply each element of $\vec{a}$ and $\vec{b}$ by one another and add up the result, the sum is 0: $1 \times 0 + 0 \times 1 = 0 + 0 = 0$. This kind of multiplication is known as the "dot product" or "inner product" of a pair of vectors, and can be done in R using the `%*%` operator.

```
a <- c(1,0)
b <- c(0,1)
a %*% b
     [,1]
[1,]    0

plot(0,0,type="n")
segments(0,0,1,0)
segments(0,0,0,1)
```

Also, notice that vectors $\vec{c}$ and $\vec{d}$ are also perpendicular, and also that their dot product is 0.

```
 c <- c(-1,1)
 d <- c(-1,-1)
 c %*% d
     [,1]
[1,]    0


plot(0,0,type="n")
segments(0,0,-1,1)
segments(0,0,-1,-1)
```

Figure 12.1: Two pairs of orthogonal vectors: $\vec{a}$ and $\vec{b}$, and $\vec{c}$ and $\vec{d}$.

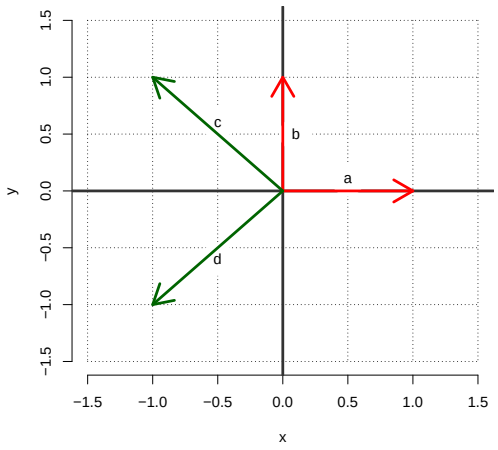
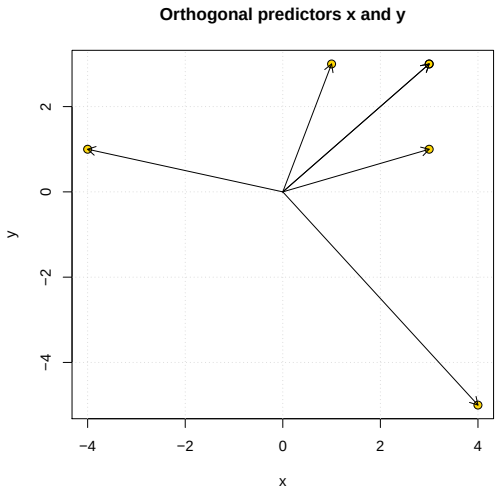


Figure 12.2: Orthogonal predictors plotted x versus y



By definition, the dot product of two orthogonal vectors is always 0, regardless of the number of dimensions. It is not obvious that the following vectors are orthogonal, but they are.

```
x <- c(1,3,3,4,-4,3)
y <- c(3,1,3,-5,1,3)
x %*% y
     [,1]
[1,]    0

plot(x,y)
```

Because they have six dimensions, we cannot visualize this like we looked at the previous vectors. We can plot them *against* one another, but this gives us no indication that they are independent, as shown in the figure. Note that this graph is different from the previous one, because I'm plotting each point of each dimension on the same axis instead of different axes. We can only really tell these are orthogonal because the dot product is 0. If you multiply each pair and add them up, you will find that they cancel out and their sum is 0.

As we will see next, orthogonality makes for a more interpretable linear model, so it is good to have in a set of predictors. Here, x and y are the predictor or design variables of a data set that are used in a linear model (or other analysis). If x and y are conditions under which you are testing an experiment, you can always find a value for a pair of x and y that you could add to your design to make the resulting design orthogonal, and then add these values to your study (any two values whose product is equal to the negative of the dot product of the current set). The problem is we often do not have control over the x and y conditions, and even if we do, we may not be able to feasibly or physically collect that data. Alternatively, we can sometimes make our predictors orthogonal by changing the numbers we use to describe the levels—usually by centering (subtracting the mean). This is usually easier, but not guaranteed to work.

This is because two predictors that are orthogonal can be correlated. As described by Rodgers et al. (1984), two vectors are uncorrelated if their dot product is 0 once the mean of each vector is subtracted from the vector. For example, the following vectors `orthx1` and `orthx2` are orthogonal but correlated:

```
orthx1 <- c(1,-5,3,-1)
orthx2 <- c(5,1,1,3)

> cor(orthx1,orthx2)
[1] 0.2548236

orthx1 %*% orthx2
     [,1]
[1,]    0
```

But since vectors that are uncorrelated are orthogonal if you subtract their means, you can always transform uncorrelated predictors to orthogonal predictors by decentering so their means are 0. For example, the following are uncorrelated but non-orthogonal vectors:

```
uncorx1 <- c(0,0,1,1)
 uncorx2 <- c(1,0,1,0)
> cor(uncorx1,uncorx2)
[1] 0
> (uncorx1)%*%(uncorx2)
```

```
     [,1]
[1,]    1
```

But simply subtract the mean from each and you have orthogonal and uncorrelated vectors:

```
 uncorx1b <- uncorx1 - mean(uncorx1)
 uncorx2b <- uncorx2 - mean(uncorx2)
 cor(uncorx1b,uncorx2b)
[1] 0
 uncorx1b%*% uncorx2b
     [,1]
[1,]    0
```

We will see why this matters in the next section. When possible, if your predictors are orthogonal, this means that the parameter estimates in a least-squares regression model will not depend on one another. Just remember that to obtain orthogonality, non-correlation or linear independence is not enough.

Exercise 12.1

For the following x and y design vectors, (1) find an additional observation (x,y) that will make them orthogonal; and (2) center the original values and determine whether they are orthogonal. For the new variables, test whether they are uncorrelated.

```
x <- c(0,1,0,1,0,1,0,1)
y <- c(1,1,2,2,3,3,4,4)
```

Consider the new x and y design variables in each case. Think about what x and y might represent. Which approach would probably be easier to implement?

12.1.1 Orthogonal Predictors in Regression

Why should orthogonality matter when you are doing a regression? Largely, the issue is not dependent on the outcome/predicted variable–it is only a matter of the predictors. If we have a pair of predictors that are related, they need to battle it out for whatever aspect of the outcome they can account for, and so you may find that neither is significant although together both are, or (more likely) that either is significant on its own but together neither one is.

So, let's create x1 and x2 to be both orthogonal and uncorrelated. See what happens to the coefficients if we fit each predictor on its own. We begin by creating two predictors that are uncorrelated, and then centering them:

```
set.seed(100)
x1.raw <- rep(1:10,each=10)
x2.raw <- rep(1:10,10)
x1.raw %*% x2.raw
 x1.raw %*% x2.raw
     [,1]
[1,] 3025
> cor(x1.raw,x2.raw)
```

```
[1] 0

x1 <- x1.raw-mean(x1.raw)
x2 <- x2.raw-mean(x2.raw)
 x1 %*% x2
     [,1]
[1,]    0
 cor(x1,x2)
[1] 0

y <- x1 + x2 + rnorm(100)
```

Now, if we look at the coefficients of three models, we see that the estimates are the same regardless of whether x1, x2, or both are used as predictors:

```
 lm(y~x1)$coef
(Intercept)          x1
0.002912563 0.972432797
 lm(y~x2)$coef
(Intercept)          x2
0.002912563 1.006314012
 lm(y~x1+x2)$coef
(Intercept)          x1          x2
0.002912563 0.972432797 1.006314012
```

Your intuition might tell you regression should always give you the same coefficients no matter what else is in the model, but that is wrong. It will not even be true if our predictors are just a little bit correlated. Here, when the predictors are orthogonal, the intercept remains unchanged because the predictors have been centered, and are both uncorrelated and orthogonal. In some sense, the intercept is more interpretable now, because it is the actual value of the mean of the data if no predictors are given. Subtracting means from your predictors essentially centers your predictors around 0, making the intercept equal to the mean at the center of the predictors. This is not what happens if we use the uncorrelated (but non-orthogonal) raw predictors:

```
 lm(y~x1.raw)$coef
(Intercept)      x1.raw
 -5.3454678   0.9724328
 lm(y~x2.raw)$coef
(Intercept)      x2.raw
  -5.531815    1.006314
 lm(y~x1.raw+x2.raw)$coef
(Intercept)      x1.raw      x2.raw
-10.8801949   0.9724328   1.0063140
```

Now, although the β values are the same, the intercepts change. So, the big advantage of having orthogonal predictors in a regression is that it won't matter if you add or remove any of them–your estimates won't change. In general, this can only be done if you have the ability to design your experiment-specifying each of the types and levels of conditions. If we had a more complex model (e.g., with interactions), more than the intercept might change. Thus, when possible, using orthogonal predictors in regression make for stable predictors. This can usually only be achieved through designing an experiment–you are unlikely to get this by sampling all variables at random from a population. But even if you have some dependence

between predictors, you may be able to achieve more stable predictors by centering the variables, and so that can be a good practice regardless, and it will often make for more interpretable models, as it fits a model around the center of the data. Now, instead of having to interpret the slope with respect to the 0 point of each variable, you interpret the intercept as the value you get in the 'middle' of each dimension of predictors.

12.2 Non-orthogonality in regression

Although it won't produce stable coefficients if you change predictors, regression works with non-orthogonal predictor variables. Depending on what you want to do with the model, having orthogonal predictors may not matter much. But a pair of two variables that are not orthogonal–even a little bit–lead to different estimates of the coefficients. Let's consider a new, 'noisy' version of x1 called x1b:

```
x1b <- x1 + rnorm(100)*.5
lm(y~x1b+x2)$coef
 (Intercept)          x1b           x2
-0.002309976  0.937548645  0.991396878

 x1b %*% x2
        [,1]
[1,] 13.1264
 cor(x1b,x2)
[1] 0.01557746
```

Here x1b is based on x1 which was orthogonal to x2. But now, it is slightly correlated and non-orthogonal to x2. In the resulting linear regression, not only did the x1/x1b coefficient change, but the x2 and intercept terms changed as well. This is because the new variable x1b was not completely orthogonal to x2, and so its presence changed the β weight for x2.

This is even a bigger problem when you have more strongly correlated predictors. We can look at this by creating a hidden variable x1 which other predictors depend on.

```
set.seed(999)
x1 <- runif(50)*10
x2 <- x1  + rnorm(50)*1
x3 <- x1 + rnorm(50)*1
```

You can see that the predictor variables are highly correlated, and are not orthogonal:

```
 cor(x2,x3)
[1] 0.8882033
 x2 %*% x3
         [,1]
[1,] 1481.824
```

We can make another outcome value based on x2 and x3, with no noise, and estimate the three beta values (100, .8, and 1.5) using a linear model:

```
y <- 100 + .8*x2 +1.5*x3
model1 <- lm(y~x2+x3)
model1
Call:
```

```
lm(formula = y ~ x2 + x3)

Coefficients:
(Intercept)           x2           x3
      100.0          0.8          1.5
```

Here, even though x2 and x3 are non-orthogonal, the estimates are exactly the same as the coefficients we used to create y. This is because they are linearly independent. So without noise, there seems to be no harm. Yet if we look at the smaller models, we get different estimates:

```
lm(y~x2)$coef
(Intercept)           x2
  99.665997    2.324701
lm(y~x3)$coef
(Intercept)           x3
   100.9275      2.1209
```

Here, because x1 and x2 are related to a common predictor, they seem to do fine at recovering the slopes. But (1) they are not orthogonal, so the coefficients change when one is dropped, and (2) we don't know will happen when we add noise. Notice tat when x2 and x3 are together in the model, the sum of their slopes is 2.3. Alone, x2 is about 2.3 and x3 is 2.1. Because they are very similar predictors, either one does OK, but together they split the predictive effect of their shared variance.

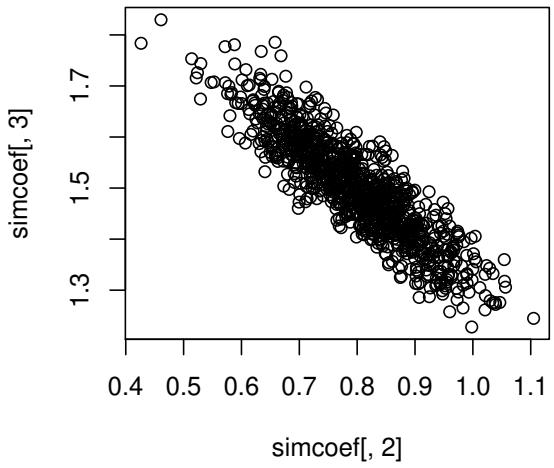
Exercise 12.2

Create a new independent predictor x4. Verify that it is not correlated with x3 or x3 (i.e., correlation below .05). Then, make a new y that is a linear combination of x2, x3, and x4. Explore how and whether your parameter estimate for x4 changes when you add or remove x2 and x3 from the model.

Let's create a simulation to see how biased estimates can be when we have correlated predictors and the outcome is not noiseless. This will repeat the above 1000 times, each time for a new set of y values based on the x values we already created. Here, we will use x2 and x3 as a design. On each trial, we will create a new set of y values at random, and estimate the coefficients of x2 and x3.

```
n <- 1000
simcoef <- matrix(0,nrow=n,ncol=3)
for(i in 1:n)
{
  y <- 100 + .8*x2 +1.5*x3 + rnorm(50)*1
  model <- lm(y~x2+x3)
  simcoef[i,]<-model$coef
}
```

Figure 12.3: Values of beta estimates across 1000 simulations where the predictors were highly correlated



Now, look at the results:

```
> colMeans(simcoef)
[1] 100.0208010   0.7912309   1.5000207
> pairs(simcoef)
cor(simcoef[,2],simcoef[,3])
[1] -0.8870764

plot(simcoef[,2],simcoef[,3])
```

The means of the values are estimated fairly well, but the first and second beta estimates appear to be highly dependent, as seen in Figure 12.3. That is, when we estimate one value high, the other tends to be low. Thus, having correlated and non-orthogonal predictors will tend to produce beta coefficients that are dependent on one another, and these will perhaps change our interpretation of the result.

To compare, we will do another simulation, but make sure the predictors are not dependent:

```
set.seed(500)
n <- 1000
simcoef2 <- matrix(0,nrow=n,ncol=3)
for(i in 1:n)
{
  x2 <- runif(50)
  x3 <- runif(50)
  y <- 100 + .8*x2 +1.5*x3 + rnorm(50)*1
  model <- lm(y~x2+x3)
  simcoef2[i,]<-model$coef
}
```

```
 colMeans(simcoef2)
[1] 99.9921734  0.8076693  1.5015694
```

Again, the estimates are OK, but what about the correlation between estimates across the many simulated fits:

```
> cor(simcoef2)
            [,1]        [,2]        [,3]
[1,]  1.0000000 -0.68840104 -0.63369767
[2,] -0.6884010  1.00000000  0.01616287
[3,] -0.6336977  0.01616287  1.00000000
```

Now, the estimates of the slopes of x2 and x3 are uncorrelated over runs, as seen in Figure 12.4, and the correlation between parameter 2 and 3 is low–.016. Note, however that the correlations with the intercept are high. This is because the predictors were uncorrelated but non orthogonal. If we subtracted the means from x2 and x3 each time, we would have uncorrelated predictors all around.

```
set.seed(500)
n <- 1000
simcoef3 <- matrix(0,nrow=n,ncol=3)
for(i in 1:n)
{
  x2 <- runif(50); x2 <- x2 - mean(x2)
  x3 <- runif(50); x3 <- x3 - mean(x3)
  y <- 100 + .8*x2 +1.5*x3 + rnorm(50)*1
  model <- lm(y~x2+x3)
  simcoef3[i,]<-model$coef
}

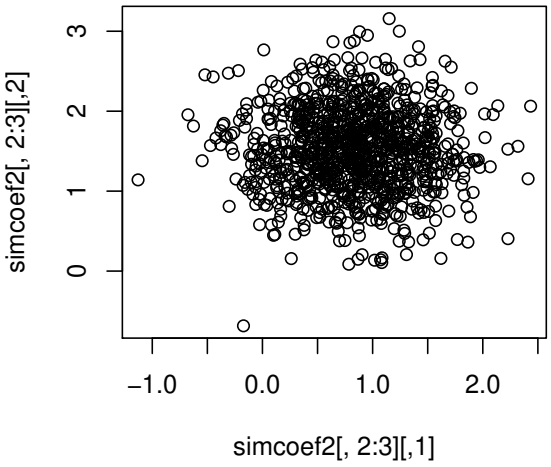
cor(simcoef3)
              [,1]        [,2]        [,3]
[1,]  1.000000000 -0.05563259 0.008318631
[2,] -0.055632591  1.00000000 0.016162868
[3,]  0.008318631  0.01616287 1.000000000
```

12.2.1 Summary of Orthogonality

When you are using regression you usually don't have the ability to create the design. When you do have the ability, if you are able to create a design with orthogonal predictors, you will have a more robust model, because it won't matter which predictors are in the model; you will have the same coefficients. This makes inference regarding removing variables easier, because you can add or remove variables in any order and the estimates do not change.

If you have uncorrelated predictors, then it is easy to make them orthogonal by centering them (subtracting their mean). Even when the predictors have a very low correlation, centering may help, as it may give more stable estimates. But if you have predictor variables with higher correlations, you may have to take other courses of action. Detecting and dealing with this situation is discussed at the end of this chapter.

Figure 12.4: Values of beta estimates across 1000 simulations where the predictors were uncorrelated



12.3 Identifiability

Identifiability also concerns the set of predictor variables in your model, but it is at the opposite extreme to orthogonality and non-correlation. It refers to the case where you have so much correlation among your predictors that you cannot distinguish them. This could happen for pairs of variables, or for entire sets of predictors.

Non-identifiability means that your beta values could hypothetically take on an infinite set of values–you cannot identify a unique value to give them. For example, if two variables are identical (or one is a linear transform of another), there are actually many possible ways you could frame the model for exactly the same results. If one variable had a beta-weight of 2, and you were able to add another copy of it to a linear model, then any combination of the two coefficients that add up to 2 (1 and 1, 1.5 and .5, 0, and 2, etc.) would produce the same results. The estimators in a regression model can't handle when predictors are identical, so it will usually fail to estimate the second predictor. This is known as the predictors being *non-identifiable*.

Consider the following orthogonal variables x1 and x2, on which y depends. If we make a model with these two, and compare it to another model with two copies of x1 (called x1a), the coefficient for x1a is returned as NA.

```
set.seed(100)
x1 <- rep(1:10,each=10)
x2 <- rep(1:10,10)
x1a <- x1
y <- x1 + x2 + rnorm(100)

lm12 <- lm(y~x1+x2)
lm12b <- lm(y~x1+x1a+x2)

> lm12
```

```
12 Call:
   lm(formula = y ~ x1 + x2)
14
   Coefficients:
16 (Intercept)            x1            x2
        0.1198        0.9724        1.0063
18
   > lm12b
20
   Call:
22 lm(formula = y ~ x1 + x1a + x2)

24 Coefficients:
   (Intercept)            x1           x1a            x2
26      0.1198        0.9724            NA        1.0063
```

In theory, we could make a linear model with both predictors, but they are redundant. The parameter estimation fails, and so the model drops predictors that make the predictors non-redundant. It might seem like a reasonable thing to do would be just to give each predictor a coefficient half as large. This won't work because there doesn't need to be two copies of x1 for this to work. Any system of predictors that are non-identifiable will create a similar situation. For example, a common strategy when creating a survey is to ask a set of similar questions (e.g., 10), and then compute the mean or sum across those questions. If we add all 11 predictors to the model, they will be non-identifiable, because we can always get the 11th predictor by a combination of the first 10.

What this means is that any predictor that can be formed by a linear combination of other predictors will fail, and so it can be impossible to split a coefficent in two. In the survey example, `lm` will probably drop the last variable, which in this case is the 11th. But if you have them in a different order, it may drop another variable, but include the mean. This may mean that you have dropped one of the predictors you care about.

In the following example, if we make x12 be a weighted sum of x1 and x2, we see that it fails to estimate it:

```
1  x12 <- x1* 3 + x2
   lm(y~x1+x2+x12)
3
   Call:
5  lm(formula = y ~ x1 + x2 + x12)

7  Coefficients:
   (Intercept)            x1            x2           x12
9       0.1198        0.9724        1.0063            NA


11
   lm(y~x12 + x1+x2)
13
   Call:
15 lm(formula = y ~ x12 + x1 + x2)

17 Coefficients:
   (Intercept)           x12            x1            x2
19      99.0423        0.5820        0.1024            NA
```

Here, x12 is not the same as either x1 or x2, but because it is formed as a combination of these, it also produces a non-orthogonal set of predictors. If we frame the model in a different order, it drops x2, which may not be what we want. This is sometimes hard to diagnose, but is probably the source of most NA results in a linear regression models.

12.3.1 Example: Dependent Predictors

Just as before, if we make x2 dependent on x1, we will expect NA estimates:

```
x1 <- 1:10
x2 <- x1*2
x3 <- runif(10)

y <- 50 + 100*x1 + 33*x2 + 17*x3
lm(y~x1+x2+x3)

Call:
lm(formula = y ~ x1 + x2 + x3)

Coefficients:
(Intercept)           x1           x2           x3
         50          166           NA           17
```

In this model, y has no noise in it, and so we should be able to reconstruct the values exactly. But the parameter for $x2$ shows up as NA. It should be clear that, through algebra, since $x2 = x1 * 2$, we could re-write the formula

$y = 50 + 100 \times x1 + 33 * (x1 \times 2) + 17 \times x3$),

which is equivalent to $y = 50 + 166 \times x1 + 17 \times x3$, which is what the model produced. But it is also equivalent to: $y = 50 + 50 \times x2 + 33 \times x2 + 17 \times x3$.

In fact, there are an infinite number of equations that are equivalent, and so lm cannot determine what the right answer is. The `lm` procedure in R will typically fail gracefully, and produce a single results and give duplicative predictor values an NA. Other regression software may take other approaches, including failing to give an answer.

12.3.2 More predictors than observations

A singular model can also be produced when you have more predictors than outcome values. If you have ten values you are trying to predict, with twelve predictors, you can always find a combination that will exactly predict the outcome (as long as the predictors are not identical), unless the predictor variables system is not linearly independent (and then we'd get NA values from the predictors anyway). For example:

```
y <- runif(10)
x <- matrix(runif(100),10,10)
model <- lm(y~x)
> summary(model)

Call:
lm(formula = y ~ x)

Residuals:
ALL 10 residuals are 0: no residual degrees of freedom!

Coefficients: (1 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)  0.05853         NA       NA       NA
x1          -0.07416         NA       NA       NA
x2           0.10816         NA       NA       NA
x3          -0.50205         NA       NA       NA
x4          -0.72696         NA       NA       NA
x5           0.16897         NA       NA       NA
x6           0.13069         NA       NA       NA
x7          -0.19430         NA       NA       NA
x8          -0.01854         NA       NA       NA
x9           1.30529         NA       NA       NA
x10               NA         NA       NA       NA

Residual standard error: NaN on 0 degrees of freedom
Multiple R-squared:     1,  Adjusted R-squared:   NaN
F-statistic:   NaN on 9 and 0 DF,  p-value: NA
```

Here, we are able to exactly predict ten completely random values with an intercept and ten completely random predictors. Notice that the model coefficients still are fitted acceptably (with the last one given an NA), but all of the other stats are given NAs as well. This is because there is no 'noise', no residual standard error, and so no way to judge how far off you are from your estimates (because you are exactly fitting!)

```
>model$res
 1  2  3  4  5  6  7  8  9 10
 0  0  0  0  0  0  0  0  0  0
>cor(y,model$fit)
[1] 1
```

To get a better intuition for this, lets try to predict y based on an increasingly larger set of random predictors:

```
fitx <- function(n=1)
{
  model <- lm(y~x[,1:n])
  plot(y,model$fit,ylim=c(0,1),xlim=c(0,1))
  abline(0,1)
}
```

Then, using the manipulate library, we can set using the GUI the number of predictors we want to use:

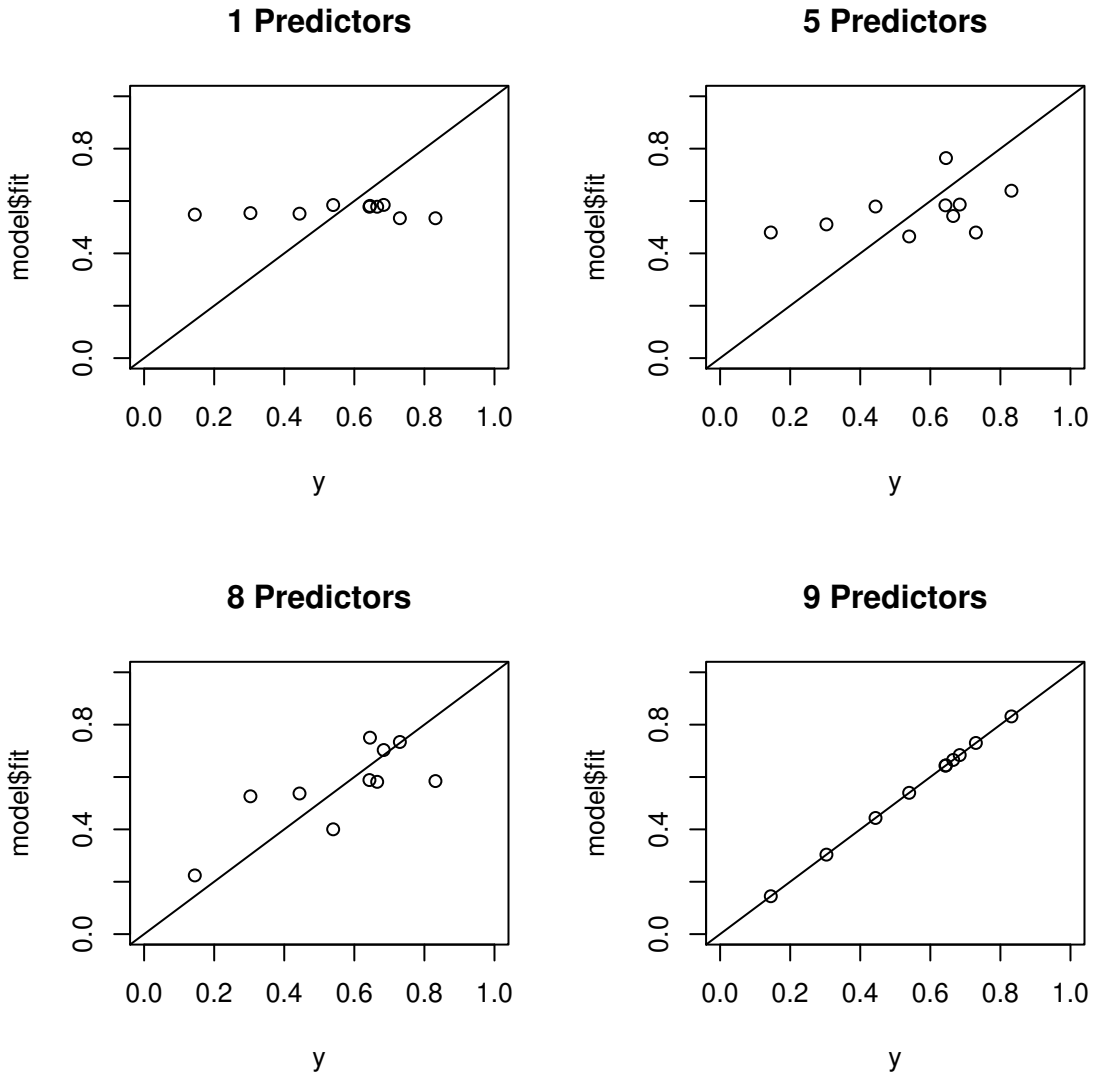
```
library(manipulate)
manipulate(fitx(x), x=slider(1,10))
```

Example output is shown in Figure 12.5

This situation is not uncommon, especially when using survey methods. It is easy to ask a 100-question survey and only get 50 respondents. What this means is that it will be impossible to predict a particular outcome variable based on the survey responses, because they will be able to fit it perfectly, and so the prediction usually makes little sense.

This is especially problematic because we want to use our statistics to make inference about people or situations we didn't see. But if our model fits the 50 people in our survey exactly, we can probably guess that it will tend to miss many of the other people, because the model is overfitting, or just fitting random fluctuations of the data.

Figure 12.5: Predicting a random set of values with an increasingly larger set of random predictors. When the number of predictors grow as large as the size of the data, we can exactly fit the results



12.3.3 How to handle non-identifiability

Having a model that is not identifiable may be acceptable, depending on the circumstances. First, remember that even if a model is not identifiable, it may not be completely saturated (having more predictors than data), and so you may not be explaining all the random noise. When a model's parameters return NA, it typically means the predictors are not identifiable. In that case you can:

- Try to determine if a predictor is a function of one or more other predictors and remove the redundancy.
- If you are saturated, remove variables (including the intercept) until you get a non-saturated model
- Place other constraints on predictors; like using only one of two highly correlated variables.
- Use a data-reduction technique such as PCA or factor analysis. Then, you can use just the main uncorrelated underlying factors as predictors in your model.
- If you are not saturated, the non-identifiable model may still be reasonable as a bookend comparison, especially if you are using a model comparison/variable selection scheme. After all, even though you have redundancy, it provides an upper limit on how good the model is.

12.4 Detecting and Managing Multi-Colinearity

Unless you have the ability to design an experiment so that it has orthogonality, it will often have predictor variables that are correlated and/or non-orthogonal. When this is especially strong, this is called often referred to by the term "multicollinearity" or "multi-colinearity". In geometry, points on the same line are colinear, and so two or more sets of points that are associated are referred to as multi-colinear. This usually means that predictors are both correlated and non-orthogonal, and may mean that they are not identifiable as well.

If your variables are non-identifiable, the solution may be simple–recode your predictors, or remove variables if necessary. But this won't take care of predictors that are just highly correlated, and we saw in this chapter that correlated and non-orthogonal predictors will impact your coefficient estimates (and thus, your selection of variables).

You can often tell by looking at the predictors that there are correlations. But how much correlation is too much? A result of correlated predictors is that if you remove on one predictor, the correlated one may get more significant than previously. But more systematic methods have been developed. The mctest library applies some of the most common tests for multi-colinearity. Note that you provide it the predictors first, then the outcome variable

```
set.seed(999)
x1 <- runif(100)*10
x2 <- x1  + rnorm(100)
x3 <- x1 + rnorm(100)
z <- rnorm(100)
y <- x1 + rnorm(100)*.2 + z

plot(x2,x3)
mctest(cbind(x2,x3,z),y)
```

```
Call:
omcdiag(x = x, y = y, Inter = TRUE, detr = detr, red = red, conf = conf,
    theil = theil, cn = cn)


Overall Multicollinearity Diagnostics

                       MC Results detection
Determinant |X'X|:         0.1993         0
Farrar Chi-Square:       153.7527         1
Red Indicator:             0.5177         1
Sum of Lambda Inverse:    11.0162         0
Theil's Method:           -0.0886         0
Condition Number:          9.9299         0

1 --> COLLINEARITY is detected by the test
0 --> COLLINEARITY is not detected by the test

>
```

This suggests that co-linearity is present according to two measures. More detail is provided with the type="i" argument

```
mctest(cbind(x2,x3,z),y,type="i" ,method="VIF")

Call:
imcdiag(x = x, y = y, method = method, corr = FALSE, vif = vif,
    tol = tol, conf = conf, cvif = cvif, leamer = leamer, all = all)


All Individual Multicollinearity Diagnostics Result

       VIF    TOL       Wi        Fi Leamer    CVIF Klein
x2 5.0096 0.1996 194.4666 392.9428 0.4468 -1.4149     0
x3 5.0036 0.1999 194.1763 392.3563 0.4471 -1.4132     0
z  1.0030 0.9971   0.1431   0.2892 0.9985 -0.2833     0

1 --> COLLINEARITY is detected by the test
0 --> COLLINEARITY is not detected by the test

* all coefficients have significant t-ratios

R-square of y on all x: 0.846

* use method argument to check which regressors may be the reason of
    collinearity
==================================
```

In this case, colinearity suggested by a couple measures, but not the popular "VIF" measure. IF x2 and x3 had been more strongly correlated, this would have also been true:

```
 mctest(cbind(x2,x3,z),y)

Call:
omcdiag(x = x, y = y, Inter = TRUE, detr = detr, red = red, conf = conf,
    theil = theil, cn = cn)


Overall Multicollinearity Diagnostics
```

```
9
                        MC Results detection
11 Determinant |X'X|:         0.0080         1
   Farrar Chi-Square:       460.6759         1
13 Red Indicator:             0.5759         1
   Sum of Lambda Inverse:   251.6083         1
15 Theil's Method:            0.0011         0
   Condition Number:         53.3344         1
17
   1 --> COLLINEARITY is detected by the test
19 0 --> COLLINEARITY is not detected by the test

21 > mctest(cbind(x2,x3,z),y,type="i",method="VIF")

23 Call:
   imcdiag(x = x, y = y, method = method, corr = FALSE, vif = vif,
25     tol = tol, conf = conf, cvif = cvif, leamer = leamer, all = all)


27
    VIF Multicollinearity Diagnostics
29
           VIF detection
31 x2 125.2860         1
   x3 125.3193         1
33 z    1.0030         0

35 Multicollinearity may be due to x2 x3 regressors

37 1 --> COLLINEARITY is detected by the test
   0 --> COLLINEARITY is not detected by the test
```

Now, we can see that we have strong colinearity, and x2 and x3 are responsible for it.

12.4.1 Dealing with Correlated and non-orthogonal predictors

There are many approaches to dealing with correlated and non-orthogonal predictors. First, you may choose to do nothing. If you are just trying to predict outcomes and you don't care about interpreting the predictor variables, then the additional correlated predictor may not harm you much. You may center the variables. This will only help for a subset of problems (i.e., if you have uncorrelated but not orthogonal variables). You might also add observations in conditions that make your design orthogonal. Unfortunately, this is not always feasible. If you have a pair of variables that are correlated, it may make sense to just replace them with their average (or some weighted average). If necessary, you can replace them with their average and their difference, so you have one predictor accounting for the common variability, and a second accounting for their differential variability. If the difference predictor is not significant, then this may allow you to argue that the two predictors are not accounting for anything independent, and so the average value is appropriate. Of course, if you are just using parameter selection, you will tend to get rid of one of the variables that helps the least, and if they are strongly correlated, it may not impact much. Finally, there are variations on regression, like ridge regression, the lasso technique, elastic net regression, partial least squares, and principal components regression that will attempt to handle this correlation directly as part of the model.

The ridge regression and LASSO regression approaches (called 'regularized regression') work by recognizing that predictors that are correlated are likely to be competing for the same variance in the outcome. Because of this correlation, we might—just by chance—get

one predictor that happens to be better for our sampled data, and it will take on a larger coefficient and swamp the other(s), even if other mixtures are about the same. LASSO and ridge regression apply a complexity penalty (much like BIC and Bayes factor), but attempt to minimize the total size of the coefficients, using different approaches, and use cross-validation approaches to be sure the generalization is best. In essence, instead of removing variables, it shifts their values toward 0 (a value of 0 would be the same as removing) in order to create a simpler model (like we would with variable selection). The coefficients are 'shrunk' toward 0 to make them less important.

12.5 Uses and limitations of the linear model in human behavioral Data

A linear regression model lets you create a model that explains the data, and then interpret parameters to help test hypotheses. Some uses for the model:

Epidemiology. For example, you can model risk for a disease based on a set of predictors. Predictors that are reliably different from zero can be identified and policy recommendations or guidelines made based on these predictors. Furthermore, comparison between predictors can be useful, allowing one to determine which factors, if changed, would provide the best chance of impacting the outcome. However, models such as this can be problematic in a number of ways. First, they will tend to use as predictors variables that are easier to measure, and ones that cannot be manipulated, but are instead observed in the sample. They are likely to have started with multiple correlated predictors where one set gets removed if it does not improve the model. This process can tend to hide interactions. If most people who smoke also drink, then smoking and drinking will be redundant predictors and so one will be removed. A risk produced by only one, or by a combination of smoking and drinking may get attributed to just one of them alone, and perhaps the wrong one.

Inferring Mental Architecture. Many famous findings in psychology relate to slopes and intercepts of functions of data. For example, the memory-scanning time identified by Sternberg, memory transfer times identified by Sperling, the relationship between memory span and the length of words identified by Baddeley, the parameters of Fitts's law, and so on. These are typically found based on comparing different transformations of the predictor and/or outcome variable. Sternberg's quadratic function and consistent coefficient of the quadratic term led him no conclude that memory access is serial and self-terminating. Here both the form of the model and the coefficients are useful. However, this type of inference is always heavily dependent on the assumptions of the linear model and the transformations that ensue. Many times, non-linear models making other assumptions produce equivalent results.

Hypothesis Testing. Using a regression model can provide much more powerful hypothesis testing than simply comparing means of groups. It allows you to examine whether slopes of functions change across conditions. A typical example might look at whether a learning slope increases in response to some manipulation.

Factoring out covariates. The so-called "ANCOVA" is simply a factorial ANOVA that includes one or more continuous predictors that are factored out as covariates. This is just a standard regression model that includes categorical and non-categorical predictors, but

the categorical effects are arranged as in an ANOVA, and typically the reliability of the covariates are not reported.

Predicting and extrapolating. Similar to epidemiological examples, one can use regression to predict the result of an unknown variable for some new cases, and even ones outside the region where data was measured. Extrapolation can always be dangerous. Predicting can be useful as a way to produce an 'adjusted' score. For example, you may make a regression that predicts memory span by education level and age. Researchers know that those with more education have higher memory spans, and that memory span declines across the lifespan by about .1 units per decade. Thus, if you want to select a set of participants who are especially good (or bad) for their age/education, you can create an adjusted score which takes their actual score and adds amounts related to the slopes obtained for age and education.

Other limitations. In practice, the main limitations in applying linear regression models include having too little data with respect to the number of predictors, having non-linear relationships between predictors and outcomes, having outcome variables (e.g., accuracy) where linear predictors don't make sense, and having sets of predictors that are correlated. Many of these limitations can be worked around using specific methods will will cover in future parts of this course.

12.6 Summary

This section covered various transformations of the predicted and predictor variables. These types of transformations can provide the standard linear model with much flexibility when it comes to fitting models that are either able to capture complex data trends, or are interesting from a psychological perspective.

Chapter 13

Polynomials, non-parametric regression, and Transformations

13.1 Polynomial Regression

Suppose we observed ten points. Looking at them, do you see a pattern, or some smooth non-linear function that could explain these values?

```
set.seed(1004)
x <- 1:10
y <- runif(10)

plot(x,y)
```

One way to handle curvature in a relationship between variables is via what is known as *polynomial regression*. Instead of assuming a line, maybe there is a parabola–a second-order polynomial, that describes the data well. Or maybe higher orders of polynomials that have more curves in them. If we continue to add more high-order polynomials, it is provable that we will be able to fit any set of data. Because the higher-power transformations of a vector are all different from all lower-order polynomials, this is similar to the example in the last chapter in which we added random predictors until the prediction was perfect. Thus, if you add enough higher-order terms, you will create a complex polynomial that will always fit the data you have. But just like the models where we incorporate non-useful predictors, the models will usually horribly mis-fit the spaces between the data you have observed.

So suppose you wanted to fit a 2nd-order polynomial to data. To do this, we could create a new predictor that is simply x^2

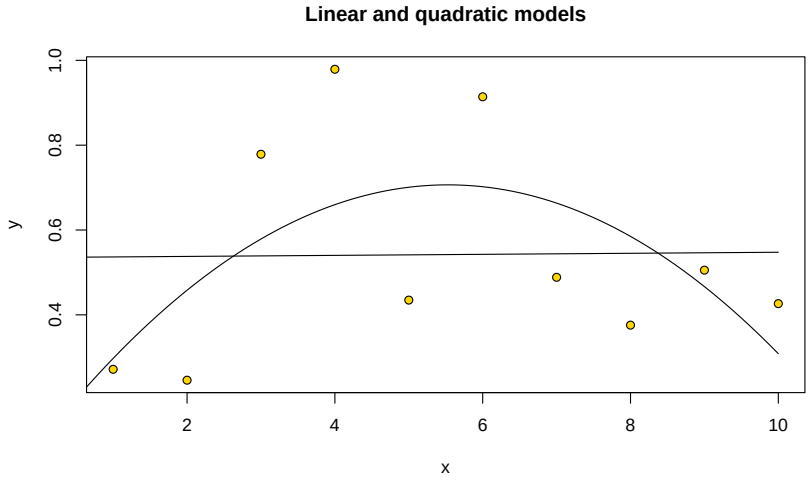
```
x2 <- x^2

lm1 <- lm(y~x)
lm2 <- lm(y~x+x2)

anova(lm1,lm2)
Analysis of Variance Table

Model 1: y ~ x
Model 2: y ~ x + x2
Res.Df     RSS Df Sum of Sq      F  Pr(>F)
```

Figure 13.1: Random data points with a linear and best-fit quadratic model fit.




```
1       8 0.60275
2       7 0.39303  1    0.20972 3.7352 0.09455 .
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

> lm1$coefficients
(Intercept)           x
0.535275275 0.001227978
> lm2$coefficients
(Intercept)           x          x2
0.09681889  0.22045617 -0.01992984

preddat <- data.frame(x=0:100/10,x2=(0:100/10)^2)
points(preddat$x,predict(lm1,preddat),type="l")
points(preddat$x,predict(lm2,preddat),type="l")
```

Notice that the polynomial model actually improves the fit–maybe enough to want to use it according to an anova. But notice that x and x2 neither uncorrelated nor orthogonal.

```
> cor(x,x2)
[1] 0.9745586
> x %*% x2
[,1]
[1,] 3025
>
```

And if we look at the coefficient x1 its value changes between the two models. We'd expect that the coefficient will change as we add higher powers if they are not orthogonal. We can improve things by centering them:

```
xb <- x-mean(x)
xb2 <- xb^2
```

```
cor(xb,xb2)
xb %*% xb2

lm(y~xb)$coef
(Intercept)          xb
0.542029155 0.001227978

lm(y~xb+xb2)$coef
(Intercept)           xb          xb2
0.706450299  0.001227978 -0.019929836
```

The intercept still changes, but xb coefficient stays the same. The actual fit of the model doesn't change between these two ways of framing the model, but now it is easier to interpret, because the lower-order term does not change when you add a higher-order term.

It is usually better to R will also let you do this within the lm function without creating separate predictors, if you encapsulate the value in an `I()` function. The `I()` function will compute the mathematical relationship before using as a predictor; if you don't do this, it may treat it as an interaction.

So, let's try to predict ten random numbers with high-order polynomials.

```

##Form the polynomial values for 0..10
r2 <- 0:100/10  #identify a range to predict with
par(mfrow=c(3,3),mar=c(2,3,2,1))
poly1 <- lm(y~x)
plot(x,y,pch=16,main="1st order polynomial fit")
points(r2,predict(poly1,list(x=r2)),type="l")
x2 <- x^2
poly2 <- lm(y ~ x + I(x^2))
plot(x,y,pch=16,main="2nd order polynomial fit")
points(x,poly2$fit,type="p")
points(r2,predict(poly2,list(x=r2)),type="l")

poly3 <- lm(y~x + I(x^2) + I( x^3))
plot(x,y,pch=16,main="3rd order polynomial fit")
points(x,poly3$fit,type="p")
points(r2,predict(poly3,list(x=r2)),type="l")


poly4 <- lm(y~x + I(x^2) + I( x^3) + I(x^4))
plot(x,y,pch=16,main="4th order polynomial fit")
points(x,poly4$fit,type="p")
points(r2,predict(poly4,list(x=r2)),type="l")

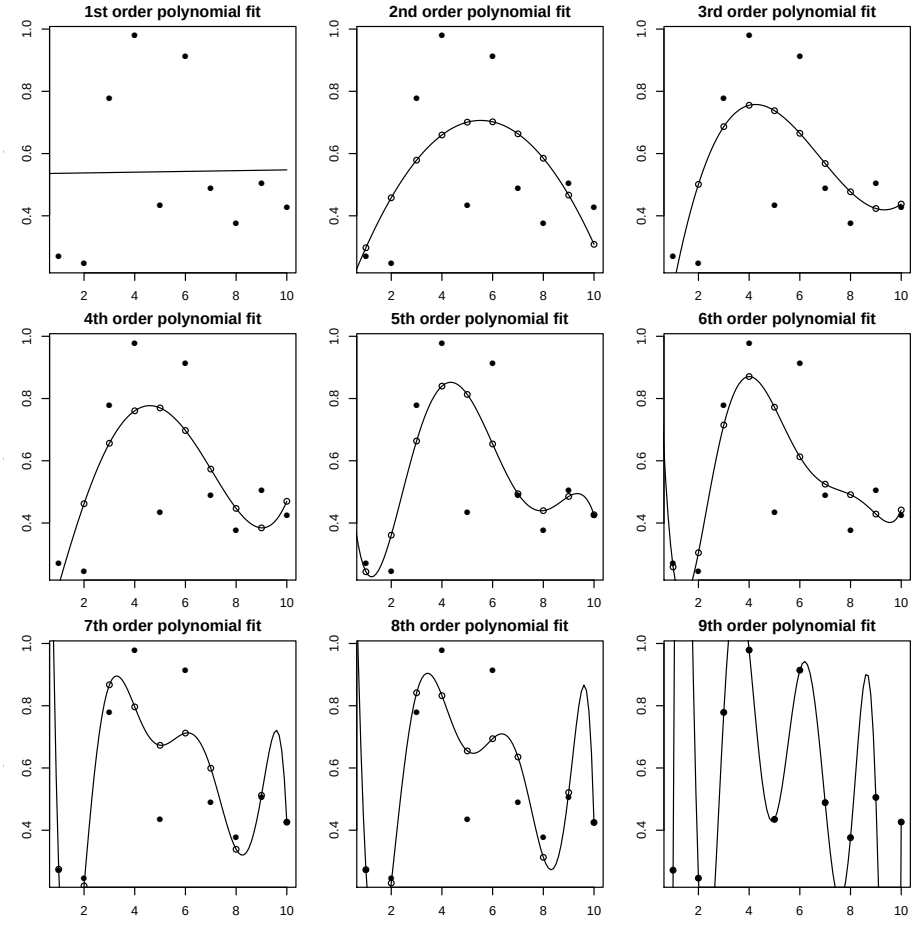
poly5 <- lm(y~x + I(x^2) + I( x^3) + I(x^4) + I(x^5))
plot(x,y,pch=16,main="5th order polynomial fit")
points(x,poly5$fit,type="p")
points(r2,predict(poly5,list(x=r2)),type="l")

poly6 <- lm(y~x + I(x^2) + I( x^3) + I(x^4) + I(x^5) +I(x^6))
plot(x,y,pch=16,main="6th order polynomial fit")
points(x,poly6$fit,type="p")
points(r2,predict(poly6,list(x=r2)),type="l")


poly7 <- lm(y~x + I(x^2) + I( x^3) + I(x^4) + I(x^5) +I(x^6) + I(x^7))
plot(x,y,pch=16,main="7th order polynomial fit")
points(x,poly7$fit,type="p")
points(r2,predict(poly7,list(x=r2)),type="l")
```

```
40
42 poly8 <- lm(y~x + I(x^2) + I( x^3) + I(x^4) + I(x^5) +I(x^6) + I(x^7) + I(x^8)
       )
   plot(x,y,pch=16,main="8th order polynomial fit")
44 points(x,poly8$fit,type="p")
   points(r2,predict(poly8,list(x=r2)),type="l")
46
48 poly9 <- lm(y~x + I(x^2) + I( x^3) + I(x^4) + I(x^5) +I(x^6) + I(x^7) + I(x^8)
       + I(x^9))
   plot(x,y,pch=16,main="9th order polynomial fit")
50 points(x,poly9$fit,type="p")
   points(r2,predict(poly9,list(x=r2)),type="l")
```

Figure 13.2: Successive Polynomial fits to a set of ten random numbers.



Notice that as the polynomial gets larger, we end up fitting each point exactly–but the model we specify gets somewhat complicated and unlikely to be able to predict anything.

A shortcut function lets you do polynomial regression with a single command, with the additional advantage that the polynomials are orthogonal because they are centered at 0–so

that adding higher-order predictors won't impact the lower-order predictor estimates. The potential disadvantage of this is that it makes it more difficult to interpret these predictors. Notice how they produce exactly the same fit (look at the residuals and the overall statistics) but the coefficients and their p-values differ. This suggests that you need to be careful when selecting polynomial coefficients based on p-values alone.

```
poly5a <- lm(y~x + I(x^2) + I(x^3) + I(x^4) + I(x^5))
poly5b <- lm(y~poly(x,5))

> summary(poly5a)

Call:
lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5))

Residuals:
         1          2          3          4          5          6          7
 0.0131388 -0.0532399  0.0712665 -0.0462165  0.1066487 -0.2630201  0.2933692
         8          9         10
-0.1541042  0.0330406 -0.0008832

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.774e-01  1.368e+00   0.422    0.695
x           -1.748e-01  2.072e+00  -0.084    0.937
I(x^2)       1.747e-01  1.036e+00   0.169    0.874
I(x^3)      -3.593e-02  2.258e-01  -0.159    0.881
I(x^4)       2.134e-03  2.218e-02   0.096    0.928
I(x^5)      -1.674e-05  8.040e-04  -0.021    0.984

Residual standard error: 0.2245 on 4 degrees of freedom
Multiple R-squared: 0.8414, Adjusted R-squared: 0.6432
F-statistic: 4.244 on 5 and 4 DF,  p-value: 0.09312

> summary(poly5b)

Call:
lm(formula = y ~ poly(x, 5))

Residuals:
         1          2          3          4          5          6          7
 0.0131388 -0.0532399  0.0712665 -0.0462165  0.1066487 -0.2630201  0.2933692
         8          9         10
-0.1541042  0.0330406 -0.0008832

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.511098   0.071007   7.198  0.00197 **
poly(x, 5)1 -0.786367   0.224545  -3.502  0.02485 *
poly(x, 5)2 -0.557678   0.224545  -2.484  0.06795 .
poly(x, 5)3  0.307271   0.224545   1.368  0.24301
poly(x, 5)4  0.214836   0.224545   0.957  0.39287
poly(x, 5)5 -0.004676   0.224545  -0.021  0.98438
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.2245 on 4 degrees of freedom
Multiple R-squared: 0.8414, Adjusted R-squared: 0.6432
F-statistic: 4.244 on 5 and 4 DF,  p-value: 0.09312
```

Notice that 5a and 5b produce identical fits, but the parameter estimates are different,

because poly() creates orthogonal polynomial predictors.

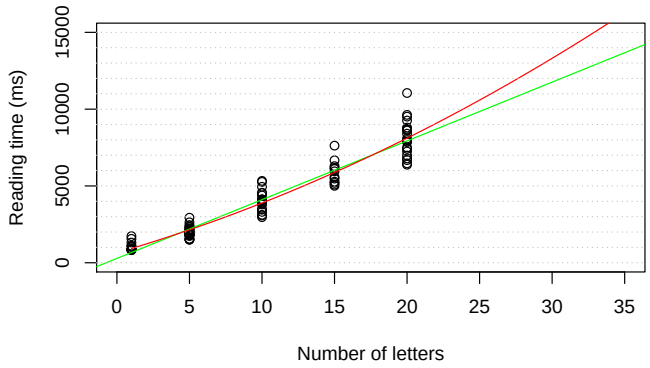
Exercise 13.1

Make a polynomial model to fit the `WWWusage` data set. First, let's fit a set of models with 1 to 7th order polynomials:

13.1.1 Polynomial regression for scientific hypotheses

Linear regression on its own assumes that there is a linear relationship between an predictor and the predicted. In social science domains, this is often not true. For example, learning curves typically asymptote after practice, and so will violate the assumption of regression. Some measures might behave in linear fashion–for example, if you measure how long it takes someone to read five versus ten versus twenty letters, you might expect a line. However, sometimes the process will suggest that there should be a specific type of polynomial relationship. For example, this happens for search tasks. If you are trying to organize a deck of cards by rank and suit, but you can only search for cards haphazardly, each search will take time proportional to the size of the deck. If you work through this, you will find that the time to sort a deck is not proportional to the size of the deck–it is proportional to the size of the deck squared. Many psychological processes involve such factors. For example, figure 13.3 shows reading times for random character strings of different lengths.

Figure 13.3: Reading times (in ms) of random letter strings of different lengths. Both a linear and quadratic model are fitted to the data, and produce nearly the same fit.



The data in Figure 13.3 shows a nearly linear trend. We can verify this by fitting a regression model, whose predicted values are plotted on the graph.

```
let <- read.csv("reading.csv",header=F)
colnames(let) <- c("sub","length","trial","stim","rt")
lm1 <- lm(rt~length,data=let)
summary(lm1)

Call:
lm(formula = rt ~ length, data = let)

Residuals:
```

```
Min       1Q   Median       3Q      Max
-1549.66  -433.22   -30.48   297.17  3119.34

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)   282.50     136.28   2.073   0.0409 *
length        382.56      11.33  33.767   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 742.1 on 94 degrees of freedom
Multiple R-squared:  0.9238,  Adjusted R-squared:  0.923
F-statistic:  1140 on 1 and 94 DF,  p-value: < 2.2e-16



plot(let$length,let$rt,ylim=c(0,15000),xlab="Number of letters",
ylab="Reading time (ms)")

abline(lm1$coeff,col='green')
```

However, maybe we suspect there is a curvature, which would be true if our deck-search story holds for reading-based search. We can create another predictor that is the square of the number of letters. This will let the predicted model be quadratic, and essentially form a parabola. We will start by adding a squared term directly to the data frame `let`:

```

> let$length2 <- let$length^2
> lm2 <- lm(rt~length + length2,data=let)
> summary(lm2)

Call:
lm(formula = rt ~ length + length2, data = let)

Residuals:
Min      1Q  Median      3Q     Max
-1673.4  -387.5   -66.1   394.1  2995.6

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)   64.461    261.916   0.246    0.806
length       419.687     40.379  10.394 9.27e-16 ***
length2       -1.002      1.306  -0.767    0.446
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 799.2 on 69 degrees of freedom
Multiple R-squared: 0.9414, Adjusted R-squared: 0.9398
F-statistic: 554.7 on 2 and 69 DF,  p-value: < 2.2e-16

##Add the predicted values to the graph
points(1:40,lm2$coef[1] + lm2$coef[2]*1:40+lm2$coef[3]*(1:40)^2,
type="l",col='red')
```

The squared term adds almost nothing to the graph, but the predictions appear to start diverging slightly as the length gets large. Of course we have no data in those regions, and so it is no reason to make one prediction over the other. However, because the polynomial

model includes the linear model, we might prefer the linear-only model by virtue of Occam's Razor–an appeal to simplicity.

We can test which one is better via an ANOVA;

```
>
> summary(lm1)

Call:
lm(formula = rt ~ length, data = let)

Residuals:
Min       1Q   Median       3Q      Max
-1549.66  -433.22   -30.48   297.17  3119.34

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   282.50     136.28   2.073   0.0409 *
length        382.56      11.33  33.767   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 742.1 on 94 degrees of freedom
Multiple R-squared:  0.9238,  Adjusted R-squared:  0.923
F-statistic:  1140 on 1 and 94 DF,  p-value: < 2.2e-16

> summary(lm2)

Call:
lm(formula = rt ~ length + length2, data = let)

Residuals:
Min      1Q  Median      3Q     Max
-1726.7  -367.8   -45.0   296.5  2942.3

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  646.128    202.050   3.198  0.00189 **
length       275.484     46.137   5.971 4.27e-08 ***
length2        4.887      2.044   2.390  0.01884 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 724.1 on 93 degrees of freedom
Multiple R-squared:  0.9282,  Adjusted R-squared:  0.9267
F-statistic: 601.5 on 2 and 93 DF,  p-value: < 2.2e-16

> ##length2 is significant!
> anova(lm2,lm1)
Analysis of Variance Table

Model 1: rt ~ length + length2
Model 2: rt ~ length
Res.Df      RSS Df Sum of Sq      F  Pr(>F)
1     93 48764821
2     94 51761049 -1  -2996228 5.7141 0.01884 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
> ##a significant difference; prefer the larger (polynomial) model.
>
```

Both the summary() functions and the ANOVA suggest that the model2 is a good

improvement–because they were significantly different. Furthermore, the coefficient of length2 tells us how long the extra memory search takes–essentially an additional 5 ms per letter of the list, for each word on the list. This adds 50 ms to the reading of a 10-item list, and 100 ms to the reading of a 20-item list.

13.1.2 Exercise:

Create a polynomial regression that approximates the data in the built-in data set `WWWusage`. Fit, 1, 5, 10, 15, and 20-order polynomials, and compare the fitted model to the data.

13.2 Non-parametric regression approaches

Polynomial regression gives us a lot of flexibility, but there are other approaches that might work that don't require making a polynomial set of predictors. If you just want a smoothed representation of data, you might consider using a moving average. A moving average specifies a window, and replaces each value in your data with the average of values within that window. You can specify windows of different sizes, as well as the 'shape' of the window. The window is really a weighting distribution, and is known in this context as a smoothing kernel.

In R, you can quickly create a moving average using the 'filter' function. Consider presidential approval ratings:

```
1
3 plot(presidents,type="o",main="Approval ratings of presidents")
  ##we could do this with polynomial regression, ut what if we just
5 ##used a moving average. That is, we replaced each value with a weighted
  ##sum of the values around it.
7 lines(filter(presidents,c(1,1,1)/3),col="grey20",lwd=3)
  lines(filter(presidents,c(1,1,1,1,1)/5),col="red",lwd=3)
```

The first argument to filter is the data sequence you want to apply. The second argument to filter is the 'smoothing kernel'. The simplest smoothing kernel is a moving average, equally weighing everything in the window. Better methods might use a triangular or gaussian kernel, so that the influence decreases the farther the sample gets out. In the example, the first window has a size of 3, the second has a size of 5. The larger the window, the smoother the function, but the more it compresses peaks and valleys.

we can make this into a function that specifies the window size:

```
  ma <- function(x,n=5){filter(x,rep(1/n,n), sides=2)[1:length(x)]}
2 ##apply ma to WWWUsage:
  plot(WWWusage,pch=16,type="o")
4 points(ma(WWWusage,n=5),type="o",col= "blue")
  points(ma(WWWusage,n=15),type="o",col="green")
```

How good is a moving average at recovering a function that had noise added to it?

```
plot(WWWusage)
newdat <- WWWusage + rnorm(100)*20
points(newdat)
points(ma(newdat,3),col="red",type="l",lwd=3)
points(ma(newdat,5),col="red",type="l",lwd=3)
points(ma(newdat,6),col="red",type="l",lwd=3)
points(ma(newdat,10),col="red",type="l",lwd=3)

points(ma(newdat,20),col="red",type="l",lwd=3)
```

We can specify any kernel whose values add up to 1.0. Common kernels include a guassian and other similar distributions. We might try a triangular kernel

```
ma2 <- function(x,n=5){
  tmp <- c(1:n, (n-1):1)
  kernel <- tmp/sum(tmp)
  filter(x,kernel, sides=2)}
plot(WWWusage)
points(newdat)
lines(ma(newdat,5),col="red")
lines(ma2(newdat,3),col="red")
lines(ma2(newdat,5),col="red")
```

13.2.1 Moving average properties

A moving average is simple, and by specifying a smoothing kernel, it can be quit flexible. but it has some limitations. These include:

- It assumes that samples are equidistant, and in a time series. What if they are not?
- What if you have two DVs that are not regular, or recorded in order of the first?
- what if you have multiple data sets you want to combine to create a merged smoothed trend?
- It can fail near the ends.
- It can cause problems when you have missing data.

A hybrid between polynomial regression and moving average is called loess (or lowess) regression. R has two different functions, one called loess and another called `lowess`. We will use loess, as it is more similar to `lm()`.

13.3 The `loess` regression

This hybrid approach essentially fits a polynomial regression locally to each point, trying to create a mini-regression for just the points around each point–so it is sort of like a moving average. It handles non-time-series data though, and it fitted much like you use lm(). The loess method offers a robust alternative that has a somewhat different underlying mechanism, but can be much more robust. It can be invoked like a regression model.

```
lmodel <- loess(daty~datx)
plot(datx,daty)
xs <- 0:30
points(xs,predict(lmodel,xs),type="l",col="red",lwd=3)
```

We can control the window size using span.

```
lmodel2 <- loess(daty~datx,span=.3)
```

Predictions are made with the same `predict()` function used for `lm`:

```
xs <- -10:50
points(xs,predict(lmodel2,xs),type="l",col="blue",lwd=3)
```

Notice that the model works regardless of whether we observed a particular data value:

```
 predict(lmodel2,list(xs=3.3322))
xs
-0.7340314
```

Although we can't do all the same hypothesis testing and estimation with loess, it can be used both for summarizing trends and for interpolating and predicting missing data, especially in time series.

Here is an example of the track of a mouse moving around the screen. The mouse location is only recorded after each screen refresh–about once every 16 ms. But suppose we wanted to know our best guess of where the mouse was for each recording of a eeg sensor, which records every millisecond. We could use the loess model to interpolate:

```
dat <- read.csv("ptracker.csv")[200:500,]

time <- 117:168*100  ##Regularize in tenths of seconds

mx <- loess(posx~time,span=.1,data=dat)
my <- loess(posy~time,span=.1,data=dat)



par(mfrow=c(2,2))
plot(dat$time,dat$posx,main="X position")
points(time,predict(mx,data.frame(time=time)),type="l")

plot(dat$time,dat$posy,main="Y position")
points(time,predict(my,data.frame(time=time)),type="l")


plot(dat$posx,dat$posy,type="b",main="X and Y")

points(predict(mx,data.frame(time=time)),
predict(my,data.frame(time=time) ),
col="red",type="o",pch=1,cex=.3)

time <- 11700:168000  ##Regularize in 100s of seconds
points(predict(mx,data.frame(time=time)),
```

```
predict(my,data.frame(time=time) ),
col="red",type="o",pch=1,cex=.3)

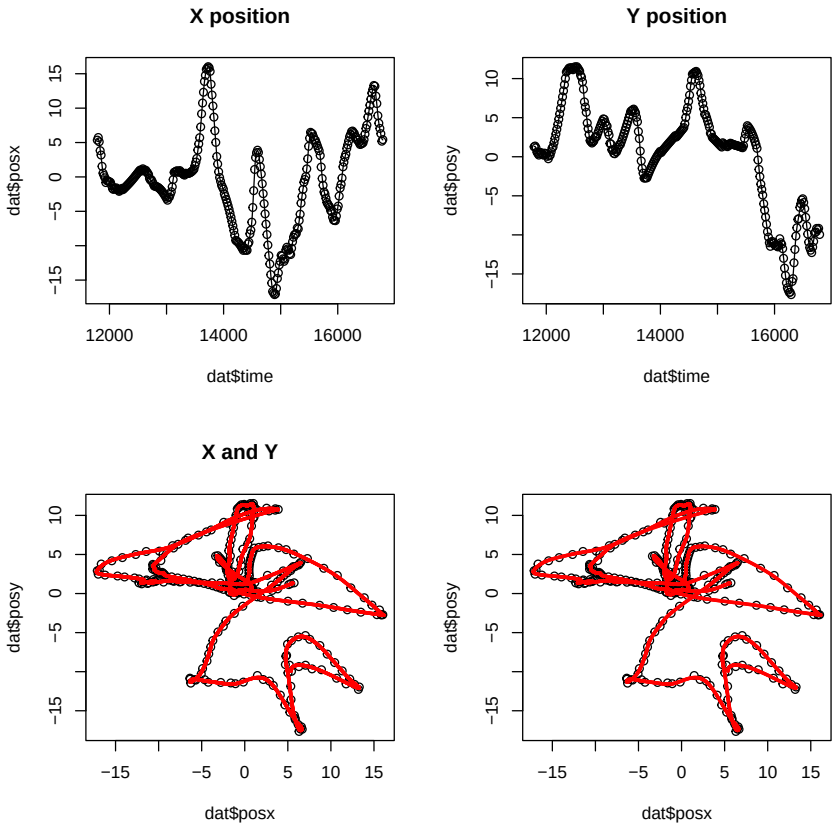
mx <- loess(posx~time,span=.02,data=dat)
my <- loess(posy~time,span=.02,data=dat)

plot(dat$posx,dat$posy,type="p")

points(predict(mx,data.frame(time=time)),
predict(my,data.frame(time=time) ),
col="red",type="o",pch=1,cex=.3)
```

Figure 13.4 shows the models described here. This could permit guessing values if we had missing or touchy sensor data, or interpolating between our sensor readings. The best part is that we don't have to try anything tricky–just fit the model and predict the values we want to use.

Figure 13.4: Using loess to fit mouse movements in 2D.



13.4 Generalized Additive Models (GAMs) and Spline regression

The options we have so far is to either use standard regression, to do some ad hoc transforms, to do polynomial regression, or to do some sort of smoothing–either as a moving average or via loess. What if we want to mix some of these? What if we want to compare models? This can be a little difficult. Generalized Additive Models (GAMs) were designed to enable a more systematic set of non-parametric curves to be incorporated, but mixed with normal regression predictors, all in a model that is fit and evaluated in a systematic way. We will only be using this for a few examples to demonstrate its power, but it will mostly work like our previous models have.

```

library(gam)
library(ggplot2)

  cw <- ChickWeight
  g <-  gam(weight~s(Time)+ Diet,data=cw)
  cw$fit <- g$fitted.values

  summary(g)
```

Output:

```
>summary(g)

  Call: gam(formula = weight ~ s(Time) + Diet, data = cw)
  Deviance Residuals:
  Min          1Q     Median          3Q         Max
  -143.98247  -17.45120   -0.07828   15.24170  134.68420

  (Dispersion Parameter for gaussian family taken to be 1261.057)

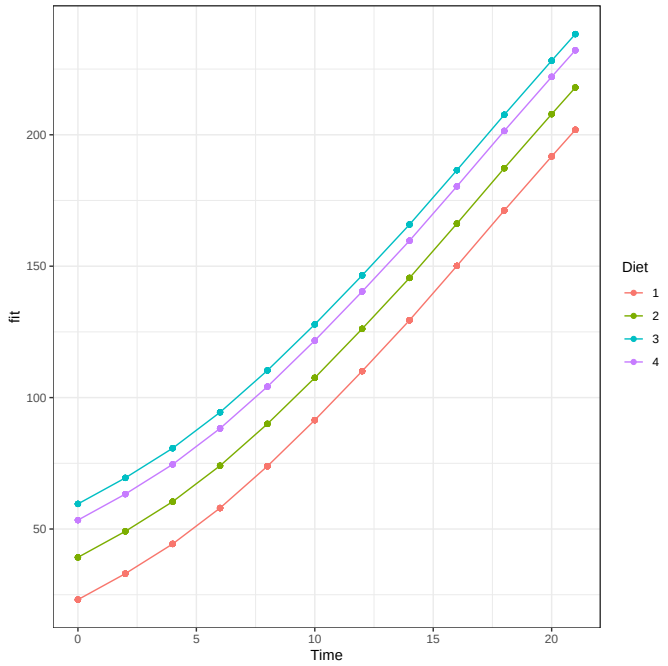
  Null Deviance: 2914556 on 577 degrees of freedom
  Residual Deviance: 718802.2 on 569.9998 degrees of freedom
  AIC: 5776.987

  Number of Local Scoring Iterations: NA

  Anova for Parametric Effects
  Df  Sum Sq Mean Sq  F value    Pr(>F)
  s(Time)     1 2042344 2042344 1619.549 < 2.2e-16 ***
  Diet        3  129675   43225   34.277 < 2.2e-16 ***
  Residuals 570  718802    1261
  ---
  Signif. codes:  0 ‘***‘ 0.001 ‘**‘ 0.01 ‘*‘ 0.05 ‘.‘ 0.1 ‘ ‘1


  Anova for Nonparametric Effects
  Npar Df Npar F     Pr(F)
  (Intercept)
  s(Time)           3 6.2202 0.0003675 ***
  Diet
  ---
  Signif. codes:  0 ‘***‘ 0.001 ‘*‘ 0.01 ‘*‘ 0.05 ‘.‘ 0.1 ‘ ‘1
```

Figure 13.5: GAM model 1, with a common diet parameter for each diet (all chicks within a diet are identical)



Here, we predicted chick weight by a constant for diet and a smoothed spline for time. This sort of assumes all chicks have the same growth path but diet may increase or decrease that somewhat. Plotting the predicted values, we can see four lines that are curved but parallel. The model output tests each component just like an ANOVA table, using an f-test as well. It also shows that the curve predicting time essentially takes 3 degrees of freedom– something like a cubic polynomial.

We could build other models that are more sophisticated. This time, let's try a loess curve instead of the spline curve (there is really not much difference between these two), but include an multiplier interaction with chick, so each chick still follows the same growth curve but with a multiplier on how fast the growth curve is followed.

```
1
g <-  gam(weight~lo(Time)*Chick+ Diet,data=cw)
3 cw$fit2 <- g$fitted.values
  ggplot(cw,aes(x=Time,y=fit2,group=Chick,color=Diet)) +geom_point() + geom_line
      ()+ theme_bw()
5 summary(g)

7 > summary(g)

9 Call: gam(formula = weight ~ lo(Time) * Chick + Diet, data = cw)
  Deviance Residuals:
11 Min        1Q    Median        3Q       Max
  -36.1116  -5.7449  -0.3421   5.9533  36.6389
13
  (Dispersion Parameter for gaussian family taken to be 130.6865)
15
  Null Deviance: 2914556 on 577 degrees of freedom
```

```
Residual Deviance: 62170.32 on 475.7212 degrees of freedom
AIC: 4550.769

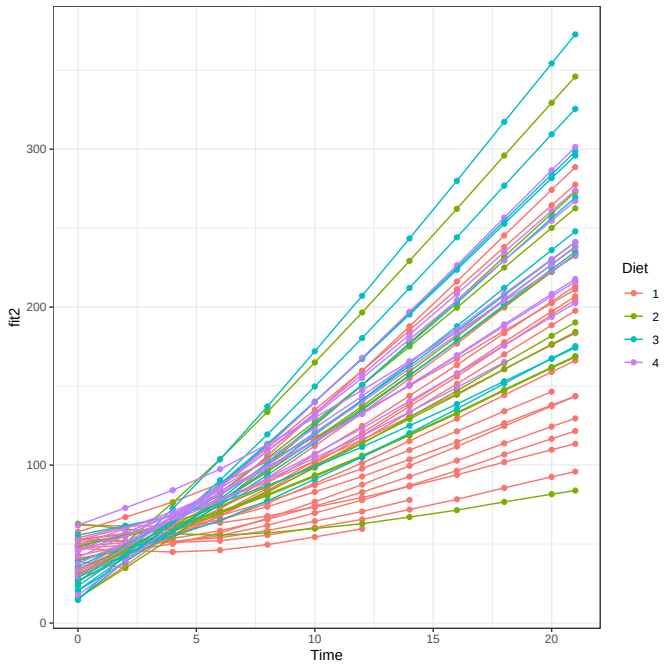
Number of Local Scoring Iterations: NA

Anova for Parametric Effects
Df  Sum Sq Mean Sq  F value    Pr(>F)
lo(Time)         1.00 2042344 2042344 15627.814 < 2.2e-16 ***
Chick           49.00  449254    9168    70.156 < 2.2e-16 ***
lo(Time):Chick  49.00  338562    6909    52.870 < 2.2e-16 ***
Residuals      475.72   62170     131
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Anova for Nonparametric Effects
Npar Df Npar F     Pr(F)
(Intercept)
lo(Time)           2.3 53.735 < 2.2e-16 ***
Chick
Diet
lo(Time):Chick
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
ggplot(cw,aes(x=Time,y=fit,group=Diet,color=Diet,group=Chick)) +geom_point() +
    geom_line() +theme_bw()
plot(g)
```

Figure 13.6: GAM model 2, with a different scale parameter for each chick



The `lo()` and `ns()` functions are actually supported by another library called splines, and can be mixed into lm models directly instead of using gam. However, the output is not nearly as interpretable.

```
  g <-  lm(weight~ns(Time,5)+ Diet,data=cw)
2 summary(g)
  anova(g)
4
```

Here, `anova(g)` is OK but not as good as the `gam`, but `summary(g)` is hard to understand.

```
2
> anova(g)
4 Analysis of Variance Table

6 Response: weight
  Df  Sum Sq Mean Sq F value    Pr(>F)
8 ns(Time, 5)   5 2066375  413275 327.276 < 2.2e-16 ***
  Diet          3  129664   43221  34.227 < 2.2e-16 ***
10 Residuals   569  718516    1263
  ---
12 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

14
  > summary(g)
16
  Call:
18 lm(formula = weight ~ ns(Time, 5) + Diet, data = cw)

20 Residuals:
  Min       1Q   Median       3Q      Max
22 -143.469  -17.405   -0.297   15.554  135.197

24 Coefficients:
  Estimate Std. Error t value Pr(>|t|)
26 (Intercept)    24.641      5.060   4.869 1.45e-06 ***
  ns(Time, 5)1   56.396      8.306   6.790 2.83e-11 ***
28 ns(Time, 5)2   94.421      9.798   9.637  < 2e-16 ***
  ns(Time, 5)3  141.708      7.886  17.969  < 2e-16 ***
30 ns(Time, 5)4  179.697     12.059  14.902  < 2e-16 ***
  ns(Time, 5)5  168.083      5.675  29.616  < 2e-16 ***
32 Diet2          16.095      4.034   3.990 7.48e-05 ***
  Diet3          36.428      4.034   9.030  < 2e-16 ***
34 Diet4          30.268      4.055   7.464 3.17e-13 ***
  ---
36 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
  Residual standard error: 35.54 on 569 degrees of freedom
38 Multiple R-squared:  0.7535,  Adjusted R-squared:   0.75
  F-statistic: 217.4 on 8 and 569 DF,  p-value: < 2.2e-16
40
```

13.5 Fitting regression interactions

Similar to polynomial regression, you might believe that the outcome variable is linearly related to the product of two predictors. This is essentially an interaction, because the

impact of one predictor depends on the impact of another predictor.

```
x <- -100:100
y <- runif(201)*201-100
z <-  3*x -5*y + 400*x*y - 50

library(rgl)
plot3d(x,y,z)
```

Figure 13.7: 3-D scatterplot illustrating interaction between predictors

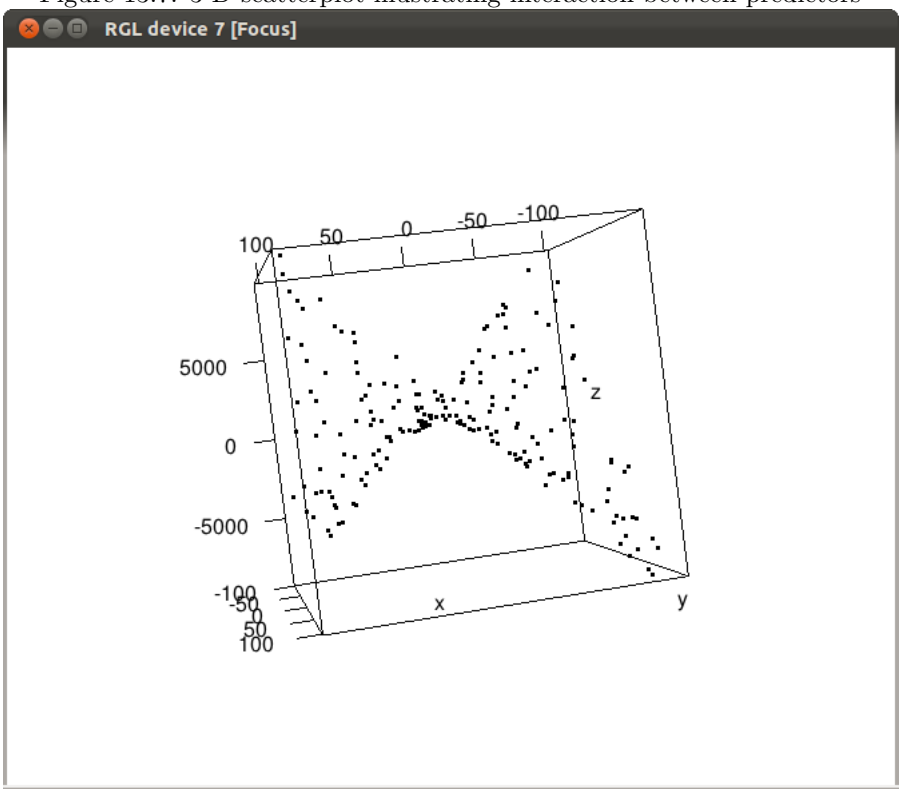


Figure 13.7 shows the relationship between x and y, in which there is a saddle point in the center. Let's add some noise and try to estimate the predictors. Each of the models below estimate the same parameters.

```
z <-  3*x -5*y + x*y - 50 + rnorm(201)*2000

xy <- x * y

interaction <- lm (z~x+y + x:y)
interaction2 <- lm(z~x * y)
interaction3 <- lm(z~x + y + xy)

summary(interaction2)

Call:
```

```
lm(formula = z ~ x * y)

Residuals:
    Min      1Q  Median      3Q     Max
-4854.1 -1291.9   106.4  1171.0  3630.4

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -186.12240  129.77901  -1.434  0.15312
x              6.13313    2.24701   2.729  0.00692 **
y             -6.08756    2.10473  -2.892  0.00425 **
x:y            0.98706    0.03593  27.472  < 2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

Notice how the estimates of the main effects are somewhat distorted (the true numbers were 3 and 5); but we were fairly accurate on the interaction.

13.6 Transformations of the Outcome or Predicted Variable

Sometimes, a transformation on the input or output variable is effective. Often, when using t-tests, we think about transformations to allow error variance to be symmetric/normal. So, we may transform response time (RT) to $log(RT)$, because it will take the long-tailed distribution and make it more symmetric, and so it will no longer violate assumptions of normality too much.

But when using regression, which assumes linear functions, a transformation may allow us to fit a non-linear function. In contrast to polynomial regression, which transforms the predictor, can also think about transforming the predicted variable. This can often be justified in terms of an underlying process.

For example, if we think back to the chick weight data set, a chick's weight gain on each day has to depend somewhat on its current weight. This is an exponential process. Growth, when it starts, is usually well described as an exponential process. Later, as physics, energy, space, or predators limit the growth it usually asymptotes, but initially, the amount you grow will often depend proportionally on your size. to model an exponential process, this is just a linear relationship with the log of the outcome. This is the so-called exponential growth model, whose parameters can be fit by taking the log of the dependent variable (see model `lm2`). Model `lm3` is a power function model, which is implicated in many psychological learning functions.

```
data(ChickWeight)
lm1 <- lm(weight~Time:Diet,data=ChickWeight)
lm2 <- lm(log(weight)~Time:Diet,data=ChickWeight)
lm3 <- lm(log(weight)~log(Time+1):Diet,data=ChickWeight)
```

Comparing these three models, the best fit is the exponential growth model. You can recover the growth rate parameters by exponentiating:

```
 > exp(lm2$coef)
(Intercept)  Time:Diet1  Time:Diet2  Time:Diet3  Time:Diet4
  44.824265    1.068984    1.079703    1.093056    1.090992
```

In other words, Diet 1 leads to a 6.8% increase in weight each day, diet 2 a 7.9% increase, and so on.

We might consider fitting a different exponential function to each chick's growth rate:

```
lm22 <- lm(log(weight)~Time:Chick,data=ChickWeight)
> lm22

Call:
lm(formula = log(weight) ~ Time:Chick, data = ChickWeight)

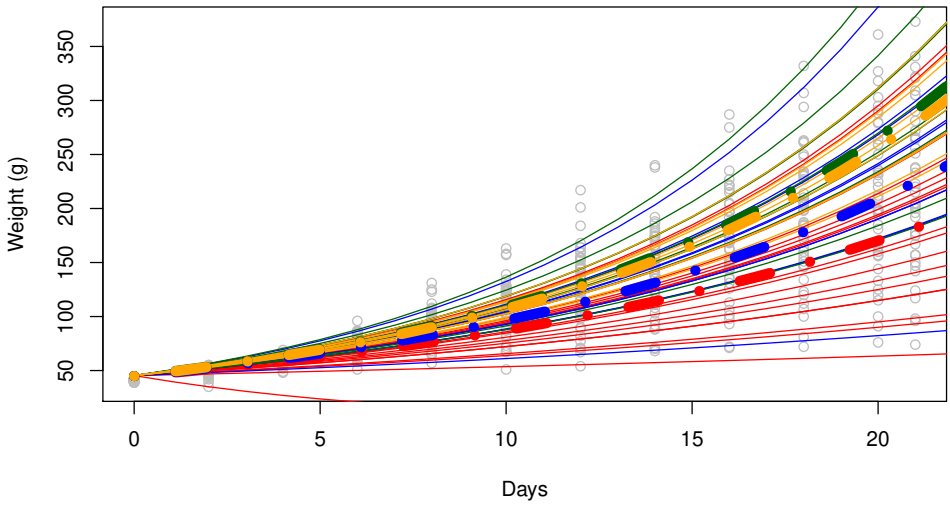
Coefficients:
 (Intercept)  Time:Chick18  Time:Chick16  Time:Chick15  Time:Chick13   Time:
    Chick9
     3.81442      -0.12954       0.01675       0.03698       0.03430
    0.04649
Time:Chick20  Time:Chick10   Time:Chick8  Time:Chick17  Time:Chick19   Time:
    Chick4
     0.04643       0.05070       0.06240       0.05783       0.05391
    0.06387
 Time:Chick6  Time:Chick11   Time:Chick3   Time:Chick1  Time:Chick12   Time:
    Chick2
     0.07275       0.08175       0.07527       0.07230       0.07399
    0.07754
 Time:Chick5  Time:Chick14   Time:Chick7  Time:Chick24  Time:Chick30  Time:
    Chick22
     0.08150       0.09364       0.09283       0.02981       0.06647
    0.06669
Time:Chick23  Time:Chick27  Time:Chick28  Time:Chick26  Time:Chick25  Time:
    Chick29
     0.07170       0.07171       0.08326       0.08364       0.08983
    0.08888
Time:Chick21  Time:Chick33  Time:Chick37  Time:Chick36  Time:Chick31  Time:
    Chick39
     0.10715       0.07005       0.06628       0.08573       0.08206
    0.08513
Time:Chick38  Time:Chick32  Time:Chick40  Time:Chick34  Time:Chick35  Time:
    Chick44
     0.08916       0.09638       0.09616       0.10094       0.11016
    0.07826
Time:Chick45  Time:Chick43  Time:Chick41  Time:Chick47  Time:Chick49  Time:
    Chick46
     0.07692       0.08832       0.08171       0.08158       0.08698
    0.08519
Time:Chick50  Time:Chick42  Time:Chick48
     0.09176       0.09258       0.09635


```

This gives us an individual growth curve for each chick. We can plot these against the actual data, as seen in Figure 13.8

```
diet <- t(table(ChickWeight$Diet,ChickWeight$Chick)>0) %*% c(1:4)
plot(ChickWeight$Time,(ChickWeight$weight),col="grey",
        xlab="Days",ylab="Weight (g)")
for(i in 2:length(lm22$coef))
{
  x <- 0:25
```

Figure 13.8: Individually-fitted growth rates for chick growth data set.




```
  fit <- exp(lm22$coef[1] + lm22$coef[i]*x)
  points(x,fit,type='l',col=c("red","blue","darkgreen","orange")[diet[i-1]])
}

 points(x, exp(lm2$coef[1] + lm2$coef[2]*x),type="l",col="red",lwd=8,lty=4)
 points(x, exp(lm2$coef[1] + lm2$coef[3]*x),type="l",col="blue",lwd=8,lty=4)
 points(x, exp(lm2$coef[1] + lm2$coef[4]*x),type="l",col="darkgreen",lwd=8,lty
    =4)
 points(x, exp(lm2$coef[1] + lm2$coef[5]*x),type="l",col="orange",lwd=8,lty=4)
```

The mean growth rates can be found using `tapply` on the coefficients (we are just transforming each growth rate coefficient by reversing the log-transform, and then finding the mean by diet):

```
tapply(exp(lm22$coef[2:length(lm22$coef)]),list(diet),mean)
     1        2        3        4
1.055595 1.079074 1.092302 1.089788
```

Are these any different from the earlier estimates? Why?

Logarithmic transforms are often done for response time data, which tend to be skewed to the right. if we look at the letter-length values and compute time-per-letter across trials:

```
let <- read.csv("reading.csv",header=F)
hist(let$rt/let$length,breaks=100)
hist(log(let$rt/let$length),breaks=100)
```

Notice that here, the transform does a good job of approximating a normal distribution. Now, your data will conform better to the assumptions of your inferential statistics.

A logarithm transform is useful whenever you have a long tail or a process that is proportional, like growth, and the values are all positive.

This might include:

- money/sales across companies
- employees across divisions/companies/jurisdictions
- population
- search/response/completion times

If there are meaningful proportional differences at all scales, a log transform might help. Otherwise, if you have large-scale values mixed in with small-scale ones, the large-scale values will also tend to swamp the model. For example, what is the relationship between county population and number of police/sheriffs/enforcement in a county? The 1000+ counties in america with under 5000 people in them might either not matter at all compared to the 6 counties with 3,000,000+; or the top counties may completely drive any relationship and the bottom 1000 may not matter.

```
census <- read.csv("census2013.csv")
labels <- census$GEO.display.label
states <-sapply(labels, function(x){strsplit(as.character(x),", ")[[1]][2]})


sum(census$respop72011<5000)
hist(census$respop72011,breaks=100)
hist(log(census$respop72011),breaks=100)

untrans <- aggregate(census$respop72011,list(state=states),mean)
logged <- aggregate(census$respop72011,list(state=states),function(x){exp(mean
    (log(x)))})
untrans$logged <- logged$x

barplot(t(as.matrix(untrans[,-1])),names=untrans$state,beside=T,horiz=T,las=1)
```

Look at places like NY, NV, and California, etc. Here, the log-transformed population size may better represent the typical county size, rather than being influenced by a few very populous counties.

13.6.1 Additional Transformation

There are only a few other transformations that are commonly used, although there are many that might be used in different narrow domains

Inverse transform . The inverse transform may be more appropriate to think about fitting $1/y$ instead of y.

Log-odd . If you are dealing with probability, the log-odds transform is often used. You can transform a probability p into log-odds $log(p/(1-p))$ with a function like this (the logistic is the inverse, allowing us to transform back to the observed data values).

```
logodds <- function(p){log(p/(1-p))}
logistic <- function(x){1/(1+exp(-x))}
```

This transform forms the basis for logistic regression. This is probably most useful in the mid-range of probabilities.

arcsine tranform. An arcsine transform is often used when dealing with accuracy or probability data near 1.0 (or near 0, if you subtract from 1.) arcsine ¡- function(p) asin(sqrt(p))

Some examples:

```
cond <- sample(1:15,1000,replace=T)
success <- runif(1000)< 1-.4^(cond/3)
cond[1:15] <- 1:15  ##make sure we have no perfect accuracies
success[1:15] <- 0

pc <- aggregate(success,list(cond=cond),mean)
plot(pc)

lm1 <- lm(pc$x~pc$cond)
lm2 <- lm(logodds(pc$x)~pc$cond)
lm3 <- lm(asin(pc$x)~pc$cond)

plot(pc$cond,pc$x,xlab="Condition",ylab="Observed value",ylim=c(0,1.1)) ##
    pretty bad
abline(lm1$coef)

##prediction of log-odds model
points(pc$cond, logistic(predict(lm2)),type="l",lwd=2,lty=3,col="cadetblue3")
    ##pretty good
points(pc$cond, sin(predict(lm3)),type="l",lwd=2,lty=4,col="orange")
legend(8,.4,c("Linear","Log-odds","Arcsine"),lty=c(1,3,4),col=c("black","
    cadetblue3","orange"),
lwd=3)
```

The disadvantage to using simple transformations such as this is that in order to do inferential statistics, you still assume the error distributions are normally distributed, which may not be true. Methods known as *generalized linear regression* (glm) provide much more flexibility, and enable proper testing of models that use transformations of the predicted variable. These include logistic regression, which provides a more appropriate error model for the data.

Tukey's Ladder of Powers Tukey (1977) Framed a set of transforms related through a power function, indexed by a coefficient λ. The size and sign of lambda controls how much the transform unskews the results.

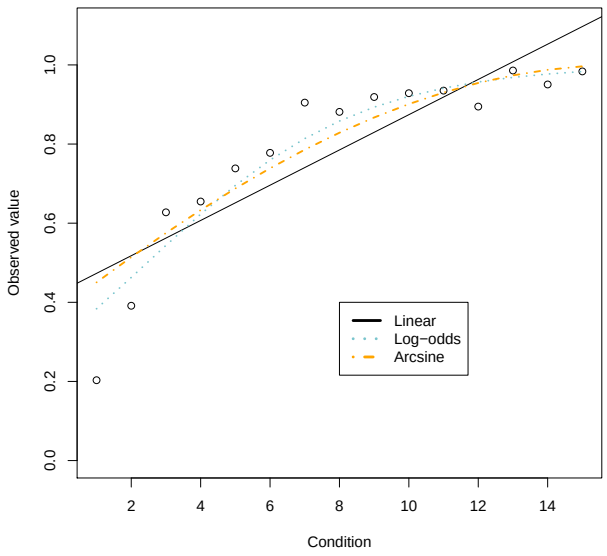
$$\lambda > 0 \quad = \quad x^{\lambda} \tag{13.1}$$

$$\lambda = 0 \quad = \quad log(x) \tag{13.2}$$

$$\lambda < 0 \quad = \quad -1/x^{\lambda} \tag{13.3}$$

For example, a power of 2 squares the results, which might be good if you have a compressed range against an upper bound. A power of 1 is no transform indicates a is log(x) transform, and so on. You can try different values of lambda and make qq-plots and histograms to find the best transform, but within the rcompanion library, they have implemented a function that does this for you. Furthermore, it tests the resulting data against a test for non-normality (Shapiro's W) and picks the transform value that is best according to that statistic.

Figure 13.9: Non-linear data with best-fitting model having no transform, a log-odds transform, and an arcsine transform. The transformed data fit the curvature of the data better than the linear model.




```
library(rcompanion)
z2 <- transformTukey(z,plotit=T)

lambda      W Shapiro.p.value
395   -0.15 0.9114       5.038e-06

if (lambda >  0){TRANS = x ^ lambda}
if (lambda == 0){TRANS = log(x)}
if (lambda <  0){TRANS = -1 * x ^ lambda}
```

Here, z2 is transformed value, and we see that the best transform was $\lambda = -.15$. We can see that we could transform our data using this, or just use z2 directly, as they are the same thing:

```
plot(-z^(-.15),z2)
model3 <- lm(z2~x)
> summary(model3)

Call:
lm(formula = z2 ~ x)

Residuals:
Min       1Q   Median       3Q      Max
-0.57619 -0.07383  0.01895  0.09803  0.60475

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.417429   0.031525 -44.962  < 2e-16 ***
```

```
x           -0.010664   0.002795  -3.816 0.000238 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.1567 on 98 degrees of freedom
Multiple R-squared:  0.1294,  Adjusted R-squared:  0.1205
F-statistic: 14.56 on 1 and 98 DF,  p-value: 0.0002376
```

We might look back at the census data:

```
census2 <- transformTukey(census$respop72011)

lambda      W Shapiro.p.value
399  -0.05 0.9956       5.571e-08

if (lambda >  0){TRANS = x ^ lambda}
if (lambda == 0){TRANS = log(x)}
if (lambda <  0){TRANS = -1 * x ^ lambda}
```

Here, a transform close to 0 (log) is ideal–and we previously used log as a good transform..

Box-Cox Transform Another general parametric transform is the Box-Cox transform, the result of the only collaboration of two famous statisticians [1]. It takes the form:

$$(x^\lambda - 1)/\lambda \tag{13.4}$$

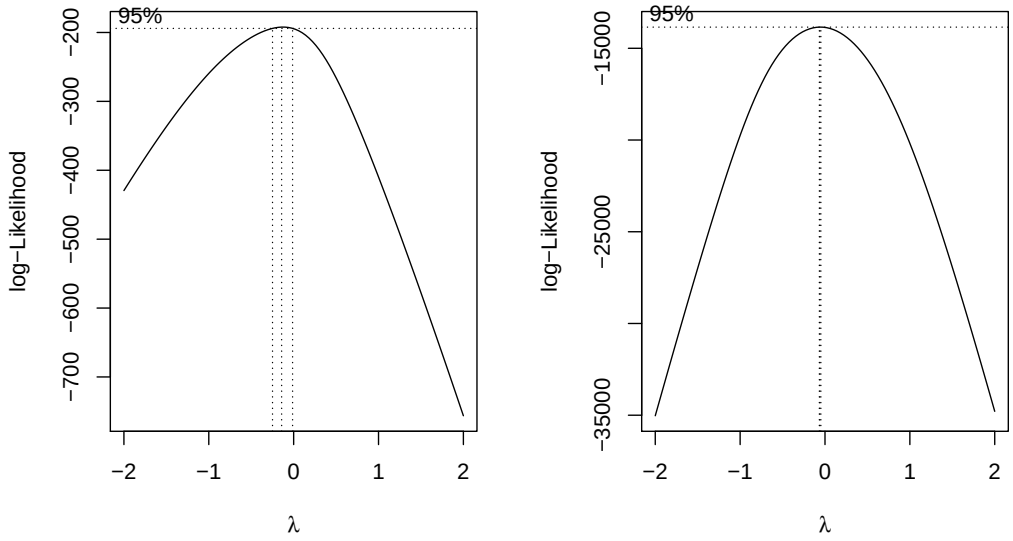
This is similar to the Tukey transform, but is scaled by λ, which will automatically incorporate the -1 value into it. Furthermore, as lambda approaches 0, this transform approaches $log(x)$, and so the transformations are perhaps less ad hoc that Tukey's method. There are a number of libraries that contain similar Box-Cox transform functions, including the `MASS` library function:

```
bc<-boxcox(z~x,plotit=T)
bc2 <- boxcox(census$respop72011~census$GEO.id2,
  plotit=T)
```

This plots the goodness of fit for different transforms values. We can see that the best point is identifies the same values of lambda as the Tukey transform did–which is nice because they are essentially the same transform. Once a reasonable transform is identified, you simply need to use the transform function on your data with the best λ value.

[1] Box, G. E. P. and Cox, D. R. (1964). An analysis of transformations, Journal of the Royal Statistical Society, Series B, 26, 211-252.

Figure 13.10: Goodness of fit plots for Box-Cox transforms for the z/x and census data.



13.7 Solution to exercises

Exercise 13.1 Solution

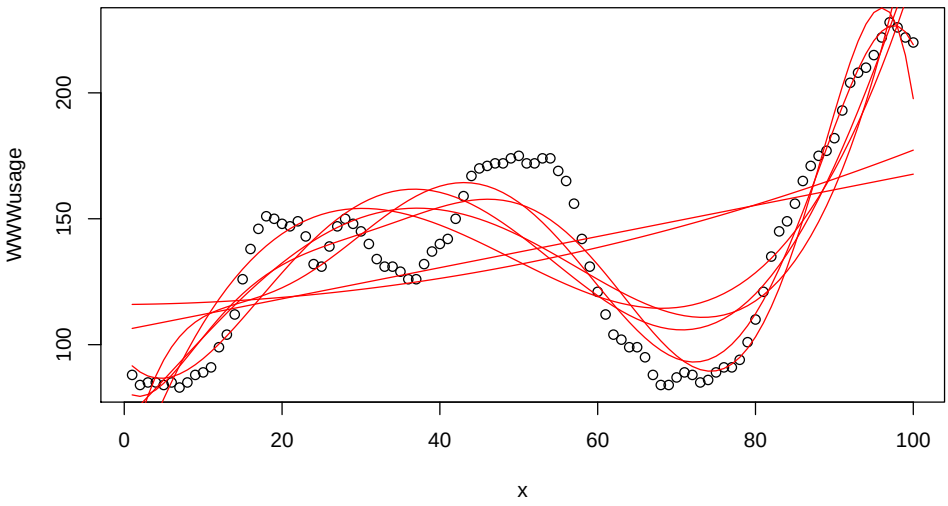
Make a polynomial model to fit the `WWWusage` data set. First, let's fit a set of models with 1 to 7th order polynomials:

```
x<- 1:100
g1 <- lm(WWWusage~poly(x,1))
g2 <- lm(WWWusage~poly(x,2))
g3 <- lm(WWWusage~poly(x,3))
g4 <- lm(WWWusage~poly(x,4))
g5 <- lm(WWWusage~poly(x,5))
g6 <- lm(WWWusage~poly(x,6))
g7 <- lm(WWWusage~poly(x,7))
```

Now, let's plot the data, and each model:

```
plot(x,WWWusage)
points(g1$fit,type="l",col="red")
points(g2$fit,type="l",col="red")
points(g3$fit,type="l",col="red")
points(g4$fit,type="l",col="red")
points(g5$fit,type="l",col="red")
points(g6$fit,type="l",col="red")
points(g7$fit,type="l",col="red")
```

Figure 13.11: Successive polynomial fits to `WWWusage` data



Chapter 14

Verifying assumptions via diagnostic tests and Outlier detection

We will use the car library in this chapter.

When using regression-type models, we should always take care to verify, to whatever extent we can, that the assumptions made by the model are not violated. Many times, the assumptions cannot be tested, but when there are violations, we can sometimes do something about it.

To make sure you have a robust and accurate model, you should check to see whether the model's assumptions are violated. Some of the important assumptions made by a linear regression model include:

- Errors is distributed normally. This can be violated severely if errors are asymmetric, or less seriously if errors have longer or shorter tails than normal distributions.
- Variance is independent of x and y values. The model assumes that error has the same variance in for all levels of the predictor, and for all combinations of factors
- There is a linear relationship between IV and DV in the region of the data
- There is an independence of predictors (unless a interaction term is included in the model) (see issues related to multi-colinearity)
- Observations are independent. You can't improve the accuracy of a poll by asking the same person who they are voting for 1000 times, because you are not sampling independently.
- There is an unrestricted range of values. The linear model does not know if your data are limited in range.
- Data are sampled in an unbiased way from the population you want to generalize to.

Any of these can fail, and they often do. The consequences of violating these assumptions include:

- The model's parameters will be incorrect

- Your inferences about the population's parameter values will be incorrect
- Your ability to detect reliable differences between groups will suffer
- Your ability to make predictions about new situations will be harmed.

Depending on which assumptions fail, we have several responses:

- Simply recognize the limitations of the model.
- Use a more general model that accounts for or models the problems explicitly. These include generalized linear models (glms), ridge/LASSO regression, non-linear regression, quantile regression, mixture modeling and others).
- Variable/model selection that removes predictors with particular problems.
- Add additional predictors and/or mixed-effects models to capture non-independence from repeated-measures designs.
- Weighted regression to help reduce non-equal sampling.
- Transform variables (predictors or predicted) to reduce the impact of the violation.
- Use an outlier removal process that trims extreme and influential values.

In any case, the inference will be more reasonable if you determine the process upfront and apply the solution judiciously and not because the model does not support your initial hypotheses.

14.1 Assessing the overall goodness of fit of the model

The first thing to look at in the model is how much of the variance it accounts for.

```
2  set.seed(302)
   x <- rnorm(1000)
4  y <- 33 + x + rnorm(1000)
   modela <- lm(y~x)
6  summary(modela)
   anova(modela)
8
   > summary(modela)
10
   Call:
12 lm(formula = y ~ x)

14 Residuals:
   Min      1Q  Median      3Q     Max
16 -3.1614 -0.7238  0.0220  0.6774  3.8889

18 Coefficients:
   Estimate Std. Error t value Pr(>|t|)
20 (Intercept) 32.96103    0.03252 1013.45   <2e-16 ***
   x            0.98448    0.03358   29.32   <2e-16 ***
22 ---
   Signif. codes:  0 ‘***‘ 0.001 ‘**‘ 0.01 ‘*‘ 0.05 ‘.‘ 0.1 ‘ ‘ 1
24
   Residual standard error: 1.028 on 998 degrees of freedom
26 Multiple R-squared:  0.4628,  Adjusted R-squared:  0.4623
```

```
F-statistic: 859.8 on 1 and 998 DF,  p-value: < 2.2e-16

> anova(modela)
Analysis of Variance Table

Response: y
      Df  Sum Sq Mean Sq F value    Pr(>F)
x      1  909.01  909.01  859.76 < 2.2e-16 ***
Residuals 998 1055.17    1.06
---
Signif. codes:  0 ‘***‘ 0.001 ‘**‘ 0.01 ‘*‘ 0.05 ‘.‘ 0.1 ‘ ‘ 1

cor(x,y)^2
[1] 0.4627926

(909/(1055+909))
[1] 0.462831
```

Notice that the summary provides an R^2 value of .46, which we interpret as proportion of variance. We could calculate that by hand using $cor(x, y)^2$ and get the same value. In the anova table, it tells us the sum square error attributable to both x and the residuals, and SSE is just a measure of variance accounted for–we can calculate the same value by dividing 909 by 909+1055, which helps us understand the link between R^2 and SSE. It is not an accident that these are the same–correlation is just a scaled measure of variance, and by dividing variance explained by total variance, we are just scaling it, and the formulas will turn out to be identical.

Let's look at some more models with more variability:

```
y2 <- 33 + x + rnorm(1000)*5
modelb <-lm(y2~x)

set.seed(100)
y3 <- y
noisy <- sample(1000,15)
y3[noisy] <- y3[noisy] + rnorm(15)* 35
modelc <- lm(y3~x)

summary(modelb)$r.squared
[1] 0.02660593

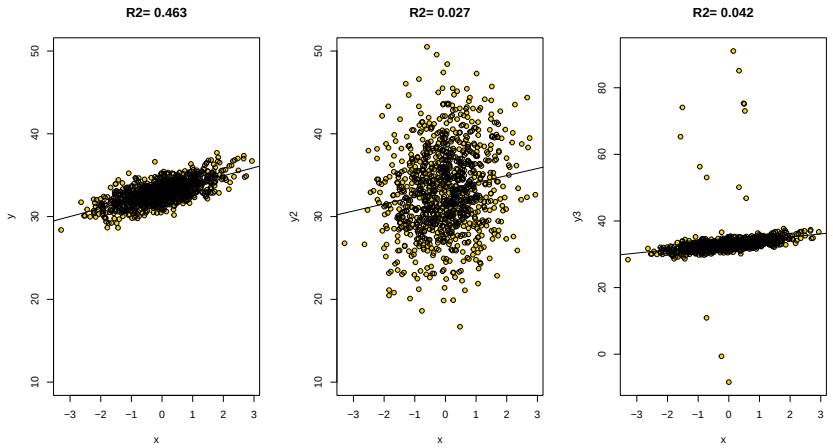
summary(modelc)$r.squared
[1] 0.04171009
```

```
par(mfrow=c(1,3))
plot(x,y,col="gold",pch=16,ylim=c(10,50),
main=paste("R2=",round(summary(modela)$r.squared,3)))
points(x,y)
abline(modela$coef)

plot(x,y2,col="gold",pch=16,ylim=c(10,50),
main=paste("R2=",round(summary(modelb)$r.squared,3)))
abline(modelb$coef)
points(x,y2)

```

Figure 14.1: Simple model with low or high noise accounts for different amounts of variability.




```
plot(x,y3,col="gold",pch=16,
main=paste("R2=",round(summary(modelc)$r.squared,3)))
points(x,y3)
abline(modelc$coef)
```

In the first model, we have accounted for 46% of the variance in the first model, but just 2.7% in the second model. The slope relationship between x and y is basically the same in both cases, but for y2 the noise is larger. And the R^2 values go way down, despite the fact that our estimate is still highly reliable. Whether a model with a R^2 value this small is acceptable depends on the context. For example, if we look at the third model, it has about the same R^2 as the second model, but looks like the first model with a few extreme values. Notice that the y scale is larger, so a few large outliers reduces R^2 exactly the same way a lot of systematic noise does.

How might context matter? For something like trying to predict life expectancy given a lifestyle factor (e.g., number of cigarettes smoked per day), you might expect a low R^2 given there are so many factors that determine how long somebody lives. For a carefully-designed lab experiment, where many things can be controlled, a low R^2 may indicate the effect has relatively little importance or new experiments should be conducted. In any case, comparing the second and third models, they have the same measure of goodness, but for different reasons. It might make sense, if we believe the dozen or so extreme points occur for another reason, to ignore them, or maybe model them specifically. These are often called 'outliers', and they can cause trouble for our models because they violate the assumptions of the model, and more importantly we might think they arise from another process we don't really care about. There are many statistical approaches to help us identify these points, which we will discuss in the rest of this chapter.

14.2 Testing assumptions of models

A regression model has four basic assumptions: linear impact of predictors; independent values; uniform variance (homoscedasticity) and normal error[1]. Each should be evaluated to the extent you can. Many times we have non-linear relationships, and the transforms and polynomial regression methods we used in the previous chapter deal with this. Non-independence is more difficult. We have already talked about multicolinearity, and there are some models that can be used to address that directly. But there are other aspects of independence, like a repeated measures experiment where a person acts as their own control. We will need to use mixed-effect models we will deal with later to handle that. In these cases, you might have many observations from a small number of people, the model is essentially constrained by the number of people, not the number of observations, as multiple observations within each person will be correlated, and so you need to deal with this in some way–perhaps by estimated separate parameters for each person. Finally. we usually want to test whether our residuals are normal. If not, this will violate some of the assumptions of our tests, and may give us incorrect inferences. The standard approach is to do histograms and qq-plots. We can also use the Shapiro test, which will turn out significant if the distribution differs from normality substantially. Here is an example of using the shapiro test

```
set.seed(111)
x <- rnorm(20)
y <- x + rnorm(20)
g1 <- lm(y~x)


hist(g1$resid)
qqnorm(g1$resid)
shapiro.test(g1$resid)


Shapiro-Wilk normality test

data:  g1$resid
W = 0.98268, p-value = 0.9639
```

This sort of testing should always be part of your initial look at the data, and you can use this to help choose the right transforms of your input or output variables.

14.3 Detecting and Handling Influential Observations and Outliers

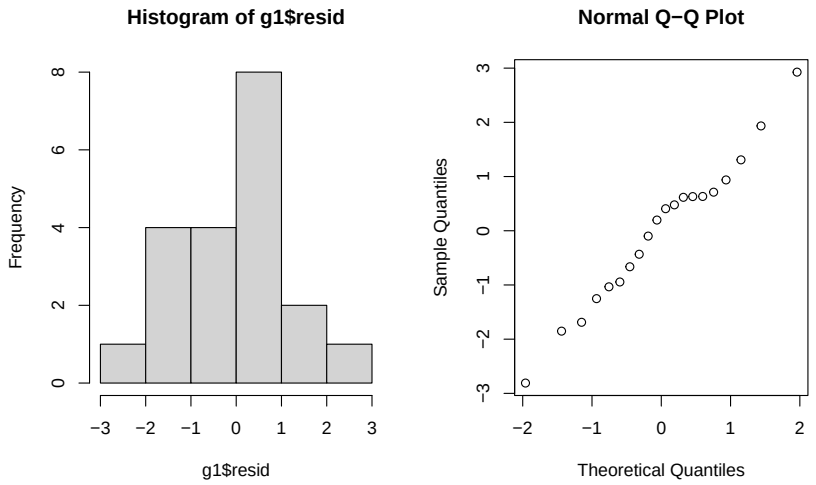
Linear regression and ANOVA models are fairly robust, even to non-normal variance. But if you have a relatively few number of observations, individual points can have a large influence on the model. It is good to know this, especially in correlational designs, because if your correlation depends on a single observation, it cannot be very robust.

Lets start with a simple model:

```
 set.seed(111)
 x <- rnorm(20)
 y <- x + rnorm(20)
```

[1] See Ernst & Albers, 2017, who demonstrated that many users of regression fail to perform tests of these or evaluate their models in terms of these assumptions https://peerj.com/articles/3323/

Figure 14.2: Histogram and qq-plot of the residuals of a lm with normal error. This conforms to a normal distribution, so the W is close to 1, as is the p-value.




```
plot(x,y)
cor(x,y)

[1] 0.5127194
```

The correlation here is about .5–pretty good. But looking at it, if we excluded the one point at the upper right (number 13), the correlation surely would be much worse:

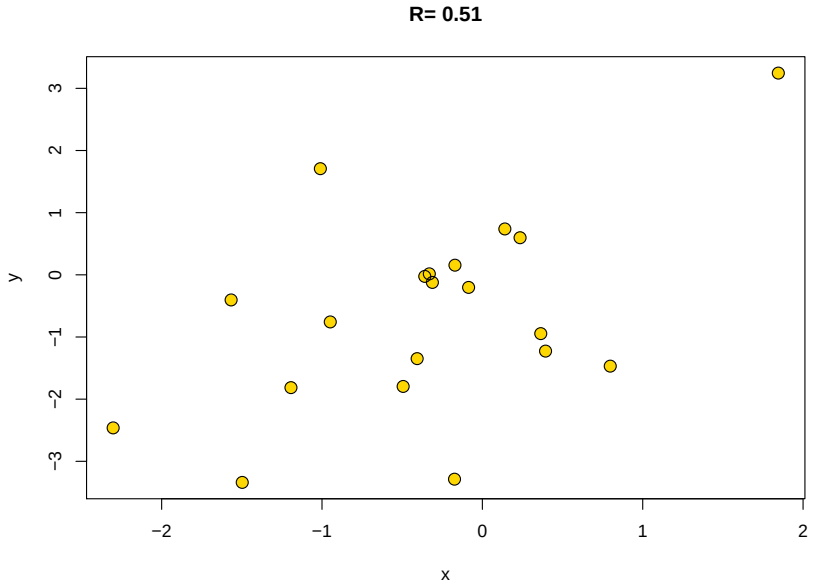
```
cor.test(x[-13],y[-13])

Pearson's product-moment correlation

data:  x[-13] and y[-13]
t = 1.1994, df = 17, p-value = 0.2468
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
-0.2002990  0.6509457
sample estimates:
cor
0.2793224
```

It looks like observation 13 is a big deal–without it, correlation will drop down to .279, and it is no longer a reliable correlation. What should we do? If you were hoping to find no correlation, it would be very easy to throw it out and justify it. After all, the correlation depends on that single point. If you were hoping for a correlation, it would be very easy to ignore the fact that the correlation is sensitive to the inclusion of that single point. Plus, point 13 is the highest observation for x as well as y. Maybe this is the only person in your sample that is an expert in the domain, and so this comparison is actually critical. This dilemma shows how difficult it is to deal with this issue, because we can see what we want to see–and that has little to do with the statistics.

Figure 14.3: A small data set with what looks like correlated predictors. But if we remove the value that is highest on both dimensions, will it still be correlated?



Remember that the process that created this particular data set contained a correlation, and because both variables were normally distributed, the extreme values of the distribution will necessarily contain relatively few points. In reality, this data set provides fairly poor evidence for a correlation because it is not very robust, despite the fact that the correlation is significant. A reviewer, editor, or reader is likely to question this as well, and publishing data such as this would invite future critical responses. This is an example of an influential observation, and there are a number of methods developed to examine them.

14.3.1 Examining Residuals and standardized residuals

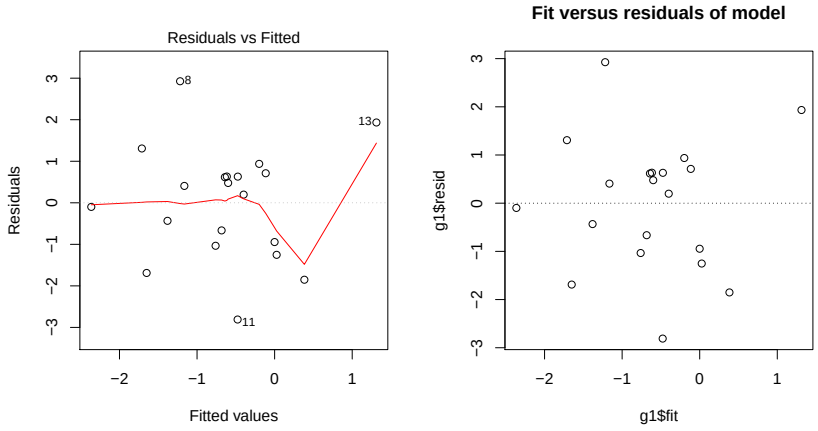
Regression and ANOVA assume that the error has equal variance across all levels of the IV. Thus, looking at the residuals can help you see if those assumptions are violated. The first plot provided by the plot() method of a `lm` or `aov` model will let you see which values have large residuals in comparison to the predicted values:

```
g1 <- lm(y~x)
plot(g1,which=1)
```

But this is essentially identical to:

```
plot(g1$fit,g1$resid)
```

Figure 14.4: Plotting fit fitted value by residuals to identify suspicious data.



Residuals for evaluating variance of estimates

Notice that this does not do well at detecting point 13–it isn't even the largest residual. This is partly because the point in influential, and so it bends the best-fit line toward itself. But it could be useful for understanding when the variance actually depends on the value. Really what we are looking for in this figure is whether the residuals are greater in one part of the fit than in others, as this could suggest doing a transform or changing the model in some way, as in the next example

```
x2 <- runif(200)*10
y2 <- 33 + x2 + rnorm(200)*(x2)

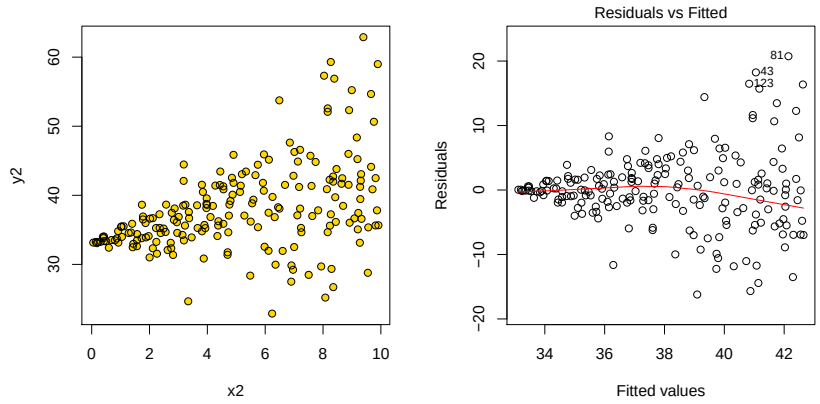

par(mfrow=c(1,2))
plot(x2,y2,col="gold",pch=16)
points(x2,y2)

g2 <- lm(y2~x2)
plot(g2,which=1)
```

The next example (model g2) example is troubling because variance depends on the absolute level of the data, and the fitted-by-residuals graph shows this. This violates the basic assumptions of the regression model, which assumes variance is uniform. Perhaps a transformation, or a more advanced generalized regression model that captures this type of variance would be better. It would be important for inference, because it means that your precision for lower fitted values should be much better than for higher fitted values. This can be especially important for categorical (ANOVA) models, because distinct variance values for different levels of a factor may compromise your tests.

One test for non-constant variance (`ncvTest` in `car`)uses the χ^2 (Chi-squared) distribution as a null hypothesis. The distribution is the same as we have used previously for examining cross-tabs counts, but we are using it in a very different context and you can think of this use as unrelated to the test of categorical relationship. If you have non-constant variance, there are steps you can take if necessary, including non-parametric tests, and using weighted least squares to give those values less weight.

Figure 14.5: Example data set with non-constant variance




```
library(car)
ncvTest(g2)
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 66.08247, Df = 1, p = 4.3245e-16
```

Standardized residuals

Sometimes it can be useful to standardize these residuals, so that your values have a variance of 1.0. These then become like z-scores, and you can use standard insights about the normal distribution to judge whether something is an outlier.

```
(g1$resid/sd(g1$resid))
plot(g1$fit,g1$resid/sd(g1$resid))

plot(g1$fit,rstandard(g1))

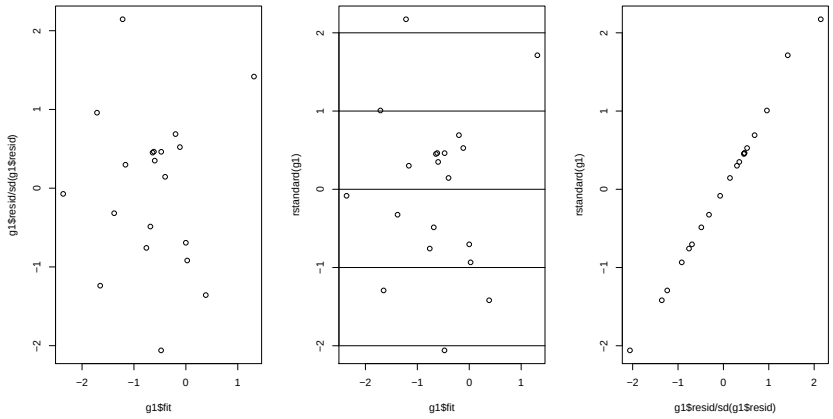
abline(-2,0)
abline(-1,0)
abline(0,0)
abline(1,0)
abline(2,0)

plot(g1$resid/sd(g1$resid),rstandard(g1))
```

The 'standardized residuals' are almost, but not exactly, what we get from simply dividing the residuals by their s.d.:

```
cor(g1$resid,rstandard(g1))
[1] 0.9983141
```

Figure 14.6: Figure depicting standardized residuals. The left panel shows the fitted value by residuals standardized by hand. The center shows plots by `rstandard`, which is identical (right panel).



Residuals can also be useful in identifying outliers. But if you look at the plot, you'll notice that some of the largest residuals for this data set are in the middle of the data set, and probably couldn't impact your model fit very much. Others, like point 13, are not the largest residuals, but because it is so far out at the edge of the data, it could have a larger influence. This notion is called 'leverage'.

14.3.2 Leverage

One measure you can look at to assess the importance of specific data points is called 'leverage'.

Details of computing leverage are covered in other more detailed texts. It relies on the so-called *hat* matrix, which is central to computing the best estimates of the linear regression model. It results from a set of operations for scaling and transforming the predictor variables (without considering the DV), and it can indicate which data points will have a large influence on the parameter estimates.

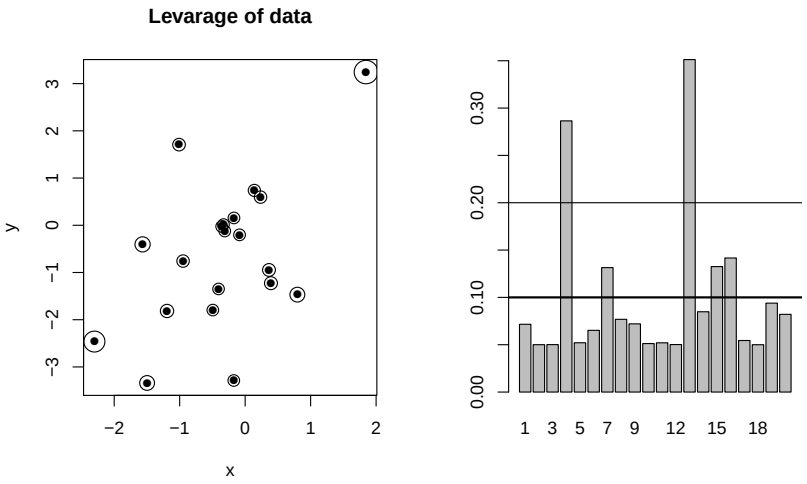
```
g1 <- lm(y~x)
leverage <- hatvalues(g1)
plot(x,y,pch=16,main="Leverage of data")
points(x,y,cex=1.2+leverage*5)
```

Notice that leverage is essentially MSE when you have a single predictor variable:

```
cor((x-mean(x))^2,leverage)
[1] 1.0
```

You can use leverage to assess which points may have a large influence, by looking at those whose means are large–in this case some are maybe twice as big as the average.

Figure 14.7: Figure depicting the leverage of each point. In the left panel, the size of the outer circle is proportional to its influence or leverage in the model. In the right panel, leverage is plotted as a bar graph.




```
barplot(leverage)
abline(mean(leverage),0,lwd=2)
```

We can use a criterion of 2 * mean to judge observations with strong leverage.

```
abline(2*mean(leverage),0)
```

When you have more than one predictor, leverage is related to its overall periphery:

```
x2 <- runif(20)
model2 <- lm(y~x+x2)
lev2 <- hatvalues(model2)  ##Does not depend on y!
plot(x,x2,pch=16)
points(x,x2,cex=1.2+lev2*5)
```

Points that are high in leverage are likely to have a large influence on the beta weights, although you can't know that until you use them to fit y. In other words, the point draws the best-fit line toward it, so it might not have large residuals, but it is still influential. Of course, if that point is a critical observation, its leverage might be justified. One method for combining residuals with leverage is the so-called 'studentized residual'

14.3.3 Studentized Residuals

Studentized residuals are standardized residuals that also scale by a function of the leverage. They are scaled so they conform to the Student's t distribution, which is why they are called "Studentized" with n-p degrees of freedom. These are sometimes as 'internally' studentized.

Figure 14.8: Leverage for two predictors. Leverage does not depend on the outcome; only the input variables.

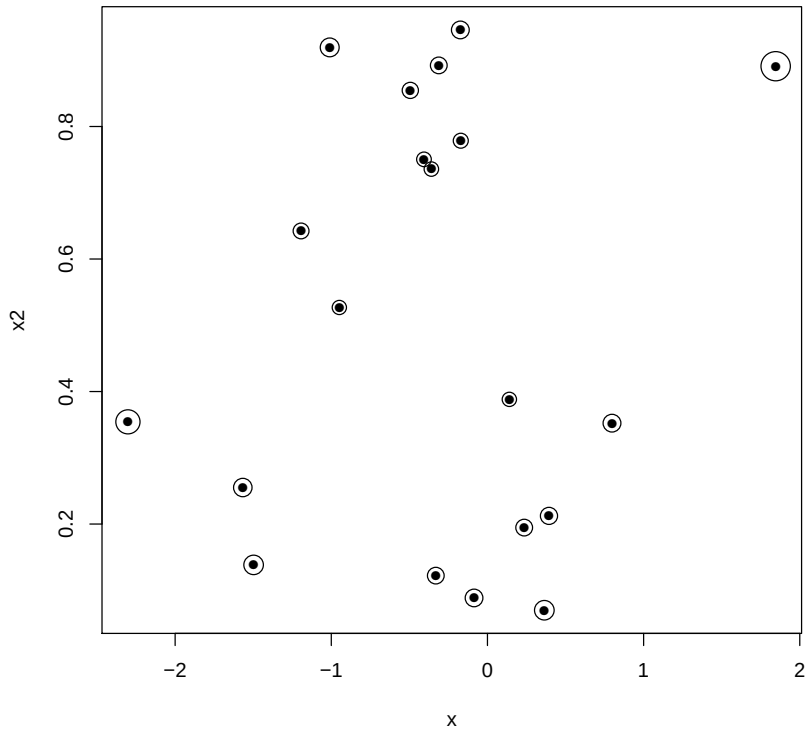
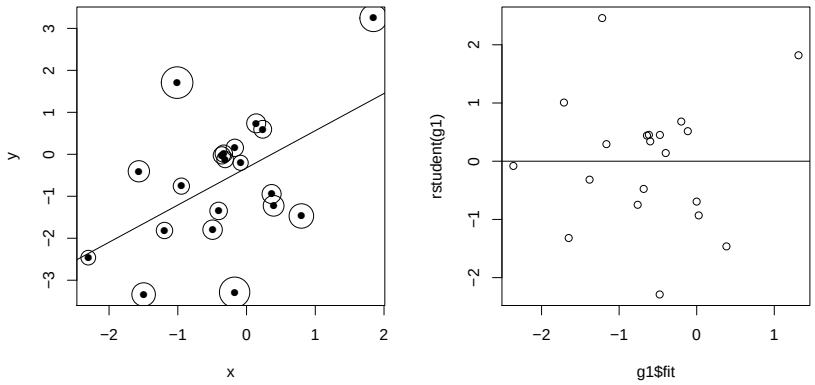


Figure 14.9: Studentized residuals combine leverage and residuals. Left panel shows leverage versus residuals; right panel shows studentized residuals.




```
rstudent(g1)

par(mfrow=c(1,2))
plot(x,y,pch=16)
points(x,y,cex=abs(rstudent(g1))+2)
abline(lm(y~x)$coef)
plot(g1$fit,rstudent(g1))
abline(0,0)
leverage <- hat(model.matrix(g))
```

Now, the large outliers near the middle end up having larger studentized residuals than the values at the extreme values of x that are fit well by the model. You could compare the residuals to a t-distribution and use that as a way of automatically removing points (maybe any value that is more than 3 or 4 t-values from 0). In this case, we'd be most likely to remove two points near the middle, which probably wouldn't impact our fit much but would reduce the MSE.

14.4 Measures of Influence

Some of R has a number of measures to assess the influence of individual observations. See:

```
> ?influence.measures

ig <- influence(g1)


ig
 ig
$hat
1           2           3           4           5           6           7           8
0.07160086 0.05003288 0.05011033 0.28648170 0.05208194 0.06519767 0.13146519
    0.07683949
9          10          11          12          13          14          15          16
```

```
0.07203051 0.05122500 0.05201870 0.05017418 0.35124947 0.08482338 0.13254122
    0.14162732
17         18         19         20
0.05446849 0.05000183 0.09393275 0.08209710

$coefficients
(Intercept)             x
1   0.0482770175  0.0281181854
2   0.0336327981  0.0009529260
3   0.0256030097  0.0013178119
4  -0.0009758523  0.0167408986
5   0.0359294129  0.0075724511
6   0.0610587109  0.0308543225
7  -0.0482305841  0.1384937253
8   0.1126541425 -0.1295989963
9   0.0161650445 -0.0162275677
10 -0.0511365571  0.0095213488
11 -0.1599487948 -0.0332326825
12 -0.0341579823  0.0023037460
13  0.2934401160  0.4082637723
14 -0.0909598002 -0.0637254227
15 -0.1608837265 -0.1530641004
16  0.0354721703 -0.1151017647
17  0.0116680838  0.0034820260
18  0.0323779869 -0.0002193539
19 -0.0150689971  0.0250243022
20 -0.0677637407 -0.0460339332

$sigma
1        2        3        4        5        6        7        8        9
       10
1.430941 1.433495 1.437207 1.441835 1.433537 1.422807 1.373514 1.238560
    1.438475 1.418932
11       12       13       14       15       16       17       18       19
       20
1.260884 1.432603 1.319337 1.406728 1.359090 1.400878 1.441272 1.433925
    1.437876 1.422128

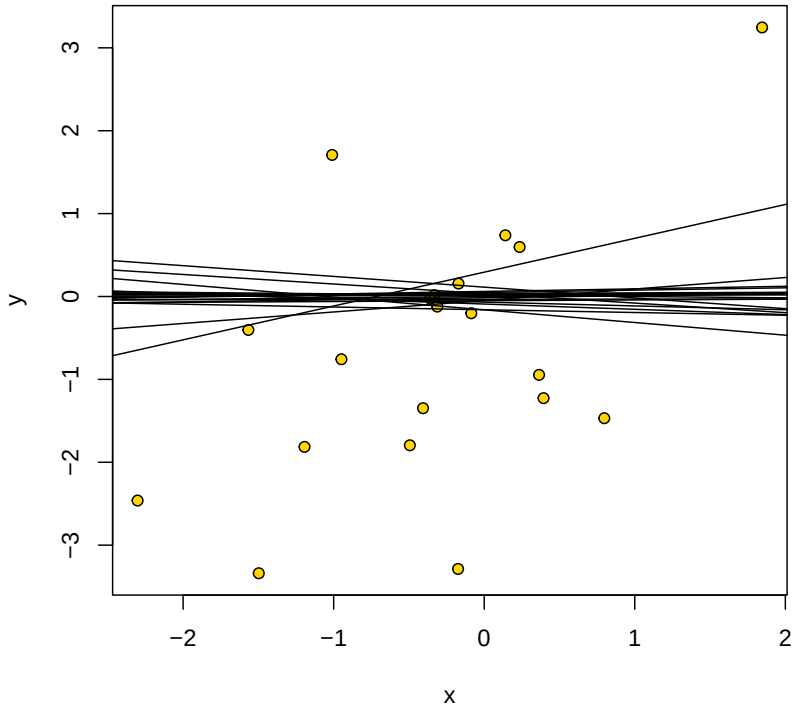
$wt.res
1           2           3           4           5           6           7
          8
0.71173379  0.63259708  0.47754529 -0.09842745  0.63038325  0.93751559
    -1.68873856  2.92632727
9           10          11          12          13          14          15
         16
0.40654255 -1.03424797 -2.80970187 -0.66436228  1.93368866 -1.25231425
    -1.85190143  1.30790596
17          18          19          20
0.19735974  0.61665474 -0.43347032 -0.94508977


plot(x,y)
for(i in 1:20)
  abline(ig$coef[i,])
```

The variable ig will contain the hat matrix (leverage), coefficients of models obtained by removing each point (`$coefficients`), residual standard deviation (RSE) (`$sigma`) obtained by removing each point, and `$wt.res`, which is essentially the square root of sigma (but with the positive or negative value.)

Here, we plot the coefficients a model describing the change in the model when each point

Figure 14.10: This shows the points, along with lines showing the amount the standard model changes when each point is removed. Point 13 produces a large change when present/absent, suggesting it is influential.



is removed–the `$coefficients` value. We can see that the line associated with one point (13) is fairly high. It also contains sigma–the similar value for dropping each case from the regression, and a weighted residuals value as well.

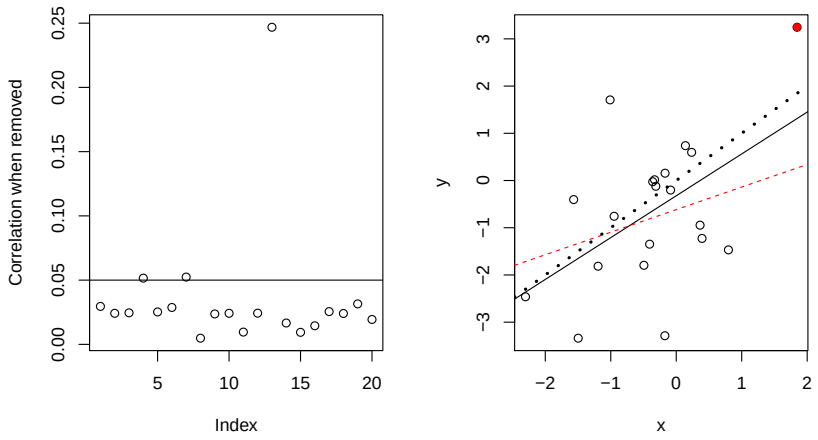
14.4.1 Jackknife Methods

What if we were to see the range of correlations we get by consecutively leaving one point out:

```
cors <- matrix(rep(0,20*3),ncol=3)
for(i in 1:20)
{
  keep <- rep(T,20)
  keep[i] <- F
  cortmp <- cor.test(x[keep],y[keep])
  cors[i,] <- c(  cortmp$estimate,cortmp$statistic,cortmp$p.value)
}

plot(cors[,3])
abline(.05,0)
```

Figure 14.11: Illustration of the Jackknife procedure.



The basic process of removing each data point and determining how it impacts the overall model is often called the "jackknife" procedure; possibly named after how a jackknife might have multiple blades which you can pull out and use one at a time.

The correlation in our sample data set changed when sample 13 was removed, but how would our regression?

```
plot(x,y)

g1 <- lm(y~x)
g2 <- lm(y[-13]~x[-13])

points(x[13],y[13],pch=16,col="red")

abline(0,1,lty=3,lwd=3)            ##The 'true' line
abline(g1$coef)                    ##The full data set
abline(g2$coef,col="red",lty=2)  ##Removing the 'outlier''
```

The smaller data set gets us further from the true model–in this case. Overall, the fit for the majority of the data points is fairly unchanged. But is 13 an outlier?

It is simple to do a jackknife procedure and evaluate how goodness of fit or residuals or beta values or other statistics change. For example, look at how the standardized residual of each point changes when that point is left out:

```
set.seed(111)
x <- rnorm(20)
y <- x + rnorm(20)


resid <- rep(0,20)
for(i in 1:20)
```

```
{
  keep <- rep(T,20)
  keep[i] <- F
  xtmp <- x[keep]
  model <-  lm(y[keep]~xtmp)
  resid[i] <-(y[i]-predict(model,list(xtmp=x[i])))/sd(model$resid)
}

plot(resid/sd(resid))
```

But the predicted change is not the only thing you could look at. You could also look at beta weights:

```
model1 <- lm(y~x)
int <- rep(0,20)
beta <- rep(0,20)
for(i in 1:20)
{
  keep <- rep(T,20)
  keep[i] <- F
  xtmp <- x[keep]
  model <-  lm(y[keep]~xtmp)
  int[i] <- model1$coef[1]-model$coef[1]
  beta[i] < model1$coef[1]-model$coef[2]
}
```

Let's plot the different measures:

```
plot(int)
plot(beta)
plot(x,y)
 for(i in 1:20)abline(int[i],beta[i])

fits <- dfbetas(model1)
for(i in 1:20)abline(fits[i,],col="red")
```

Th `dffits` scales the residual much like studentized residuals, but also by estimates of sigma. The Cook's statistics does similar alchemy. A number of these are available, and many of them tell you similar things.

```

plot(hatvalues(model1),type="o",col="red",ylim=c(-1,2))
points(cooks.distance(model1),type="o",col="darkgreen")
points(dffits(model1),type="o",col="orange")
```

The built-in function influence.measures conducts a bunch of tests like this to identify possible outliers. The `dfb.1` and `dfb.x` correspond roughly to the change in intercept and slope we just did.

```

influence.measures(model1)

Influence measures of
lm(formula = y ~ x) :
```

Figure 14.12: Illustration using a Jackknife procedure on four statistics.

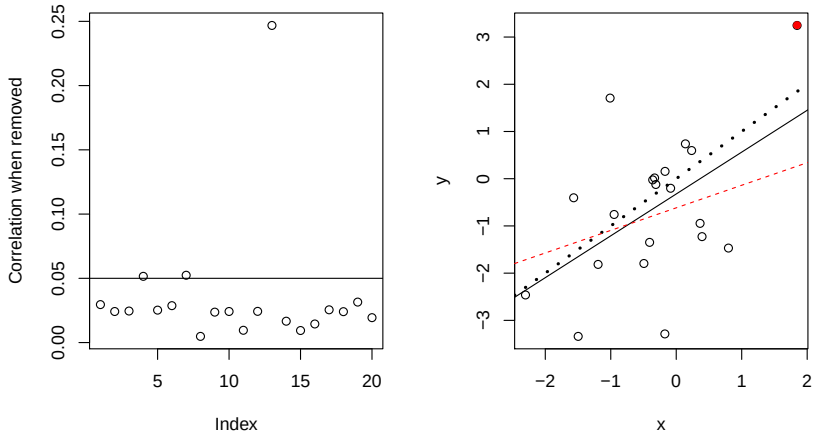
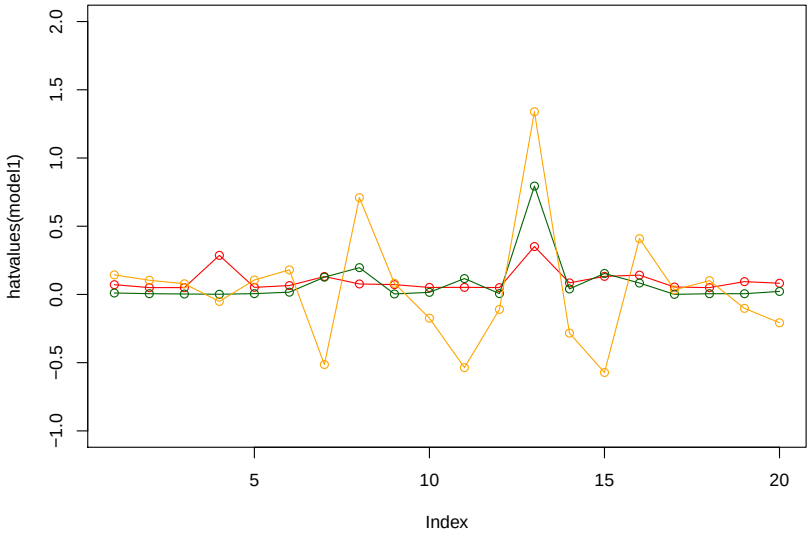


Figure 14.13: Comparison of three influence measures: hat values (leverage); cooks distance, and the dffits diagnostic fit value



.

```
6
   dfb.1_     dfb.x   dffit cov.r   cook.d    hat inf
8  1   0.14034  0.078740  0.1434 1.171 0.010712 0.0716
   2   0.09760  0.002664  0.1039 1.152 0.005648 0.0500
10 3   0.07410  0.003674  0.0783 1.164 0.003224 0.0501
   4  -0.00282  0.046526 -0.0512 1.570 0.001388 0.2865   *
12 5   0.10426  0.021167  0.1059 1.155 0.005863 0.0521
   6   0.17851  0.086896  0.1800 1.136 0.016693 0.0652
14 7  -0.14607  0.404044 -0.5133 1.062 0.126519 0.1315
   8   0.37835 -0.419291  0.7094 0.661 0.196549 0.0768   *
16 9   0.04675 -0.045205  0.0817 1.196 0.003519 0.0720
   10 -0.14991  0.026889 -0.1739 1.107 0.015495 0.0512
18 11 -0.52768 -0.105614 -0.5361 0.691 0.116326 0.0520
   12 -0.09918  0.006444 -0.1094 1.150 0.006249 0.0502
20 13  0.92519  1.239984  1.3389 1.211 0.794381 0.3512   *
   14 -0.26897 -0.181524 -0.2833 1.109 0.040432 0.0848
22 15 -0.49241 -0.451291 -0.5719 1.020 0.153775 0.1325
   16  0.10533 -0.329240  0.4093 1.163 0.083704 0.1416
24 17  0.03368  0.009681  0.0338 1.183 0.000604 0.0545
   18  0.09393 -0.000613  0.1012 1.154 0.005363 0.0500
26 19 -0.04359  0.069738 -0.1020 1.223 0.005473 0.0939
   20 -0.19821 -0.129709 -0.2074 1.155 0.022155 0.0821
```

14.5 Impact on inferential statistics

Many types of data you may measure as a social scientist will frequently or logically produce data whose error is not normally distributed, and/or distributions where the variance depends on the level. These include:

- Response time. It is typically highly skewed, and RT cannot be negative, which violates normality. Variance is typically proportional to the mean, and not independent. Often, a log transform will improve all of these problems
- Probability Correct. Probability is bounded between 0/1, and can usually be modeled as conforming to a binomial distribution. Variance of a binomial is $\sqrt{(p * (1 - p))}$, getting smaller near 0 and 1
- Distance-based precision or accuracy. Measures of how far off an estimate is will be bounded at the bottom by zero, will likely be skewed and have variance proportional to the mean.
- Multiple choice responses. Like P(corr), but with more contraints, and additionally a guessing parameter might be needed.cause it is bounded.

It is difficult to find any common measure in psychology that does not violate the normality assumption in one way or another.

```
1 dat <- read.csv("tmt.csv")
  dat$age <- as.factor(dat$age)
3 par(mfrow=c(1,2))
  plot(dat$time~(dat$age)+dat$type)
```

These completion times look highly skewed. What if we do an lm:

```
lm1 <- lm(time~age+type,data=dat)
qqnorm(lm1$resid)
```

As expected, some residuals look problematic.

```
par(mfrow=c(1,2))
plot(lm1)
hist(lm1$resid)
```

But maybe a simple transformation would help: If we take a log transform after subtracting the smallest number to make sure they are all positive and close to 0,

```
hist(log(lm1$resid-min(lm1$resid)),breaks=20)

par(mfrow=c(1,2))

plot(log(dat$time)~(dat$age))
points(dat$age,log(dat$time))
plot(log(dat$time)~(dat$type))
points(dat$type,log(dat$time),col="grey")
```

There are still 'outliers' on the high side, but it looks better'

```
lm2 <- lm(log(time)~age+type,data=dat)
qqnorm(lm2$resid)
#This is not perfect, but an improvement.

##This is a bit better; the fastest time was around 50
lm3 <- lm(log(time-50)~age+type,data=dat)
qqnorm(lm3$resid)
hist(lm3$resid)

summary(lm1)
summary(lm2)
summary(lm3)
```

These transformations both improved the R^2 and made the residuals more normal.

14.6 Downside of transformation to normalize variance

```
lm1.int <- lm(time~age*type,data=dat)
lm3.int <- lm(log(time-50)~age*type,data=dat)
anova(lm1.int)
anova(lm3.int)


agg1 <- tapply(dat$time,list(dat$age,dat$type),mean)
agg3 <- tapply(log(dat$time-50),list(dat$age,dat$type),mean)

matplot(t(agg1),type="o",ylab="Response time (s)")
matplot(t(agg3),type="o",ylab="log(Response time)",yaxt="n",ylim=c(2.5,5))
axis(2,log(c(65,75,100,150,200,250)-50), c(65,75,100,150,200,250),las=1)
```

When we take the transform to make the variance fit the model, the interaction disappears. After all, interactions are only defined additively. Notice that a test for inequality is fine after the transform.

```
library(car)
ncvTest(lm1.int)
ncvTest(lm3.int)
```

14.7 Outlier detection and removal

It is common practice to identify individual observations (and in a repeated measures experiment, individual people), assess how these conform to your assumptions, and remove or exclude that data. In psychology and human factors, some approaches to doing so include:

- Pre-identifying a performance criterion. Participants who cannot meet that performance criterion are excluded from further testing. Often, this is done via time or accuracy measures, and may be embedded in sophisticated testing software with time limits and accuracy requirements.
- Use attention-check and reverse-coded questions. This can help you identify people who are just responding blindly and whose other data should not be trusted.
- Post-hoc identification of performance ranges. I.e., compare the distribution of performance to a normal distribution by calculating z-scores. Participants outside of roughly 2.5 z-scores from the mean are outside of the performance range that would be expected if sampling from a normal distribution, and might justifiably be removed.
- If you have redundant measures (i.e., repeated trials from every combination of IV levels), removing individual observations that are outside a typical range. This is common in multi-trial tasks in which you record response times. RT distributions are skewed and typically have a long tail, and the trials in the tail can be influential. Many researchers in these areas will always go through an outlier removal process, and there are various schemes with different levels of sophistication. The `prepdat` or `trimr` libraries in R has some built-in routines for outlier removal from response time tasks.

14.8 Case Study: Strategies for Response Time outlier removal

Researchers who employ response time (RT) as a primary dependent variable to assess mental architecture and cognitive processes are well familiar with heuristics to select and remove individual trials from the data. This is typically possible when:

- Any individual trial is not consequential, so removing one or more trials does not produce systematic bias. Typically, this means dozens of trials per condition per person.
- Under normal operations, the participants should be able to complete the task in a fairly limited time range, so trials substantially longer may reasonably represent another process

- There is a need to obtain a good estimate for the mean of a condition that is not influenced by potential outliers

The basic strategies for doing this involve decisions about a few different criteria:

- Should errors be considered? Sometimes, an erroneous response is thought to probably involve incorrect processes, so removing them might be justified. On the other hand, removing these could also bias RT mean estimates.
- Is there a time faster than which it is unreasonable to consider responses? Normally something like 100 or 200 ms might indicate an anticipation response that could be discounted.
- is there a time slower than which it is unreasonable to consider responses? E.g., if a person can normally complete a response in 500 ms, maybe if it takes longer than 5 seconds to respond, the trial can be removed.
- Is there a number of standard deviations from the mean that indicate a distinct process? Typically, somewhere between 2 and 3 SDs of the mean would produce 1% or fewer cases in a normal distribution, and might be thought of not arising from the same process.

The `prepdat` and `trimr` libraries offer a few functions based on the work of Van Seltz & Jolicoeur (1994) [2]. These functions are essentially sophisticated aggregation programs that take response time data and automatically trim outliers and return the means of the trimmed distributions. The `trimr` library offers two simple functions for trimming and averaging (`absoluteRT` and `sdTrim`) that remove outliers outside of particular bounds, Both libraries offer three trimming procedures that use either non-recursive, recursive, or hybrid approaches. In these, the criteria for removing data depends on the number of observations. The recursive approach repeatedly applies the process until no more outliers are found, because a large outlier can inflate the SD criteria you use to identify an outlier.

prepdat provides the means with outlier removal for a condition. For the following data, we will consider the three age groups as individual participants. Here, we show the mean estimate for the raw data vs three different methods supported by prepdat. Prepdat must be applied to each condition on its own and later aggregated:

```
library(prepdat)
dat <- read.csv("tmt.csv")
dat$age <- as.factor(dat$age)


agg1 <- aggregate(dat$time,list(dat$age),mean)
agg2 <- aggregate(dat$time,list(dat$age),non_recursive_mc)
agg3 <- aggregate(dat$time,list(dat$age),modified_recursive_mc)
agg4 <- aggregate(dat$time,list(dat$age),hybrid_recursive_mc)

grouped <- rbind(agg1,agg2[,1:2],agg3[,1:2],agg4[,1:2])
grouped$Method <- factor(rep(c("Mean","Non-recursive","Modified Recursive","
    Hybrid"),each=4),levels=c("Mean","Non-recursive","Modified Recursive","
    Hybrid"))
colnames(grouped) <- c("Age","RT","Method")
round(tapply(grouped$RT, grouped[,c(1,3)],mean),0)

```

[2] Van Selst, M. & Jolicoeur, P. (1994). A solution to the effect of sample size on outlier elimination. Quarterly Journal of Experimental Psychology, 47 (A), 631-650.

```
17 Method
   Age Mean Non-recursive Modified Recursive Hybrid
19 1   188            174                 178    176
   2   101             96                  95     96
21 3   104            100                  98     99
   4   136            133                 136    135
```

the `tidyr` library likewise provides these methods, but lets you specify participant and condition variables.

```
2  library(trimr)
   ##trimr requires accuracy values.
4  dat$accuracy <- 1

6
   absoluteRT(data = dat, minRT = 150, maxRT = 500, rtVar="time",accVar="accuracy
       ",
8              pptVar="age",condVar="type",returnType = "mean")
   participant        A       B
10 1            1 221.055 260.191
   2            2 214.795 220.171
12 3            3 248.305 209.432
   4            4 182.976 195.619
14


16
   > sdTrim(data = dat, minRT = 150, sd=2.5, rtVar="time",accVar="accuracy",
18         pptVar="age",condVar="type",     returnType = "mean")
   participant        A       B
20 1            1 216.710 270.831
   2            2 214.795 220.171
22 3            3 248.305 188.450
   4            4 182.976 195.619
24

26 >  hybridRecursive(data = dat,rtVar="time",pptVar="age",condVar="type",minRT
       =100)
   participant        A       B
28 1            1 154.611 228.940
   2            2 122.537 127.284
30 3            3 123.428 128.873
   4            4 126.893 160.378
32
```

Other kinds of data have other procedures for outlier removal. For example, continuously sampled data (e.g., soundwave recordings) sometimes have routines for removing pops and clicks.

Chapter 15

Example: Houghton County Snowfall

This example was taken from the midterm exam given to the 2012 class.

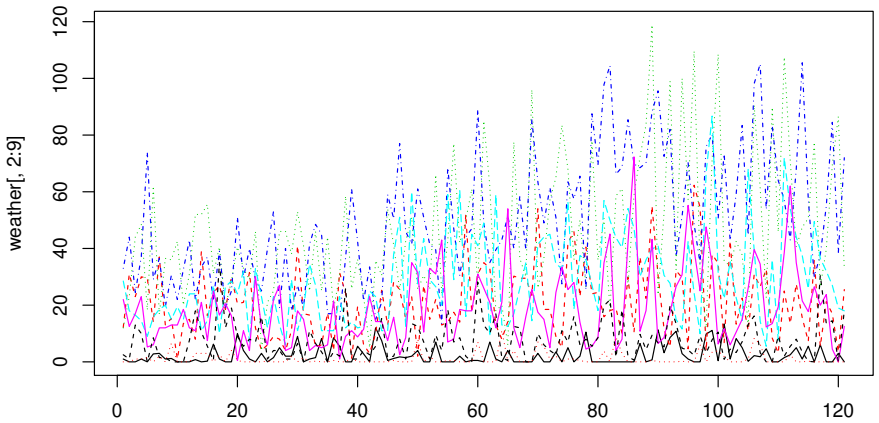
The available data set has snowfall each of 8 winter months (October through May) in inches, each year since 1890. It contains each year in a row, and each month in a column.

15.1 Graphing the major trends

After reading in the data, we can use `matplot` to display it, but it is sort of confusing:

```
weather <- read.csv("weather.csv")
matplot(weather[,2:9],type="l")
```

Figure 15.1: Snowfall in each month over the past 120 years in Houghton



I'd like to make a stacked line plot. To do this, I can apply the `cumsum` function to each row of the data frame:

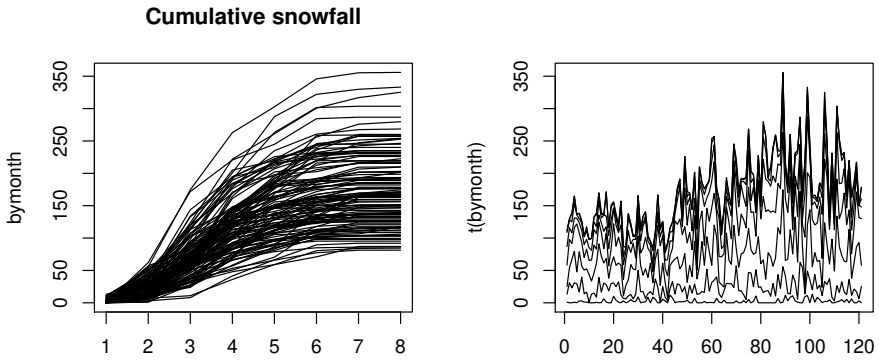
```
bymonth <- apply(weather[,2:9],1,cumsum)

> bymonth[,1:5]
     [,1]  [,2]  [,3]  [,4]   [,5]
OCT   1.0   0.0   0.0   1.0   0.00
NOV  13.2  31.0  23.0  31.0  29.50
DEC  25.2  54.0  67.0  79.5  47.80
JAN  58.2  98.0  91.5 116.5 121.80
FEB  86.7 116.0 108.0 131.0 130.80
MAR 108.7 128.5 125.0 154.0 136.05
APR 111.2 129.5 138.0 164.0 136.05
MAY 111.2 129.5 139.0 165.0 137.55
```

Now, each column represents a different year, with the numbers in each row representing the cumulative snowfall that year. I'll start by plotting both this matrix and its transpose:

```
par(mfrow=c(1,2))
matplot(bymonth,type="l",col="black",lty=1,main="Cumulative snowfall")
matplot(t(bymonth),type="l",col="black",lty=1)
```

Figure 15.2: Cumulative Snowfall in each month over the past 120 years in Houghton



These are pretty interesting. The left panel shows one trajectory for each year, and the right shows a contour of the snowfall over time. I'd like to improve the right one in a number of ways. First, I'd like to color each month according to a different color in some palette. Also, maybe I can fit a polynomial regression to approximate the trend over time.

To fill the plots, I need to use the polygon function. To do it right, I need to add the bottom corners of the polygon to the beginning and end. Then iterate through each column of the matrix, starting from May and moving back to October. I've made this into a function, so that it might be easier to adapt to other data sets.

```

x <-1891:2011
```

```
ymatrix <- t(bymonth)
ylab<-"Inches of snow per Winter"
xlab<-"Year"
main<-"Houghton County Annual Snowfall"

matplot(x,ymatrix,lty=1,type="l",xlab=xlab,ylab=ylab,main=main,col="black")

##Figure out some vertical lines.
xrange <- round(range(x),-1)
abline(v=seq(xrange[1],xrange[2],10),lty=2,col="grey")

## The following command will make a blue gradient,
## using the gplots library. Hardcoding here to avoid
## needing to load library
##cols <- rev(rich.colors((ncol(ymatrix)),"blues"))
cols <- c("#FFFFFFFF", "#CEEAFFFF", "#A1D0FFFF", "#79B1FFFF", "#548DFFFF",
          "#3463E1FF", "#1834A9FF", "#00005CFF")

for(i in ncol(ymatrix):1)
  {
      polygon(c(min(x),x,max(x)), c(0,ymatrix[,i],0),col=cols[i],lty=0)
  }

#Make one last unfilled polygon around the whole thing.
polygon(c(min(x),x,max(x)), c(0,ymatrix[,ncol(ymatrix)],0),lty=1)

##Let's fit a polynomial model to describe the trend
fit <- lm(ymatrix[,ncol(ymatrix)]~poly(x,degree=10))
points(x,fit$fit,lty=1,lwd=3,type="l")

## Add a legend.
legend(min(x),355,rev(c("Oct","Nov","Dec","Jan","Feb","Mar","Apr","May")),bty=
    "o",pt.cex=1.9,
          pch=15,col=rev(cols),y.intersp=.9,lty=0)
## Add a second legend that just draws boxes around the points.
legend(min(x),355,rev(c("","","","","","","","")),bty="n",pt.cex=1.9,
          pch=0,y.intersp=.9)
```

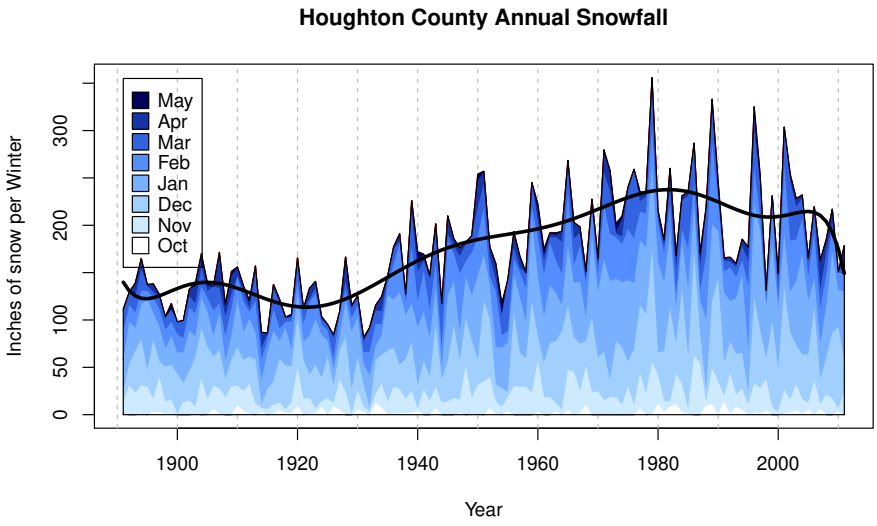
Also, we'd like to be able to look at this by month, to see the typical trends across the calendar year. For a plot like this, I like to show as much of the raw data as possible, and to provide summaries only as overlays, unless the raw data is too cluttered to make sense of. Also, anticipating one of the later questions, I'm going to pick out the month of highest snowfall in each year, and mark it with a colored circle. The result is shown in Figure 15.4

```
matplot(t(weather[,2:9]),type="o",pch=16,cex=.2,col="black",xaxt=  "n",lty=3,
        ylab="Monthly snowfall (in)",las=1)
axis(1,1:8,c("OCT","NOV","DEC","JAN","FEB","MAR","APR","MAY"))
for(i in seq(0,120,20))
abline(i,0,lty=2,lwd=.8,col="grey")

points(rowMeans(t(weather[,2:9])),type="l",col="blue",lwd=5)

##anticipating later question
weather$max <-   apply(weather[,2:9],1,which.max)
points(weather$max,
         weather[,2:9 [cbind(1:121,weather$max)],
         col="navy")
```

Figure 15.3: Cumulative Snowfall in each month over the past 120 years in Houghton



15.2 Climate Change?

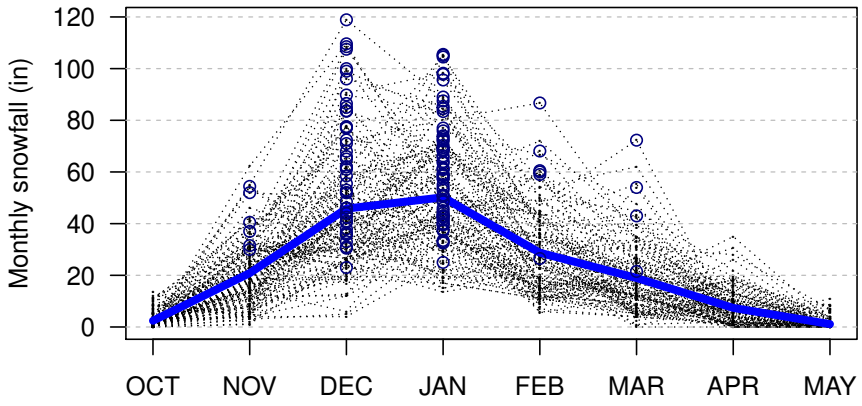
There appears to be a shift in snowfall patterns over the past century, with an increase happening around 1920. Let's two time eras, one for the 30 years before 1920, and one for the 91 years after 1920. I'd like to conduct statistical tests that tell me (at p=.05) whether snowfall increased in each month of the year, or just some of the months. Also, I'd like to show a plot of the means of the early and late era by month, and use some graphical means to indicate which values differ reliably.

To start, I'll make two filtering vectors to let me pick out the two regions. Then it is a matter of conducting t-tests for each month. The t test function spits out a lot of information, but we can access the important pieces using the $ symbol.

```
early <- rep(c(T,F),c(30,91))
late <- rep(c(F,T),c(30,91))
o<-t.test(weather$OCT[early],weather$OCT[late])
n<-t.test(weather$NOV[early],weather$NOV[late])
d<-t.test(weather$DEC[early],weather$DEC[late])
j<-t.test(weather$JAN[early],weather$JAN[late])
f<-t.test(weather$FEB[early],weather$FEB[late])
m<-t.test(weather$MAR[early],weather$MAR[late])
a<-t.test(weather$APR[early],weather$APR[late])
m2<-t.test(weather$APR[early],weather$APR[late])

round(rbind(
   c(o$stat,o$par,o$p.val),
   c(n$stat,n$par,n$p.val),
   c(d$stat,d$par,d$p.val),
   c(j$stat,j$par,j$p.val),
   c(f$stat,f$par,f$p.val),
   c(m$stat,m$par,m$p.val),
   c(a$stat,a$par,a$p.val),
   c(m2$stat,m2$par,m2$p.val)),2)
        t     df
[1,] -1.22  65.98 0.23
```

Figure 15.4: Cumulative Snowfall in each month over the past 120 years in Houghton, plotted by calendar month




```
[2,] -1.08  60.30 0.29
[3,] -3.79  92.05 0.00
[4,] -6.99  92.98 0.00
[5,] -5.73 109.16 0.00
[6,] -3.12  97.58 0.00
[7,] -0.36  45.22 0.72
[8,] -0.36  45.22 0.72
```

Notice that the months early and late in the season do not differ reliably between eras, but December through March do. Let's plot these means. A simple interaction.plot or matplot would work. To make it easier to control plot settings, I'll use a simple plot and build up to what I want. To indicate which ones differ reliably, I'll put a gold rectangle around the difference. The results are shown in Figure 15.5.

```
#To start with, use type "n" so nothing gets plotted.
plot(colMeans(weather[early,2:9]),type="n",ylim=c(0,80),
     ylab="Average monthly snowfall" ,xaxt="n",xlab="Month",
     las=1,main="Average snowfall by month",cex=2.5,pch=16,col="darkblue")
#points(colMeans(weather[late,2:9],na.rm=T),type="o",
     pch=15,cex=2.5,col="darkblue")
axis(1,1:8,c("OCT","NOV","DEC","JAN","FEB","MAR","APR","MAY"))
for(i in c(0:8*10))
abline(i,0,col="grey",lty=2)

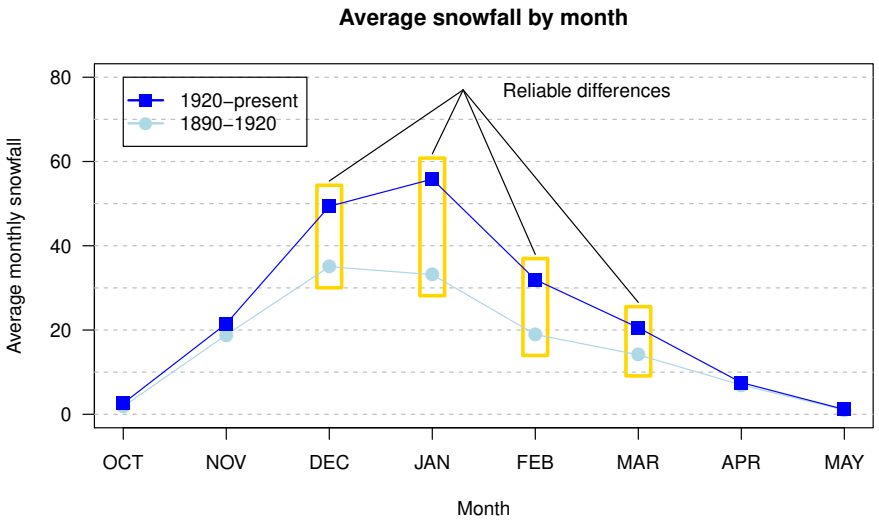
means <-rbind(colMeans(weather[early,2:9]),
              colMeans(weather[late,2:9]))
reliable <- c(F,F,T,T,T,T,F,F)
##Add the reliability boxes
rect((1:8)[reliable]-.12,
     means[1,reliable]-5,
```

```
     (1:8)[reliable]+.12,
     means[2,reliable]+5,
     border="gold",lwd=3)

text(5.5,77,"Reliable differences")
segments(4.3,77,3:6,means[2,reliable]+6) #add the arrow lines
#Now, plot the points so they are on top
points(colMeans(weather[early,2:9]),type="o",
         ylim=c(0,80),col="lightblue",pch=16,cex=1.6)
points(colMeans(weather[late,2:9],na.rm=T),
        type="o",col="blue",pch=15,cex=1.6,)

legend(1,80,c("1920-present","1890-1920"),
       lwd=2,lty=1,col=c("blue","lightblue"),
        pt.cex=1.5,pch=c(15,16))
```

Figure 15.5: Comparison of snowfall across months for two eras



15.3 Did the snowiest month change?

It appears that in the 'old' era, December tended to get more snow than January, but this has switched. Conduct an appropriate statistical test determining whether each of these are reliable. Are these differences reliable. That is, for each era, was January larger than December? Describe the results as you would in a research report.

First, let's test this for all years. A `t.test` is probably the easiest test to use, but because we are comparing months of the same year, and because we know the overall weather trend has changed over time, we want to use a paired t test.

```
##This is the 'wrong' test
t.test(weather$DEC,weather$JAN)
##This is a better one to use:
t.test(weather$DEC,weather$JAN,paired=T)
```

```

> t.test(weather$DEC[early],weather$JAN[early],paired=T)

  Paired t-test

data:  weather$DEC[early] and weather$JAN[early]
t = 0.4958, df = 29, p-value = 0.6237
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -5.884926  9.651593
sample estimates:
mean of the differences
               1.883333

> t.test(weather$DEC[late],weather$JAN[late],paired=T)

  Paired t-test

data:  weather$DEC[late] and weather$JAN[late]
t = -2.0184, df = 90, p-value = 0.04652
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -12.7880271  -0.1014235
sample estimates:
mean of the differences
              -6.444725
```

We can see that there is no difference in the early era, but there is a difference in the late era ($p < .05$). We must be careful not to infer that there was a change though. In this case, since the difference went from +1.88 to -6.44 inches, we might be fairly confident; after all, if this new difference is less than zero, it must also be less than 1.88. But, we don't know whether the old value was positive, and it could easily have been negative. So, we should do a specific test for this, which we could do using a regression or anova model,

Because these data are sort of equivalent to a cross-tabulation table, we might use a chi-squared test. For arbitrary DVs, this would not be appropriate, but we are sort of counting units of snow that fell during different times, and so it will work in a pinch.

To do this, let's compute a table with the total snowfall in December and January by era:

```
tab <- aggregate(data.frame(DEC=weather$DEC,JAN=weather$JAN),
                 list(late),sum)
> tab
 Group.1     DEC     JAN
1   FALSE 1051.00  994.50
2    TRUE 4492.01 5078.48

>chisq.test(tab)

  Pearson's Chi-squared test

data:  tab
X-squared = 13.5619, df = 2, p-value = 0.001135

Warning message:
In chisq.test(tab) : Chi-squared approximation may be incorrect
```

Table 15.1: Number of times each month had the highest snowfall of the season

Month	1920-2011	1890-1920
OCT	0	0
NOV	4	2
DEC	28	17
JAN	50	10
FEB	5	1
MAR	4	0
APR	0	0
MAY	0	0

This test shows that distribution of the snowfall did indeed change by era.

15.4 Highest snowfall month

Since there seems to be this switch between December and January, maybe the time of winter in which the peak snowfall arrives has also switched. I'd like to calculate which month received the highest snowfall in each winter, and tabulate that over the two eras. The following commands will work:

```
weather$max <-   apply(weather[,2:9],1,which.max)
highmonth <- table(weather$max,early)
```

In this case, there were no 1 values, indicating October never was the highest month of snow, so the table `highmonth` has now 1 row. The data look like It looks like the distribution of peak snowfall has shifted later. We can test this with a chi squared test.

```
chisq.test(highmonth)

  Pearson's Chi-squared test

data:  highmonth
X-squared = 7.9598, df = 4, p-value = 0.09306
```

15.5 Prediction: March Snowfall

Can we predict snowfall in March based on current year's snowfall? To do this, let's create a regression model that attempts to predict March snowfall based on a combination of linear predictors of OCT,NOV,DEC,JAN, and FEB (the months that precede March). Because of these differences between early and late eras, we know we probably shouldn't use the oldest data, so let's make one model for the entire data set, and a second model for just the modern era.

The full model is easy to specify:

```
g1 <- lm(weather$MAR~weather$OCT+weather$NOV +
         weather$DEC+weather$JAN+weather$FEB)
> summary(g1)

```

```
Call:
lm(formula = weather$MAR ~ weather$OCT + weather$NOV + weather$DEC +
    weather$JAN + weather$FEB)

Residuals:
    Min      1Q  Median      3Q     Max
-22.891  -7.676  -1.931   7.393  47.022

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  7.05681    3.80608   1.854 0.066289 .
weather$OCT -0.08314    0.35321  -0.235 0.814321
weather$NOV  0.01502    0.09254   0.162 0.871305
weather$DEC  0.03976    0.05110   0.778 0.438098
weather$JAN  0.04914    0.05685   0.864 0.389151
weather$FEB  0.26096    0.07672   3.401 0.000923 ***
---
Signif. codes:  0 ?***? 0.001 ?**? 0.01 ?*? 0.05 ?.? 0.1 ? ? 1

Residual standard error: 12.81 on 115 degrees of freedom
Multiple R-squared: 0.1419, Adjusted R-squared: 0.1045
F-statistic: 3.802 on 5 and 115 DF,  p-value: 0.003183
```

The overall model is interesting, but not very encouraging. The reliability of the parameters seems to improve as we get closer to March, and February is the only one that is reliable. It looks like on average we get 7 inches + one inch for every four inches that falls in February, plus some additional amount that the other months are accounting for. It would be good to get rid of some of these other predictors to give us more stable estimates. Let's do that for a model that uses only the late-era data. Here is a set of nested regression models, and each one removes the least reliable predictor of the larger model. We can compare them all with an ANOVA table.

```
g3a <- lm(MAR~OCT+NOV+DEC+JAN+FEB,data=weather[late,])
g3b <- lm(MAR~OCT+DEC+JAN+FEB,data=weather[late,])
g3c <- lm(MAR~OCT+JAN+FEB,data=weather[late,])
g3d <- lm(MAR~JAN+FEB,data=weather[late,])
g3e <- lm(MAR~FEB,data=weather[late,])

> anova(g3a,g3b,g3c,g3d,g3e)
Analysis of Variance Table

Model 1: MAR ~ OCT + NOV + DEC + JAN + FEB
Model 2: MAR ~ OCT + DEC + JAN + FEB
Model 3: MAR ~ OCT + JAN + FEB
Model 4: MAR ~ JAN + FEB
Model 5: MAR ~ FEB
  Res.Df   RSS Df Sum of Sq      F Pr(>F)
1     85 17222
2     86 17222 -1    -0.146 0.0007 0.9787
3     87 17286 -1   -64.283 0.3173 0.5747
4     88 17301 -1   -14.960 0.0738 0.7865
5     89 17496 -1  -194.655 0.9608 0.3298

> anova(g3a,g3e)
> anova(g3a,g3e)
Analysis of Variance Table

Model 1: MAR ~ OCT + NOV + DEC + JAN + FEB
Model 2: MAR ~ FEB
```

```
  Res.Df   RSS Df Sum of Sq      F Pr(>F)
1     85 17222
2     89 17496 -4   -274.04 0.3381 0.8515
```

Each model provides a fit no worse than the model one parameter bigger. as a check, I also compared the smallest model with the biggest, which showed that there is no reliable difference between these two models. The best model, g3e, looks like:

```
> summary(g3e)

Call:
lm(formula = MAR ~ FEB, data = weather[late, ])

Residuals:
    Min      1Q  Median      3Q     Max
-23.145  -9.099  -2.084   7.853  47.698

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 12.00139    3.12103   3.845 0.000226 ***
FEB          0.26752    0.08614   3.106 0.002547 **
---
Signif. codes:  0 ?***? 0.001 ?**? 0.01 ?*? 0.05 ?.? 0.1 ? ? 1

Residual standard error: 14.02 on 89 degrees of freedom
Multiple R-squared: 0.09777,  Adjusted R-squared: 0.08763
F-statistic: 9.645 on 1 and 89 DF,  p-value: 0.002547
```

So, our best estimate is $12 + .26$ times February's snowfall, which ended up being 25.5 inches. Our best guess should thus be 18.8 inches. But we can't be very confident–the R^2 value is .087, which means we are predicting less than 10% of the variance with the model. Can we place confidence regions around our prediction? The predict function will report two kinds of confidence regions for us: prediction and confidence:

```
> predict.lm(g3e, data.frame(FEB=25.5),interval="prediction")
       fit       lwr      upr
1 18.82322 -9.210079 46.85652
> predict.lm(g3e, data.frame(FEB=25.5),interval="confidence")
       fit      lwr      upr
1 18.82322 15.70038 21.94606
```

What do these tell us? These disagree substantially, and neither seems very appropriate. This is an area of research where there is no consensus. We know from Figure **??** that the range of snowfall in March is between around 0 and 60, and if we look at the 95% range of March snowfall in the recent era, we get:

```
 quantile(weather$MAR[late],c(.025,.5,.975))
 2.5%   50% 97.5%
 2.85 17.70 54.90

```

This maps more closely on the prediction interval, except the prediction interval goes awry because its lower bound is negative, because it is assuming the error should be symmetric and

not bounded. On the upper bound, it is a bit larger than our model, and so the prediction interval brings the upper bound in a little bit. The confidence interval is probably much too small. How does it relate to our residuals for the data set?

```
> quantile(g3e$resid,c(.05,.95))
  2.5%     97.5%
-19.62415  34.62150
```

This shows that within the data set we are predicting, 95% of the time our best prediction was no more than 19.6 inches too high and 34 inches too low. The prediction interval seems about right.

15.6 Predictions based on el nino and sunspots records

The only thing we can do to improve the model is to add more predictors. Monthly records of the the South Pacific El Nino/La Nina oscillation activity go back to the 1880s, and I've included them in the weather.csv data set. Also, monthly records of sunspot counts go back hundreds of years. Maybe knowing about these predictors will be helpful?

The sunspots data are separated in another file which has each month on a separae line. The following transforms them into a matrix like the rest of our data.

```
sunspots <- read.table("sunspots.csv",sep=",",header=T)

##Our weather data set goes from OCT 1890 to present
start<-which(sunspots$Year.Month==189010)
sunsub <- sunspots[start:3158,]
sunmat <- matrix(sunsub$Sunspots[1:(12*121)],byrow=T,ncol=12)
colnames(sunmat) <- colnames(weather)[12:23]
rownames(sunmat) <- rownames(weather)
```

Now, we can just add the sunspot data together with the master data.

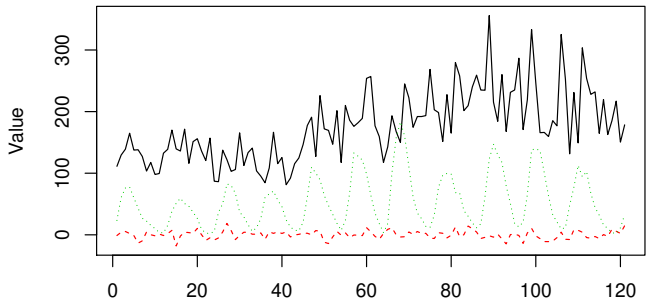
```
masterdat <- data.frame(weather,sunmat)
>masterdat[1:5,]
    Year OCT  NOV  DEC  JAN  FEB   MAR  APR MAY  TOTAL    Elnino Oct  Nov  Dec
      Jan
1 1890-91   1 12.2 12.0 33.0 28.5 22.00  2.5 0.0 111.20 1890-1891 3.6  2.6
    0.6 15.6
2 1891-92   0 31.0 23.0 44.0 18.0 12.50  1.0 0.0 129.50 1891-1892 0.6 -4.7
    -4.5  2.7
3 1892-93   0 23.0 44.0 24.5 16.5 17.00 13.0 1.0 139.00 1892-1893 8.5 -0.7
    3.7 11.3
4 1893-94   1 30.0 48.5 37.0 14.5 23.00 10.0 1.0 165.00 1893-1894 7.9  2.6
    1.6 17.5
5 1894-95   0 29.5 18.3 74.0  9.0  5.25  0.0 1.5 137.55 1894-1895 1.8  7.2
    0.1  5.6
    Feb  Mar  Apr  May  Jun  Jul  Aug   Sep max Oct.1 Nov.1 Dec.1 Jan.1 Feb.1
    Mar.1
1  -3.6 -9.5  4.5 -0.3 -1.5 -6.3 -8.9 -10.6   4  11.2   9.6   7.8  13.5  22.2
     10.4
2 -10.2 11.1  6.9 10.0 19.6  7.4  5.9   6.3   4  51.5  41.9  32.5  69.1  75.6
     49.9
3   7.7 -1.4  1.2 -3.5 10.7 14.0  7.8   5.7   3  70.5  65.4  78.6  75.0  73.0
     65.7
4  10.0  5.6 -3.0 -5.1 -1.5 -2.3 -5.7  -1.6   3  80.0  75.1  93.8  83.2  84.6
     52.3
```

```
14 5   3.0 -0.3 -7.1 -8.2 -4.7 -0.4 -6.3  -4.0   4  75.5  56.6  60.0  63.3  67.2
       61.0
     Apr.1 May.1 Jun.1 Jul.1 Aug.1 Sep.1
16 1  20.5  41.1  48.3  58.8  33.0  53.8
   2  69.6  79.6  76.3  76.5 101.4  62.8
18 3  88.1  84.7  89.9  88.6 129.2  77.9
   4  81.6 101.2  98.9 106.0  70.3  65.9
20 5  76.9  67.5  71.5  47.8  68.9  57.7
```

We now have three sets of monthly predictions–the column headers could be improved, but they will work in a pinch. Let's look at the annual totals of each and see if there is a relationship?

```
snowfall <- rowSums(masterdat[,2:9])
2 elnino <- rowMeans(masterdat[,12:23])
sunspots <- rowSums(masterdat[,24:35])
4 > cor(cbind(snowfall,elnino,sunspots))
            snowfall     elnino   sunspots
6 snowfall 1.0000000 0.11226902 0.29689342
  elnino   0.1122690 1.00000000 0.01364441
8 sunspots 0.2968934 0.01364441 1.00000000
```

Figure 15.6: Comparison of snowfall, sunspots, and el nino activity across years.



Actually, snowfall seems modestly related to both, and the correlation with sunspot activity during a given year is big enough to get excited about. Plus, sunspot activity is not related to el nino in this data set, so the predictors are close to orthogonal.

I'd like to be able to use the entire data set, rather than just the late era. Assuming the shift in snowfall does not stem from sunspots or el nino, but some other aspect (overall climate change, measurement change, or changes in record-keeping). So what if we fit a polynomial to the overall trend over the years to factor out that effect? Let's add a predictor year that is just the integers 1 to 121, and use powers of this them in a model. To get a feel for which power to use, let's fit a model of March snowfall using just a polynomial regression on years:

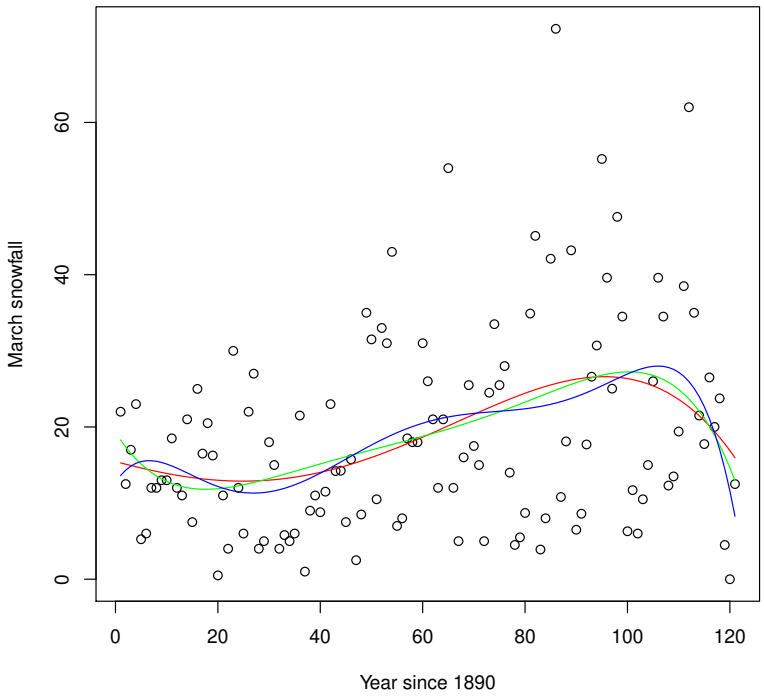
```
masterdat$year <- 1:121

poly4 <- lm(MAR~year + I(year^2) + I(year^3) + I(year^4),data=masterdat)
poly5 <- lm(MAR~year + I(year^2) + I(year^3) + I(year^4)+I(year^5),data=
    masterdat)
poly6 <- lm(MAR~year + I(year^2) + I(year^3) + I(year^4)+I(year^5)+I(year^6),
    data=masterdat)
##You can choose orthognoal polynomials like this:
poly6 <- lm(MAR~poly(year,6),data=masterdat)
```

We can overplot each of these polynomials. There looks to be little benefit of going higher than a 4th order polynomial fit.

```
plot(masterdat$MAR)
points(poly4$fit,type="l",col="red")
points(poly5$fit,type="l",col="green")
points(poly6$fit,type="l",col="blue")
```

Figure 15.7: Polynomial regression to fit the 120-year snowfall trend.



Now, let's do a larger model where we all the predictors and the polynomial regression together.

```
g1 <- lm(MAR~OCT+NOV+DEC+JAN+FEB+Oct+Nov+Dec+Jan+Feb+ Oct.1+Nov.1+Dec.1+Jan.1+
    Feb.1+
  year + I(year^2) + I(year^3) + I(year^4),data=masterdat)
>summary(g1)

Call:
lm(formula = MAR ~ OCT + NOV + DEC + JAN + FEB + Oct + Nov +
    Dec + Jan + Feb + Oct.1 + Nov.1 + Dec.1 + Jan.1 + Feb.1 +
    year + I(year^2) + I(year^3) + I(year^4), data = masterdat)

Residuals:
    Min      1Q  Median      3Q     Max
-21.752  -7.241  -0.491   6.142  36.670

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.782e+01  7.320e+00   2.434   0.0167 *
OCT         -9.392e-02  3.671e-01  -0.256   0.7986
NOV         -7.738e-02  9.791e-02  -0.790   0.4312
DEC          3.977e-02  5.489e-02   0.725   0.4704
JAN         -6.662e-02  6.975e-02  -0.955   0.3418
FEB          2.084e-01  8.049e-02   2.590   0.0110 *
Oct          2.333e-01  1.812e-01   1.288   0.2008
Nov         -2.039e-01  1.689e-01  -1.207   0.2301
Dec         -8.518e-02  1.976e-01  -0.431   0.6673
Jan         -1.556e-01  1.605e-01  -0.969   0.3348
Feb          3.682e-01  1.634e-01   2.253   0.0264 *
Oct.1        4.733e-03  7.946e-02   0.060   0.9526
Nov.1        6.001e-02  9.077e-02   0.661   0.5100
Dec.1       -1.293e-01  8.525e-02  -1.516   0.1326
Jan.1        1.275e-01  8.670e-02   1.470   0.1446
Feb.1       -1.294e-01  6.153e-02  -2.104   0.0379 *
year        -1.116e-01  7.048e-01  -0.158   0.8745
I(year^2)   -5.304e-03  2.311e-02  -0.230   0.8189
I(year^3)    2.065e-04  2.826e-04   0.731   0.4666
I(year^4)   -1.305e-06  1.150e-06  -1.135   0.2591
---
Signif. codes:  0 ?***? 0.001 ?**? 0.01 ?*? 0.05 ?.? 0.1 ? ? 1

Residual standard error: 12.01 on 101 degrees of freedom
Multiple R-squared: 0.337,  Adjusted R-squared: 0.2122
F-statistic: 2.701 on 19 and 101 DF,  p-value: 0.0007264
```

The good news is our R-squared value has risen to .337, which is somewhat respectable. The bad news is we have 19 predictors, and only a handful are reliable. It will be a nightmare trying to pick the best one.

We can use a stepwise regression method implemented in the R `step` function. If we give it a model, it will automatically remove predictors until some criterion is met.

```
gsmall <- step(g1,direction="both")

> gsmall

Call:
lm(formula = MAR ~ FEB + Feb + Feb.1 + I(year^2) + I(year^3) +
     I(year^4), data = masterdat)

Coefficients:
(Intercept)          FEB          Feb        Feb.1
```

```
  1.414e+01    1.937e-01    2.178e-01   -7.861e-02
 I(year^2)    I(year^3)    I(year^4)
-7.430e-03    2.150e-04   -1.291e-06
```

OK, now model shows three reliable predictors along with the polynomials. By default, step uses the Akaike Information Criterion (AIC) as a means to help select the smallest most descriptive model. Uses goodness of fit, but penalizes models that have more predictors. Recent research has tended to prefer a related criterion, called the Schwarz or Bayesian Information Criterion (BIC). This tends to be a bit more conservative and produce smaller models. You can use BIC by setting the `k` argument to be equal to log(N).

Also, you can control how the model does the search through the space. In actuality there are more than half a million ($2^19 = 524,288$) possible sub-models. It would be possible to check each of these models and pick the best one. My computer can run 1000 of them in about 7 seconds, and so doing the complete search should take about an hour. By default, the `step` function uses a downward-only search, removing one predictor on each step until remove another makes the AIC/BIC worse. We can set it to be a bit more robust by testing at each step whether adding or subtracting predictors would be better. This is the "both" argument, and allow us to later add back predictors that we earlier removed. In this case, it does not matter, and we get the same prediction regardless. However, the BIC eliminates one additional predictor: February sunspots.

```
gsmall.bic <- step(g1,direction="both",k=log(121))

> summary(gsmall)
Call:
lm(formula = MAR ~ FEB + Feb + Feb.1 + I(year^2) +
    I(year^3) + I(year^4), data = masterdat)

Residuals:
    Min      1Q  Median      3Q     Max
-26.164  -7.535  -0.592   6.389  38.428

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.414e+01  3.115e+00   4.541  1.4e-05 ***
FEB          1.937e-01  7.428e-02   2.608  0.01033 *
Feb          2.178e-01  1.059e-01   2.057  0.04199 *
Feb.1       -7.861e-02  2.432e-02  -3.232  0.00161 **
I(year^2)   -7.430e-03  5.184e-03  -1.433  0.15450
I(year^3)    2.150e-04  1.027e-04   2.093  0.03853 *
I(year^4)   -1.290e-06  5.270e-07  -2.449  0.01585 *
---
Signif. codes:  0 ?***? 0.001 ?**? 0.01 ?*? 0.05 ?.? 0.1 ? ? 1

Residual standard error: 11.75 on 114 degrees of freedom
Multiple R-squared: 0.2834,  Adjusted R-squared: 0.2457
F-statistic: 7.514 on 6 and 114 DF,  p-value: 8.442e-07

> summary(gsmall.bic)

Call:
lm(formula = MAR ~ FEB + Feb.1 + I(year^3) + I(year^4),
    data = masterdat)

Residuals:
    Min      1Q  Median      3Q     Max
-25.347  -8.011  -1.217   7.318  39.745
```

```
37
   Coefficients:
39                Estimate Std. Error t value Pr(>|t|)
   (Intercept)  1.130e+01  2.473e+00   4.568 1.23e-05 ***
41 FEB          2.094e-01  7.417e-02   2.823  0.00561 **
   Feb.1       -7.896e-02  2.469e-02  -3.198  0.00178 **
43 I(year^3)    6.562e-05  1.957e-05   3.354  0.00108 **
   I(year^4)   -5.342e-07  1.692e-07  -3.158  0.00203 **
45 ---
   Signif. codes:  0 ?***? 0.001 ?**? 0.01 ?*? 0.05 ?.? 0.1 ? ? 1
47
   Residual standard error: 11.94 on 116 degrees of freedom
49 Multiple R-squared: 0.2476, Adjusted R-squared: 0.2216
   F-statistic: 9.542 on 4 and 116 DF,  p-value: 1.05e-06
```

In this case, I'd probably prefer the AIC model, for one reason: the predictors it used were all February activity. There are reasons to suspect that if sunspot and el nino activity was related to snowfall, it would most likely be the activity that happened in February. But the BIC model gets rid of one of my polynomial predictors, which are likely to cause me problems when I extrapolate anyway.

So, let's make a new prediction based on these models.

```
   ##Lets predict this year!
2  year2012 <- data.frame(FEB=25.5,Feb=2.5,Feb.1=33.1,year=121)
   > predict(gsmall,year2012)
4         1
   12.42054
6
   > predict(gsmall.bic,year2012)
8       1
   15.7602
```

Both of these predictions are lower than the snowfall-only model. The reason why they are lower is probably partly because of the dropoff in March snowfall in recent years, which is causing the polynomial to slope down at that point. Only time will tell how long this trend will bear out.

Chapter 16

Categorical Predictors in lm, the One-Way ANOVA, and post-hoc tests

16.1 Categorical Predictors and their Underlying Contrasts

A good reference for understanding contrasts is Davis, 2010[1].

We saw several times already how you can use a factor as a predictor in a regression model. When you look at the `summary()` of the model, it codes the first level of the factor as 0, and codes the others with respect to that. As a reminder, here is a simulated data set with hourly pay recorded from a survey for of 15 professionals, with three job categories, and we can use specialty to predict salary in an `lm` model.

```
specialty <- as.factor(c("Advertising","Advertising","Advertising",
                         "Advertising","Advertising",
"Business","Business","Business","Business","Business",
"Certification","Certification","Certification",
"Certification","Certification"))

salary <- c(30,35,32,40,34,70,56,45,65,28,19,23,28,18,32)
> aggregate(salary,list(specialty),mean)
Group.1    x
1   Advertising 34.2
2      Business 52.8
3 Certification 24.0


lm(salary~specialty)

Call:
lm(formula = salary ~ specialty)

Coefficients:
(Intercept)       specialtyBusiness  specialtyCertification
34.2                      18.6                     -10.2
```

[1] Davis, M. (2010). Contrast coding in multiple regression: Strengths, Weaknesses, and Utility of Popular Coding Structures. Journal of Data Science, 8, 61-73. http://www.jds-online.com/files/JDS-563.pdf

As we've seen before, the first level of the predictor doesn't appear in the table. This is because in order put the categorical predictor into the regression, it must be coded according to multiple hidden binary predictor variables called contrasts. Contrasts are sets of vectors that uniquely identify categories in a way that makes sense. You could do this by hand, use the default contrasts, or specify contrasts that you like. Contrasts are linked to the factor, not the regression itself.

To see the contrasts associated with a factor, you can use the `contrasts` function:

```
contrasts(specialty)
                    Business Certification
 Advertising           0             0
 Business              1             0
 Certification         0             1
```

Notice that the three-level categorical variable has been recoded into two variables (the columns). These are labeled according to the variable they pick out, and the first level (Advertising) is 0 on these two variables. Contrasts are just a we to code categorical values into a set of (sometimes orthogonal, but otherwise independent) numerical predictors. If you have two levels of a category, you can code it with a single numerical variable, but if you have three levels, you need additional variables for each level, which accounts for $n-1$ degrees of freedom.

The default contrasts are called "treatment" or "dummy" coding. We can produce this type of contrast using the `contr.treatment` function.

```
> contr.treatment(3)
  2 3
1 0 0
2 1 0
3 0 1

> contr.treatment(levels(specialty))
        Business Certification
Advertising           0             0
Business              1             0
Certification         0             1
```

If we give contr.treatment a number, it will create treatment contrasts for a variable with that many levels. If we give it a list of levels, it will create them and label them by those levels, just like the default factor function will do. One thing should be clear–we need (and use) two binary-coded variables to represent three levels. This may seem a little strange, but if we had just two levels, we could easily represent them as a single binary variable. Furthermore, we need to do this because we have typically committed to estimating an intercept, which pins down our contrasts to some level, so we have just $N-1$ degrees of freedom left.

Consider the following, if we force a 0 intercept:

```
lm(salary~0+specialty)

Call:
lm(formula = salary ~ 0 + specialty)
```

```
Coefficients:
  specialtyAdvertising       specialtyBusiness  specialtyCertification
                  34.2                    52.8                    24.0
```

Now, the model will estimate each level with its actual mean. With a single predictor, the contrast coding used will always produce the same estimates if you force a 0-intercept.

Notice that the two columns of the contrast are orthogonal–the sum of the products is 0. This won't always be true of all contrasts we consider. There are many possible contrasts you could choose; your choice may help you make default tests between relevant models. Thus, you should both recognize what the default contrasts are so you can interpret models with categorical predictors, and recognize that they can be changed, so that you can create new models that are interpretable in the way you are interested in.

One contrast that is handy is the opposite to the default in R, but is common in the SAS language, which can be produced via `contr.SAS()`. This contrast gives the last factor the zero-level, and the first N-1 factors their own unique dimension. By setting the contrasts of specialty to something different, we will get a different output to the model.

```
contr.SAS(levels(specialty))
               Advertising Business
Advertising              1        0
Business                 0        1
Certification            0        0


contrasts(specialty)<- contr.SAS(levels(specialty))

lm(salary~specialty)
Coefficients:
(Intercept)  specialtyAdvertising     specialtyBusiness
24.0                  10.2                  28.8
```

Notice that the intercept now corresponds to the mean of Certification, with the others coded with respect to that. Like before, this is not a true intercept term–it is a baseline that depends on how you have coded the internal representation of a factor. You simply need to determine which factor level is not reported to understand which value the intercept corresponds to.

16.1.1 Helmert coding

Another coding that is sometimes used is called 'Helmert' coding. In published research, people use the term Helmert coding to refer to a number of similar coding schemes, including those called reverse helmert codes, and so you may need to be careful about choosing and referring to the one you are using, especially depending on the library or analysis software you are using. R provides one such coding scheme in `contr.helmert()`, and the `car` library has a similar`contr.Helmert()`, which produces identical contrasts but labels contrasts to be easier to read..

```
contr.helmert(levels(specialty))
               [,1] [,2]
 Advertising     -1   -1
```

```
 Business          1   -1
 Certification     0    2

>library(car)
> contr.Helmert(levels(specialty))
[H.1] [H.2]
Advertising      -1    -1
Business          1    -1
Certification     0     2
```

Again, the three levels are coded in two dimensions. The first dimensions ignores the third level, and subtracts the first from the second. Consequently, this first level of the coding determines whether the first level of the factor differs from the second. The second dimension compares the first two levels to two times the third. This essentially determines whether the third differs from the average of the first two. As you add more levels, this pattern continues, each time asking whether the next level differs from the average of the previous levels. This might be useful in identifying where or when an effect starts, or (if reversed) when an effect stops changing. It is a way of sneaking an ordinal representation into a categorical coding, and so you need to arrange your levels so that they are in the order you care about.

Notice that the dot product of each column with any other column is 0, so they are orthogonal. Furthermore, they are each uncorrelated with one another, which means is great for estimating parameters (because they will not depend on one another). Finally, each column sums to 0, so each contrast corresponds to a test against the null hypothesis that the contrast equals 0–which would happen if all values are the same.

```
 > contr.helmert(5)
     [,1] [,2] [,3] [,4]
1   -1   -1   -1   -1
2    1   -1   -1   -1
3    0    2   -1   -1
4    0    0    3   -1
5    0    0    0    4


> cor(contr.helmert(5))
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1

```

For helmert (and some other codings), the order of the factor makes a difference. By default, factors are coded in alphabetical order, which may not be what you want.

```

values <- c(1,1.1,1,1.3,.9,1.2, 4,5,6,8,10,9)
months <- factor(c("Jan","Feb","Mar","Apr","May","Jun",
                   "July","Aug","Sep","Oct","Nov","Dec"))
months

 [1] Jan  Feb  Mar  Apr  May  Jun  July Aug  Sep  Oct  Nov  Dec
Levels: Apr Aug Dec Feb Jan July Jun Mar May Nov Oct Sep

```

```
contrasts(months) <- contr.helmert(levels(months))
lm(values~months)

Call:
lm(formula = values ~ months)

Coefficients:
(Intercept)  months1      months2      months3      months4      months5
        4.04167      1.85000      1.95000     -1.00000     -0.62000
    0.08667

months6      months7   months8      months9     months10     months11
-0.22778      0.72778      0.41364      0.17803   -0.33810     -0.27857
```

At first, this all looks like it might be OK. We have an intercept, which is 4.04. Months1 codes the difference between month 1 and 2, but looking at the original values, we know that there was just a .1 increase between January and February. What happened? Let's look at the contrast coding:

```
> contrasts(months)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
Apr    -1   -1   -1   -1   -1   -1   -1   -1   -1    -1    -1
Aug     1   -1   -1   -1   -1   -1   -1   -1   -1    -1    -1
Dec     0    2   -1   -1   -1   -1   -1   -1   -1    -1    -1
Feb     0    0    3   -1   -1   -1   -1   -1   -1    -1    -1
Jan     0    0    0    4   -1   -1   -1   -1   -1    -1    -1
July    0    0    0    0    5   -1   -1   -1   -1    -1    -1
Jun     0    0    0    0    0    6   -1   -1   -1    -1    -1
Mar     0    0    0    0    0    0    7   -1   -1    -1    -1
May     0    0    0    0    0    0    0    8   -1    -1    -1
Nov     0    0    0    0    0    0    0    0    9    -1    -1
Oct     0    0    0    0    0    0    0    0    0    10    -1
Sep     0    0    0    0    0    0    0    0    0     0    11
```

We can see that the factor is in alphabetical order, so this compares April to August–something we probably don't care about. We need to take care to set the order when defining the factor:

```
months.0 <- c("Jan","Feb","Mar","Apr","May","Jun",
              "July","Aug","Sep","Oct","Nov","Dec")
months<- factor(months.0,levels=months.0)
contrasts(months) <- contr.helmert(levels(months))
lm(values~months)

Call:
lm(formula = values ~ months)

Coefficients:
(Intercept)      months1      months2      months3      months4      months5
    4.04167      0.05000     -0.01667      0.06667     -0.04000      0.02333

months6      months7   months8      months9     months10     months11
0.41667      0.43750  0.45139      0.56111      0.64091      0.45076
```

Now, months1 is 1/2 of the deviation between January and February, months2 is the deviation between March and the mean of Jan/Feb, and so on. We can see that the coefficient rises at months6, which indicates an increase in July.

16.1.2 Successive difference coding

Successive difference coding in the `MASS` library is similar to helmert coding, but it allows each level to be compared to the single previous level. The documentation is a bit confusing, but if you look at the coding, you can see that the first sees if a change happens between 1 and 2; whereas the second looks at whether the change happens between 3 and 2.

```
library(MASS)


> contr.sdif(3)
      2-1        3-2
1 -0.6666667 -0.3333333
2  0.3333333 -0.3333333
3  0.3333333  0.6666667

> contr.sdif(4)
    2-1  3-2   4-3
1 -0.75 -0.5 -0.25
2  0.25 -0.5 -0.25
3  0.25  0.5 -0.25
4  0.25  0.5  0.75
```

Note that if we repeat the months example, we see that the lm coefficients are exactly the between-month differences:

```
contrasts(months) <- contr.sdif(months)
lm(values~months)

Call:
lm(formula = values ~ months)

Coefficients:
   (Intercept)   monthsFeb-Jan   monthsMar-Feb   monthsApr-Mar   monthsMay-Apr
     4.042          0.100          -0.100           0.300          -0.400

monthsJun-May  monthsJuly-Jun  monthsAug-July   monthsSep-Aug   monthsOct-Sep
 0.300          2.800           1.000            1.000           2.000

monthsNov-Oct   monthsDec-Nov
 2.000          -1.000
```

Looking at the salary-specialty case, we get pairs that are compared in alphabetical order, which may or may not be interesting. However, the levels are fairly clearly labeled and also easier to understand than helmert coding.

```

> contrasts(specialty)<-contr.sdif(levels(specialty))
> lm(salary~specialty)

Call:
lm(formula = salary ~ specialty)

Coefficients:
(Intercept)    specialtyBusiness-Advertising
37.0                           18.6
specialtyCertification-Business
-28.8
```

16.1.3 Sum-to-zero or Deviation coding

The "sum to zero" contrasts are available via `contr.sum()`. Here, the contrasts compare different pairs of elements. This is a version of 'deviation' coding. You may choose to reorder the rows of this if it makes more sense for your study. Here, the first contrast compares level 1 to 3, and the second compares level 2 to 3.

```
> contr.sum(levels(specialty))
               [,1] [,2]
Advertising       1    0
Business          0    1
Certification    -1   -1
```

If we set this contrast to our predictor, we can see something interesting:

```

contrasts(specialty)<-contr.sum(levels(specialty))
> lm(salary~specialty)

Call:
lm(formula = salary ~ specialty)

Coefficients:
(Intercept)   specialty1   specialty2
37.0          -2.8          15.8

> mean(salary)
[1] 37
> tapply(salary,specialty,mean)
Advertising      Business Certification
   34.2           52.8           24.0

 37-2.8    ## == 34.2
[1] 34.2
 37 +15.8  ## == 52.8
[1] 52.8
 37 +2.8-15.8  ##==24
[1] 24
```

Notice that this coding does something interesting–now, the intercept is the mean of all values. Each remaining contrast ends up being the difference between the mean and the particular level.

There are a handful of other similar coding schemes available, and at times they make specific planned tests easy to do. These are sometimes referred to as "planned contrasts", and they permit automatic comparison of a model with and without the contrast using the ANOVA F-test method. A planned contrast test can be carried out in other ways, but if you have a good idea of the specific levels you want to compare, these make for nice ways of testing them, because they fit into the basic scheme of ANOVA tests of nested models. That is, remember that testing the significance of a predictor is equivalent to testing the model with that predictor to one without it.

A few things to recognize about contrasts:

- Some decision about contrasts has to be made, or is made for you, by the software you use.
- In many cases, the contrast you choose doesn't really matter.
- Orthogonal contrasts are nice because their estimates will not change in the absence of one another. But not all contrasts of interest are orthogonal
- Contrasts whose values sum to 0 are nice because it makes the null hypothesis easy to test.
- You should probably use planned contrasts only when you have a well-designed experiment and well-understood paradigm. Typically, you will end up doing additional tests that don't correspond directly to these "planned" contrasts anyway, so you need to protect yourself from having too high of a false alarm rate.

In many cases, the particular contrast doesn't matter, because if you are performing an "ANOVA" test, you are comparing a model with the entire set of predictors to one without. In those cases, the actual coefficients are sometimes of secondary importance, and so any coding will produce the same outcome:

```
contrasts(specialty)<-contr.sum(levels(specialty))
specialty2 <- specialty
contrasts(specialty2) <- contr.helmert(levels(specialty2))


anova(lm(salary~specialty))
Analysis of Variance Table

Response: salary
          Df Sum Sq Mean Sq F value   Pr(>F)
specialty  2 2132.4  1066.2  9.6227 0.003209 **
Residuals 12 1329.6   110.8
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anova(lm(salary~specialty2))

Analysis of Variance Table

Response: salary
          Df Sum Sq Mean Sq F value   Pr(>F)
specialty  2 2132.4  1066.2  9.6227 0.003209 **
Residuals 12 1329.6   110.8
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 . 0.1  ' ' 1
```

Notice that both produced identical results, even though they used different codings. The aov function provides one shortcut to feeding the linear model into an anova table function.

```
> aov(salary~specialty)
Call:
   aov(formula = salary ~ specialty)

Terms:
                specialty Residuals
Sum of Squares     2132.4    1329.6
Deg. of Freedom         2        12
```

```
Residual standard error: 10.52616
Estimated effects may be unbalanced
> summary(aov(salary~specialty))
            Df Sum Sq Mean Sq F value  Pr(>F)
specialty    2   2132  1066.2   9.623 0.00321 **
Residuals   12   1330   110.8
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

16.1.4 Example regressions with different contrasts

The tooth growth data set looks at tooth growth as a function of different dosages of vitamin c (.5, 1 and 2 mg).

```
lm.tooth1 <- lm(len~dose,data=ToothGrowth)
summary(lm.tooth1)

 summary(lm.tooth1)

Call:
lm(formula = len ~ dose, data = ToothGrowth)

Residuals:
    Min      1Q  Median      3Q     Max
-8.4496 -2.7406 -0.7452  2.8344 10.1139

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   7.4225     1.2601    5.89 2.06e-07 ***
dose          9.7636     0.9525   10.25 1.23e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.601 on 58 degrees of freedom
Multiple R-squared:  0.6443,  Adjusted R-squared:  0.6382
F-statistic: 105.1 on 1 and 58 DF,  p-value: 1.233e-14

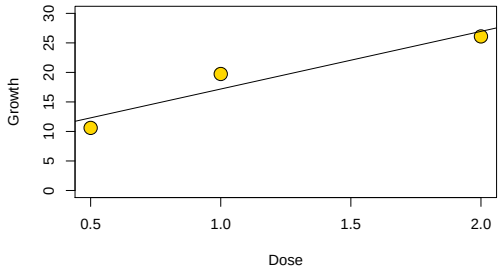


plot(aggregate(ToothGrowth$len,list(ToothGrowth$dose),mean),ylim=c(0,30),
xlab="Dose",ylab="Growth")

means <- aggregate(ToothGrowth$len,list(ToothGrowth$dose),mean)
>
> means
Group.1      x
1     0.5 10.605
2     1.0 19.735
3     2.0 26.100
>
```

This doesn't seem completely linear. We might try to fit a polynomial regression, but what if we instead treated these as categories. This is often an acceptable approach when we do not know (or really care) about the particular functional form of a continuous predictor. Instead of establishing the linear relationship of tooth growth as vitamin c increases, we just ask whether 1 mg is better than 0.5, or if 2.0 is better than 1 ,or 0.5.

Figure 16.1: Means of tooth growth data, and linear model describing the effect




```
dosage <- as.factor(ToothGrowth$dose)
lm.tooth2 <- lm(ToothGrowth$len~dosage)
summary(lm.tooth2)

Call:
lm(formula = ToothGrowth$len ~ dosage)

Residuals:
Min      1Q  Median      3Q     Max
-7.6000 -3.2350 -0.6025  3.3250 10.8950

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.6050     0.9486  11.180 5.39e-16 ***
dosage1       9.1300     1.3415   6.806 6.70e-09 ***
dosage2      15.4950     1.3415  11.551  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.242 on 57 degrees of freedom
Multiple R-squared:  0.7029,  Adjusted R-squared:  0.6924
F-statistic: 67.42 on 2 and 57 DF,  p-value: 9.533e-16
```

Now, by default this uses the R contrast coding, so the model says that dosage 1 and dosage 2 each differ from dose 0.5, whose mean is 10.6. Really, what happens in the regression is that neither of the two contrasts include 0.5 level, and so when the regression finds the least-squares estimate, all of level 0.5 (and nothing else) gets included in the intercept to balance out the least-squares equation.

```
contrasts(dosage)
    1 2
0.5 0 0
1   1 0
2   0 1
```

This default contrast is often a bit confusing. What if we wanted to know whether each additional dosage makes a difference? We can use successive difference coding:

```
contrasts(dosage) <- contr.sdif(levels(dosage))
```

```
contrasts(dosage)
> contrasts(dosage)
1-0.5        2-1
0.5 -0.6666667 -0.3333333
1    0.3333333 -0.3333333
2    0.3333333  0.6666667
> summary(lm(ToothGrowth$len~dosage))

Call:
lm(formula = ToothGrowth$len ~ dosage)

Residuals:
Min      1Q  Median      3Q     Max
-7.6000 -3.2350 -0.6025  3.3250 10.8950

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)  18.8133     0.5477  34.352  < 2e-16 ***
dosage1-0.5   9.1300     1.3415   6.806 6.70e-09 ***
dosage2-1     6.3650     1.3415   4.745 1.44e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.242 on 57 degrees of freedom
Multiple R-squared:  0.7029,  Adjusted R-squared:  0.6924
F-statistic: 67.42 on 2 and 57 DF,  p-value: 9.533e-16
```

Now, because each contrast is balanced to sum to 0, the intercept is the grand mean. Once we have the grand mean, the estimates are constrained to relative differences between groups, for which there are only two distinct values. Here, the first parameter is the difference between dosage 1 and 0.5–the same as the first parameter of the first model. But the second is the difference between level 2.0 and level 1.0 (26.1-19.735=6.365).

But maybe instead we wanted to look at when or whether the increase stopped. This would involve helmert or reverse helmert coding. (Some sources suggest that the R helmert coding is actually ‘reverse helmert’).

```
contrasts(dosage) <- contr.helmert(levels(dosage))
contrasts(dosage)
summary(lm(ToothGrowth$len~dosage))
> summary(lm(ToothGrowth$len~dosage))

Call:
lm(formula = ToothGrowth$len ~ dosage)

Residuals:
Min      1Q  Median      3Q     Max
-7.6000 -3.2350 -0.6025  3.3250 10.8950

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  18.8133     0.5477  34.352  < 2e-16 ***
dosage1       4.5650     0.6707   6.806 6.70e-09 ***
dosage2       3.6433     0.3873   9.408 3.35e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.242 on 57 degrees of freedom
Multiple R-squared:  0.7029,  Adjusted R-squared:  0.6924
F-statistic: 67.42 on 2 and 57 DF,  p-value: 9.533e-16
```

```
25
b0 =  mean(means$x)  #grand mean
27 b1 = (means$x[2] - mean(means$x[1]))/2   # (2 - 1)/2
b2 = (means$x[3]  - mean(means$x[1:2]) )/3  # (3-(1:2))/3
29
c(b0,b1,b2)
31 [1] 18.813333  4.565000  3.643333
```

Here, b0 is the mean of group 1. b1 is related to group 2 - mean(b1); and b3 is mean of group 3 - mean(b1,b2). This might be better for detecting when a growth starts–if the values are flat until some point and then take off, the parameter where this first happens will be the first significant one. We could play with the rows of this helmert coding or the order of the factor to ask these alternate questions.

16.1.5 Regression and the One-way ANOVA

Note: This describes the application of the ANOVA method to simple designs, where all observations are balanced so that you have an equal number of observations in each group, and you have no other systematic dependencies, such as a within-subject manipulation or a repeated-measures design. Those issues are dealt with in future chapters.

In the salary x specialty example, the regression shows us the coefficients for whatever contrast we decided to use, and `summary()` provides other information. If we want to compare this to the intercept-only model, it provides such a comparison by default in the F test. But as we've seen before, we can also give this to the `anova()` function to report an ANOVA table.

```
> contrasts(specialty) <- contr.treatment(levels(specialty))
2 > summary(lm(salary~specialty))

4 Call:
lm(formula = salary ~ specialty)
6
Residuals:
8 Min      1Q Median      3Q     Max
-24.8   -4.6   -0.2    4.9   17.2
10
Coefficients:
12 Estimate Std. Error t value Pr(>|t|)
(Intercept)              34.200      4.707   7.265 9.94e-06 ***
14 specialtyBusiness        18.600      6.657   2.794   0.0162 *
specialtyCertification  -10.200      6.657  -1.532   0.1514
16 ---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
18
Residual standard error: 10.53 on 12 degrees of freedom
20 Multiple R-squared:  0.6159,  Adjusted R-squared:  0.5519
F-statistic: 9.623 on 2 and 12 DF,  p-value: 0.003209
```

If we put the lm model in an `anova` function, it will create the Analysis of Variance table comparing the model to each neighboring sub-model. In this case, we only have the neighboring intercept-only model, so the specialty line simple determines whether specialty accounts for additional variance that the intercept does not:

```
> anova(lm(salary~specialty))
 Analysis of Variance Table

 Response: salary
           Df Sum Sq Mean Sq F value   Pr(>F)
 specialty  2 2132.4  1066.2  9.6227 0.003209 **
 Residuals 12 1329.6   110.8
 ---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that the F test is the same in both cases. However, this second table is called the "Analysis of Variance Table", and is one of the primary outputs used in experimental data analysis. Notice that it hides the complexity of contrasts and parameter estimates. In fact, under normal conditions the coding won't even matter and we will get the same values for the "omnibus" ANOVA test. All the main ANOVA tells us is that the model with specialty is significantly better than the model without specialty, and it does this by estimating the chance of finding the results we found if specialty was not related. The low p-value indicates that this result would have been very unlikely to have occurred by chance if there was no real difference.

Because of tradition and training, many researchers don't even recognize that the ANOVA is just a regression with categorical predictors, or may recognize it as a curious fact. For many experimentalists, the ANOVA is the first and last procedure used, and software has been developed since the 1970s to support this. In fact, in some commercial software versions, the ANOVA is standard and the linear model (on which it is based) is an add-on you pay extra for! In R, `aov()` function will create this model by itself.

```
 > aov(salary~specialty)
 Call:
 aov(formula = salary ~ specialty)

 Terms:
 specialty Residuals
 Sum of Squares     2132.4    1329.6
 Deg. of Freedom         2        12

 Residual standard error: 10.52616
 Estimated effects are balanced

 > summary(aov(salary~specialty))
             Df Sum Sq Mean Sq F value  Pr(>F)
 specialty    2   2132  1066.2   9.623 0.00321 **
 Residuals   12   1330   110.8
 ---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The traditional ANOVA table includes sum of squares and degrees of freedom. These indicate the amount of variance accounted for and so provide a measure of effect size. However, the make-up of the ANOVA table is mostly a historical anachronism, dating back to days in which people had to create an ANOVA by hand. But, just like we saw with regression, the 9.623 value comes from finding the ratio of the two Mean Square values: $9.623 = 1066.2/110.8$. Just like in regression, this F test is the comparison of the complete model to one including the intercept only. The intercept-only model will have 1330+2132 sum squared error, and the specialty model reduces the error to 1330.

When you have a single predictor, this is called a one-way ANOVA, and it is identical to the model you get from the F-test in the regression. Now, let's look at what happens to the F test for different contrasts:

```
> contrasts(specialty) <- contr.treatment( levels(specialty))
> s <- summary(lm(salary~specialty))
> s$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000
> contrasts(specialty) <- contr.SAS( levels(specialty))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000
> contrasts(specialty) <- contr.sdif( levels(specialty))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000
> contrasts(specialty) <- contr.helmert( levels(specialty))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000
> contrasts(specialty) <- contr.sum( levels(specialty))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000
>
> contrasts(specialty) <- contr.treatment( levels(specialty))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000
> contrasts(specialty) <- contr.SAS( levels(specialty))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000
> contrasts(specialty) <- contr.sdif( levels(specialty))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000
> contrasts(specialty) <- contr.helmert( levels(specialty))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000
> contrasts(specialty) <- contr.sum( levels(specialty))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
9.622744  2.000000 12.000000


##some random set of contrasts:
contrasts(specialty ) <- cbind(c(1,1,2),c(3,-1,4))
summary(lm(salary~specialty))$fstatistic
    value     numdf     dendf
 9.622744  2.000000 12.000000

##another random set:
contrasts(specialty ) <- cbind(c(1,1,2),c(0,0,0))
> summary(lm(salary~specialty))$fstatistic
value     numdf     dendf
7.508544  1.000000 13.000000
```

Notice that here, it doesn't really matter what contrast we used–we get the same omnibus

F test each time; except for the last case where we create a set of contrasts that actually equate levels 1 and 2. If our contrasts really are planned, this test is sufficient evidence for these differences. If a different test were planned, could make the contrasts align with our hypotheses. If we have no specific hypothesis, or we want to test all possible comparisons, then it gets trickier. In order to do that, we need to consider post-hoc tests. But before we do so, let's examine the assumptions of the ANOVA model.

16.2 Testing ANOVA Assumptions

Because ANOVA is a form of regression, many of the same assumptions apply–except for the assumption of linear effects, which if we restrict ourselves to categorical predictors, are irrelevant. The normality assumptions can be tested like in regression, and you should always look at residuals with qqplots and histograms. But in addition to these, specific tests of homogeneity of variance are available in ANOVA–they specifically test whether the variance is the same in different categorical groupings.

There are probably dozens of such tests that are available in the literature. Some common common tests available in R, including:

- bartlett.test(); a general test of homogeneity of variance
- var.test(); a special case for comparing two variances
- fligner.test(); a non-parametric test
- ansari.test() and mood.test(); two non-parametric tests for equality of variance in two groups.
- leveneTest() in the `car` package, another test that commonly reported by users of SPSS.

ANOVA pools variance across all residuals, so it essentially assumes that each group has the same variance. This is a strange assumption if you think about it, because you are hoping that there is a difference in the means, and to determine this you assume that there is no difference in the variance. Most of these tests work similarly and will provide similar conclusions.

16.2.1 Bartlett's K-squared Test of Homogeneity of Variance

Bartlett's K-Squared test [2] can be executed in two ways–one way puts a regression formula into it, and the other specifies input and output variables. If you have a more complicated set of predictors, you may need to recode a predictor variable that includes each combination of groups. Authors suggest that the Bartlett test is best used on data you know is normally distributed; it is sensitive to non-normality even when variances do not differ much.

```
2 bartlett.test(salary~specialty)
  bartlett.test(salary,specialty)
4
  > bartlett.test(salary~specialty)
6
  Bartlett test of homogeneity of variances
```

[2] Bartlett, M. S. (1937). Properties of sufficiency and statistical tests. Proceedings of the Royal Society of London Series A 160, 268—282.

```
8
  data:  salary by specialty
10 Bartlett's K-squared = 8.1217, df = 2, p-value = 0.01723
```

Here, the Bartlett K^2 test is significant, suggesting the variances differ.

16.2.2 Levene's equality of variance Test

Levene's test is commonly reported by SPSS, and is available in the car package. Authors suggest that Levene's test is less dependent on the data being normally-distributed. If we run Levene's test on the same data, we find that it is not significant (but just barely):

```
  library(car)
2 leveneTest(salary,specialty)
  Levene's Test for Homogeneity of Variance (center = median)
4 Df F value  Pr(>F)
  group  2  3.3685 0.06901 .
6 12
```

16.2.3 Fligner test

A non-parametric test is called the Fligner test, or the Fligner-Killeen test. It uses a chi-squared test for homogeneity of variance.

```
  fligner.test(salary~specialty)
2
  Fligner-Killeen test of homogeneity of variances
4
  data:  salary by specialty
6 Fligner-Killeen:med chi-squared = 5.2297, df = 2,
  p-value = 0.07318
```

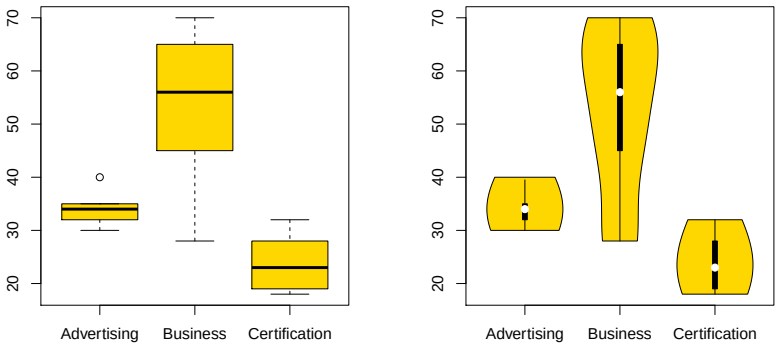
Note here that the equal variance assumption is violated according to Bartlett, but OK according to Levene and Fligner. What could lead to this difference? Maybe some of the assumptions of one or both of *these* tests are not met–probably normality. This may be confirmed by looking at the boxplots or violinplots:

Apparently, there is a difference in variance AND normality across groups. This violates two of our assumptions of the ANOVA/regression, and so we should be careful about how we proceed.

16.3 Dealing with unequal variance

If you have detected unequal variance, a few different things might work. You might try a transform, or maybe you might decide to remove some influential outliers, or you might try to add additional covariates to make a better model. Alternately, there are parametric tests that don't assume normality. One such test is available in the oneway.test(), which implements what is known as Welch's one-way ANOVA–a modified version of the Welch's t-test that handles unequal variance in pairwise comparisons.

Figure 16.2: Boxplot of the salary by specialty data set. This reveals that not only do the variances appear to differ, business appears skewed, which leads to trouble with some of the tests.




```
> oneway.test(salary~specialty,var.equal=F)

One-way analysis of means (not assuming equal variances)

data:  salary and specialty
F = 8.3724, num df = 2.0000, denom df = 6.8695, p-value = 0.01439
```

Notice that in this test, the residual degrees of freedom are adjusted–this is a trick that is sometimes done to adjust for the fact that variance differs, allowing us to use the same F distribution. Here, our interpretation is not that business leads to a higher salary than the others, but simply that different business sector have different average salaries. The pairwise comparisons would need to be handled in a regression or planned comparison.

16.4 Kruskal-Wallis H

Another alternative is to use a non-parametric test; the Kruskal-Wallis H is a common approach for a one-way ANOVA. Because we are not assuming anything about the distribution, and looking at rank-order differences, this test can be robust to strange data with a lot of outliers or skewed tails.

Note: There is a good tutorial of the Kruskal-Wallis test here: http://www.r-tutor.com/elementary-statistics/non-parametric-methods/kruskal-wallis-test.

```
summary(kruskal.test(len~dose,data=ToothGrowth))

Kruskal-Wallis rank sum test

data:  len by dose
Kruskal-Wallis chi-squared = 40.6689, df = 2, p-value = 1.475e-09

##############################
```

```
##Salary data set:
kruskal.test(salary~specialty)

Kruskal-Wallis rank sum test

data:  salary by specialty
Kruskal-Wallis chi-squared = 8.4151, df = 2, p-value = 0.01488
```

Notice that again, dose is a significant predictor, as is specialty when predicting salary.

16.5 Bayesian One-way ANOVA

Finally, the BayesFactor package provides an analysis corresponding to one-way ANOVA: anovaBF. This computes a Bayes Factor comparing the Null hypothesis to the alternative that the categorical predictor is a better description of the data. However, because we are making specific assumptions about normality, he bayesian anova does not help us avoid the normality and equal variance assumptions. These could be handled by making specific distributional assumptions in a Bayes model, but neither the aov or anovaBF will handle that easily.

```

dat <- data.frame(salary,specialty)
abf <- anovaBF(salary~specialty,data=dat)

abf
Bayes factor analysis
--------------
[1] specialty : 13.21095 +- 0%
```

Here, specialty produces a Bayes Factor value of 13.2, which indicates strong support for the alternative (that specialty influences salary).

16.6 Testing differences between levels of a predictor in ANOVA and Multiple Comparisons

In both regression and ANOVA models, we test for whether an entire predictor set is useful by comparing models with and without that predictor, or equivalently testing whether a β value (parameter) is different from zero. But the basic logic of the one-way ANOVA usually leads to an answer we are not interested in. It asks, for a factor having more than two levels, whether including that factor is better than not including it. We are usually interested in additional hypotheses–is one level of the predictor better than another specific level. If we have very tight control over an experiment, we may be able to choose a contrast in a regression equation that will map onto exactly the set of hypotheses we have. Oftentimes, we may have a control condition that can be coded as the first level, and then comparisons are directly coded in contrast to that control.

If we look at the tooth growth regression table, the 0.5 dosage is the baseline level.

```
contrasts(dosage) <- contr.treatment(levels(dosage))
summary(lm( ToothGrowth$len ~ dosage))

```

```
Call:
lm(formula = ToothGrowth$len ~ dosage)

Residuals:
Min      1Q  Median      3Q     Max
-7.6000 -3.2350 -0.6025  3.3250 10.8950

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.6050     0.9486  11.180 5.39e-16 ***
dosage1       9.1300     1.3415   6.806 6.70e-09 ***
dosage2      15.4950     1.3415  11.551  < 2e-16 ***

Residual standard error: 4.242 on 57 degrees of freedom
Multiple R-squared:  0.7029,  Adjusted R-squared:  0.6924
F-statistic: 67.42 on 2 and 57 DF,  p-value: 9.533e-16
```

Notice how the standard error estimate for the different dose levels are the same? This is no accident. In fact, these are all calculated based on the RSE. 4.242/sqrt(10)=1.34. In this test, we know that both dosage1 and dosage2 are reliably different from 0. What if we also want to know whether dosage1 and dosage2 differ? We can use the normal t-test machinery to do that.

```
> ((15.495-9.13)/ 1.34
4.75
1-pt(4.75,57)
[1] 7.079305e-06
```

We could have done all this before we learned ANOVA/regression models, just doing pairwise t tests. Here is a function that will let us do so:

```
pairwise.t.test(ToothGrowth$len,ToothGrowth$dose)

Pairwise comparisons using t tests with pooled SD

data:  ToothGrowth$len and ToothGrowth$dose

0.5     1
1 1.3e-08 -
2 4.4e-16 1.4e-05

P value adjustment method: holm
```

This produces all possible pairs of t tests. Here, we see that each differ from the other. But there is a problem–we need to protect ourselves somewhat from finding false alarms. If we have 7 levels, there are 6+5+4+3+2+1=21 possible comparisons. We could use t-test logic to compare any possible pairings, but in this case if we use a criteria like p=.05, we'd expect at least one of these to differ significantly just by chance. The pairwise.t.test allows us to adopt different ways of protecting ourselves from multiple comparisons. By default, it uses 'holm' method, which is one reasonable approach, but there are others you may prefer. The most conservative is the Bonferonni correction, which simply assumes all of the tests are uncorrelated, and essentially adjusts the p-value needed to achieve an overall false alarm rate we specify.

```
pairwise.t.test(ToothGrowth$len,ToothGrowth$dose,p.adj="bonf")
Pairwise comparisons using t tests with pooled SD

data:  ToothGrowth$len and ToothGrowth$dose

0.5     1
1 2.0e-08 -
2 4.4e-16 4.3e-05

P value adjustment method: bonferroni
```

The lesson here is that you can often use simple t tests based on the RSE to test differences between parameters, or adjust independent t-tests using an adjustment technique such as Holm or Bonferonni. However, some people object to this type of approach in some cases. The Bonferroni correction is very conservative–it does each test at a 1/N value that would otherwise be considered significant. So if you were doing ten tests using a .05 criterion, the Bonferroni correction requires a .005 level of significance. This means that your effects need to be much larger in order to detect them, which may be a waste of resources. In those cases, a better approach might be to look at the significant effects and demonstrate they can be replicated in a second experiment.

16.6.1 Multiple comparisons and post-hoc tests in ANOVA

Neither of these two approaches are ideal. In one case, we faked a t-test based on parameters of the regression. This is good because it allows us to base our variability on true residual variability–if we had added other predictors to the model, we would have reduced residual variance and been able to detect smaller values. But we are left to correct for multiple testing on our own, usually by just picking a smaller value or using the conservative bonferroni correction. The other alternative allows us to choose a correction scheme, but won't let us use the regression or ANOVA model to estimate residual variance, which means we may not detect true differences because variability we know about is being included in residual variance.

The solution is a set of related tests typically referred to as 'Post-Hoc' tests. Post-hoc tests are those that are not planned contrasts. A good post-hoc test will use the residual error variance from the model to do the test, but will correct for multiple comparisons as well. There are many such tests–he most common is probably Tukey's Honest Significant Different (HSD) test. It can be run on an `aov()` model directly, and will provide confidence intervals and adjusted p values for all pairwise comparisons.

```
TukeyHSD(aov(salary~specialty))
Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = salary ~ specialty)
Fit: aov(formula = salary ~ specialty)

$specialty
                           diff          lwr        upr     p adj
Business-Advertising       18.6   0.8391598  36.36084 0.0400226
Certification-Advertising -10.2 -27.9608402   7.56084 0.3112892
Certification-Business    -28.8 -46.5608402 -11.03916 0.0026182


> TukeyHSD(aov(len~dose,data=ToothGrowth)
```

```

> TukeyHSD(aov(len~dosage,data=ToothGrowth))
Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = len ~ dosage, data = ToothGrowth)

$dosage
diff       lwr       upr    p adj
1-0.5  9.130  5.901805 12.358195 0.00e+00
2-0.5 15.495 12.266805 18.723195 0.00e+00
2-1    6.365  3.136805  9.593195 4.25e-05
```

Here, in the salary data set, certification and advertising do not differ significantly, but the two other pairings do. For the tooth growth, each dosage level differs significantly from each other level.

There are many other post-hoc tests you can use. Within the car package, there are a number, and the TukeyLSD test provides a number of options, including a Bonferroni correction.

16.6.2 Post-Hoc test with BayesFactor ANOVA

The Bayes factor ANOVA has a different way of doing post-hoc tests. You are able to do this by sampling the posterior distribution of different values, and looking at how many of the observed samples satisfied your constraint.

The Bayesian model works by attempting to create a distribution around plausible parameter estimates. It will do this by sampling cases repeatedly that can be mathematically proven to conform to the posterior distribution we care about. If we use the `posterior` function on a BayesFactor model, it will sample parameter estimates from this posterior distribution. So, we can just look at the proportion of these estimates that satisfy each particular contrast we are interested in. This proportion is a measure of how many of the likely parameter estimates that would have lead to the outcome we observed would have satisfied our test.

To sample, we use the posterior function:

```

specialty <- as.factor(c("Advertising","Advertising","Advertising",
"Advertising","Advertising",
"Business","Business","Business","Business","Business",
"Certification","Certification","Certification",
"Certification","Certification"))

salary <- c(30,35,32,40,34,70,56,45,65,28,19,23,28,18,32)
dat <- data.frame(salary,specialty)
abf <- anovaBF(salary~specialty,data=dat)
chains <- posterior(abf,iterations=10000)

plot(chains[,2:4]])

summary(chains)


> summary(chains)

Iterations = 1:10000
```

```
Thinning interval = 1
Number of chains = 1
Sample size per chain = 10000

1. Empirical mean and standard deviation for each variable,
plus standard error of the mean:

                            Mean     SD Naive SE Time-series SE
mu                        37.022  3.155  0.03155        0.03203
specialty-Advertising     -2.317  3.902  0.03902        0.03902
specialty-Business        12.532  4.747  0.04747        0.07340
specialty-Certification -10.215  4.492  0.04492        0.06629
sig2                     148.276 74.147  0.74147        1.15385
g_specialty                2.961 11.528  0.11528        0.14136

2. Quantiles for each variable:

                              2.5%      25%      50%      75%   97.5%
mu                         30.7537  35.0297  37.004  38.9967  43.386
specialty-Advertising    -10.0926  -4.7806  -2.285   0.1859   5.257
specialty-Business         2.8084   9.4752  12.726  15.7212  21.566
specialty-Certification -18.8291 -13.2754 -10.317  -7.3131  -1.098
sig2                      60.1829  98.0835 130.014 177.9454 344.247
g_specialty                0.1057   0.4803   1.071   2.4048  16.407
```

If we do `plot(chains)`, this will automatically show us the distribution of posterior values, as shown in Figure 16.3. The summary method shows similar things—it provides the best estimates and standard deviations or standard errors that could be used to make figures or table. Here, mu is the grand-mean, the three specialty levels are the deviations from the grand-mean, `sig2` is the error variance, and `g_specialty` is a parameter controlling the three specialties. So far, this looks reasonably similar to what we might have done with a standard ANOVA, but the inferences are somewhat different. The Bayesian samples show us how likely initial sets of parameters are to have produced the data we observed. The model obtains these through monte carlo simulation–not just calculating statistics of data. These posterior distributions are sampled, but if we sample enough we can be confident about the possible values, and make similar inferences as we do in standard statistics, based on how much the distributions overlap.

If we look at the columns of chains, these are associated with these variables. Each row is a random sampling from the posterior distribution of these parameters.

```
chains[1:5,]
          mu    specialty-Adv      specialty-Bus      specialty-Cert
[1,] 36.83802   -2.7339783          13.183127           -10.449148
[2,] 39.82864    0.4432873           9.969553           -10.412841
[3,] 35.20709   -1.4354836           4.479783            -3.044299
[4,] 31.89395   -0.4472300          13.012616           -12.565386
[5,] 37.48467   -3.7155770          22.926612           -19.211035
          sig2 g_specialty
[1,]  68.80031   4.5549329
[2,] 140.39422   1.7207386
[3,] 147.79995   0.2602432
[4,]  86.65993   2.8790289
[5,] 105.47087   5.7729450
```

We can use these samples to test hypotheses. For example, if we want to know whether business makes more or less than the mean, it is just looking at how many of the samples

Figure 16.3: Sampled posterior distributions for each parameter.This shows the likely values that could have produced the data.

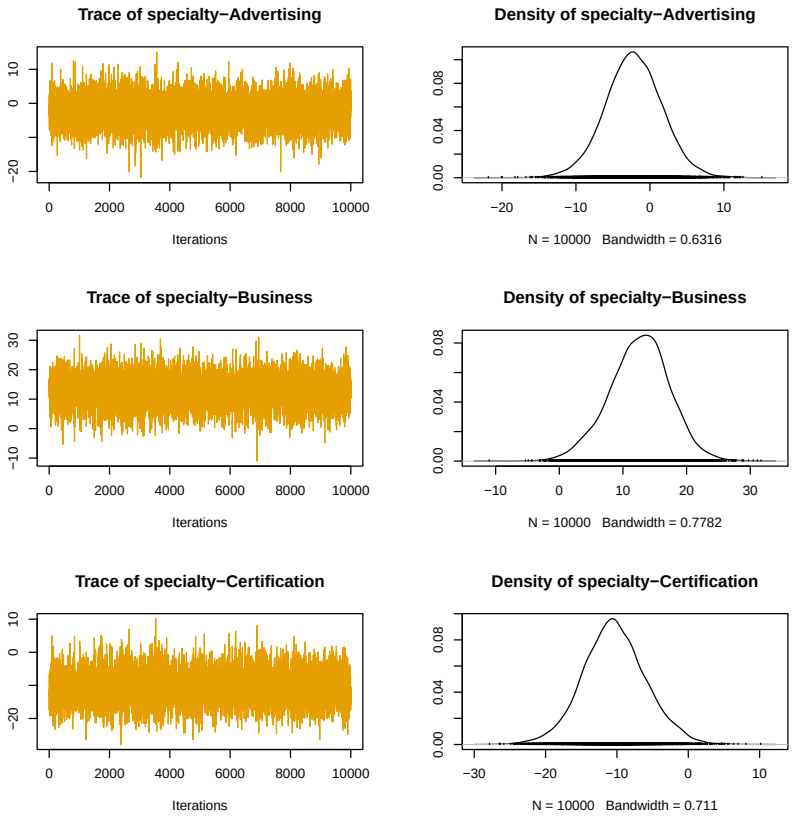
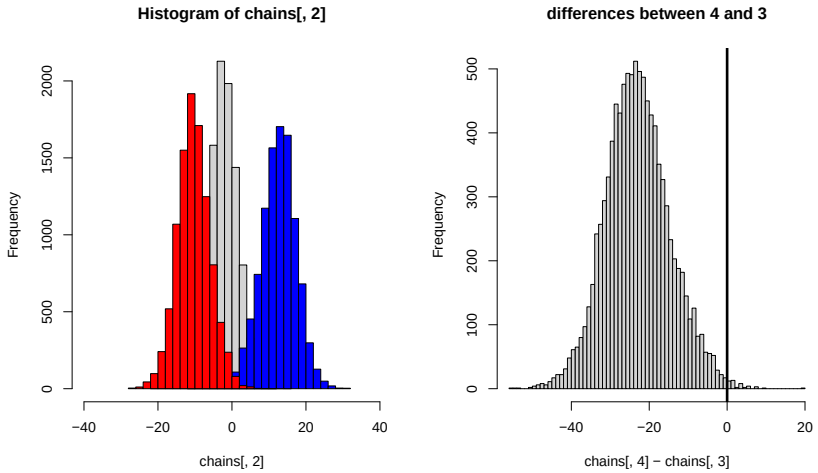


Figure 16.4: Left panel shows overlapping histograms of the posterior estimates of the coefficients for each group (Advertising, Business, certification). The right panel shows the posterior distribution of the difference.



were greater than 0:

```
 mean(chains[,3]>0)
[1] 0.9942
```

Here, more than 99% of plausible parameters for business were greater than the mean, which is strong evidence for this hypothesis.

We could look at the posterior distribution of the groups on their own, as shown in left panel of Figure 16.4

```
  hist(chains[,2],xlim=c(-40,40))
  hist(chains[,3],add=T,col="blue")
  hist(chains[,4],add=T,col="red")
```

This is much like Figure 16.3, but overlays them on the same graph. But even though this is a between-subject design, the individual samples in the model depend on one another. So in essence, to test the difference between groups, we can do essentially a paired test of the samples, to see how often they differed. The distribution of the difference between certification and business is shown in the right panel of Figure 16.4, which shows almost no overlap in sampled values.

```
# certification > business
 mean(chains[,4]>chains[,3])  ##probability that  > 5==1%
 [1] 0.0046
# certification > Advertising
 > mean(chains[,4]>chains[,2])  ##probability that 4 > 2==11%
 [1] 0.1184
#business > Advertising
 > mean(chains[,3]>chains[,2])  ##probability that 3 > 2==97%
 [1] 0.9759
```

```
11
   hist(chains[,4]-chains[,3],breaks=100,main="differences between 4 and 3")
13 abline(v=0,lwd=3)
```

Here, there is strong support that business is greater than both certification (99.5% of cases) and advertising (97%), but the support for a difference between certification and advertising is less strong (it was true for only 89% of plausible parameter estimates). This is consistent with the Tukey test performed earlier.

Chapter 17

Multi-Way (Factorial) ANOVA

Note: as with the One-way ANOVA, this describes the application of the ANOVA method to simple designs, where all observations are balanced so that you have an equal number in each group, and you have no other systematic dependencies, such as a within-subject manipulation or a repeated-measures design.

The Tooth Growth data set actually had two predictors–it also included a categorical predictor supp (OJ versus Vitamin C supplement) along with dose numerical predictor. We can easily look at the two sets of effects using `interaction.plot`.

```
interaction.plot(ToothGrowth$dose,ToothGrowth$supp,ToothGrowth$len,pch=15,
      type="b",xlab="Dosage (mg)", ylab="Tooth length (mm)",
      bty="n", las=1,xtick=T,leg.bg="white",leg.bty="n",
      ylim=c(0,30), col=c("orange","red"),trace.label="Supplement")
```

Because ANOVA is just a regression, it should be obvious that the anova procedure can accept multiple predictors. When we have a single predictor, it is called a "One-way" ANOVA, and with two predictors a "two-way" ANOVA, and so-on. Furthermore, we refer to the levels of each factor in the design. For the tooth growth data, we have a 2x3 ANOVA, where 2 indicates the number of levels of supp, 3 indicates the number of levels of dosage, and the number of numbers (2x3 has two numbers) indicates it is 2-way ANOVA. This ANOVA will have (2-1) * (3-1) = 3 degrees of freedom in the Omnibus F test at the end. In some conditions, we would refer to this as a "Factorial" ANOVA, and generally a "Full Factorial" ANOVA if it contained all main effects and interactions.

If we fit this with a linear regression, the summary would look like this:

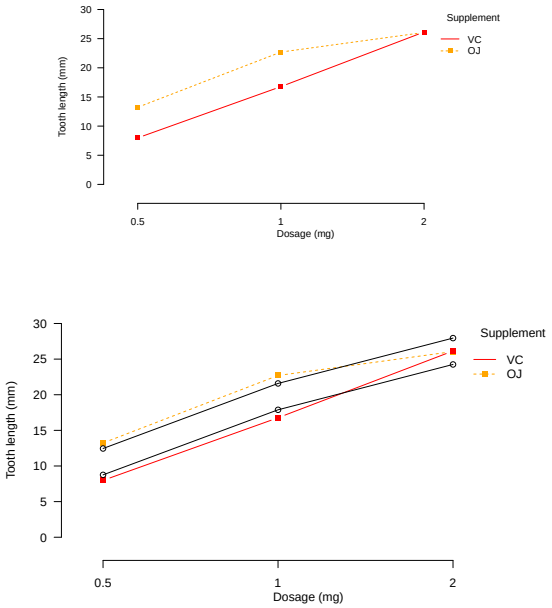
```
ToothGrowth$dosage <- as.factor(ToothGrowth$dose)
lm.tooth3 <- lm(len~supp+dosage,data=ToothGrowth)
summary(lm.tooth3)
anova(lm.tooth3)

> summary(lm.tooth3)

Call:
lm(formula = len ~ supp + dosage, data = ToothGrowth)

Residuals:
Min      1Q  Median      3Q     Max
-7.085 -2.751 -0.800  2.446  9.650
```

Figure 17.1: Tooth growth in Guinea pigs in response two doses of either Vitamin C (VC) or Orange Juice (OJ). The right panel shows the model's predictions.




```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  12.4550     0.9883  12.603  < 2e-16 ***
suppVC       -3.7000     0.9883  -3.744 0.000429 ***
dosage1       9.1300     1.2104   7.543 4.38e-10 ***
dosage2      15.4950     1.2104  12.802  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.828 on 56 degrees of freedom
Multiple R-squared:  0.7623,  Adjusted R-squared:  0.7496
F-statistic: 59.88 on 3 and 56 DF,  p-value: < 2.2e-16

##plot the model predictions:
modelfit <- tapply(lm.tooth3$fit,list(dosage=ToothGrowth$dosage,supp=
    ToothGrowth$supp),mean)
matplot(modelfit,type="o",pch=1,add=T,lty=1,col='black')
```

Here, supplement seems to make a difference, because it is significant. Note that the ANOVA with two predictors assumes the factor effects are additive, and so it cannot account for the way the effects converge at high doses. If we compare this to the model made with just dosage:

```

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.6050     0.9486  11.180 5.39e-16 ***
dosage1       9.1300     1.3415   6.806 6.70e-09 ***
```

```
dosage2      15.4950     1.3415  11.551  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.242 on 57 degrees of freedom
Multiple R-squared:  0.7029,  Adjusted R-squared:  0.6924
F-statistic: 67.42 on 2 and 57 DF,  p-value: 9.533e-16
```

We see a few things:

- The intercept changed (not a big deal)
- The estimates for dosage1 and dosage2 did not change (because the predictors were orthogonal)
- The standard error for our estimates got smaller, and thus the t-values larger and the p-values smaller
- R^2 is larger, RSE is smaller.
- F value changed, but we can't make any generalizations about that.

By adding an additional predictor we reduced the RSE, just like you'd expect in a regression. This reduces our estimate of the variability of our parameters, because we are taking residual variance and explaining it, thus reducing the residuals. So, by adding the additional predictor, we may be able to have a more sensitive test of some effects that were already in the model. This may or may not be justifiable if the secondary predictor is not itself significant.

From the perspective of an ANOVA user, remember that for the one-way ANOVA we compared the model with a categorical predictor to one with just the intercept. But now, we have several possible models and submodels to compare, and the ANOVA table produced with `anova()` will display some of these comparisons:

```
> anova(lm.tooth3)
Analysis of Variance Table

Response: len
Df  Sum Sq Mean Sq F value    Pr(>F)
supp       1  205.35  205.35  14.017 0.0004293 ***
dosage     2 2426.43 1213.22  82.811 < 2.2e-16 ***
Residuals 56  820.43   14.65
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, we have essentially two different F tests–one related to supp, and the second related to dosage. Like the omnibus F test in the one-way ANOVA, these are supposed to compare a model with that predictor to a model without that predictor. But which models exactly are being compared? Let's try to figure it out, using the aov() function

17.1 Interpreting the Analysis of Variance (ANOVA) Table

The ANOVA table for a multi-way ANOVA contains several F-tests. Which tests are they? They are supposed to tell us the effect of a variable within the larger model, but what does

that actually mean?

To look into this, let's build four models for this data set, on with both predictors, one each for the two predictors, and one with only the intercept.

```
m.int <- aov(len~0,data=ToothGrowth)
m.supp <- aov(len~supp,data=ToothGrowth)
m.dose <- aov(len~dosage,data=ToothGrowth)
m.both <- aov(len~dosage+supp,data=ToothGrowth)

> anova(m.both)
Analysis of Variance Table

Response: len
Df  Sum Sq Mean Sq F value    Pr(>F)
dosage     2 2426.43 1213.22  82.811 < 2.2e-16 ***
supp       1  205.35  205.35  14.017 0.0004293 ***
Residuals 56  820.43   14.65
```

We might expect each test to examine whether the individual predictor is better than the intercept-only model–much like in the one-way ANOVA. This would be m.dose and m.supp:

```
 summary(m.dose)
 Df Sum Sq Mean Sq F value   Pr(>F)
 dosage       2   2426    1213   67.42 9.53e-16 ***
 Residuals   57   1026      18
 ---
 Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> summary(m.supp)
             Df Sum Sq Mean Sq F value Pr(>F)
supp         1    205  205.35   3.668 0.0604 .
Residuals   58   3247   55.98
```

Notice that the 2426 and 205 figures appear in both–the sum of squares associated with both predictors don't change because the predictors are orthogonal and balanced. But the F ratio is different. The F ratio is the ratio between the mean square deviation (which is identical in the two models), and the residuals (which change). In our more complex model, the residuals go down because we are explaining them through another reliable predictor. Consequently, the F statistics do change. It would seem to make sense then that we want to look at the effect of a variable in the context of all other predictors; not in isolation. The estimates and MSE won't differ, but our F tests will. To look at this, let's do specific anova() tests comparing the full model to the model without each predictor. To get the effect of dose, compare m.both to m.supp; to get the effect of supplement, compare m.both to m.dose:

```
anova(m.both,m.supp)
Analysis of Variance Table

Model 1: len ~ dosage + supp
Model 2: len ~ supp
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     56  820.4
2     58 3246.9 -2   -2426.4 82.811 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> anova(m.both,m.dose)
```

```
Analysis of Variance Table

Model 1: len ~ dosage + supp
Model 2: len ~ dosage
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1     56  820.43
2     57 1025.78 -1   -205.35 14.017 0.0004293 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```

Now, we can see that not only are the MSE and Sum of Squares the same, the F tests are the same. So, by default, the ANOVA table for a multi-way ANOVA compares the full model to the model without that particular predictor. By doing so, it asks whether the variable has an impact in light of all the other predictors (not in isolation).

17.1.1 Post-hoc testing in multi-way ANOVA

Post-hoc tests will work essentially the same for multi-way ANOVA. Both the `TukeyHSD()` and the `HSD.test()` functions in agricolae will compute these post-hoc tests.

```

TukeyHSD(m.both)

 Tukey multiple comparisons of means
 95% family-wise confidence level

 Fit: aov(formula = len ~ dosage + supp, data = ToothGrowth)

 $dosage
         diff       lwr       upr p adj
 1-0.5  9.130  6.215909 12.044091 0e+00
 2-0.5 15.495 12.580909 18.409091 0e+00
 2-1    6.365  3.450909  9.279091 7e-06

 $supp
 diff       lwr       upr     p adj
 VC-OJ -3.7 -5.679762 -1.720238 0.0004293



> library(agricolae)
> HSD.test(m.both,"dosage", group=TRUE,
        main="Effect of dosage",console=T)

Study: Effect of dosage

HSD Test for len

Mean Square Error:  14.65045

dosage,  means

      len      std  r  Min  Max
0.5 10.605 4.499763 20  4.2 21.5
1   19.735 4.415436 20 13.6 27.3
2   26.100 3.774150 20 18.5 33.9

```

```
alpha: 0.05 ; Df Error: 56
Critical Value of Studentized Range: 3.404809

Honestly Significant Difference: 2.914091

Means with the same letter are not significantly different.

Groups, Treatments and means
a    2     26.1
b    1     19.74
c    0.5   10.6
```

Here, the Tukey test is performed on each variable, or on the one we specify in the argument. It probably does not control for the number of variables tested in a multi-way ANOVA; just the number of comparisons within each variable.

17.1.2 Interpreting Bayes Factor Multi-way ANOVA

The Bayes factor ANOVA produces a Bayes Factor for each candidate sub-model, and we can find the ratio of these factors in order to do the tests we care about.

```
abf2 <- anovaBF(len~dosage+supp,data=ToothGrowth)
summary(abf2)
> summary(abf2)
Bayes factor analysis
--------------
[1] supp                        : 1.198757     0.01%
[2] dosage                      : 4.983636e+12 0%
[3] supp + dosage               : 2.877134e+14 2.25%
[4] supp + dosage + supp:dosage : 7.570807e+14 1.39%

Against denominator:
Intercept only
---
```

To test whether supp matters in the context of dosage, we find the ratio of model 3 versus 2; for supp we find the ratio of model 3 versus model 1.

```
> abf2[3]/abf2[2]
Bayes factor analysis
--------------
[1] supp + dosage : 57.73161 2.25%

Against denominator:
len ~ dosage
---
Bayes factor type: BFlinearModel, JZS

> abf2[3]/abf2[1]
Bayes factor analysis
--------------
[1] supp + dosage : 2.400098e+14 2.25%

Against denominator:
len ~ supp
---
```

```
Bayes factor type: BFlinearModel, JZS
```

In both cases, the predictor improves substantially over the smaller model according to the Bayes Factor test.

17.1.3 Exercise

Perform an ANOVA on the insect multi-way ANOVA OrchardSprays outcome variable 'decrease'. Use rowpos, colpos, and treatment as categorical predictors. Use a Tukey test to determine the relative effects of row, column, and treatment. Do a Bayesian ANOVA as well.

```
data(OrchardSprays)
```

17.2 Non-orthogonal predictors

Previously, we had examined how sets of orthogonal predictor variables can enter into a regression model without impacting the estimates provided by other predictors. This is important for the ANOVA because we are comparing sets of nested models, and so if the predictors are not orthogonal, removing them in different orders can matter. That is, multiple sub-models might differ by a single factor, and so it might matter which pair of models we compare.

As an example, suppose we had a DV of response time in a stimulus detection experiment, predicted by color, set size, and stimulus size. The model might be:

```
1
color <- c("r","r","r","r","r","r","r","r","r",
3          "g","g","g","g","g","g","g","g","g",
           "b","b","b","b","b","b","b","b","b")
5
setsize <- c(1,2,3,1,2,3,1,2,3, 1,2,3,1,2,3,1,2,3,1,2,3,1,2,3)
7 stimsize <- c(1,1,1,2,2,2,3,3,3,1,1,1,2,2,2,3,3,3,1,1,1,2,2,2,3,3,3)
  set.seed(100)
9 rt <- rnorm(27)

11 table(color,setsize,stimsize)
  > table(color,setsize,stimsize)
13 , , stimsize = 1

15 setsize
  color 1 2 3
17 b 1 1 1
  g 1 1 1
19 r 1 1 1

21 , , stimsize = 2

23 setsize
  color 1 2 3
25 b 1 1 1
  g 1 1 1
27 r 1 1 1
```

```
, , stimsize = 3

setsize
color 1 2 3
b 1 1 1
g 1 1 1
r 1 1 1



lm(rt~color+setsize+stimsize)

  Call:
  lm(formula = rt ~ color + setsize + stimsize)

  Coefficients:
  (Intercept)       colorg       colorr      setsize     stimsize
  0.24228     -0.02269     -0.06716      0.08401     -0.16156
```

The table shows there is an observation in each of the 3x3x3 cells of the design. Suppose we want to know whether set size has an impact. We could compare:

```
anova(lm(rt~color+setsize+stimsize),lm(rt~color+stimsize))

Analysis of Variance Table

Model 1: rt ~ color + setsize + stimsize
Model 2: rt ~ color + stimsize
  Res.Df    RSS Df Sum of Sq    F Pr(>F)
1     22 12.153
2     23 12.280 -1  -0.12704 0.23 0.6363
```

Or we could compare:

```
anova(lm(rt~setsize+stimsize),lm(rt~stimsize))
Analysis of Variance Table

Model 1: rt ~ setsize + stimsize
Model 2: rt ~ stimsize
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     24 12.174
2     25 12.302 -1  -0.12704 0.2504 0.6213
```

Or:

```
anova(lm(rt~color+setsize),lm(rt~color))

Analysis of Variance Table

Model 1: rt ~ color + setsize
Model 2: rt ~ color
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     23 12.623
2     24 12.750 -1  -0.12704 0.2315  0.635
```

or even:

```
> anova(lm(rt~setsize))
Analysis of Variance Table

Response: rt
          Df Sum Sq Mean Sq F value Pr(>F)
setsize    1  0.127 0.12704  0.2512 0.6206
Residuals 25 12.644 0.50577
```

These are three different ANOVA tests to assess whether set size has an effect. If your experiment used an orthogonal design (i.e., full factorial, or appropriately counterbalanced), these will all agree in terms of the change in the sum-square-error (here it is 0.127). The F values, however, will differ because the residual error differs. What is more, if you have an orthogonal design, it doesn't matter the order of the variables in the model:

```
anova(lm(rt~setsize+color))
anova(lm(rt~color+setsize))
anova(lm(rt~setsize+color))
Analysis of Variance Table

Response: rt
          Df Sum Sq Mean Sq F value Pr(>F)
setsize    1  0.127 0.12704  0.2315 0.6350
color      2  0.021 0.01051  0.0191 0.9811
Residuals 23 12.623 0.54883
> anova(lm(rt~color+setsize))
Analysis of Variance Table

Response: rt
          Df Sum Sq Mean Sq F value Pr(>F)
color      2  0.021 0.01051  0.0191 0.9811
setsize    1  0.127 0.12704  0.2315 0.6350
Residuals 23 12.623 0.54883
```

Here, all the F, P, SS, and other values are identical. This is not always true. If instead of a completely balanced design like we had, what if we had used a quasi-experimental design, or randomly-sampled conditions for each person, or for some other reason have unbalanced factors? In this case, these three comparisons could in fact differ!

When you are creating your own experiments, having a orthogonally balanced design makes things much easier. A complete factorial design will be balanced, provided you have equal numbers of observations in each group. A partial factorial design (such as a latin square) can also be orthogonal. But usually random assignment to experimental groups, or subject attrition, sampling problems will create an unbalanced design. What do you do when that happens?

Here is an example:

Create a fake data set where the outcome is related to the levels of two factors, but those factors are sampled and not chosen by design. We will use just two predictors for now–color and setsize:

```

set.seed(100)
color <- sample(c("r","g","b"),27,replace=T)
setsize <- sample(1:3,27,replace=T)
stimsize <- sample(1:3,27,replace=T)
```

```
rt <- rnorm(27)


table(color,setsize)
> table(color,setsize)
    setsize
color 1 2 3
  b 6 0 5
  g 3 3 6
  r 1 3 0
```

Here, we got especially unlucky because there are no cases of 'b' and '2', which will obviously cause problems. Now, let's create two linear models, with the predictor terms entered in different orders:

```
lmz1 <- lm(rt~color+setsize)
lmz2 <- lm(rt~setsize+color)
```

We can verify that they produce exactly the same regression model. This is as expected, because the estimate is done simultaneously for all predictors, regardless of the order the predictors are specified.

```
> lmz1$coefficients
  (Intercept)      colorg      colorr     setsize
  0.14930149  0.07585296 -0.34465411 -0.07424683

> lmz2$coefficients
  (Intercept)     setsize      colorg      colorr
  0.14930149 -0.07424683  0.07585296 -0.34465411
```

But now let's look at these in terms of the ANOVA tables. Remember, the ANOVA table compares each model term with respect to how its next higher model. Now, the order does matter.

```
> anova(lmz1)
Analysis of Variance Table

Response: rt
Df Sum Sq Mean Sq F value Pr(>F)
color      2  0.454 0.22716  0.1270 0.8814
setsize    1  0.110 0.10975  0.0613 0.8066
Residuals 23 41.149 1.78910


> anova(lmz2)
Analysis of Variance Table

Response: rt
Df Sum Sq Mean Sq F value Pr(>F)
setsize    1  0.045 0.04460  0.0249 0.8759
color      2  0.519 0.25974  0.1452 0.8657
Residuals 23 41.149 1.78910

---
```

The same thing happens if you go straight to the `aov()` function:

```
z.anova1 <- aov(rt ~ setsize + color)
z.anova2 <- aov(rt ~ color+setsize)

> summary(z.anova1)
Df Sum Sq Mean Sq F value Pr(>F)
setsize      1   0.04  0.0446   0.025  0.876
color        2   0.52  0.2597   0.145  0.866
Residuals   23  41.15  1.7891

> summary(z.anova2)
Df Sum Sq Mean Sq F value Pr(>F)
color        2   0.45  0.2272   0.127  0.881
setsize      1   0.11  0.1098   0.061  0.807
Residuals   23  41.15  1.7891
```

The results look familiar to the earlier ANOVA on the balanced design (except the data are re-sampled). Yet if you look carefully, we see something unexpected. Now, what the `aov` is doing is running an `lm` and feeding it to the `anova` table. Each row of the table is compared against its predecessor, so the first is compared to the intercept-only model (thus a difference of 2/3 d.f.), and the next is compared to the next one, and so on. Because we have non-orthogonal predictors, the order of model comparison impacts the estimates, because it impacts the variables that happen to be in the model.

We should be able to build the `anova` test by hand to get a clearer understanding. To do that, I need to create the 'lattice' of models. At the top, we have the intercept-only model, then we have two ways of adding on factor, and finally the full model at the bottom.

```
lm0 <- lm(rt~1)
lmcolor <-lm(rt~color)
lmsetsize <- lm(rt~setsize)
lmcolorsetsize <- lm(rt~color+setsize)
lmsetsizecolor <- lm(rt~setsize+color)
```

If we look at ANOVAs to compare models, we get different SSEs for each comparison that should produce the same difference:

```
                                    SS
anova(lm0,lmcolor)                  # .454
anova(lmsetsizecolor,lmsetsize)     # -.519


anova(lm0,lmsetsize)               # .044
anova(lmcolor,lmsetsizecolor)      # .10975
```

If we look at the ANOVA table for the largest models, we can trace the lines back to particular comparisons between models.

```


anova(lmsetsizecolor)
Analysis of Variance Table


Response: rt
```

```
8             Df Sum Sq Mean Sq F value Pr(>F)
setsize        1  0.045 0.04460  0.0249 0.8759
10 color       2  0.519 0.25974  0.1452 0.8657
Residuals 23 41.149 1.78910
12

14 anova(lmzcolorsetsize)
Analysis of Variance Table
16
Response: rt
18            Df Sum Sq Mean Sq F value Pr(>F)
color          2  0.454 0.22716  0.1270 0.8814
20 setsize     1  0.110 0.10975  0.0613 0.8066
Residuals 23 41.149 1.78910
```

You should be able to trace where each F value comes from in the built-in ANOVA by looking at the specific model pairs it tests.

17.2.1 What do you do?

This is a problem that cannot just be swept under the rug. The logic of the ANOVA test requires comparing nested models, and when sets of predictors are not orthogonal, those predictors' coefficients will change and will depend on each other, and thus the order you do the comparison. The type of ANOVA reported by R by default has been called the Type I ANOVA; a term that appears to have arisen from SAS and been adopted by SPSS. More on this in a little bit.

The lesson from this is that ANOVA was really designed and created for handling true experiments with good designs, random assignment to groups, and equal numbers of observations in each group. This is something you might be able to acheive in botany or when studying rats, but often human research creates situations that violate this. We can still manage when we do not have a perfect design, but it involves compromises regardless of what we do.

This problem becomes even a greater challenge when you have more than two factor-based predictors, because the model lattice becomes much more complex. For example, What if x2 is only reliable if it enters after x1, but before x3?

Luckily, It is often not as bad as it might seem it could be. First, remember that this will only matter when your design is not orthogonal. Many times you can avoid this by simply designing or sampling equally.

Because you cannot always do that, and a reasonable strategy would be to use the different models as a way to make your argument stronger. Comparing sets of models demonstrates statistical control–that a variable has an effect even when other predictors are accounted for. For the above comparison, color never makes a reliable difference to the model fit, regardless of whether you compare it first to the intercept or compare it after you let the other variable setsize have its say.

In fact, oftentimes the order in which the predictors should be tested will have some logical sense, as it will map onto your theory. For example, suppose it has been well established that gender impacts spatial skills (this has been established by research, but the reasons for it are the subject of controversy), but nobody has considered whether participation in sports is also predictive. A nice default way to present this is to first assess whether the gender effect is reliably different from the intercept-only model, and then determine if sports participation can still account for variability after that. Next, if you fit the model the opposite way, it

would be powerful if you were able to show that not only does the sports activity predict after gender is taken into account, but gender has no effect if sports activity is first given its shot. So the order of a Type I ANOVA can be a useful rhetorical/argumentation device.

This example suggests that, because order matters in the standard "Type I" ANOVA, we should be deliberate about how order of variables are used. Furthermore, a reasonable way to handle this is to examine the impact of a factor only after all other factors have had their chance. This is what the Type II ANOVA does.

17.3 Type I, II, and III ANOVA tests

The Type I ANOVA table is unsatisfying because it depends on how you write your model. Consequently, there are several possible answers. The nice thing about the "Type I" ANOVA is that it divides your variance into mutually exclusive bins, allowing you to figure out directly how much each predictor is accounting for. Notice that for any individual model, the sum square error add up to the same value:

```
> 0.1705 + 5.8233+18.0159
[1] 24.0097
> 5.748+0.2457+18.0159
[1] 24.0096
>
```

This is nice because you are truly dividing your variance into the things you know about and things you don't; the problem is that it can produce different results if you chose to fit the model one way versus the other. This has created substantial debate within the applied statistics community, and is especially important when dealing with interactions in the face of an unbalanced design (one with different numbers of observations in each cell).

In many cases, you can do an ANOVA test comparing your two specific models of interest, rather than relying on a pre-packaged ANOVA table. Oftentimes, the order in which predictors are included or removed from a model is important from a theoretical perspective, and if so, you should rely on testing those hypotheses directly.

Nevertheless, sometimes our default hypothesis involves testing the impact of a predictor, after all other variables have been entered into the model. This can make it harder to find a reliable impact if a variable is correlated with others; it can also improve the impact, if two variables together can account for variance that neither one alone does. This basic notion is encapsulated in the so-called "Type II" and "Type III" ANOVA models provided by default by software like SAS and SPSS. When using this type of ANOVA, we usually let go of our goal of making our table account for all the pieces of variance. Instead, we reports specific F tests associated with reasonable hypotheses. And of course, in a completely balanced design, it will not matter because all of the tests will be the same.

We can build get these ANOVA tables by hand from the Type-I ANOVAs:

```
>anova(lm(rt~color + setsize))
 Analysis of Variance Table

 Response: rt
 Df Sum Sq Mean Sq F value Pr(>F)
 color      2  0.454 0.22716  0.1270 0.8814
 setsize    1  0.110 0.10975  0.0613 0.8066
 Residuals 23 41.149 1.78910

```

```
> anova(lm(rt~setsize+color))
 Response: rt
 Df Sum Sq Mean Sq F value Pr(>F)
 setsize    1  0.045 0.04460  0.0249 0.8759
 color      2  0.519 0.25974  0.1452 0.8657
 Residuals 23 41.149 1.78910
```

By taking just the last line of each, we can make our Type II/III ANOVA table:

```
Proposed ANOVA table
Response: z
          Df Sum Sq Mean Sq F value    Pr(>F)
color      2  0.519 0.25974  0.1452 0.8657
setsize    1  0.110 0.10975  0.0613 0.8066
Residuals 23 41.149 1.78910
```

Notice how the Residuals are the same, but the SS no longer adds up to the same thing:

```
> .045 + .519 + 41.149
[1] 41.713

> .519 + .110 +41.149
[1] 41.778
```

This kind of model is what is known by SAS and SPSS as the Type II and/or Type III ANOVA. When you have no interaction terms or higher-order polynomials, the two types are identical.

The `car` library is a nice R package that is linked to a companion to Fox's "Companion to Applied Regression in R"–a textbook that covers much of the same material as the current course. It contains the special-purpose `Anova` function that will estimate Type II and Type III ANOVAs for you.

```
library(car)

> Anova(lmcolorsetsize)
Anova Table (Type II tests)

Response: rt
Sum Sq Df F value Pr(>F)
color      0.519  2  0.1452 0.8657
setsize    0.110  1  0.0613 0.8066
Residuals 41.149 23


> Anova(lmsetsizecolor)
Anova Table (Type II tests)

Response: rt
Sum Sq Df F value Pr(>F)
setsize    0.110  1  0.0613 0.8066
color      0.519  2  0.1452 0.8657
Residuals 41.149 23
```

Notice how these two results are the same, regardless of order. This is because they use just the two ANOVA model comparisons produced by leaving a single factor out at a time, and combine them for display. We will cover the differences between Type II and Type III after a section on specifying interactions in a later chapter.

Finally, an admonishment: when using an ANOVA, and there are more than two predictor factors, be sure to report: 1. whether the predictors were orthogonal and balanced (if they were, none of this matters); 2. If not and you are using the base "Type I" ANOVA, discuss how the order matters, and use this as a way to help illustrate your argument; 3. If not and you are using a Type II/Type III ANOVA, state this explicitly. This is true regardless of whether you perform your ANOVA in R or SPSS or whatever.

17.4 ANOVA Model Lattice

Just as with the variable selection problem in regression, the ANOVA model is concerned with a lattice of models. To understand this lattice, we will construct all possible models including the predictors a, b, and c, and compute pairwise ANOVA tests between them.

17.5 The Model Lattice and ANOVA Types

In the previous section, we discussed how ANOVA tests differences between models, and when your predictors are not orthogonal, the order of the models matters. We will start with an exercise that will help us build a model lattice for a data set whose predictors are not orthogonal:

The following random data set creates a 2x2 design which, because we use the set.seed() function, should create the same values for all users.

```
1
set.seed(1000)  #Everyone should get the same numbers.
3 ##  Build the full model lattice for the set of models with three predictors

5 a <- factor(sample(c("I","II"),replace=T,50))
b <- factor(sample(c("i","ii"),replace=T,50))
7 c <- factor(sample(c("a","b","c"),replace=T,50))
y <- c(.3,-5)[a] - c(50,35)[b] + c(-.35,1.5,-5)[c] + rnorm(50)
```

Remember that the `anova()` function creates a table which iteratively removes terms from the model. When you have a non-orthogonal design, the order you remove things matters. Compare the results of these different commands

```
1 anova(lm(y~a+b+c))

3 anova(lm(y~a+c+b))
anova(lm(y~b+a+c))
5 anova(lm(y~b+c+a))
anova(lm(y~c+a+b))
7 anova(lm(y~c+b+a))
```

First, let's look at the first command.

```
1 > anova(lm(y~a+b+c))
Analysis of Variance Table
```

```
Response: y
Df  Sum Sq Mean Sq F value    Pr(>F)
a          1  458.88  458.88  718.00 < 2.2e-16 ***
b          1 2836.58 2836.58 4438.33 < 2.2e-16 ***
c          2  433.42  216.71  339.08 < 2.2e-16 ***
Residuals 45   28.76    0.64
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Where do these values come from? We could compute each by hand, by comparing models explicitly:

```
anova(lm(y~a),lm(y~1))
anova(lm(y~a+b),lm(y~a))
anova(lm(y~a+b+c),lm(y~a+b))

> anova(lm(y~a),lm(y~1))
Analysis of Variance Table

Model 1: y ~ a
Model 2: y ~ 1
Res.Df    RSS Df Sum of Sq      F  Pr(>F)
1     48 3298.8
2     49 3757.6 -1   -458.88 6.6771 0.01286 *

> anova(lm(y~a+b),lm(y~a))
Analysis of Variance Table

Model 1: y ~ a + b
Model 2: y ~ a
Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     47  462.2
2     48 3298.8 -1   -2836.6 288.46 < 2.2e-16 ***

---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> anova(lm(y~a+b+c),lm(y~a+b))
Analysis of Variance Table

Model 1: y ~ a + b + c
Model 2: y ~ a + b
Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     45  28.76
2     47 462.18 -2   -433.42 339.08 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

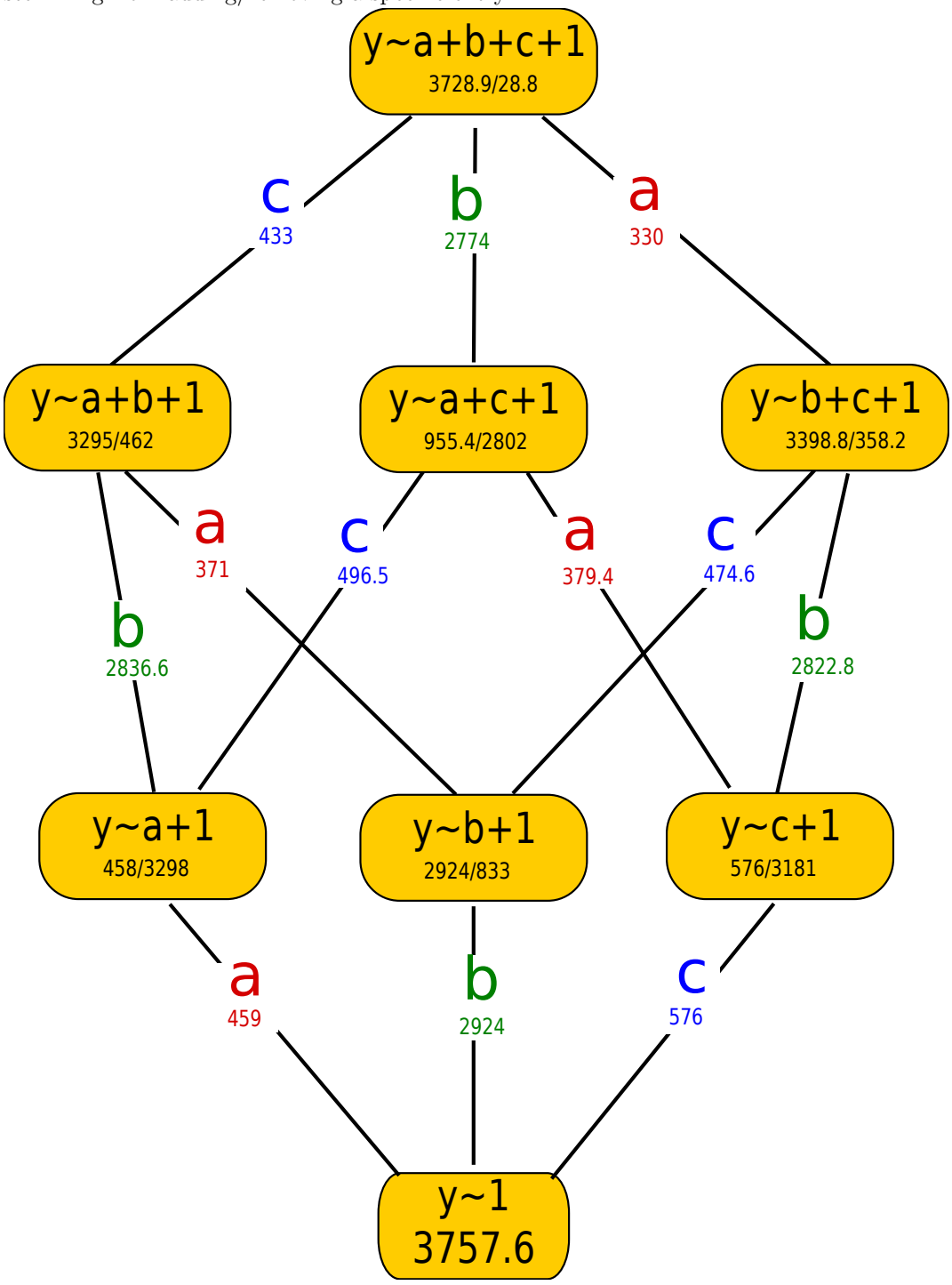
Notice that at each step, the Sum Sq is the difference between two models. You could think of this as either adding the term to the smaller model, subtracting it from the larger model; it doesn't matter.

What should be clear is that the 'standard' Type-I ANOVA follows the path up from the intercept-only model, entering the first term, then the second term, finally the third term. You can take any of the ANOVA's shown earlier, and follow them up through the lattice until you get to the first model.

We can see that because the levels of a,b,and c are sampled from the population and not chosen by design, they are not orthogonal. Depending on your design, you could potentially sample equally from each cell to avoid the unbalanced problem. Remember that if your predictors are not orthogonal, then their estimates will depend on one another, and so if you two models where you remove a factor from the larger model, the estimate of the remaining predictor will change.

As we can see in Figure 17.2, order does matter. For example, there are four different model pairs who differ by only a. And these four times, a accounts for either 31.6, 342, 25.7, or 307 Sum of Squares! Which one is the right one?

Figure 17.2: Model lattice for sample problem. Each model shows the total SS it accounts for, relative to the residual SS it cannot account for. Each link shows the relative change stemming from adding/removing a specific entry.



Clearly, if you had just done `aov(y~a+b+c)`, this might not be the right statistic to report, especially if order is sort of arbitrary. One alternative uses a conservative criterion–report the SS that a factor can account for AFTER all other factors have done their work. In this case, it would be the top row of the lattice. This is called "Type II" or "Type III" ANOVA, which is what SAS and SPSS report by default. You can get it by building the lattice by hand, or by using the `car` package ANOVA function, or by looking at the full model and comparing it to each model that is one smaller, using drop1:

```
> drop1(lm(y~a+b+c))
Single term deletions

Model:
y ~ a + b + c
Df Sum of Sq     RSS     AIC
<none>                28.76 -17.652
a        1    330.06  358.82 106.539
b        1   2773.47 2802.23 209.307
c        2    433.42  462.18 117.197
```

Looking at the lattice again, we can see the big picture that Type I and Type II/III miss. Each predictor alone counts for more variability than when it is entered after any others. This suggests they are somewhat redundant. In this case, all of the SSEs for each predictor are about the same, but this does not always happen. If they change substantially for different links in the lattice, this can help you understand the relationships between variables.

17.6 Solutions to exercises

17.6.1 Orchard spray ANOVA

```

#this isn't exactly what we wanted:
model1 <-aov(decrease~rowpos+colpos+treatment,data= OrchardSprays)
model1


model <- aov(decrease~as.factor(rowpos)+as.factor(colpos)+treatment,data=
    OrchardSprays)
model
summary(model)

TukeyHSD(model)
library(agricolae)
HSD.test(model,trt="treatment",group=T,console=T)

> HSD.test(model,trt="treatment",group=T,console=T)

Study: model ~ "treatment"

HSD Test for decrease
```

```
Mean Square Error:  380.8311

treatment,  means

decrease       std r Min Max
A     4.625  3.204350 8   2  12
B     7.625  3.292307 8   4  14
C    25.250 24.429198 8   9  84
D    35.000 13.437687 8  20  57
E    63.125 26.909571 8  39 114
F    69.000 29.189039 8  20 114
G    68.500 20.142351 8  24  92
H    90.250 24.223660 8  69 130

alpha: 0.05 ; Df Error: 42
Critical Value of Studentized Range: 4.509098

Honestly Significant Difference: 31.11078

Means with the same letter are not significantly different.

Groups, Treatments and means
a     H    90.25
a     F    69
a     G    68.5
ab    E    63.12
bc    D    35
c     C    25.25
c     B    7.625
c     A    4.625
> HSD.test(model,trt="treatment",group=T,console=T)
```

Chapter 18

Factorial ANOVA: Main effects and interactions

18.1 Interactions Between Factors in a balanced ANOVA model

So far, we have looked at *main effects* within ANOVA and regression models. Earlier, we examined how the product of two continuous variables could be used to make an orthogonal predictor that amounts to a curved surface in a regression space. Within categorical predictions, this product is called an interaction.

An interaction simple allows that the impact of one factor depends on the level of a second factor. It should make intuitive sense–just like a drug interaction causes an unexpected result that differs from either of the main effects of two drugs, a statistical interaction does the same.

We have also used interactions previously to fit a different slope or intercept for each level of a condition. But now let's consider interactions of pure categorical variables. The toothgrowth data set we examined previously is a good example. From the mean values in each condition, it looks like the impact of supplement changes for large versus small doses:

```
 tapply(ToothGrowth$len,list(ToothGrowth$supp,ToothGrowth$dose),mean)
     0.5     1     2
OJ 13.23 22.70 26.06
VC  7.98 16.77 26.14
```

The general question we want to ask here is whether there is an interaction between dose and supplement. We can include this in a regression or ANOVA model using the ':' or '*' symbols. The '*' symbol includes both main effects and the interaction, but the ':' includes just the interaction. Lets use the ANOVA here to look at categorical effects (and be sure to convert dose to a factor).

```
 tg1 <- aov(len~supp+as.factor(dose),data=ToothGrowth)
 tg2 <- aov(len~supp*as.factor(dose),data=ToothGrowth)
 tg3 <- aov(len~supp:as.factor(dose),data=ToothGrowth)

> summary(tg1)
                 Df Sum Sq Mean Sq F value   Pr(>F)
supp              1  205.4   205.4   14.02 0.000429 ***
```

```
as.factor(dose)  2 2426.4  1213.2   82.81  < 2e-16 ***
Residuals       56  820.4    14.7
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> summary(tg2)
                     Df Sum Sq Mean Sq F value   Pr(>F)
supp                  1  205.4   205.4  15.572 0.000231 ***
as.factor(dose)       2 2426.4  1213.2  92.000  < 2e-16 ***
supp:as.factor(dose)  2  108.3    54.2   4.107 0.021860 *
Residuals            54  712.1    13.2
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> summary(tg3)
                     Df Sum Sq Mean Sq F value Pr(>F)
supp:as.factor(dose)  5 2740.1   548.0   41.56 <2e-16 ***
Residuals            54  712.1    13.2
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

anova(tg2,tg1)
Analysis of Variance Table

Model 1: len ~ supp * as.factor(dose)
Model 2: len ~ supp + as.factor(dose)
  Res.Df    RSS Df Sum of Sq     F  Pr(>F)
1     54 712.11
2     56 820.43 -2   -108.32 4.107 0.02186 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Notice several things. First of all, the baseline model found both supplement and dose were significant. When their interaction was added, it was also significant. Next, see that supp:dose has two degrees of freedom associated with it. The DF of an interaction is the product of the DFs of the main effects, and so because 1x2=2, this 2x3 interaction has 2 df. Finally, compare tg2 to tg3. They account for exactly the same amount of variability, but in this case, the interaction term has 5 degrees of freedom. We can maybe learn a bit more about this by doing the lm versions to examine each fitted beta weight:

```
> summary(lm(len~supp*as.factor(dose),data=ToothGrowth))

Call:
lm(formula = len ~ supp * as.factor(dose), data = ToothGrowth)

Residuals:
    Min      1Q  Median      3Q     Max
 -8.20   -2.72   -0.27    2.65    8.27

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                 13.230      1.148  11.521 3.60e-16 ***
suppVC                      -5.250      1.624  -3.233  0.00209 **
as.factor(dose)1             9.470      1.624   5.831 3.18e-07 ***
as.factor(dose)2            12.830      1.624   7.900 1.43e-10 ***
suppVC:as.factor(dose)1     -0.680      2.297  -0.296  0.76831
suppVC:as.factor(dose)2      5.330      2.297   2.321  0.02411 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 3.631 on 54 degrees of freedom
Multiple R-squared:  0.7937,  Adjusted R-squared:  0.7746
F-statistic: 41.56 on 5 and 54 DF,  p-value: < 2.2e-16



summary(lm(len~supp:as.factor(dose),data=ToothGrowth))

Call:
lm(formula = len ~ supp:as.factor(dose), data = ToothGrowth)

Residuals:
    Min      1Q  Median      3Q     Max
 -8.20   -2.72   -0.27    2.65    8.27

Coefficients: (1 not defined because of singularities)
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                 26.140      1.148  22.763  < 2e-16 ***
suppOJ:as.factor(dose)0.5  -12.910      1.624  -7.949 1.19e-10 ***
suppVC:as.factor(dose)0.5  -18.160      1.624 -11.182 1.13e-15 ***
suppOJ:as.factor(dose)1     -3.440      1.624  -2.118   0.0388 *
suppVC:as.factor(dose)1     -9.370      1.624  -5.770 3.98e-07 ***
suppOJ:as.factor(dose)2     -0.080      1.624  -0.049   0.9609
suppVC:as.factor(dose)2         NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.631 on 54 degrees of freedom
Multiple R-squared:  0.7937,  Adjusted R-squared:  0.7746
F-statistic: 41.56 on 5 and 54 DF,  p-value: < 2.2e-16
```

Notice that it carves the effects up into slightly different bins, and none of the numbers are the same, but overall it accounts for all the levels of one factor crossed with all the levels of the other. This is because we lose some degrees of freedom estimating the means of the two main effects, and so the interaction is then estimated with respect to those means. Without estimating those first, we just compare each pairing of levels, and estimate the values. Then, to determine whether the interaction is significant, ANOVA compares a model containing those terms to a model that does not contain those terms.

18.1.1 Exercise

Verify that the two models' estimates for suppVC:dose1 and suppVC:dose2 are identical.

The fact that these two produce the same estimates makes sense when we look at the 'table of effects' produced by the `model.tables()` function. In each case, this ignores the intercept (which was 18.8), but then codes effects related to the different predictors:

```
> model.tables(tg2)
Tables of effects

 supp
supp
   OJ    VC
 1.85 -1.85
```

```
 as.factor(dose)
as.factor(dose)
   0.5      1      2
-8.208  0.922  7.287

 supp:as.factor(dose)
    as.factor(dose)
supp 0.5    1      2
  OJ  0.775  1.115 -1.890
  VC -0.775 -1.115  1.890

> model.tables(tg3)
Tables of effects

 supp:as.factor(dose)
    as.factor(dose)
supp 0.5     1       2
  OJ  -5.583   3.887   7.247
  VC -10.833  -2.043   7.327
```

If we use the 'means' argument, it produces the estimated at each level of prediction, and you can see they are the same. Thus, the sort of have to have the same degrees of freedom.

```
> model.tables(tg2,type="means")
Tables of means
Grand mean

18.81333

 supp
supp
    OJ     VC
20.663 16.963

 as.factor(dose)
as.factor(dose)
   0.5      1      2
10.605 19.735 26.100

 supp:as.factor(dose)
    as.factor(dose)
supp 0.5   1     2
  OJ 13.23 22.70 26.06
  VC  7.98 16.77 26.14


> model.tables(tg3,type="means")
Tables of means
Grand mean

18.81333

 supp:as.factor(dose)
    as.factor(dose)
supp 0.5   1     2
  OJ 13.23 22.70 26.06
  VC  7.98 16.77 26.14
```

So, in this case, adding an interaction term will let us model all six points exactly. We add to the base effects differential effects of each level, and then can recompute the exact 6 means precisely. Then, the question we want to ask, statistically, is whether this more complex model accounts for sufficiently more data. With a NHST, when testing the interaction, we are asking whether the difference from the main-effect model would have occurred by chance if no true interaction existed.

18.2 The model lattice with interactions

Understanding the model lattice is important for standard ANOVA, but even more critical when we start to deal with interactions. Suppose that a predictor we call `c` were an interaction term (we'd have to recreate the term in order to do this).

In that case, the Type II/III ANOVA logic we discussed in the previous chapter makes a strange assumption. It shows the impact that the *a* predictor has after the a:b interaction (called c) has entered the model, and similarly for *b*, it shows the effect of b in comparison to the model with a and a:b. This violates the principle of marginality.[1].

That is, if we ignore that C is an interaction term and just treat it as another variable, to test the effect of A and B we are comparing a model A+C to A+B+C, or B+C to A+B+C. This violates the principle 'marginality' because C is composed of B and A already. The principle of marginality is that the main effects are marginal to the interactions–they are the remaining effects after the interaction has been accounted for, and it is potentially incorrect or uninformative to try to test the main effects when interactions exist.

The basic advice frequently given is as follows, with respect to which pool of the sum-of-squares to use (i.e., which type of ANOVA) when you have unbalanced sampling: [2].

- Type I: each variable is used consecutively, and this usually is not what we want, especially if our sampling is unbalanced.
- Type II: To evaluate A, use SS(A given B); consider A after B is accounted for. This should only be used if no interaction term is found, because this logic assumes there are no interactions between A and B.
- Type III: To evaluate A, use SS(A given B & AB); consider A after the other main effect and interaction are determined, and this should be used in the presence of significant interactions. **BUT...** if the interaction is significant, the main effects are often not interesting or interpretable. At the very most, you can identify that an average effect still exists once the interaction is accounted for, but the interpretation depends on the kind of interaction.

In addition, this means the estimates will depend on how the factor's contrasts are created if the sampling we achieved is not balanced–so we need to be careful to use orthogonal contrasts. Using Helmert or polynomial contrasts will ensure this is done properly.

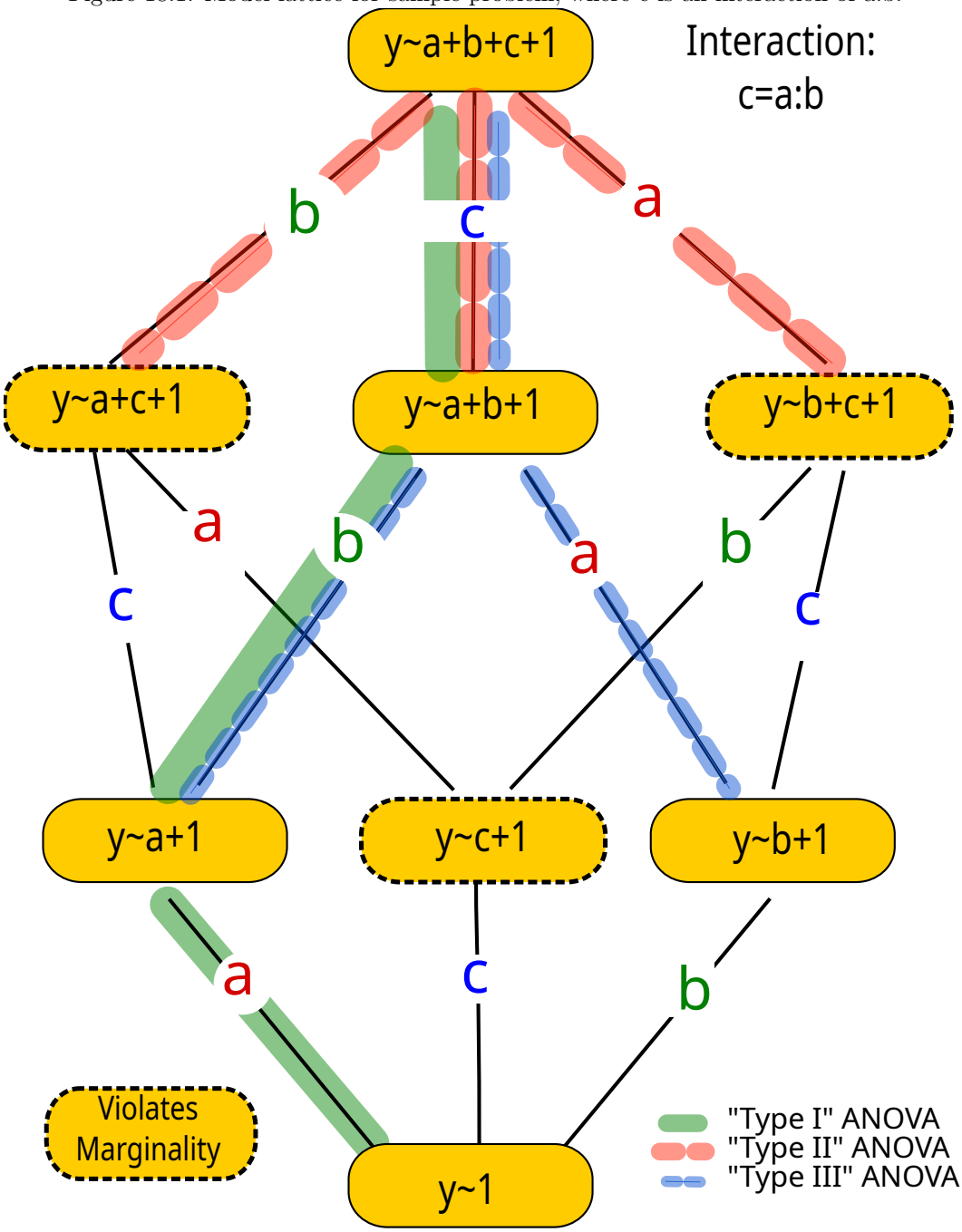
It may seem strange to test the main effect in comparison to a model with an interaction of that effect. The Type-II ANOVA uses a different approach, looking elsewhere in the lattice. It tests the main effects by comparing the model with all main effects to one without the predictor of interest, and then tests the interactions in comparison to the model with all

[1] See W. N. Venables (1998), *An exegesis on Linear Models* http://www.stats.ox.ac.uk/pub/MASS3/Exegeses.pdf; and also see Fox's message to R mailing list on Mar-2-2010 "[R] Type-I v/s Type-III Sum-Of-Squares in ANOVA" and other discourses such as https://www.r-bloggers.com/2011/03/anova-type-iiiiii-ss-explained/

[2] https://www.r-bloggers.com/2011/03/anova-type-iiiiii-ss-explained/

main effects. This can be seen as the small-hashed blue highlighted path through the lattice in Figure 18.1.

Figure 18.1: Model lattice for sample problem, where c is an interaction of a:b.



Overall, it is more important to test your hypotheses than to worry about these particular default tests. Recognize that none of the three tests will always automatically test the comparison between models you are interested in, and then use specific ANOVA models to test your specific hypotheses. However, Fox recommends using Type-II as a default, in part because it is does not violate marginality (and so you don't have to worry about contrasts), and in part because it is reasonably intuitive.

When you have an interaction, the main effects become ambiguous. One approach is to simply ignore the main effects, and either interpret the interaction on its own, or break it apart and look at each level separately. However, sometimes there is still a reasonable interpretation of the main effect–you might mean the marginal means–the average effect across all levels of the other predictor(s). This gets tricky if you have an unbalanced design–different numbers of observations in each cell, because the SSE will overweight some groups. Thus, a reasonable test of a main effect in this context might to examine the effect of A after accounting for both B AND AB. this is called a "Type-III" ANOVA, and is also available as an option in Fox's Anova in the car package. It needs to be used with care though, because since marginality is violated, the contrasts used to represent your factors use could matter. In these cases, you should code your factor variable with contrasts that are balanced and orthogonal, to ensure you get the same estimates regardless of order. The `contr.poly`, `contr.sum` and `contr.helmert` arguments will do this, (but the default treatment contrasts do not). Thus, you should either change the default contrast or hand-code the contrasts.

So, to summarize:

- The Type-I anova which is given directly be the R `aov` command is rarely exactly what you want, especially in the context of interactions. But it calculates and correctly divides up sums-of-squares in exactly the order you specify, and so could be used to calculate the entire model lattice.
- The Type-II ANOVA is a reasonable default when you have no interactions, and should only be used if the interaction is not significant.
- The Type-III is also intuitive, but violates marginality, and so should only be used when you specify orthogonal contrasts.
- If you have a balanced sample and orthogonal design, all of these are identical.

18.2.1 Dealing with Interactions: Worked Example

In this study, participants played the first player in the 'Ultimatum' game. Two players play for up to 10 points. The first player decides how to divide up the points between him or herself and the other player They can choose any whole number value from 0 to 10, either keeping it all or giving it all to the other player. The second player can only decide whether to accept the offer (in which case the two players keep whatever money or points were specified by player 1), or reject it, (in which case neither player gets anything). A rational first player would minimize the amount given to the other player, and a rational second player would accept any positive offer, because it is better than the 0 payoff otherwise obtained. In reality, people tend to reject offers that are not equitable, and tend to offer around 50% of the pot to the other player.

In this data set, based on a real experiment, we select participants who self-identified as either men or women to play the first player of the game (the offeror). At the outset, participants were randomly assigned to one of three conditions: (1) no information about the opponent (2) background information about opponent who is female; (3) background

information about opponent who is male. The background in the two cases, and the name, is identical–only the picture indicating gender is changed.

The questions of interest are whether the background information has an effect, whether the gender of the opponent has an effect, and whether the gender of the opponent and gender of player interact.

here is a (simulated) sample data set:

```
set.seed(100)
ultim1 <- data.frame(cond =rep(rep(c("N","F","M"),2),100),
                        gender= rep(rep(c("F","M"),each=3),100),
                        offer = rep(c(5.5, 6.5,  6.5,
                                      5, 6.1, 6.0),100)+rnorm(600))


table(ultim1$cond,ultim1$gender)

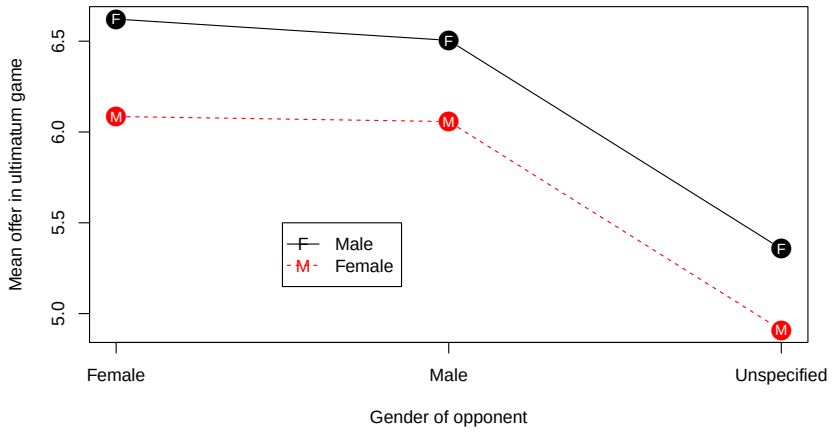
      F   M
  F 100 100
  M 100 100
  N 100 100
```

Note that the cells are balanced, which makes as a bit less concerned about the particular contrasts we are using.

```
matplot(tapply(ultim1$offer,list(ultim1$cond,ultim1$gender),mean),
        type="o",xaxt="n",pch=c("F","M"),ylab="Mean offer in ultimatum game",
        xlab="Gender of opponent")

axis(1,1:3,c("Female","Male","Unspecified"))
legend(2.5,6,c("Male","Female"),pch=c("F","M"),col=1:2,lty=1:2)
```

Figure 18.2: Simulated data from the ultimatum game. Here, men and women behave about the same with respect to the gender of each partner



Let's build a model and try to test these hypotheses. We would usually only test the post-hoc after we examine the model, but in this case we will look first to see what we might expect

```
library(car)
model1 <- aov(offer~cond*gender,data=ultim1)
model1b <- aov(offer~cond+gender,data=ultim1)

TukeyHSD(model1)

TukeyHSD(model1)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = offer ~ cond * gender, data = ultim1)

$cond
            diff        lwr        upr     p adj
M-F -0.07235367 -0.3102928  0.1655855 0.7550238
N-F -1.21990466 -1.4578438 -0.9819655 0.0000000
N-M -1.14755099 -1.3854901 -0.9096118 0.0000000

$gender
           diff        lwr        upr p adj
M-F -0.4771212 -0.6395116 -0.3147309     0

$`cond:gender`
                diff        lwr          upr      p adj
M:F-F:F -0.11701910 -0.5264759  0.292437727 0.9643759
N:F-F:F -1.26425527 -1.6737121 -0.854798446 0.0000000
F:M-F:F -0.53646527 -0.9459221 -0.127008442 0.0027012
M:M-F:F -0.56415352 -0.9736103 -0.154696692 0.0012792
N:M-F:F -1.71201933 -2.1214762 -1.302562501 0.0000000
N:F-M:F -1.14723617 -1.5566930 -0.737779348 0.0000000
F:M-M:F -0.41944617 -0.8289030 -0.009989344 0.0410458
M:M-M:F -0.44713442 -0.8565912 -0.037677594 0.0230551
N:M-M:F -1.59500023 -2.0044571 -1.185543402 0.0000000
F:M-N:F  0.72779000  0.3183332  1.137246829 0.0000074
M:M-N:F  0.70010175  0.2906449  1.109558579 0.0000193
N:M-N:F -0.44776405 -0.8572209 -0.038307230 0.0227433
M:M-F:M -0.02768825 -0.4371451  0.381768576 0.9999627
N:M-F:M -1.17555406 -1.5850109 -0.766097233 0.0000000
N:M-M:M -1.14786581 -1.5573226 -0.738408983 0.0000000
```

Notice that the Tukey test shows many different specific comparisons between levels of the interaction.

By default, we should be examining the whole test. The `Anova` function will default to a Type-II:

```
Anova(model1)
> Anova(model1)
Anova Table (Type II tests)

Response: offer
            Sum Sq  Df F value    Pr(>F)
cond        187.35   2 91.3448 < 2.2e-16 ***
gender       34.15   1 33.2970 1.274e-08 ***
cond:gender   0.26   2  0.1288    0.8792
Residuals   609.16 594
---
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Here, both the model with gender is compared to gender+condition to give use the SS for cond; the model with cond is compared to the model with gender + cond to give us the test for gender, and the model with gender+cond is compared to the model with gender + cond + gender:cond to give us the interaction. Here, we see that the interaction is not significant, but the other two are.

But maybe we'd prefer to test the effect of gender in a Type-III ANOVA. We can't just use `type=="III"`, we need to use orthogonal contrasts, as we see below:

```
Anova(model1,type="III")  ##incorrect
Anova Table (Type III tests)

Response: offer
             Sum Sq  Df   F value    Pr(>F)
(Intercept) 4384.6   1 4275.5231 < 2.2e-16 ***
cond          97.6   2   47.5887 < 2.2e-16 ***
gender        14.4   1   14.0317 0.0001973 ***
cond:gender    0.3   2    0.1288 0.8791895
Residuals    609.2 594
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

contrasts(ultim1$cond) <- contr.poly(3)
contrasts(ultim1$gender) <- contr.poly(2)
model1 <- aov(offer~cond*gender,data=ultim1)

Anova Table (Type III tests)

Response: offer
              Sum Sq  Df    F value    Pr(>F)
(Intercept) 21046.8   1 20523.0746 < 2.2e-16 ***
cond          187.4   2    91.3448 < 2.2e-16 ***
gender         34.1   1    33.2970 1.274e-08 ***
cond:gender     0.3   2     0.1288    0.8792
Residuals     609.2 594
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Notice that in this case, the type-II and type-III make no difference, nor does the contrasts we happen to use. This is because we have a balanced design–so if you have a balanced design (the same number of each cell), you can get away with a lot. But using the wrong contrasts in a type-III will give different F values for the main effects.

```

contrasts(ultim1$cond) <- contr.treatment(levels(ultim1$cond))
contrasts(ultim1$gender) <- contr.treatment(levels(ultim1$gender))
```

Resample the data so it is no longer balanced

```
ultim1b <-ultim1[sample(1:nrow(ultim1),size=nrow(ultim1),replace=T),]
table(ultim1b$cond,ultim1b$gender)

     F   M
  F  94 106
```

```
  M  93 103
  N 101 103
>
```

```
model1b <- aov(offer~cond*gender,data=ultim1b)
Anova(model1b,type="II")
Anova Table (Type II tests)

Response: offer
            Sum Sq  Df F value    Pr(>F)
cond        153.22   2 75.2343 < 2.2e-16 ***
gender       33.90   1 33.2874  1.28e-08 ***
cond:gender   0.10   2  0.0501    0.9511
Residuals   604.85 594
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Anova(model1b,type="III") ##not right
Anova Table (Type III tests)

Response: offer
            Sum Sq  Df   F value    Pr(>F)
(Intercept) 4073.6   1 4000.5255 < 2.2e-16 ***
cond          76.5   2   37.5816 4.281e-16 ***
gender        10.5   1   10.3190  0.001388 **
cond:gender    0.1   2    0.0501  0.951144
Residuals    604.9 594
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

contrasts(ultim1b$cond) <- contr.poly(3)
contrasts(ultim1b$gender) <- contr.poly(2)
model1c <- aov(offer~cond*gender,data=ultim1b)
Anova(model1c,type="III")
Anova Table (Type III tests)

Response: offer
             Sum Sq  Df    F value    Pr(>F)
(Intercept) 21188.8   1 20808.6181 < 2.2e-16 ***
cond          153.2   2    75.2122 < 2.2e-16 ***
gender         33.9   1    33.3322 1.253e-08 ***
cond:gender     0.1   2     0.0501    0.9511
Residuals     604.9 594
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
```

So, when we have an unbalanced design, we need to be careful about (1) choosing II vs. III, and (2) if we choose Type-III, the type of contrasts we use. In this case, the results all align, there are main effects of participant gender and condition, but no interaction. Also, the differences have their effect in the tests of main effects rather than interactions.

Because the ANOVA model shows two main effects and NO interaction. it would be most appropriate to consider the main-effects-only model:

```
model1d <- aov(offer~cond+gender,data=ultim1b)
Anova(model1d)
Anova Table (Type II tests)
```

```
Response: offer
          Sum Sq  Df F value    Pr(>F)
cond      153.22   2  75.475 < 2.2e-16 ***
gender     33.90   1  33.394 1.214e-08 ***
Residuals 604.95 596
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Because the interaction is not significant, each main effect is meaningful and we can look at them individually:

this looks at each pairwise comparison

```
> pairwise.t.test(ultim1$offer,ultim1$cond)

  Pairwise comparisons using t tests with pooled SD

data:  ultim1$offer and ultim1$cond

  F      M
M 0.49   -
N <2e-16 <2e-16

P value adjustment method: holm
> pairwise.t.test(ultim1$offer,ultim1$gender)

  Pairwise comparisons using t tests with pooled SD

data:  ultim1$offer and ultim1$gender

  F
M 5.5e-07

P value adjustment method: holm
>
```

The drawback of this approach is that we are not factoring out the reliable variability accounted for by the other dimension.

Alternately, use the Tukey or other post-hoc test:

```
TukeyHSD(model1d)
> TukeyHSD(model1d)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = offer ~ cond + gender, data = ultim1b)

$cond
            diff         lwr         upr     p adj
M-F -0.09778521 -0.3357063  0.1401358 0.5988856
N-F -1.10108461 -1.3366378 -0.8655315 0.0000000
N-M -1.00329940 -1.2400631 -0.7665357 0.0000000

$gender
           diff         lwr         upr p adj
M-F -0.4756291 -0.6373146 -0.3139437     0
```

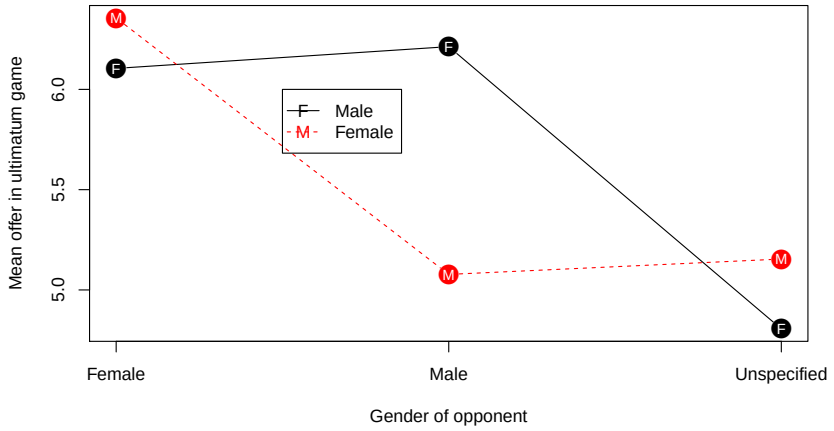
How should we interpret these results?

However, this was all for faked data that we created to specifically not have an interaction. But in reality, there is an interaction that looks roughly like we see in Figure 18.3

```
ultim.tmp <- data.frame(cond =rep(rep(c("N","F","M"),2),100),
                        gender= rep(rep(c("F","M"),each=3),100),
                        offer = rep(c(5, 6.2, 6.3,
                                      5, 6.4,5.1),100)+rnorm(600))

ultim2 <- ultim.tmp[sample(1:nrow(ultim.tmp),size=nrow(ultim.tmp),replace=T),]
matplot(tapply(ultim2$offer,list(ultim2$cond,ultim2$gender),mean),
        type="o",xaxt="n",pch=c("F","M"),ylab="Mean offer in ultimatum game",
        xlab="Gender of opponent")
axis(1,1:3,c("Female","Male","Unspecified"))
legend(2.5,6,c("Male","Female"),pch=c("F","M"),col=1:2,lty=1:2)
```

Figure 18.3: Typical interaction data in the ultimatum game.



This looks like when we don't tell participants anything about the opponent, the two genders behave the same. When we tell women anything about the person, they offer more money. In contrast, men only offer more when the opponent is a woman. How can we test this more specifically?

We might start by building a model with the two predictors and the interaction (notice you can run Anova on an aov model):

```
model2 <- aov(offer~cond*gender,data=ultim2)
Anova(model2)
Anova Table (Type II tests)

Response: offer
             Sum Sq  Df F value    Pr(>F)
cond         155.65   2 73.0621 < 2.2e-16 ***
gender         4.96   1  4.6572   0.03132 *
cond:gender   68.92   2 32.3512 4.612e-14 ***
Residuals    632.71 594
```

The interaction is significant, so we have to think about what the right approach is. Questions:

- is the experiment unbalanced (in this case yes!)
- Do we have hypotheses about main effects after accounting for interactions (maybe)
- are there specific contrasts that are more interesting (probably)
- Are the factors represented with orthogonal contrasts (no)

We take a number of approaches.

18.2.2 Approach 0: Type II and III ANOVAs

Here, we set the constrasts, and use type-II and type-III ANOVAs to examine the main effects and interactions:

```
contrasts(ultim2$cond) <- contr.poly(3)
contrasts(ultim2$gender) <- contr.poly(2)
model2a <- aov(offer~cond*gender,data=ultim2)

Anova(model2a,type="II")  ##type II: test effect of interaction after main
    effects considered
Anova Table (Type II tests)

Response: offer
            Sum Sq  Df F value    Pr(>F)
cond        155.65   2 73.0621 < 2.2e-16 ***
gender        4.96   1  4.6572   0.03132 *
cond:gender  68.92   2 32.3512 4.612e-14 ***
Residuals   632.71 594
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Anova(model2a,type="III") ##type III: test effect of main effects after
    interaction considered
Anova Table (Type III tests)

Response: offer
             Sum Sq  Df    F value    Pr(>F)
(Intercept) 18922.5   1 17764.7950 < 2.2e-16 ***
cond          155.9   2    73.1619 < 2.2e-16 ***
gender          4.8   1     4.4883   0.03454 *
cond:gender    68.9   2    32.3512 4.612e-14 ***
Residuals     632.7 594
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Do these map onto any hypotheses we have? The type-II ANOVA showed that there were main effects for the marginal means of condition and gender once the interaction is accounted for; the Type-III test showed that each main effect was significant even when the interaction was accounted for. We can look at the specific means as well

```
summary(model2a)
model.tables(model2a,type="means")
model.tables(model2a,type="effects")
```

Maybe the gender effect is most interpretable–women tended to offer more on average when the effect of condition is cancelled out, but even then that looks misleading. Overall, we'd maybe like to compare specific pairs in greater detail.

18.2.3 Approach 1: Post-hoc Tukey test on individual pairs

:

we can look at TukeyHSD, which will show all pairwise comparisons that we might be interested in–here consider just the interaction comparisons

```
TukeyHSD(model2a)
> TukeyHSD(model2a)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = offer ~ cond * gender, data = ultim2)

$cond
           diff         lwr         upr p adj
M-F -0.5980595 -0.8384785 -0.3576406     0
N-F -1.2501747 -1.4931026 -1.0072469     0
N-M -0.6521152 -0.8965002 -0.4077302     0

$gender
           diff         lwr          upr     p adj
M-F -0.1818347 -0.3473354 -0.01633399 0.0313437

$`cond:gender`
               diff          lwr         upr      p adj
M:F-F:F  0.1083800 -0.30702939  0.5237894 0.9760260
N:F-F:F -1.2989864 -1.71549681 -0.8824761 0.0000000
F:M-F:F  0.2514137 -0.15978064  0.6626081 0.5004492
M:M-F:F -1.0272761 -1.43746151 -0.6170907 0.0000000
N:M-F:F -0.9505850 -1.36821630 -0.5329537 0.0000000
N:F-M:F -1.4073664 -1.82998515 -0.9847477 0.0000000
F:M-M:F  0.1430338 -0.27434678  0.5604143 0.9243070
M:M-M:F -1.1356561 -1.55204265 -0.7192695 0.0000000
N:M-M:F -1.0589650 -1.48268847 -0.6352415 0.0000000
F:M-N:F  1.5504002  1.13192386  1.9688765 0.0000000
M:M-N:F  0.2717103 -0.14577461  0.6891953 0.4276094
N:M-N:F  0.3484014 -0.07640147  0.7732044 0.1779472
M:M-F:M -1.2786898 -1.69087140 -0.8665083 0.0000000
N:M-F:M -1.2019987 -1.62159076 -0.7824067 0.0000000
N:M-M:M  0.0766911 -0.34191219  0.4952944 0.9952285
```

This shows that all main effect levels are significantly different from one another. For interactions, we might just pick out the specific pairings we care about, which include:

- For NM vs NF: not significant ($p = .18$)
- For MM vs MF: significant $p < .001$
- For FM vs FF: NS $p = .5$

This tells a different story, because we can test specific pairings.

18.2.4 Approach 2: subset on one variable, t-test/ANOVA for each level:

```
1
Anova(aov(offer~gender,data=ultim2[ultim2$cond=="N",]))
3 Anova Table (Type II tests)

5 Response: offer
            Sum Sq  Df F value  Pr(>F)
7 gender      5.857   1  5.8179 0.01681 *
  Residuals 192.270 191
9 ---
  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’
11
  Anova(aov(offer~gender,data=ultim2[ultim2$cond=="M",]))
13

15 Anova Table (Type II tests)

17 Response: offer
            Sum Sq  Df F value    Pr(>F)
19 gender     64.768   1  55.035 3.353e-12 ***
  Residuals 234.192 199
21 ---
  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1> Anova(aov(
      offer~gender,data=ultim2[ultim2$cond=="F",]))
23 Anova Table (Type II tests)

25 Response: offer
            Sum Sq  Df F value  Pr(>F)
27 gender      3.255   1  3.2195 0.07425 .
  Residuals 206.249 204
29 ---
  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1>
```

Here, we can see that there are significant effects of player gender only in the M and F partner case.

18.2.5 Approach 3: subset to get rid of 3-level, interpret 2x2 interaction

Finally, we might edit one of the conditions and test a 2x2 interaction, which is maybe easier to interpret:

```
1 Anova(aov(offer~gender*cond,data=ultim2[ultim2$cond!="N",]))

3 Anova Table (Type II tests)

5 Response: offer
              Sum Sq  Df F value    Pr(>F)
7 gender       19.11   1  17.483 3.559e-05 ***
  cond         35.47   1  32.456 2.356e-08 ***
9 gender:cond  48.92   1  44.758 7.477e-11 ***
  Residuals   440.44 403
11 ---
  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
13
  model.tables(aov(offer~gender*cond,data=ultim2[ultim2$cond!="N",]))
```

```
Tables of effects

 gender
           F        M
      0.2235  -0.2202
rep 202.0000 205.0000

 cond
           F        M
      0.2916  -0.2988
rep 206.0000 201.0000

 gender:cond
      cond
gender F      M
   F    -0.34   0.36
   rep 104.00  98.00
   M     0.35  -0.34
   rep 102.00 103.00
```

Here, the gender x subjectgender interaction is more easily interpretable, and had a significant interaction. Using model.tables, we can also see what the interaction looks like. In this approach, we could also transform this into one-way ANOVA, use custom contrasts to test the hypotheses you care about.

```
cond3 <- as.factor( paste(ultim2$cond,ultim2$gender,sep="-"))
contrasts(cond3) <- cbind(a=c(0,0,0,0,1,-1),
                          b=c(1,-1,-1,1,0,0),
                          c=c(2,0,-1,0,-1,0),
                          d=rep(0,6),
                          e=rep(0,6),
                          f=rep(0,6))

summary(lm(ultim2$offer~cond3))

> summary(lm(ultim2$offer~cond3))

Call:
lm(formula = ultim2$offer ~ cond3)

Residuals:
     Min       1Q   Median       3Q      Max
-2.70787 -0.62712 -0.05767  0.69411  2.68717

Coefficients: (2 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.62236    0.04355 129.100   <2e-16 ***
cond3a       0.13825    0.08221   1.682   0.0932 .
cond3b      -0.81316    0.06877 -11.824   <2e-16 ***
cond3c       0.62830    0.05852  10.736   <2e-16 ***
cond3d            NA         NA      NA       NA
cond3e            NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.066 on 596 degrees of freedom
Multiple R-squared:  0.2139,  Adjusted R-squared:   0.21
F-statistic: 54.06 on 3 and 596 DF,  p-value: < 2.2e-16
```

18.3 Effect sizes and ANOVA models

In an ANOVA or regression model, effect sizes can help provide good insight into whether the model is good, and how much each set of predictors account for. For example, the R^2 statistic is a good measure of effect size, because it tells you about how much of the variance you account for.

Now, consider the following ANOVA model predicting mean response time (rt) based on classes of stimuli. Here is a model for data from an experiment that measures the time needed to find one of fourteen target letters amongst a set of thirteen identical foils. That is, on each trial, a target and a set of foils appeared on the screen, and each of 14 targets was tested amongst each of the other foils. *Note: data are not available for this example*

```
> x <- aov(rt~target+foil)
> x

Call:
   aov(formula = rt ~ target * foil)

Terms:
                         target           foil  Residuals
Sum of Squares         249534642      230659625 1419803758
Deg. of Freedom               13             12        702

Residual standard error: 1422.151
170 out of 196 effects not estimable
Estimated effects may be unbalanced
> summary(x)
                  Df    Sum Sq  Mean Sq F value Pr(>F)
target            13 2.495e+08 19194972   9.491 <2e-16 ***
foil              12 2.307e+08 19221635   9.504 <2e-16 ***
Residuals        702 1.420e+09  2022512
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```

First, notice that because it is an ANOVA table, it will not give us an R^2 directly, like we get in an `lm` model. But it is easy to figure out.

```
 cor(x$fit,tmpsearch$rt)^2
[1] 0.2527341
```

Now, remember that this is the percent of variance accounted for, and so we should be able to get it directly from the sum-of-squares in the ANOVA model:

```
> (249534642+ 230659625)/ (249534642+ 230659625 +1419803758)
[1] 0.2527341
```

But, we could generalize the total variance accounted for explanation with the proportion of variance accounted for by each set of predictors. For target:

```
(230659625)/ (249534642+ 230659625 +1419803758)
[1] 0.1213999
> (249534642)/ (249534642+ 230659625 +1419803758)
```

```
[1] 0.1313342
```

Here, we see that target accounts for 12% of the variance, and foil 13%, summing to 25%. This is exactly what Cohen's η^2 computes. More formally:

$$\eta^2 = \frac{SS_{effect}}{SS_{total}} \tag{18.1}$$

There is a sense in which this is perhaps biased, because we include in the denominator a lot of variance we can actually explain. So, what if we were to divide just the SS related to the effect by the total SS related to either the effect or the unknown variance:

```
> (230659625)/ (230659625 +1419803758)
[1] 0.1397545

> (249534642)/ (249534642+ 1419803758)
[1] 0.1494812
```

This is known as *partial* η^2:

$$\eta_p^2 = \frac{SS_{effect}}{SS_{effect} + SS_{residual}} \tag{18.2}$$

.

Both versions of η^2 are interpretable like R^2, and can give you an intuitive understanding for how good your model is.

Finally, a third effect size often used in ANOVA is ω^2 (Omega-squared), and it is more complicated to compute. The rationale is that we want a fair measure of how much the current factor is explaining over and above the variability that would be expected by chance, given the number of predictors we use to account for it. The mean squared residuals (MSE) gives a value for how much error is associated with each point in the data set. Because a we could completely explain the residual error with a set of random predictors equal in number to the degrees of freedom, we can assume that each random predictor would account for the 1/DF proportion of that total, or the MSE. So, the top of the omega equation gets penalized by MSE times the degrees of freedom of the predictor (in our case, 13 or 12). We then divide by the total SS, but add one MSE to the bottom as well:

```
> (249534642 - 13 * 2022512) / (249534642+ 230659625 +1419803758 + 2022512)
[1] 0.117371

> ( 230659625 - 12 * 2022512) / (249534642+ 230659625 +1419803758 + 2022512)
[1] 0.1085106
>
```

Or, you can use this function (courtesy http://stats.stackexchange.com/questions/2962/omega-squared-for-measure-of-effect-in-r):

```
omega_2 <- function(aov_in, neg2zero=T){
    aovtab <- summary(aov_in)[[1]]
    n_terms <- length(aovtab[["Sum Sq"]]) - 1
    output <- rep(-1, n_terms)
    SSr <- aovtab[["Sum Sq"]][n_terms + 1]
    MSr <- aovtab[["Mean Sq"]][n_terms + 1]
```

```
    SSt <- sum(aovtab[["Sum Sq"]])
    for(i in 1:n_terms){
        SSm <- aovtab[["Sum Sq"]][i]
        DFm <- aovtab[["Df"]][i]
        output[i] <- (SSm-DFm*MSr)/(SSt+MSr)
        if(neg2zero & output[i] < 0){output[i] <- 0}
    }
    names(output) <- rownames(aovtab)[1:n_terms]

    return(output)
}
```

$$\omega^2 = \frac{SS_{effect} - DF \times MSE_{residual}}{SS_{total} + MSE_{residual}} \tag{18.3}$$

This statistic is also interpretable as a percentage of variance accounted for, but accounts for the extra boost you get from additional non-useful predictors. So, for example, should you try to predict an effect with 10 random predictors, on average their value will be exactly canceled out in the numerator. Of course, you may have gotten unlucky and account for less than what might be expected by chance. In that case, ω^2 could be negative.

More conveniently, the `sjstats` package has built-in functions `eta_sq`, `omega_sq`, and an all-in-one that computes many effect sizes called `anova_stats()`.

Be careful when calculating these in different stats packages, because some ANOVA tables will report the residual error, whereas others will report total error.

Exercise 18.3

Compute η^2, η_p^2, and ω^2 for the tooth growth model

```
ToothGrowth$dose2 <- as.factor(ToothGrowth$dose)
aov.tootha <- aov(len~supp+dose2,data=ToothGrowth)
```

Considering the tooth growth data examined in the previous chapter.

```

aov.tootha <- aov(len~supp+dose2,data=ToothGrowth)
aov.toothl <- lm(len~supp+dose2,data=ToothGrowth)

summary(aov.tootha)
            Df Sum Sq Mean Sq F value   Pr(>F)
supp         1  205.4   205.4   14.02 0.000429 ***
dose2        2 2426.4  1213.2   82.81  < 2e-16 ***
Residuals   56  820.4    14.7
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> summary(aov.toothl)

Call:
lm(formula = len ~ supp + dose2, data = ToothGrowth)

Residuals:
    Min      1Q  Median      3Q     Max
-7.085 -2.751  -0.800   2.446   9.650
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  12.4550     0.9883  12.603  < 2e-16 ***
suppVC       -3.7000     0.9883  -3.744 0.000429 ***
dose21        9.1300     1.2104   7.543 4.38e-10 ***
dose22       15.4950     1.2104  12.802  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.828 on 56 degrees of freedom
Multiple R-squared:  0.7623,  Adjusted R-squared:  0.7496
F-statistic: 59.88 on 3 and 56 DF,  p-value: < 2.2e-16
```

Notice that if we take the Sum Sq from the ANOVA table, and divide the 'explained' values by the total values, we get exactly the same number as produced in the 'multiple R^2:

```
 (205.35+2426.434)/(205.35+2426.434+820.425)
[1] 0.7623478
```

We can generalize this to consider the proportion accounted for by each part of the model. This is called η^2 (Eta squared), and is associated with each predictor. For `supp`, $\eta^2 = (205.35)/(205.35 + 2426.434 + 820.425) = .059$, and for dose, $\eta^2 = (2426.434)/(205.35 + 2426.434 + 820.425) = .703$. Notice that these two add to .76. Other related measures are sometimes used, such as an adjusted η^2 akin to adjusted R^2. Several packages in R provide automated ways to calculate these.

18.3.1 Effect size for condition and gender in the ultimatum game data.

We can obtain a number of effect size measures using sjstats `anova_stats()`:

```
eta_sq(aov.tootha)
   term etasq
1  supp 0.059
2 dose2 0.703

omega_2(aov.tootha) #from definition above
supp        dose2
 0.05500642  0.69144228

sjstats::omega_sq(aov.tootha)
   term omegasq
1  supp   0.055
2 dose2   0.691
```

or the all-in-one function

```
> anova_stats(aov.tootha)
       term df    sumsq   meansq statistic p.value etasq partial.etasq omegasq
1      supp  1  205.350  205.350    14.017       0 0.059          0.200   0.055
2     dose2  2 2426.434 1213.217    82.811       0 0.703          0.747   0.691
3 Residuals 56  820.425   14.650        NA      NA    NA             NA      NA
  partial.omegasq cohens.f power
1           0.178     0.50 0.963
```

```
2            0.732      1.72 1.000
3               NA        NA    NA
```

Chapter 19

Analysis of Covariance

We have used linear models to handle both categorical and continuous predictors. In addition, we have mixed categorical and continuous predictors together in a model without much thought. In some domains, when we do this–especially when the continuous variable is a predictor we don't care about theoretically but we know has an impact–this is called an Analysis of Covariance (ANCOVA). From the perspective of regression, adding a categorical predictor to a regression seems like no big deal. But from the perspective of ANOVA, some researchers want to remain focused on the hypothesis testing ability of the ANOVA, while factoring out a nuisance variable. This makes the most sense when you have a manipulated experimental variable, and you want to factor out a covariate that you think might impact your dependent variable.

19.0.1 ANCOVA with a single covariate

A typical examples might include an experimental educational training manipulation, in which we include student pre-test score as a covariate. Alternately we might look at age as a covariate to another experimental manipulation. Here is an example looking at whether there is a difference in salary based on gender–something we know is probably true, but it is difficult to assess the extent. To answer this, you could do a t-test comparing salary by gender, but this might make an experience effect look like a gender effect (if different genders have different experience that leads to differing salaries). Here are three different salary data sets that we will look at with respect to the same gender/experience breakdown:

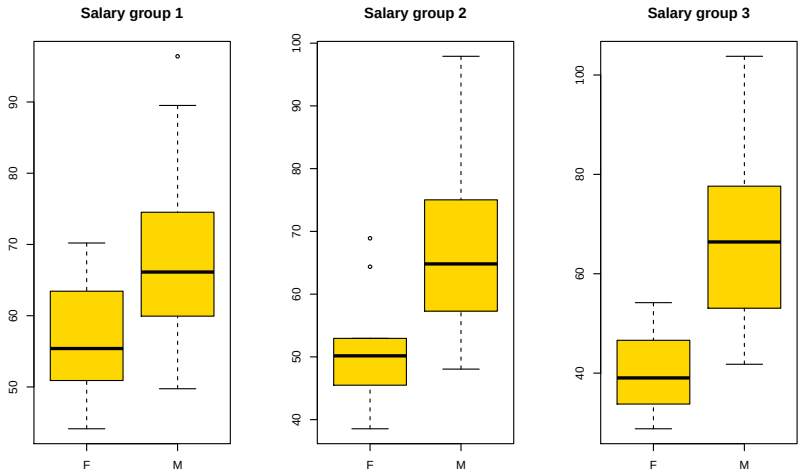
```
set.seed(100)
gender <- as.factor(c("M","M","M","M","M","M","M","M","M","M",
            "F","F", "F","F", "F","F", "F","F", "F","F"))
##we are worried about interactions, so be sure we have an orthogonal set of
    contrasts:
contrasts(gender)<- contr.poly(2)

experience <- c(4,6,10,5,3,8,12,15,9,19,
                5,3,5, 6,1,3,5, 9,10,2)

salary1<-c
    (52.26,59.28,71.10,56.69,49.40,65.80,77.43,85.94,68.99,98.21,56.22,50.42,
    55.12,59.74,43.51,50.91,55.69,67.65,70.18,46.30) + rnorm(20)*5
salary2 <- c
    (53.43,59.39,70.65,55.74,49.68,65.52,76.47,85.92,67.31,97.41,50.49,
    45.23,50.3,53.17,39.92,45.93,50.55,63.53,66.77,41.81)+rnorm(20)*2
```

```
salary3 <- c
    (46.6,55.01,71.21,51.82,42.83,62.5,78.35,90.98,67.6,107.52,40.54,37.25,
    41.83,42.81,33.34,36.46,40.47,48.37,51.42,34.56)+rnorm(20)*2

par(mfrow=c(1,3))
boxplot(salary1~gender)
boxplot(salary2~gender)
boxplot(salary3~gender)
```



This leads to two questions:

Question 1: Do men make more money than women?

Question 2: Can anything account for this difference?

Traditionally, we would test the difference with a t-test or an ANOVA, because it is a categorical difference.

```
t.test(salary1~gender)

  Welch Two Sample t-test

data:  salary1 by gender
t = -2.1525, df = 13.421, p-value = 0.0501
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -23.361112055   0.005598285
sample estimates:
mean in group F mean in group M
       56.74246        68.42021

> t.test(salary2~gender)

  Welch Two Sample t-test

data:  salary2 by gender
t = -2.9266, df = 14.65, p-value = 0.01063
```

```
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -28.534391  -4.456573
sample estimates:
mean in group F mean in group M
       51.39823        67.89372

> t.test(salary3~gender)

  Welch Two Sample t-test

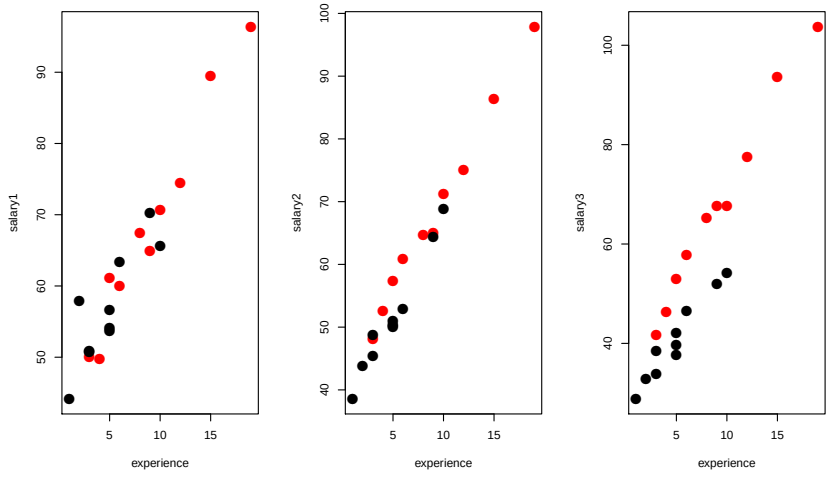
data:  salary3 by gender
t = -3.9631, df = 12.048, p-value = 0.001869
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -41.58499 -12.08927
sample estimates:
mean in group F mean in group M
       40.61854        67.45567

>
```

In each of these cases, men had a higher salary than women. But if it may be that this is just an effect of experience–men may have had more experience in the job, and so make more money. We can look at this in a scatterplot:

```
par(mfrow=c(1,3))
plot(experience,salary1,col=as.numeric(gender),pch=16,cex=2)
plot(experience,salary2,col=as.numeric(gender),pch=16,cex=2)
plot(experience,salary3,col=as.numeric(gender),pch=16,cex=2)
```

Figure 19.1: This shows the relationship between experience and salary, with gender coded as red/black.



If you look carefully, you see that the rightmost case looks like there are two different

lines, indicating that for the same amount of experience men make more. On the left, it looks like the same line, with men simply continuing the same line. The center one is maybe harder to tell. How might we incorporate this in our t-test or ANOVA?

Traditional ANOVA software could not do this, and so special- purpose routines called 'analysis of covariance' were used that allowed you to add a covariate Here, experience would be a covariate, and we'd want to know whether there was a gender effect once experience was accounted for. We should be comfortable just making linear regression models mixing categorical and continuous predictors. this is what is meant by the ANCOVA model, but users typically in an ANCOVA, users essentially ignore the covariate(s) rather than examining them explicitly, as if they were not of interest. Sometimes when people report ANCOVAs, the will not even discuss the slopes of the covariate.

To use an ANCOVA model, you should (1) verify that the covariate matters (2) test whether the covariate has the same impact on all groups; and if not (3) assess the slope interaction. In case 3, just as in ANOVA, it can be difficult to make an inference about the main effects when an interaction exists.

Let's begin by considering salary1. We can make two models and compare them with `anova` (or use type-II Anova from car)

```
library(car)
##for data set 1, try a simple model:
m1.0 <- lm(salary1~experience)
Anova(m1.0) ##experience matters!
Anova Table (Type II tests)

Response: salary1
Sum Sq Df F value    Pr(>F)
experience 3097.75  1  239.17 7.724e-12 ***
Residuals   233.14 18
---



m1 <- lm(salary1~gender+experience)
Anova(m1,type="II")

Anova Table (Type II tests)

Response: salary1
Sum Sq Df  F value    Pr(>F)
gender        0.03  1   0.0022    0.9628
experience 2415.94  1 176.1899 2.119e-10 ***
Residuals   233.11 17

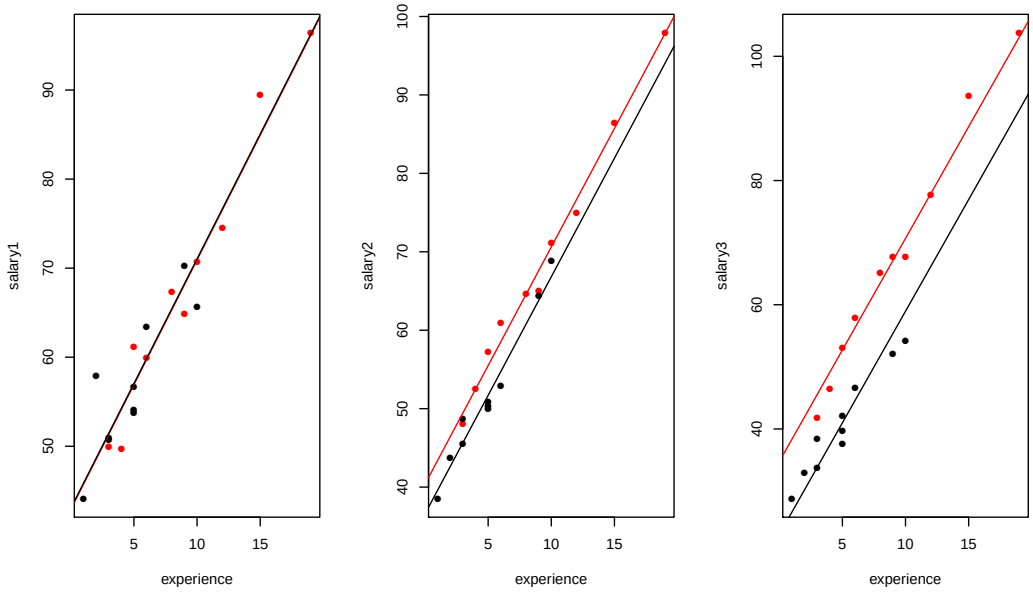


##how would you interpret this?
anova(m1,m1.0)

Analysis of Variance Table

Model 1: salary1 ~ gender + experience
Model 2: salary1 ~ experience
Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     17 233.11
2     18 233.14 -1   -0.0308 0.0022 0.9628
```

Figure 19.2: ANCOVA for salary 1, 2 and 3. When the two lines are on top of one another, as there is no effect of gender once experience is accounted for. When they differ, there is an effect of gender once experience is considered.



Here, these different anova model comparisons show that experience (the covariate) has a large effect. But gender does not once experience is considered.

We can we easily plot this to see what is going on. here, the two lines represent the model for men and for women. Because we have a single salary slope in the model, they are parallel.

```
par(mfrow=c(1,1))
plot(experience,salary1,col=as.numeric(gender),pch=16)
abline(m1$coef[1]+m1$coef[2],m1$coef[3],col=2)
abline(m1$coef[1]-m1$coef[2],m1$coef[3],col=1)
```

How about the second data set? We will just use Type-II ANOVA here:

```
 m2.0 <- lm(salary2~experience) ##experience matters!
> Anova(m2.0)
Anova Table (Type II tests)

Response: salary2
           Sum Sq Df F value    Pr(>F)
experience 4117.5  1  724.49 5.424e-16 ***
Residuals   102.3 18
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> m2 <- lm(salary2~gender+experience)
> Anova(m2,type="II")
Anova Table (Type II tests)

```

```
Response: salary2
             Sum Sq Df F value    Pr(>F)
gender        56.07  1   20.62 0.0002893 ***
experience 2813.06  1 1034.49 < 2.2e-16 ***
Residuals    46.23 17
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

In this data, gender matters once experience is accounted for, which can be seen in the second panel of the figure.

```
par(mfrow=c(1,1))
plot(experience,salary2,col=as.numeric(gender),pch=16)
abline(m2$coef[1]+sqrt(.5)*m2$coef[2],m2$coef[3],col=2)
abline(m2$coef[1]-sqrt(.5)*m2$coef[2],m2$coef[3],col=1)
```

How about the third company?

```
 m3.0 <- lm(salary3~experience)
> Anova(m3.0)
Anova Table (Type II tests)

Response: salary3
            Sum Sq Df F value    Pr(>F)
experience 7046.3  1  185.98 6.273e-11 ***
Residuals   682.0 18
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> m3 <- lm(salary3~gender+experience)
> Anova(m3,type="II") ##experience+gender matter
Anova Table (Type II tests)

Response: salary3
            Sum Sq Df F value    Pr(>F)
gender       535.1  1  61.929 4.563e-07 ***
experience 3980.2  1 460.650 9.413e-14 ***
Residuals   146.9 17
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

For model m3, the effect of experience is significant, but the effect of gender remains significant after it is accounted for. But the plot looks weird–there may be an interaction here.

19.0.2 ANCOVA with interactions

This begins to answer the question about pay by gender, but the third company looks strange—it may be that the effect of gender gets worse as experience increases. This would mean an interaction. We used polynomial contrasts so we could test these interactions. Here are the interaction models for each salary, but we need to consider what the right model is. If there is an interaction, does it really make sense to ask what the effect of gender is? I think not, but if we test gender, it is probably a Type-III test rather than a Type-II test. That is, we want to test whether gender still has an impact AFTER we account for a gender:experience interaction. We can run this test because we used polynomial contrasts.

```
> m1b <- lm(salary1~gender * experience)
> Anova(m1b,type="III")
Anova Table (Type III tests)

Response: salary1
                    Sum Sq Df  F value    Pr(>F)
(Intercept)       8607.6  1 648.1224 2.253e-14 ***
gender              15.3  1   1.1516    0.2991
experience        1588.2  1 119.5829 7.807e-09 ***
gender:experience   20.6  1   1.5521    0.2308
Residuals          212.5 16
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> anova(m1b,m1)  ##no difference--interaction does not matter.
Analysis of Variance Table

Model 1: salary1 ~ gender * experience
Model 2: salary1 ~ gender + experience
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     16 212.49
2     17 233.11 -1   -20.613 1.5521 0.2308
>
> m2b <- lm(salary2~gender * experience)
> Anova(m2b,type="III")
Anova Table (Type III tests)

Response: salary2
                    Sum Sq Df   F value    Pr(>F)
(Intercept)       6734.0  1 2363.2318 < 2.2e-16 ***
gender              22.4  1    7.8458   0.01281 *
experience        2109.3  1  740.2236 7.961e-15 ***
gender:experience    0.6  1    0.2232   0.64300
Residuals           45.6 16
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> m3b <- lm(salary3~gender * experience)
>
> Anova(m3b,type="III")
Anova Table (Type III tests)

Response: salary3
                    Sum Sq Df F value    Pr(>F)
(Intercept)       4020.0  1 799.116 4.366e-15 ***
gender              32.4  1   6.431  0.022020 *
experience        2496.0  1 496.169 1.807e-13 ***
gender:experience   66.4  1  13.199  0.002237 **
Residuals           80.5 16
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> Anova(m3b,type="II")
Anova Table (Type II tests)

Response: salary3
                    Sum Sq Df F value    Pr(>F)
gender             535.1  1 106.369 1.785e-08 ***
experience        3980.2  1 791.211 4.721e-15 ***
gender:experience   66.4  1  13.199  0.002237 **
Residuals           80.5 16
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```

As suspected, for salary 3 there was a significant interaction, which was not present in te other two. But for the first company, since we did a Type-III ANOVA, we are more confident that the non-effect of gender (accounted for by differences in experience) cannot be accounted for by the interaction. Note that for Company 1, the significant effect of gender on its own is still informative, as is the fact that it goes away when accounting for the experience covariate, even in the face of an interaction. It tells the company that their gender pay gap insofar as men are indeed making more money, but it really is an experience gap, and they might make efforts to address the experience gap.

To display this, but because we used poly() contrasts, we need to rescale by the contrasts to get the right slopes/intercepts:

```
par(mfrow=c(1,1))
plot(experience,salary3,col=as.numeric(gender),pch=16)
abline(m3b$coef[1]+sqrt(.5)*m3b$coef[2],m3b$coef[3]+sqrt(.5)*m3b$coef[4],col
    =2)
abline(m3b$coef[1]-sqrt(.5)*m3b$coef[2],m3b$coef[3]-sqrt(.5)*m3b$coef[4],col
    =1)
```

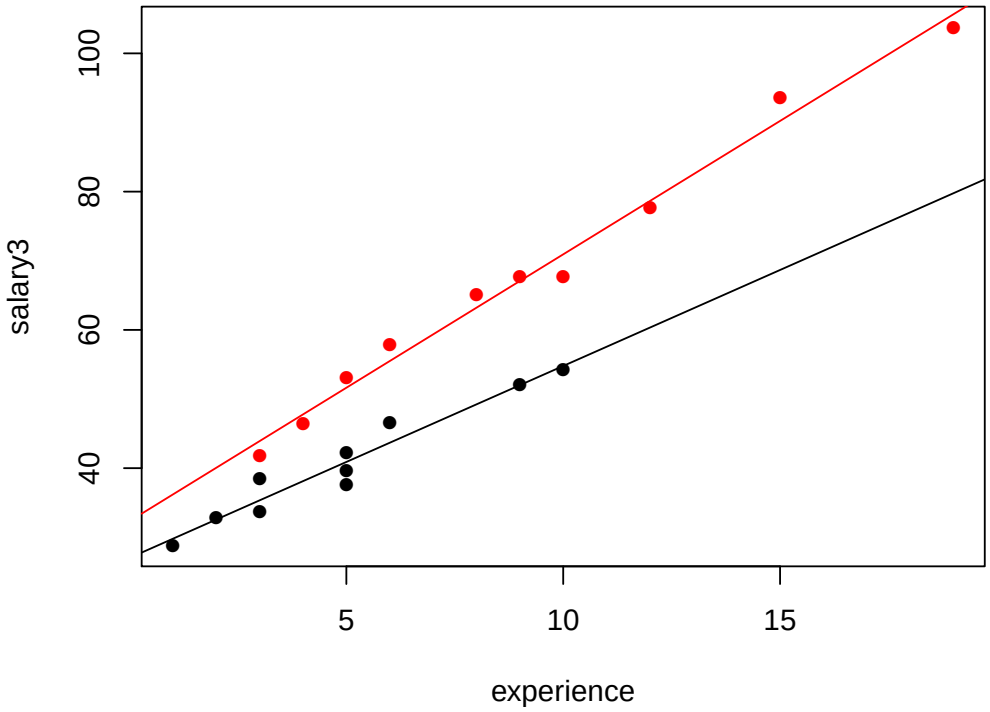
Exercise 19.0.2

The following data are from a problem solving task, in which people tried to solve the "Traveling Salesman Problem". We will remove one subject's data (B1). Each subject took part in many different tests, with different difficulties (problem size). There are several dependent measures, but focus on either solution time (elapsedtime) or how good the solution was (eff).

```
dat.raw <- read.table("tsp-all.txt",header=T)
dat <- dat.raw[ dat.raw$subnum !="B1"&dat.raw$cycle>0,]
```

Create ANCOVA models to see if cycle (as a factor), or clothing (mopp: 1 vs 4) have an impact, using problem difficult as a covariate. Note that in this case, cycle and clothing are designed to be fully crossed with problem size. *Note: In general, you would use an ANCOVA when the covariate is not counterbalanced. In the case of this data, maybe we would look at each individuals spatial reasoning scores or something to see if that accounts for overall performance. In this case, the covariate essentially plays the role of an orthogonal predictor.*

Figure 19.3: ANCOVA for salary 3. The two effects of salary on experience diverge, suggesting that the difference in gender gets worse as experience gets greater.



Exercise 19.0.2 Solution

The following data are from a problem solving task, in which people tried to solve the "Traveling Salesman Problem". We will remove one subject's data (B1). Each subject took part in many different tests, with different difficulties (problem size). There are several dependent measures, but focus on either solution time (elapsedtime) or how good the solution was (eff).

```
dat.raw <- read.table("tsp-all.txt",header=T)
dat <- dat.raw[ dat.raw$subnum !="B1"&dat.raw$cycle>0,]
```

Create ANCOVA models to see if cycle (as a factor), or clothing (mopp: 1 vs 4) have an impact, using problem difficult as a covariate.

```
dat.raw <- read.table("tsp-all.txt",header=T)
dat <- dat.raw[ dat.raw$subnum !="B1"&dat.raw$cycle>0,]
dat$cycle <- as.factor(dat$cycle)
contrasts(dat$cycle) <- contr.poly(levels(dat$cycle))

tab1 <- tapply(dat$eff,list(size=dat$numpos,cycle=dat$cycle),mean)
matplot(c(10,20,30),tab1,type="o",pch=c(1:4),xlab="Problem size",ylab="
    Efficiency")
legend(25,1.08,paste("Cycle", 1:4),lty=1:4,pch=1:4,col=1:4)


##Build and test the ANCOVA models:
tsp0 <- aov(eff~numpos,data=dat)
tsp1 <- aov(eff~numpos+cycle,data=dat)
tsp2 <- aov(eff~numpos*cycle,data=dat)

Anova(tsp1)
Anova(tsp2, type="II") #test the interaction
Anova(tsp2,type="III") #test the interaction
##Here, curiously, whether we find an effect of cycle depends on the
    type of ANOVA!
##but it is sort of misleading, because the interaction was not
    significant.

##These don't differ:
Anova(tsp1, type="II") #test the interaction
Anova(tsp1,type="III") #test the interaction

##what about post-hoc tests:
TukeyHSD(tsp1) ##won't work!

#look at cycle in each case
TukeyHSD(tsp1,which="cycle")
TukeyHSD(tsp2,which="cycle")

##what about time instead of efficiency?
```

```

tab2 <- tapply(dat$elapsedtime,list(size=dat$numpos,cycle=dat$cycle),
    mean)
matplot(c(10,20,30),tab2,type="o",pch=c(1:4),xlab="Problem size",ylab="
    Elapsed time")
legend(25,1.08,paste("Cycle", 1:4),lty=1:4,pch=1:4,col=1:4)


##Build and test the ANCOVA models:
tsp0.2 <- aov(elapsedtime~numpos,data=dat)
tsp1.2 <- aov(elapsedtime~numpos+cycle,data=dat)
tsp2.2 <- aov(elapsedtime~numpos*cycle,data=dat)

```

Chapter 20

Advanced ANOVA: Within-Subject Designs, Repeated Measures, and Random versus Fixed Factors

We have discussed how we can test many of the assumptions of the ANOVA and regression models, including linearity, normal residuals, homogeneity of variance, and independence of predictors (multi-colinearity). However, there is an important aspect of independence that we haven't been able to handle in our models yet: repeated measurement. This is analogous to the paired-samples t-test: we have measured multiple times from the same person, and this comes with both advantages and disadvantages. If we conduct a study or collect data in which we have measured a variable multiple times for each person, we will usually get a more precise estimate of that person. But this is non-independent sampling, and so we shouldn't get an advantage in our power just by measuring multiple times. If I wanted to tell if one group was taller than another, I cannot just measure members of the group a thousand times, giving a larger N and thus a smaller residual error to compare the differences by. The residual error should depend mostly on how many people I sample, not how many samples of each person I take.

This chapter and the next will cover approaches for dealing with repeated measurement, as well as some of the consequences and challenges of this.

20.1 Terminology

There is a lot of related terminology here. So far, the regressions and ANOVA tests we have performed assume randomized and independent designs–that there is no relationship between any particular set of observations. This is essentially making an assumption identical to an independent-samples t-test, rather than a paired samples t-test. For certain designs when this is not the case, special terms have been developed to help characterize the design.

20.1.1 Repeated Measures and within-subject variables

Typically, in psychology, we have multiple observations per individual. This will often happen across a set of conditions, where every participant is tested in each condition. This is usually referred to as a within-subject design. Sometimes, the repetition is not meaningful, in which case we would consider this a repeated-measures design. For example, you might assess response time by having a person respond as quickly as possible to 100 identical lights. Other times, the repetition does matter—for example a pre- versus post- intervention design, or in the case of the light response, response time to different light colors. Both of these are repeated measures designs, but we would often use an aggregate score (like mean RT) over the repeated measures we don't care about, rather than building into the model directly. But for repetitions that matter to us, we have to be sure to compute the relevant F test with the right pair of models, and the appropriate residual variance.

20.1.2 Fixed versus Random effects

A related issue involves whether a factor is fixed versus random. A fixed effect is one in which the conditions or levels are the exact ones we care about and want to generalize to. A random effect is a condition or categorical predictor in which the particular level is sampled from a group or category, and we want to generalize to the group, not the individual levels.

For example, when we test a set of participants, we are sampling from the population, and are hoping to generalize to the population. This means that participant is a random factor. If we want to compare men and women in a test, this is a fixed factor, because these are the specific values we want to compare. If we want to test the effect of math expertise on a set of math problems, the expertise level would be a fixed factor, but the individual math problems we choose will be a randomized factor.

The biggest debate comes when the stimuli or condition is sampled from possible cases, but it is treated as fixed. For example, you might want to see whether two genres of music played during study differentially improve learning. So, you might give one group a bluegrass song, and another group a K-Pop song. If you find that memory differs between the two conditions, you have only shown that the two particular songs differ, not the two genres in general. To establish this, you would need to show this happens across a wide range of songs within each genre, and you would want to incorporate this directly in your model.

20.1.3 Nested effects

Oftentimes, a factor will be nested within another factor. In the case of a within-subject variable (e.g., instruction condition), this is a fixed factor nested within a randomized factor of participant. In other situations, a randomized factor will be nested within a fixed factor. For example, you might compare charter versus public schools, and select 10 schools at random from each type. Here, school type is a fixed factor and the particular school is a random factor.

20.1.4 Mixed Models

There are many approaches to dealing with these issues. If you have a simple repeated measures design with fixed-factor predictors, we can use the ANOVA with error strata to compare the appropriate models. If you have a more complex model with multiple fixed and randomized factors, you may need to use the slightly more complex approach provided by the ezANOVA model. If you have a more complex model with both fixed and random effects that are nested, you may need to use one of the special-purpose packages designed to handle

such models, such as nlme, lme4, nlm4, or more complex models in the ezANOVA model. As these models get more complex, their interpretation becomes a greater challenge.

20.1.5 Why should we care?

In reality, it is fairly common for people to report the improper ANOVA models. In general, researchers have been critiquing others about ignoring randomized effects in psychology since Clark (1974). This is a problem because it fails to consider the randomness involved in choosing your stimuli, and trying to generalize to the entire class of stimuli you are choosing from. By treating them as a fixed effect, you will be more likely to reject the null hypothesis when the effect might really be because you happened to choose stimuli that were not representative of their group, and so you risk increasing the false positive rate of your statistical test.

A more critical problem involves failing to incorporate subject into a within-subject repeated measures design. If you collected 500 trials from 20 subjects, that is 10,000 total trials. If you use a standard ANOVA model, you will have close to 10,000 residual degrees of freedom, and your residual error will become very small because of it, giving you the impression that you have a very sensitive test. But you are not trying to generalize over all the trials of every person under the assumption that every person is identical–you expect variation across people. This between-participant variability is the sampling variability you are trying to generalize over, and so in reality you should have close to 20 degrees of freedom rather than 10,000. Considered in another way, if you had the ability to collect 10,000 trials, which would give you a better understanding of the distribution across a population– 20 participants measured for 500 trials, or 200 participants measured for 50 trials? In most situations, the added benefit you would get by increasing trials from 50 to 500 would be small relative to the added benefit of sampling a larger number of people. Furthermore, most of the time you are essentially ignoring the repeated trials, and so are essentially considering the mean value of each participant in each condition. Those 50 or 500 trials might be averaged into just 5 numbers—one for each condition.

Next, we will go over some typical designs we see in psychology.

20.2 ANOVA with Repeated Measurement

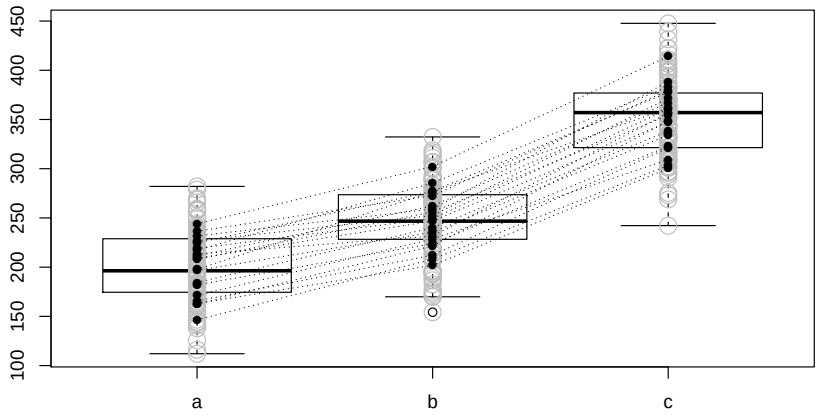
Let's suppose that we run an experiment with 20 participants, and measure response times for three conditions with five observations per condition. We have 15 observations per subject, but really just 3 means we care about (a b and c).

```
x1 <- factor(rep(c("a","b","c"),20*5))
subj <- factor(rep(1:20,each=3*5))
rt <- 200 + runif(20)[subj]*100+ c(-50,0,100)[x1] + rnorm(300) * 30
boxplot(rt~x1)
points(as.numeric(x1),rt,cex=2,col="grey")
```

Notice that if we plot the actual subjects, there is a noticeable consistent variation.

```
rttab <- tapply(rt,list(x1,subj),mean)
matplot(rttab,pch=16,col="black",type="o",add=T,lty=3)
```

Figure 20.1: In this study, we have measured each person multiple times in three conditions. The mean of each subject appears to follow a very consistent pattern across conditions



Here, each line represents a person. We can see that some individuals have higher values, and others have lower overall values, but there are actually very few lines that cross, indicating a very consistent pattern. So, how should we fit a model to this data to determine the impact of the condition variable? Let's start with a simple model:

```
aov1 <- aov(rt~x1)
summary(aov1)

             Df  Sum Sq Mean Sq F value    Pr(>F)
x1            2 1104007  552003  341.49 < 2.2e-16 ***
Residuals   297  480094    1616
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Let's consider this model. If we had collected 300 separate points, this might be correct, but there is something wrong here.

First, think about the within-subject design. We know that each subject had a different overall performance. But this design acts as if they are all independent. However, our figure shows that some people are on average faster or slower than others. This is a lack of independence. It is also double-counting (or quintuple-counting!). If we did a poll where we asked 200 people who they would vote for, but asked them each five times, we cannot claim that we sampled 1000 people. In this case we probably got a significant result in both cases, but it will not always be this way. We'd like to account for the between-subject variability and factor it out or a least account for it.

A simple way to do this would be to essentially fit a different intercept for each person. We could do this by adding the subject code as a predictor, and if we remove the intercept, we won't even have to worry about the treatment coding issue where participant 1 would be equated with the intercept, and all other subject coefficients would be relative to that value.

```
aov2 <- aov(rt~0+subj+x1)

Df   Sum Sq Mean Sq F value    Pr(>F)
subj       20 22040384 1102019 1273.54 < 2.2e-16 ***
x1          2  1104007  552003  637.92 < 2.2e-16 ***
Residuals 278   240559     865
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Anova(aov2)
Anova Table (Type II tests)

Response: rt
           Sum Sq  Df F value    Pr(>F)
subj      4239294  20  265.57 < 2.2e-16 ***
x1        1202115   2  753.07 < 2.2e-16 ***
Residuals  221882 278
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Although the Type-I and Type-II anovas differ a bit, our approach appears reasonable at first. We soak up only 20 degrees of freedom to account for most of the variability in the data set. What this does is essentially get rid of the mean subject variability. If we computed ω^2 or eta^2, we'd find that subject accounts for most of the variance in the model.

```
> library(sjstats)
eta_sq(aov2)
  term etasq
1 subj 0.938
2   x1 0.052

omega_sq(aov2)
  term omegasq
1 subj   0.937
2   x1   0.052
```

This is good–we soak up only 20 degrees of freedom to account for most of the variability in the data set. What this does is essentially get rid of the mean subject variability, which is substantial, and so the residual variance is relatively smaller, and our test is able to detect smaller effects.

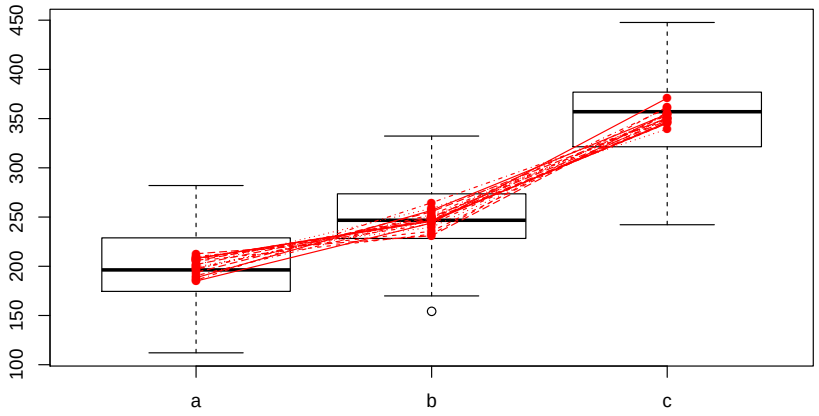
We can see the impact this has by subtracting out the individual means, then adding back in the grand mean, and overplotting in red:

```
boxplot(rt~x1)
 matplot(t(mean(rttab)+ (t(rttab)-(colMeans(rttab)))),
         pch=16,type="o",col="red",add=T)
```

This is what the data would have looked like if we adjust every subject to have their mean look like the average subject. Now, we have very consistent relative differences.

In this way, we modeled variability into three pools–variability we can explain and relates to our hypothesis, variability we don't care about but can explain (the subj code), and variability we can't explain.

Figure 20.2: Plotting participant effects once the mean of each participant is subtracted out.



But the model is still not quite right. Look at the degrees of freedom. Even though we've given each participant his/her own intercept, the residual degrees of freedom are for the whole data set. Imagine we aggregated the data so we have just a single observation per subject per cell, and re-run on this aggregated data:

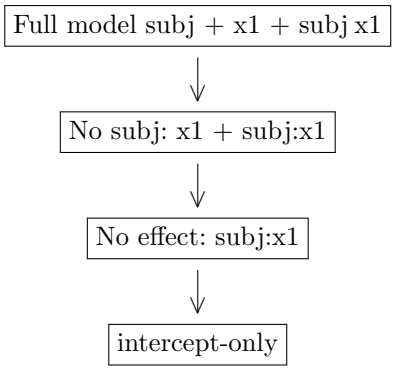
```
dat.agg<-aggregate(rt,list(sub=subj,x1=x1),mean)
aov3 <- aov(x~0+sub+x1,dat=dat.agg)
 summary(aov3)
Df Sum Sq Mean Sq F value    Pr(>F)
sub       20 4408077  220404  1906.4 < 2.2e-16 ***
x1         2  220801  110401   954.9 < 2.2e-16 ***

Residuals   38   4393     116
---
```

In this model, we have only 38 residual degrees of freedom– 20 subjects by 3 groups - 20 subject parameters - 2 groups parameters. We are comparing this to the intercept-only model which contains 1 parameter.

Think about some situations in which `aov2` does not make sense. If there is very little within-subject variability for a measure (say height), and I wanted to see whether height was related to gender, I might collect ten male and ten female participants and find non-significant results. So all I need to do is measure my participants ten times each, add those observations to the data set, thus increasing the residual d.f. by a factor of nearly 10, without appreciably increasing the variance associated with the intercept-only model. Then, I might find a significant result. This should seem suspicious, and we don't want our analyses to work this way.

What we really want to do is compare the main effect model to the smaller model consisting of the variability we cannot account for, after removing the variability we can account for. Let's start with the full `x1*subject` model, as we can always partial out the variability into any bins we know about. Here we build a partial lattice:



First, remove the variability we can account for but don't care about–the main effect of subject. We could compare these models directly, but the remaining model has just the x1 predictor plus the subject x factor interaction.

Now, think about what that interaction is. It is the extent to which the effect of x1 depends on the subject. If it does greatly, than it should be harder to find a reliable main effect. If it does not, than even a very small main effect should be detectable.

Notice that we can account for the `subj` main effect and we don't really care about it; but within each subject, they do not replicate the main effect exactly, even once their intercept is factored out. Each pattern of A-B-C is a bit different. Consequently, the `subj:x1` interaction is the interesting and unknown source of variance we care about when trying to test if there is an effect of `x1`:

```
aov4 <- aov(rt~x1*subj)
summary(aov4)
Anova(aov4)
```

This all can be understood in terms of the ANOVA model lattice. Consider a model that tries to predict response time or accuracy based on the color and size of stimuli. Each combination of color and size was measured multiple times for each participant, and so we sort of have three predictors: color, size, and participant. For this study, we really only care about the particular colors and sizes used, and so we don't worry about generalizing to any randomly chosen color or size (whatever that might mean). But our participants were randomly sampled, and we want to generalize to the population from which they were sampled, and so participant is a randomized factor.

Figure 20.3 shows the basic model lattice for this experiment which looks exactly like the previous models. Previously, we had simply compared particular models to other models in the lattice, and used the ANOVA procedure (and an F test) to determine whether the models differed, and thus whether the factor was reliable.

But now, things get tricky. Let's say we wanted to estimate the impact of color. Our standard approach is to compare a model with color to one without color, and there are three such comparisons, which would be identical if conditions were orthogonal. But now we want to compare the variance we can account with color to what we'd expect given the variability in the color effect across people. The top right comparison measures the difference color makes in the context of participant and size. But this model doesn't tell us much about the variability in color across people. We could have 20 participants with nearly identical means,

Figure 20.3: Basic Model lattice for a within-participant (repeated measures) experiment looking at the impact of color and size on responding.

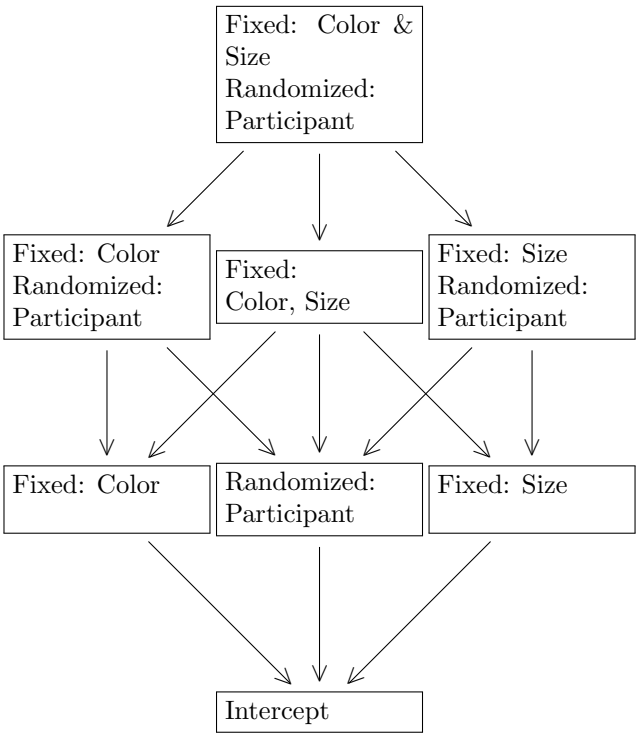
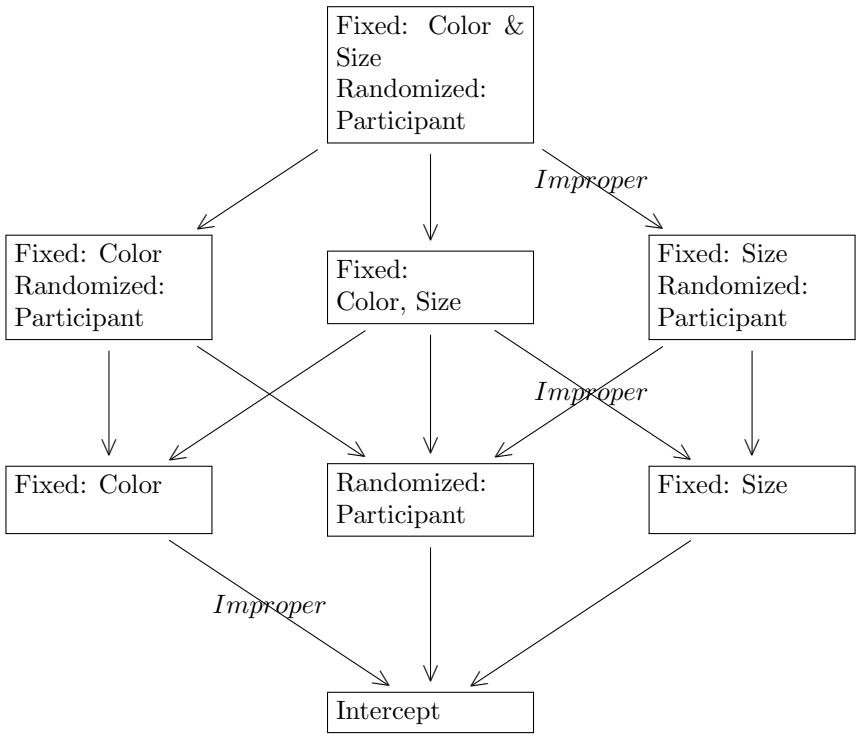


Figure 20.4: Basic Model lattice for a within-participant (repeated measures) experiment looking at the impact of color and size on responding. Each candidate comparison to test the impact of color is inappropriate

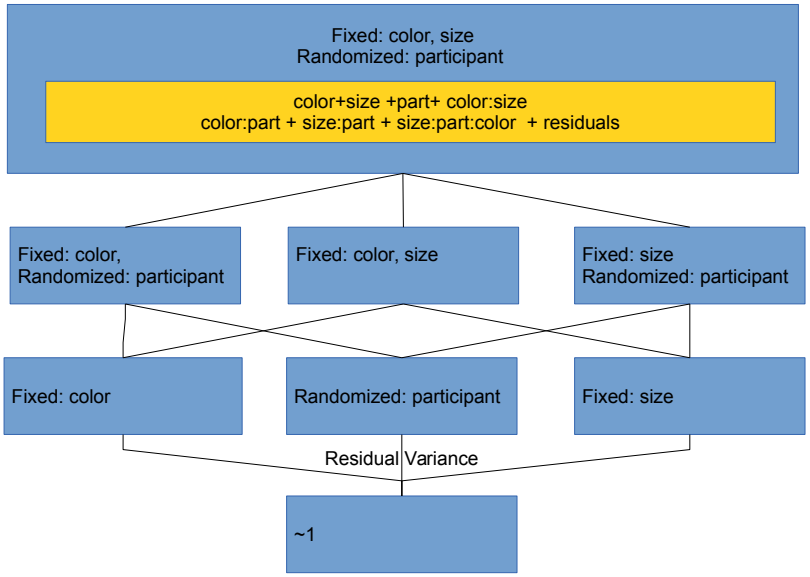


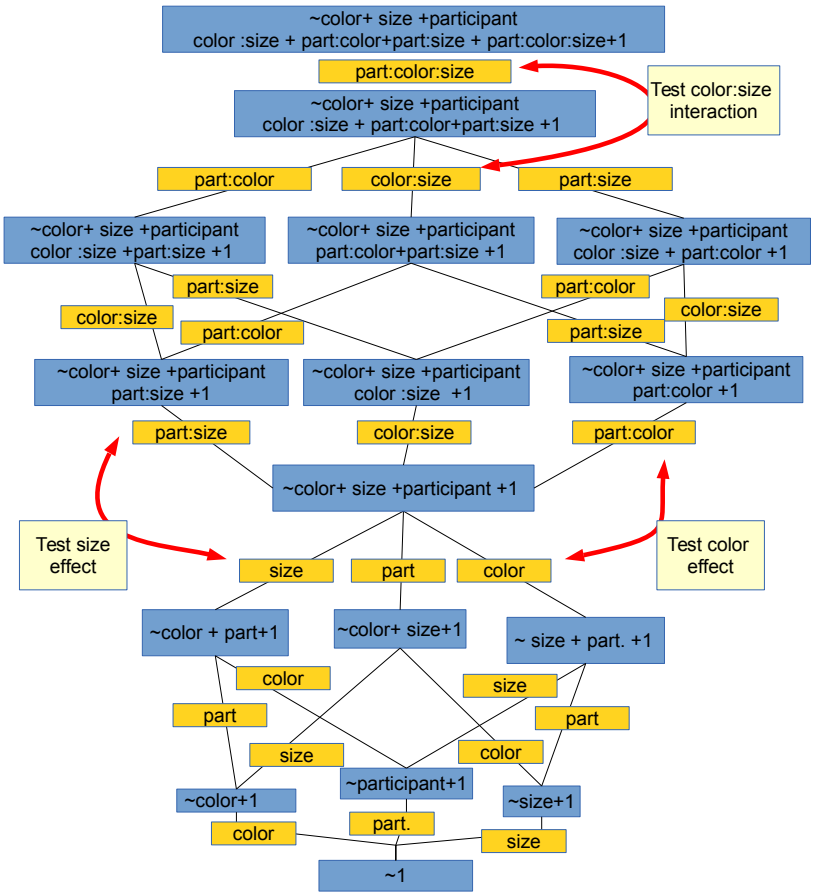
yet individual responses differ substantially; on the other hand, we could have 20 individuals with very different means, but individual responses don't differ much within a person.

The middle and lower comparisons simply ignores participant completely, and so it is definitely not what we want. What we want to do is compare the effect to the part of the variability in the upper right box that participant accounts for. This is just asking what the variability across the means of the participants is, ignoring individual responses. If you consider this carefully, you should recognize that this is the participant by color interaction, which estimates a color effect individually for each participant. Although this interaction was not in the original model lattice, it is in the data. Consider Figure 20.5, which shows how the complete model could include all main effects and interactions. Once these are accounted for, the only remaining variance stems from repeated measurement within each cell of the experiment, and so to estimate variance we need more than one measure in each condition.

Finally, let's expand the interaction model to a complete model lattice. This is pretty complex, and we display only the models that don't violate marginality. (i.e., no interaction without the main effects). Now, we also show what the difference between each model in the lattice is accounting for. To test the impact of one factor (color), we need to compare the SS associated with color alone to the SS associated with the color+participant interaction. Similarly, to test size, we form the ratio between the MS associated with size versus the

Figure 20.5: Basic Model lattice for a within-participant (repeated measures) experiment looking at the impact of color and size on responding. The total variance in the complete model can be divided into all possible main effects and interactions.





size:participant interaction, and to test the interaction between size and color, we compare to size:color:participant interaction. This produces an F test whose first degree of freedom will be related to the number of levels of the test, and whose second degree of freedom is related closely to the number of participants (rather than the total number of observations or trials in the study, which could be hundreds of times larger).

```
aov4 <- aov(rt~x1*subj)
summary(aov4)
```

20.2.1 Exercise

Identify the path through the lattice for aov4. label the MSE on each node of the lattice.

Looking at `aov4`, it provides us with a SS divided into each transition between models in the lattice. Remember that because the design is balanced, we get the same answer whether we use `x1*subj` or `subj*x1`. But the SS related to the `subj*x1` interaction term is the critical pool of variance to compare to. Look at the SSE/MSE for the 4 lines, ignoring the F scores: x1*subject

```
   Df  Sum Sq Mean Sq  F value Pr(>F)
```

```
  x1            2 1104007  552003 606.0651 <2e-16 ***
  subj         19  239535   12607  13.8418 <2e-16 ***
  x1:subj      38   21967     578   0.6347 0.9533
Residuals     240  218592     911
```

- The first line is the delta SS related the main effect, which we care about. It is currently being compared to the residuals.
- The next line is the set of intercepts related to subject variability. This is stuff we can account for but we don't care about.
- The third line is the variance that can be attributed to the subj * effect interaction, which tells us how much the effect depends on participant.
- The fourth line tells us the rest, which is the difference from the mean value predicted by subject and condition and subj*condition provide.

For generalization, we don't care about predicting new single observations, we want to understand the mean impact that condition has on the population. The x1:sub interaction tells us how much it depends, and so this is the proper pool of variance (subtracting out the intercept) to use. This means that the F values are computed incorrectly. What we really want is an F value of 552003/578=955.0225

Exercise 20.2.1

Notice that this is different from the following: Why?

```
> anova(aov(rt~x1),aov(rt~x1:subj))
```

The basic logic we have gone through involves picking out the proper error term for your factors when they are nested within randomized variables such as participant. Notice that this last model would be fine if subj were a blocking variable that was not sampled randomly. In that case, we'd want to be generalizing over the individual observations still, not the blocks. But for subjects, we usually want to make inference about the population mean.

There are several libraries for handling this, but R let's you specify so-called error strata using the Error() function. Instead of entering a randomized factor directly into the model, place it inside the Error function, and tell it which non-randomized factors are nested within it (in this case, just x1)

```
aov5 <- aov(rt~x1+Error(subj/x1)+0)
summary(aov5)

Error: subj
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 19 215235   11328

Error: subj:x1
          Df  Sum Sq Mean Sq F value Pr(>F)
x1         2 1202115  601057    1263 <2e-16 ***
Residuals 38   18090     476
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Error: Within
           Df Sum Sq Mean Sq F value Pr(>F)
Residuals 240 203792   849.1
```

Notice that the F test is the same as we calculated earlier, (1263) considering rounding error for the MSE reported in the table. Also, notice the d.f. are appropriate. $F(2, 38) = 1263$,$p < .0001$.

A number of examples are provided in the alternate readings for today. (see personalityproject and Baron's R notes')

Exercise 20.2.1

Use the data from the example provided by Baron:

```
  data1<-c(
  49,47,46,47,48,47,41,46,43,47,46,45,
  48,46,47,45,49,44,44,45,42,45,45,40,
  49,46,47,45,49,45,41,43,44,46,45,40,
  45,43,44,45,48,46,40,45,40,45,47,40)

Hays.df <- data.frame(rt = data1,
       subj = factor(rep(paste("subj", 1:12, sep=""), 4)),
       shape = factor(rep(rep(c("shape1", "shape2"), c(12, 12)), 2)),
       color = factor(rep(c("color1", "color2"), c(24, 24))))
par(mfrow=c(1,3))
boxplot(rt~shape,data=Hays.df)
boxplot(rt~color,data=Hays.df,col=c("grey40","gold"))
boxplot(rt~shape*color,data=Hays.df)
matplot(tapply(Hays.df$rt,Hays.df[,3:4],mean),type="b")
```

1. Do a full aov model including the main effects's interactions with subject, and calculate F values by hand by comparing lines of the table
2. Build a model using the appropriate Error() term, and verify you can obtain the same F tests.

Exercise 20.2.1

Identify the path through the lattice for aov4. label the MSE on each node of the lattice.

20.3 Repeated Measures

Consider the following experiment in which we have five repetitions of each participant. Here, the repetitions do not really represent anything of interest–just additional data to get a more precise estimate, based on the faked data described earlier.

```
set.seed(500)
x1 <- factor(rep(c("a","b","c"),20*5))
```

```
subj <- factor(rep(1:20,each=3*5))
rt <- 200 + runif(20)[subj]*100+ c(-50,0,100)[x1] + rnorm(300) * 30

set.seed(100)
data2 <- data.frame(subj=rep(subj,each=5),
                    cond = rep(x1,each=5),
                    rt = rep(rt,each=5) + rnorm(length(rt)*5))

data2[1:10,]
   subj cond       rt
1     1    a 190.3287
2     1    a 190.9624
3     1    a 190.7520
4     1    a 191.7177
5     1    a 190.9479
6     1    b 230.2393
7     1    b 229.3389
8     1    b 230.6352
9     1    b 229.0954
10    1    b 229.5608
>
```

Here, we have just observed each person in each condition five times In terms of the analysis we do, any repetition within the subject x condition table should not increase the power of the test (although it may increase the accuracy of our estimates.) In a sense, the test we do should be roughly the same as what we get if we'd simply aggregate down to one observation per person. Let's see if we can do this.

```
data2b  <- aggregate(data2$rt,list(subj=data2$subj,cond=data2$cond),mean)
data2b$rt <- data2b$x

aov2.a <- aov(rt~cond+Error(subj/cond),data=data2)
aov2.b <- aov(rt~cond+Error(subj/cond),data=data2b)

summary(aov2.a)

Error: subj
          Df  Sum Sq Mean Sq F value Pr(>F)
Residuals 19 1045428   55023

Error: subj:cond
          Df  Sum Sq Mean Sq F value Pr(>F)
cond       2 5565412 2782706   501.4 <2e-16 ***
Residuals 38  210897    5550
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Error: Within
            Df  Sum Sq Mean Sq F value Pr(>F)
Residuals 1440 1131706   785.9



> summary(aov2.b)

Error: subj
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 19  41817    2201

Error: subj:cond
```

```
33           Df  Sum Sq Mean Sq F value Pr(>F)
cond          2 222616  111308   501.4 <2e-16 ***
35 Residuals 38   8436     222
---
37 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

To test the effect of condition, in each case we look at the cond:subj error strata. In aov2.a, the F tests are identical: F(2,38)= 501.4.

Consequently, for true repetition of conditions, this error strata scheme will properly ignore repetitions within subject in the test, because we care about generalizing across the mean of people, color, and we don't see an effect, we shouldn't get a huge benefit if we simply decide to measure height ten times per person.

20.4 Repeated measures and the ezANOVA

The `ez` library has some easier-to-use implementations of ANOVA that support repeated measures. You need to specify within and between variables specifically, but it provides a nice output.

```
1 library(ez)
  ez2.a1 <- ezANOVA(data=data2,dv=rt,within=cond,wid=subj)
3 ez2.a2 <- ezANOVA(data=data2,dv=rt,within_full=cond,wid=subj) ##This doesn't
      work
  ez2.b <- ezANOVA(data=data2b,dv=rt,within=cond,wid=subj)
5
  ez2.a1
7 > ez2.a1
  $ANOVA
9   Effect DFn DFd       F             p p<.05       ges
  2   cond   2  38 501.396 4.852583e-28     * 0.8158351
11
  $`Mauchly's Test for Sphericity`
13  Effect         W         p p<.05
  2   cond 0.9592763 0.6878494
15
  $`Sphericity Corrections`
17  Effect       GGe         p[GG] p[GG]<.05      HFe         p[HF] p[HF]<.05
  2   cond 0.9608698 4.956839e-27         * 1.066701 4.852583e-28         *
19
  > ez2.b
21 $ANOVA
    Effect DFn DFd       F             p p<.05       ges
23 2   cond   2  38 501.396 4.852583e-28     * 0.8158351

25 $`Mauchly's Test for Sphericity`
    Effect         W         p p<.05
27 2   cond 0.9592763 0.6878494

29 $`Sphericity Corrections`
    Effect       GGe         p[GG] p[GG]<.05      HFe         p[HF] p[HF]<.05
31 2   cond 0.9608698 4.956839e-27         * 1.066701 4.852583e-28         *
```

Notice that the ez provides the ANOVA test, including a 'ges' effect size, which is a generalized eta-squared value. Also, it includes a test for sphericity, which can tell you if your test is violating some of the assumptions of regression models. We can see that the test

is identical across all the versions of this test we have done. You can get all of the sum-of-squares values if you use the detailed=T argument, and the actual anova object (with error strata) if you use `return_aov=T`.

20.5 Mixed Designs: ANOVA models with between and within variables

In typical ANOVA/Regression, we assume that all variables are between-participant. In the current example, the predictors have been within-subject. When we have both types, this is referred to as a mixed factorial design. Let's suppose we have such a data set.:

(Be sure subject code is a factor!)

```
data3 <- data.frame(subj=as.factor(c(subj, as.numeric(subj)+20)),
                    group = rep(c("Control","Experimental"),each=300),
                    cond = rep(x1,2),
                    rt = rep(rt,2) + rnorm(600))

> head(data3)
  subj    group cond       rt
1    1 Control    a 189.6783
2    1 Control    b 230.3848
3    1 Control    c 365.4999
4    1 Control    a 249.0270
5    1 Control    b 229.3105
6    1 Control    c 338.4940
>
```

defining the model is as follows: because group is a between-subject factor, it is not nested within the Error() notation, but cond is because it is a within-subject variable.

```

model3<- aov(rt~ group + cond + Error(subj/cond),data=data3)
summary(model3)

> summary(model3)

Error: subj
          Df Sum Sq Mean Sq F value Pr(>F)
group      1      0       0       0      1
Residuals 38 417296   10981

Error: subj:cond
          Df  Sum Sq Mean Sq F value Pr(>F)
cond       2 2222004 1111002    1025 <2e-16 ***
Residuals 78   84574    1084
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


Error: Within
           Df Sum Sq Mean Sq F value Pr(>F)
Residuals 480 451797   941.2
```

interpreting model3 is tricky, because the effects appear in multiple error strata. To find the right one, it is sometimes helpful to take a look at the degrees of freedom and be sure

they are sensible. The rule of thumb is that if we are trying to generalize an effect of a condition across people, so we should look at the error related to the condition x person interaction. This is the second strata, and we'd report F(2,78)=1060, p¡.001. Note that there were 40 subjects and 3 conditions; (40-1)*(3-1)=78.

To examine the group variable, the correct statistic is nested within sthe subj error strata F(1,38)=0.

This can be confusing. A way of figuring out the right test to do is to consider the one-way ANOVA you'd perform on just the factor you care about, aggregating over subject. The degrees of freedom should be the same, although the test statistic could differ.

To test the effect of between-subject group

```
data3.agg1 <- aggregate(data3$rt,list(group=data3$group,subj=data3$subj),mean)
head(data3.agg1)
          group subj        x
1       Control    1 293.0034
2       Control    2 280.8230
3       Control    3 323.2033
4       Control    4 256.1180
5       Control    5 296.4960
6       Control    6 226.4755


summary(aov(x~group,data=data3.agg1))
            Df Sum Sq Mean Sq F value Pr(>F)
group        1      0     0.0       0      1
Residuals   38  27820   732.1
```

Notice that here, the test for group is again F(1,38). How about for condition: aggregate over condition and subject:

```

data3.agg2 <- aggregate(data3$rt,list(cond=data3$cond,subj=data3$subj),mean)
summary(aov(x~cond+Error(subj/cond),data=data3.agg2))
> summary(aov(x~cond+Error(subj/cond),data=data3.agg2))

Error: subj
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 39  83459    2140

Error: subj:cond
          Df Sum Sq Mean Sq F value Pr(>F)
cond       2 444401  222200    1025 <2e-16 ***
Residuals 78  16915     217
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Just as above, here the F test is $F(2, 78) = 2619$. This is exactly the same test in this case. If you have a non-orthogonal design, it probably could differ.

How about the ezANOVA?

```


ez3.a <- ezANOVA(data=data3,dv=rt,within=cond,     wid=subj,between=group,
    detailed=T)
ez3.b <- ezANOVA(data=data3,dv=rt,within_full=cond,wid=subj,between=group,
    detailed=T) ##This collapses too much and is wrong
```

```
> print(ez3.b)
$ANOVA
  Effect DFn DFd           SSn      SSd             F          p p<.05
    ges
1  group   1  38 0.0001753375 27819.71 2.395002e-07 0.9996121       6.302636e
    -09

$`Levene's Test for Homogeneity of Variance`
  DFn DFd       SSn      SSd            F         p p<.05
1   1  38 0.0489761 9620.256 0.0001934556 0.9889755

>
> print(ez3.a)
$ANOVA
       Effect DFn DFd           SSn      SSd             F            p p<.05
          ges
1 (Intercept)   1  38 8.982357e+06 83459.14 4.089781e+03 2.681462e-40     *
    9.889489e-01
2       group   1  38 5.260125e-04 83459.14 2.395002e-07 9.996121e-01
    5.240536e-09
3        cond   2  76 4.444008e+05 16914.67 9.983774e+02 2.767544e-55     *
    8.157517e-01
4  group:cond   2  76 1.616313e-01 16914.67 3.631161e-04 9.996370e-01
    1.610291e-06

$`Mauchly's Test for Sphericity`
      Effect         W         p p<.05
3       cond 0.9604942 0.4744085
4 group:cond 0.9604942 0.4744085

$`Sphericity Corrections`
      Effect       GGe        p[GG] p[GG]<.05      HFe        p[HF] p[HF]<.05
3       cond 0.9619956 2.795560e-53         * 1.012247 2.767544e-55         *
4 group:cond 0.9619956 9.995202e-01           1.012247 9.996370e-01
```

Notice that we get the same values for group and cond.

```
summary(model3)


Error: subj
          Df Sum Sq Mean Sq F value Pr(>F)
group      1      0       0       0      1
Residuals 38 417296   10981

Error: subj:cond
          Df  Sum Sq Mean Sq F value Pr(>F)
cond       2 2222004 1111002    1025 <2e-16 ***
Residuals 78   84574    1084
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


Error: Within
           Df Sum Sq Mean Sq F value Pr(>F)
Residuals 480 451797   941.2
```

Here, the results are essentially the same, but the ez model will automatically test the interaction as well, so the residual degrees of freedom goes down by 2 to 76 instead of 78.

Exercise 20.5

Exercise: Use the data from the examples provided by Baron to test the effect of shape and color (both within-subject variables)
Use both the Error() syntax and the ezANOVA() model. Write the way you'd report the test in a paper. Verify that they produce the same results.

```
##>Example 2 from Baron
  data1<-c(
  49,47,46,47,48,47,41,46,43,47,46,45,
  48,46,47,45,49,44,44,45,42,45,45,40,
  49,46,47,45,49,45,41,43,44,46,45,40,
  45,43,44,45,48,46,40,45,40,45,47,40) # across subjects then conditions


Hays.df <- data.frame(rt = data1,
                      subj = factor(rep(paste("subj", 1:12, sep=""), 4))
    ,
                      shape = factor(rep(rep(c("shape1", "shape2"), c
    (12, 12)), 2)),
                      color = factor(rep(c("color1", "color2"), c(24,
    24))))
```

20.6 Sphericity, and corrections for sphericity

The ezANOVA provides additional output testing the assumptions of the model, especially the homogeneity of variance. Under some conditions, ezANOVA returns Levene's test, but for within/between designs it retruns "Mauchly's Test for Sphericity":

```
$`Mauchly's Test for Sphericity`
      Effect         W         p p<.05
3       cond 0.9604942 0.4744085
4 group:cond 0.9604942 0.4744085

$`Sphericity Corrections`
      Effect       GGe        p[GG] p[GG]<.05      HFe        p[HF] p[HF]<.05
3       cond 0.9619956 2.795560e-53         * 1.012247 2.767544e-55         *
4 group:cond 0.9619956 9.995202e-01           1.012247 9.996370e-01
```

In this example (see above), the condition effect as significant, but let's focus on the Mauchly test. Here, the Mauchly test was not significant (p=.477), so we are fine to interpret them as above.

Sphericity is a special kind of equal variance assumption relevant to repeated measures ANOVA: it is the assumption that all covariances between levels of variables are equal. In other words, if, for each participant, you find the pairwise difference between their scores on each pair of levels of a variable, these pairwise differences all have the same variance.

So, sphericity is an assumption in repeated-measures ANOVA that we didn't have to look at for non-repeated measures because it implies a consequence of our lack of independence. Without repeated measures, there is no sense in which we have differences between levels on an individual basis. But sphericity does not require that there be no covariance between

levels–just that it is the same across levels. For example, if you have people drive three different cars on the same track and examine their completion speed:

Table 20.1: Hypothetical time (in s) of five racers in three different vehicles on the same track.

Person	VW Golf	Ford Mustang	Tesla	VW - Ford	Ford - Tesla	VW - Tesla
Anders	105	88	66	17	22	39
Blaine	95	75	68	20	7	27
Cristina	83	89	85	-6	4	-2
Donald	125	72	88	53	-16	37
Variance	316	77	129	590	244	358

If we look at the final variance-of-differences, we see that these three scores appear to differ substantially. The higher covariance for VW-Ford indicates that these two are probably more closely related than Ford-Tesla. This means that vehicle is not just a constant effect, and that it interacts with driver, and this is problematic because we are using this interaction to determine whether the effect of vehicle is significant.

In the event you find a significant test for non-sphericity, the typical recommendation is to use an adjustment for degrees of freedom–just like we did in the Welch's t-test and the one-way ANOVA with non-constant variance. There are several common corrections, including the Greenhouse-Geisser, Huyn & Feldt correction. These are included in the 'Sphericity corrections' table in ezANOVA. If we show the entire output again, we can see these corrections map onto the Greenhouse-Geisser (GG) and Huyn-Feldt (HF). These are values that we would adjust the degrees of freedom and p-value by if the Mauchly's test was significant.

```
> print(ez3.a)
$ANOVA
       Effect DFn DFd SSn        SSd      F           p       p<.05       ges
1 (Intercept)   1  38 8.982e+06 83459.14 4.089e+03 2.681e-40    * 9.889e-01
2       group   1  38 5.260e-04 83459.14 2.395e-07 9.996e-01      5.240e-09
3        cond   2  76 4.444e+05 16914.67 9.983e+02 2.767e-55    * 8.157e-01
4  group:cond   2  76 1.616e-01 16914.67 3.631e-04 9.996e-01      1.610e-06

$`Mauchly's Test for Sphericity`
      Effect          W          p p<.05
3       cond 0.9604942 0.4744085
4 group:cond 0.9604942 0.4744085


$`Sphericity Corrections`
      Effect       GGe        p[GG] p[GG]<.05      HFe        p[HF] p[HF]<.05
3       cond 0.9619956 2.795560e-53          * 1.012247 2.767544e-55          *
4 group:cond 0.9619956 9.995202e-01            1.012247 9.996370e-01
```

So, if the Mauchly's test is not significant, we might report this test as follows:

> Mauchly's test for sphericity was not significant (W=.96, p=.47), and so the repeated-measures ANOVA showed there was a significant effects of condition ($F(2, 76) = 998, p < .001$), generalized $\eta^2 = .81$, but no effect of group ($F(1, 38) = 0, p = 1.0$) and no condition by group interaction ($F(2, 76) = 0$, $p = 1$).

If, however, the sphericity test was significant, we would adjust both degrees of freedom by GGe or HFe. In the cases above, GGe and HFE are both close to 1.0, so there is no large

adjustment. We can then report the p-value from the corrections lines–in this case they do not differ.

> Mauchly's test for sphericity was significant (W=.12, $p < .01$), and so we applied the Greenhouse-Geisser correction to degrees of freedom. The resulting adjusted ANOVA showed there was a significant effects of condition ($F(2, 75.7) = 998$, $p < .001$), generalized $\eta^2 = .81$, but no effect of group ($F(1, 37.8) = 0$, $p = 1.0$) and no condition by group interaction ($F(2, 75.7) = 0$, $p = 1$).

20.7 Post-hoc tests with repeated measures ANOVA

If we are using a repeated-measures ANOVA–either with ez or with Error(), one remaining problem is that we'd often like to compute post-hoc tests. This is problematic, because the software will not generally permit this, and it is not even clear that the the proper way to compute post-hoc tests in a repeated-measures context is widely agreed upon or widely used. This is, in part, because more modern approaches to modeling repeated measures in the guise of mixed-effects models make the models more principled, and there are libraries that will handle the post-hoc tests in those contexts.

Some strategies I have seen for dealing with this include:

- Use subsetting to generate sub-models with two levels; this will then test a specific contrast between levels. This does not correct for family-wise testing, so you would then likely apply a bonferroni correction

- Do multiple pairwise t-tests and apply a bonferroni or holm correction. For example, use `pairwise.t.test`. This will have less power than a true anova, because you are not factoring out predictors you know, but it may be effective if conservative.

- Some sources have suggested that if the sphericity assumption is met, you can treat subject as a normal effect, and no longer treat it as a randomized factor but as a fixed factor. Then, the normal Tukey HSD tests will work on the remaining variables. I have not seen strong mathematical justification for doing this.

- Use a richer modeling framework. Most sources simply advise people to use nlme/lmer/lme models and multcomp package, which will do relevant post-hoc tests on those models, or use a hierarchical Bayesian approach. These are reasonable approaches, but those models are difficult to understand without a grounding in the historical approach to repeated measures models, and overall more difficult to create and interpret than these models. We will cover those models in the next chapter.

20.8 Answers to exercises

Exercise Solution 20.2.1

Use the data from the example provided by Baron:

```
  data1<-c(
  49,47,46,47,48,47,41,46,43,47,46,45,
  48,46,47,45,49,44,44,45,42,45,45,40,
  49,46,47,45,49,45,41,43,44,46,45,40,
  45,43,44,45,48,46,40,45,40,45,47,40)

Hays.df <- data.frame(rt = data1,
       subj = factor(rep(paste("subj", 1:12, sep=""), 4)),
       shape = factor(rep(rep(c("shape1", "shape2"), c(12, 12)), 2)),
       color = factor(rep(c("color1", "color2"), c(24, 24))))
par(mfrow=c(1,3))
boxplot(rt~shape,data=Hays.df)
boxplot(rt~color,data=Hays.df,col=c("grey40","gold"))
boxplot(rt~shape*color,data=Hays.df)
matplot(tapply(Hays.df$rt,Hays.df[,3:4],mean),type="b")
```

1. Do a full aov model including the main effects's interactions with subject, and calculate F values by hand by comparing lines of the table
2. Build a model using the appropriate Error() term, and verify you can obtain the same F tests.

```
anova1 <- aov(rt~(color+shape)*subj ,data=Hays.df)
anova2 <- aov(rt~color*shape*subj ,data=Hays.df)
anova3 <- aov(rt~color+shape+subj:color + subj:shape ,data=Hays.df)

anova4 <- aov(rt~color*shape +Error(subj/(color*shape)), data=Hays.df)
```

Let's suppose that we run an experiment with 20 participants, and measure response times for three conditions with five observations per condition. We have 15 observations per subject.

Exercise Solution 20.8

Exercise: Use the data from the examples provided by Baron to test the effect of shape and color (both within-subject variables)
Use both the Error() syntax and the ezANOVA() model. Write the way you'd report the test in a paper. Verify that they produce the same results.

```
##>Example 2 from Baron
  data1<-c(
  49,47,46,47,48,47,41,46,43,47,46,45,
  48,46,47,45,49,44,44,45,42,45,45,40,
  49,46,47,45,49,45,41,43,44,46,45,40,
  45,43,44,45,48,46,40,45,40,45,47,40) # across subjects then conditions


Hays.df <- data.frame(rt = data1,
                      subj = factor(rep(paste("subj", 1:12, sep=""), 4))
    ,
                      shape = factor(rep(rep(c("shape1", "shape2"), c
    (12, 12)), 2)),
                      color = factor(rep(c("color1", "color2"), c(24,
    24))))
```

For this data, we are interested in knowing if shape and color of stimuli affect RT.

```
par(mfrow=c(1,3))
boxplot(rt~shape,data=Hays.df)
boxplot(rt~color,data=Hays.df,col=c("grey40","gold"))
boxplot(rt~shape*color,data=Hays.df)
matplot(tapply(Hays.df$rt,Hays.df[,3:4],mean),type="b")

##incorrect:
summary(aov(rt ~ shape * color, data=Hays.df))

anova.hays <- aov(rt~color*shape +Error(subj/(color*shape)), data=Hays.
    df)
summary(anova.hays)

ezANOVA(Hays.df,,dv=rt,within=.(shape,color),wid=subj,detailed=T)

#  Exercise: compute aov() and ezANOVA models for each of Baron's
    examples:
#  Baron example 2 (Maxwell & Delaney)

#Two within-subject variables:

MD.rt <- matrix(c(
420, 420,  480, 480, 600, 780,
420, 480, 480, 360, 480, 600,
480, 480, 540, 660, 780, 780,
420, 540,540, 480, 780, 900,
540, 660,540, 480, 660, 720,
360, 420, 360, 360, 480, 540,
480, 480,600, 540, 720, 840,
480, 600, 660, 540, 720, 900,
540, 600,540, 480, 720, 780,
480, 420,540, 540, 660, 780),
ncol = 6,byrow = T) # byrow=T so the matrix's layout is exactly like
    this
```

Chapter 21

Mixed effects models, lmer, and nlme models

The previous chapter considered a special class of models with just one randomized factor: participant. The other factors were fixed, and you had the possibility of repeated measures. Sometimes you have additional random factors, and also additional fixed factors. These are handled with a more advanced set of models referred to as mixed-effects models. Mixed-effects just refers to the fact that you are combining both fixed and random factors in a model.

A fixed effect is one whose independent variables won't change if re-run the experiment using new subjects/materials. A random effect is one that will. Commonly, randomized factor (often referred to as random effects models) will be participants, batches, stimuli, and things we sample from to make our conditions. The random factor isn't really a parameter/condition, it is a random variable whose variability we want to account for and usually compare against when determining if we have found some systematic effect of a fixed factor.

This especially comes into play when our random factor could have a lot of variability. Suppose we wanted to test the effectiveness of different types advertisement on websites. We could select from 100 different advertisements, and we might suspect that the variability across advertisements is large, based simply on the content, rather than on any variable we care about. Suppose there is no difference in the kind of ad. In this case, if we get unlucky, we might pick a subset of 'good' ads for one condition, and 'bad' ads for a second condition, and conclude that the manipulation in ad type actually worked. If we acknowledge that these are a randomized factor, we can account for it and factor out that variability.

Suppose we were trying to test something about how different advertisements impacted on-line behavior, using three kinds of ads, and so we will examine click-through rates for "social" advertisements versus traditional ads.

The experiment works like this: people visit a web site we have created, and it contains different types of advertisements we have identified. For each person, we get end up getting a server log indicating whether they clicked on or hovered their mouse over each type of advertisement. We want to know whether the social ads (ones sensitive to their on-line identity) were better than the non-social advertisements, or maybe from a third type which includes animated video ads.

Let's suppose there is actually no difference between social and normal styles of ads, but animated video ads are better. We can simulate the population of click-through rates like this–normal and social range in their effectiveness between 0 and 1 uniformly, but animated

range in their clickthrough from .6 (150/250) to 1.0:

```
ads.social <- 0:100/100
ads.normal <- 0:100/100
ads.animated <- 150:250/250
```

Now, suppose that in an experiment, we have selected four advertisements of each type, and created a web page including all advertisements. In the study, each of 50 people either does or does not follow each link. Ignoring for the moment within-subject effects, we have a data set like this:

```

set.seed(100)
social.eff <- sample(ads.social,4)
normal.eff <- sample(ads.normal,4)
animated.eff <- sample(ads.animated,4)

##100 of each ad type
clicks <- c(runif(50*4)<social.eff, runif(50*4)<normal.eff, runif(50*4) <
    animated.eff) ##600 long; in
type <- as.factor(rep(c("A","B","C"),
                      each=200))             ## 600 long
ad <- as.factor(rep(rep(1:4,50),3))
ad2 <- as.factor(c(rep(1:4,50),rep(11:14,50), rep(21:24,50)))
              ##ad id; in blocks of type.

##
sub <- as.factor(rep(rep(1:50,each=4),3))

##Set the contrasts to be sum-to-zero so they are orthogonal
contrasts(type) <- contr.sum(3)
contrasts(ad) <- contr.sum(length(unique(ad)))
contrasts(ad2) <- contr.sum(length(unique(ad2)))
contrasts(sub) <- contr.sum(50)

ads <- data.frame(sub=sub,type=type,ad=ad,ad2=ad2,clicks=clicks)

ads[1:10,]

> ads[1:10,]
   sub type ad ad2 clicks
1    1    A  1   1   TRUE
2    1    A  2   2   TRUE
3    1    A  3   3   TRUE
4    1    A  4   4  FALSE
5    2    A  1   1  FALSE
6    2    A  2   2   TRUE
7    2    A  3   3   TRUE
8    2    A  4   4  FALSE
9    3    A  1   1  FALSE
10   3    A  2   2   TRUE


> table(ads$ad,ads$ad2)

   1  2  3  4 11 12 13 14 21 22 23 24
1 50  0  0  0 50  0  0  0 50  0  0  0
2  0 50  0  0  0 50  0  0  0 50  0  0
3  0  0 50  0  0  0 50  0  0  0 50  0
4  0  0  0 50  0  0  0 50  0  0  0 50
>
```

In this data set, we have 50 participants, each with 12 rows of data. Type indicates the type of advertisement. We have numbered ad twice–The first (ad) represents if we had the same product appearing in 3 different ads, and we have sampled 4 ads for this. The second (ad2) is one where we have sampled 4 different advertisements in each role–12 distinct advertisements and 3 in each type. In one case, advertisement is crossed with type, and in the other case it is nested within type, and we may want to handle it differently.

We are also in the situation where we have two random factors: subject and advertisement. We have essentially sampled advertisements from the population of possible advertisements, and we want to see if it has an impact on the population of advertisements. At the end of our statistical test, we would like to generalize to the population of people we sampled from, and to the population of advertisements we sampled from.

But let's start by imagining that advertisement is not a random factor, and we want to look at just type. In the previous chapter, we would have done this:

```
1
> aggregate(ads$clicks,list(ads$type),mean)
3   Group.1    x
1        A 0.63
5 2        B 0.53
  3        C 0.82
7
model1 <-   aov(clicks~type+Error(sub/type),data=ads)
9
Error: sub
11 Df Sum Sq Mean Sq F value Pr(>F)
Residuals 49  6.473  0.1321
13
Error: sub:type
15 Df Sum Sq Mean Sq F value   Pr(>F)
type       2   8.68   4.340   32.75 1.28e-11 ***
17 Residuals 98  12.99   0.133
---
19 Signif. codes:  0 ‘***‘ 0.001 ‘**‘ 0.01 ‘*‘ 0.05 ‘.‘ 0.1 ‘ ‘ 1

21 Error: Within
Df Sum Sq Mean Sq F value Pr(>F)
23 Residuals 450  106.5  0.2367
```

The ad type had a huge effect here, and we are correctly accounting for the repeated-measures design. But we cannot know if the large effect of type is because all three levels differ, or just two of them, or what. The inability to use a post-hoc test with the `Error()` function prevents us from drawing the conclusions we want to make. The typical advise is to use pairwise t-tests with a correction:

```
pairwise.t.test(clicks,type,data=ads)
2
  Pairwise comparisons using t tests with pooled SD
4
data:  clicks and type
6
   A       B
8 B  0.03    -
  C  8.1e-05 1.6e-09
```

```
10
data:  clicks and type
```

This shows that each pair differs significantly. This is a bit worrisome because we created the data so that A and B should not differ, but A/B would differ from C. But by treating advertisement as a fixed effect, we have asserted that advertisement in this case is not random (maybe we are considering a specific ad campaign with specific advertisements, so the significant result is probably fine. But what if we were to try to consider the random effect of ad. We can incorporate that in a model instead of participant:

```
2 #type is a between-ad variable
  model2 <- aov(clicks ~ type + Error(ad))
4
  > summary(model2)
6

8 Error: ad
  Df Sum Sq Mean Sq F value Pr(>F)
10 Residuals  3  17.61   5.871

12 Error: Within
  Df Sum Sq Mean Sq F value   Pr(>F)
14 type        2   8.68   4.340   23.79 1.15e-10 ***
  Residuals 594 108.35   0.182
16 ---
```

The type variable is significant when considering advertisement a randomized factor. This means that with respect to the variability within advertisement, the type seems to have a large enough effect. We'd like to take both random factors into account, but there is no simple way to do that within `Error()`, and again no way of knowing which levels of type differ.

Traditionally, and even today, researchers fail to account for the fact that they are sampling from stimuli, and often do not treat their stimulus as a randomized factor (See Clark, 1974, "The Language-as-fixed-effect fallacy.").

But what do you do? According to Clark, you should use minF:

$$F_{min} = F1xF2/(F1 + F2) \tag{21.1}$$

Here, model2 is F1 (treating subject as a random factor) and model3 is F2.

Although people rarely report minf anymore, the linguistics community has developed advanced mixed-effects and hierarchical effects models to help model the different sources of variance explicitly. The most cutting-edge libraries for this, such as lme4, are available only in R.

21.1 Mixed effects models: A modern approach

There are several libraries in R that handle mixed effects models directly. The most prominent are the `lme4`[1] library , and `nlme`. There are many other libraries based on these, including `BayesFactor`, which provides means for testing lme4 models using Bayes Factor

[1] See Bates (2010). Lme4: Mixed-effect modeling with R http://lme4.r-forge.r-project.org/book/front.pdf

tests, and `multcomp`, which allows for post-hoc multiple comparison tests such as the Tukey HSD based on lme4 or nlme models. The `ez` library has a function that will build lme4 models for you as well.

There is a comprehensive comparison of different approaches to fitting linear mixed models at http://glmm.wikidot.com/pkg-comparison–both with R and with other tools. We will focus on lme4, which seems to have broader use in social science/psychology.

With lme4 and nlme, we specify random effect in different ways. In nlme, we need to specify these as an argument to the lme model. Note that if we have multiple random factors, they get specified as a list. Both models support nested effects, so a pure random effects is specified as `1|effect`, indicating it is random in its intercept. Let's start with the equivalent of the subject-only random factor, comparing oh lme and lmer:

```
library(nlme)
am2 <- lme(clicks ~ type, random = (~1|sub), data=ads)  #subject and
    randomized factor

library(lme4)
lmer2 <- lmer(clicks~type + (1|sub),data=ads)
```

Here, we treat subject as a random factor. Be sure to use the parentheses around the random effects. These two models specify the random effect as (1—sub). This indicates that there is a random effect of subject on the intercept–each subject can have their own additive difference from the population model. The results of the two models are fairly similar:

```
> summary(am2)
Linear mixed-effects model fit by REML
Data: ads
AIC      BIC    logLik
793.401 815.3606 -391.7005

Random effects:
Formula: ~1 | sub
(Intercept)  Residual
StdDev: 1.511953e-05 0.4593346

Fixed effects: clicks ~ type
Value  Std.Error  DF  t-value p-value
(Intercept)  0.66 0.01875226 548 35.19576  0.0000
type1       -0.03 0.02651969 548 -1.13123  0.2585
type2       -0.13 0.02651969 548 -4.90202  0.0000
Correlation:
      (Intr) type1
type1  0.0
type2  0.0   -0.5

Standardized Within-Group Residuals:
Min         Q1        Med         Q3        Max
-1.7851910 -1.1538430  0.3918712  0.8055130  1.0232192

Number of Observations: 600
Number of Groups: 50



> summary(lmer2)
Linear mixed model fit by REML ['lmerMod']
Formula: clicks ~ type + (1 | sub)
```

```
Data: ads

REML criterion at convergence: 783.4

Scaled residuals:
Min      1Q  Median      3Q     Max
-1.7852 -1.1538  0.3919  0.8055  1.0232

Random effects:
Groups   Name        Variance Std.Dev.
sub      (Intercept) 0.000    0.0000
Residual             0.211    0.4593
Number of obs: 600, groups:  sub, 50

Fixed effects:
Estimate Std. Error t value
(Intercept)  0.66000    0.01875  35.196
type1       -0.03000    0.02652  -1.131
type2       -0.13000    0.02652  -4.902

Correlation of Fixed Effects:
       (Intr) type1
type1  0.000
type2  0.000 -0.500
optimizer (nloptwrap) convergence code: 0 (OK)
boundary (singular) fit: see ?isSingular
```

Notice that it shows us the regression coefficients, and so using the right contrasts for our regression might be helpful here in testing our particular hypotheses.

21.1.1 Interpreting the linear mixed effects models

Let's look at each of the sections of the lmer4 output and discuss what these entail.

```
Linear mixed model fit by REML ['lmerMod']
Formula: clicks ~ type + (1 | sub)
   Data: ads


REML criterion at convergence: 783.4

Scaled residuals:
Min      1Q  Median      3Q     Max
-1.7852 -1.1538  0.3919  0.8055  1.0232
```

This is basically book-keeping, but it tells us we let each subject have its own intercept, and we treat these are random variables. It also tells us the optimization criteria it used, which is REML. This is a form of likelihood, and different lme models use different metrics. Also, we can look at the residuals and maybe conclude that they are reasonably symmetric. Next, we have the random effects:

```
Random effects:
Groups   Name        Variance Std.Dev.
sub      (Intercept) 0.000    0.0000
Residual             0.211    0.4593
Number of obs: 600, groups:  sub, 50
```

Here, the output is simple: just the subject intercept and the residual variance. Note that because we have incorporated these as random effects, we do not get beta-weight estimates– just the variance/standard deviation. We are estimating the variability of these, in order to account for that in the model.

We only test the fixed effects:

```
Fixed effects:
            Estimate Std. Error t value
(Intercept)  0.66000    0.01875  35.196
type1       -0.03000    0.02652  -1.131
type2       -0.13000    0.02652  -4.902
```

Here, we get the equivalent of the regression coefficients and tests. Because type was coded with sum-to-zero contrasts, these are a bit tricky to interpret. Intercept is the grand mean, type1 and type2 are the differences from the grand mean, and thus type3 is the opposite value, which is +.03+.13 = .16. To see difference between pairs we would need to either reset the factors or do a post-hoc test.

Finally, the correlations between fixed effects are reported. These correlations really just come from the contrasts directly because we have a balanced design. Note that the correlations with the intercept are 0, and type1 vs type2 have a correlation of -.5 which is by design. Large correlations between covariates or independent variables are typically bad– they indicate that the effect on one could be accounted for by the other. However, because these are just sub-elements of a designed contrast set, they necessarily have these values.

```
Correlation of Fixed Effects:
        (Intr) type1
 type1  0.000
 type2  0.000 -0.500
```

The fixed effects of type are quite similar here. Because we used sum-to-zero coding, we really don't have any pairwise comparisons, but it does tell us whether each of two values is different from the mean, which is useful (although in the case of lmer, we have to look up the p-value). But we can also use the `multcomp` library to compute family-wise post-hoc tests of either of these models (but it won't work for `aov` models with `Error` terms). We can use the `glht` function to specify tests, but we need to do that in contrasts of the fixed effects.

Specifying post-hoc contrasts is a bit tricky and depends on how the contrasts were set up in the factor. Below, we can see the contrasts associated with this factor

```
> contrasts(type)
   [,1] [,2]
A    1    0
B    0    1
C   -1   -1
```

These columns represent the second two factors in a three-factor coding, where the first factor (not shown here) is the grand mean of the data. So to get the grand mean/intercept, we want to add the intercept to neither fo these, or a weighted sum of c(1,0,0). To get A alone, we need the intercept plus the first column (`c(1,1,0)`), to get B alone we need

intercept plus the second column (`c(1,0,1)`), and to get C alone, we need the intercept plus the reverse of the other two (`c(1,-1,-1)`)).

Getting pairwise differences is just subtracting the codings for each variable alone from one another. Below, seven interesting contrast tests are extracted, and we can test them with multcomp:

```
library(multcomp)
contrasts <- rbind(
   "Grand mean/intercept"=c(1,0,0),
   "A alone"= c(1,1,0),
   "B alone"= c(1,0,1),
   "C alone"= c(1,-1,-1),
   "A to B" = c(0,1,-1),
   "A to C" = c(0,2,1),
   "B to C" = c(0,1,2)
   )

> summary(glht(am2,contrasts))

Simultaneous Tests for General Linear Hypotheses

Fit: lme.formula(fixed = clicks ~ type, data = ads, random = (~1 |
sub))

Linear Hypotheses:
             Estimate Std. Error z value Pr(>|z|)
A alone == 0  0.63000    0.03248  19.397   <0.001 ***
B alone == 0  0.53000    0.03248  16.318   <0.001 ***
C alone == 0  0.82000    0.03248  25.246   <0.001 ***
A to B == 0   0.10000    0.04593   2.177    0.119
A to C == 0  -0.19000    0.04593  -4.136   <0.001 ***
B to C == 0  -0.29000    0.04593  -6.313   <0.001 ***

 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
 (Adjusted p values reported -- single-step method)
```

Notice that this adjusts p-values, probably with something like a Bonferonni correction. Because of this, the number of contrasts we specify matters. Let's try again with just the pairwise:

```
contrasts <- rbind(
       "A to B" = c(0,1,-1),
       "A to C" = c(0,2,1),
       "B to C" = c(0,1,2))


summary(glht(lmer2,contrasts))

Simultaneous Tests for General Linear Hypotheses

Fit: lmer(formula = clicks ~ type + (1 | sub), data = ads)

Linear Hypotheses:
            Estimate Std. Error z value Pr(>|z|)
A to B == 0  0.10000    0.04593   2.177 0.075270 .
A to C == 0 -0.19000    0.04593  -4.136 0.000118 ***
B to C == 0 -0.29000    0.04593  -6.313  < 1e-04 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

Notice that the Z-values do not change but the p(z) does, because it is correcting for the number of tests we do. If we want all pairwise comparisons, the Tukey post-hoc is the best approach anyway. Here, 'type' is the name of the variable we want to apply the Tukey correction to:

```

tukey <- glht(am2,
           linfct=mcp(type="Tukey"))
print(tukey)


General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Linear Hypotheses:
Estimate
B - A == 0    -0.10
C - A == 0     0.19
C - B == 0     0.29


summary(tukey)

 Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lme.formula(fixed = clicks ~ type, data = ads, random = (~1 |
sub))

Linear Hypotheses:
             Estimate Std. Error z value Pr(>|z|)
B - A == 0 -0.10000    0.04593  -2.177   0.0752 .
C - A == 0  0.19000    0.04593   4.136   <0.001 ***
C - B == 0  0.29000    0.04593   6.313   <0.001 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

Here, both models give identical results, and they show that C differs from A and B, but A and B do not differ significantly. We can also specify tukey comparisons, which will correct for multiple comparisons.

Notice the p-values appear identical to the hand-built contrasts before when we looked at just the 3 pairwise comparisons. But if we look a little earlier, the results correct for the number of tests when we specified additional contrasts, and the larger set of contrasts lead to different final probabilities. Ihe probability appears to be calculated with some sampling scheme, so each time it is run the `Pr(>|z|)` values are a bit different.

This is fine for testing pairwise contrasts, but this is assuming we only care about these particular ads we tested, and are not trying to generalize. But that is not likely to be the case. To do this right, we need to incorporate a random effect for ad as well:

```
am4 <- lme(clicks~type,random=list((~1|ad),(~1|sub)),data=ads) ##both
lmer4 <- lmer(clicks~type + (1|ad) + (1|sub),data=ads)


> summary(am4)
Linear mixed-effects model fit by REML
Data: ads
AIC      BIC    logLik
718.8982 745.2497 -353.4491

Random effects:
Formula: ~1 | ad
(Intercept)
StdDev:   0.1947427

Formula: ~1 | sub %in% ad
(Intercept)  Residual
StdDev: 1.939485e-05 0.4270852

Fixed effects: clicks ~ type
Value  Std.Error  DF   t-value p-value
(Intercept)  0.66 0.09892010 398  6.672052  0.0000
type1       -0.03 0.02465778 398 -1.216655  0.2245
type2       -0.13 0.02465778 398 -5.272170  0.0000
Correlation:
(Intr) type1
type1  0.0
type2  0.0   -0.5

Standardized Within-Group Residuals:
Min         Q1        Med         Q3        Max
-2.1014882 -0.9536002  0.3218473  0.7088314  1.3878528

Number of Observations: 600
Number of Groups:
ad sub %in% ad
4         200



> summary(lmer4)
Linear mixed model fit by REML ['lmerMod']
Formula: clicks ~ type + (1 | ad) + (1 | sub)
Data: ads

REML criterion at convergence: 706.9

Scaled residuals:
Min      1Q  Median      3Q     Max
-2.1015 -0.9536  0.3219  0.7088  1.3879

Random effects:
Groups   Name        Variance Std.Dev.
sub      (Intercept) 0.00000  0.0000
ad       (Intercept) 0.03792  0.1947
Residual             0.18240  0.4271
Number of obs: 600, groups:  sub, 50; ad, 4

Fixed effects:
Estimate Std. Error t value
(Intercept)  0.66000    0.09892   6.672
```

```
type1       -0.03000    0.02466  -1.217
type2       -0.13000    0.02466  -5.272

Correlation of Fixed Effects:
(Intr) type1
type1  0.000
type2  0.000 -0.500
optimizer (nloptwrap) convergence code: 0 (OK)
boundary (singular) fit: see ?isSingular
```

Now, we can incorporate both random effects. If we look at the results of the last model using summary(), we see that again, the two models agree. We can do the same glht tests here too:

```
> summary(glht(am4,contrasts))

   Simultaneous Tests for General Linear Hypotheses

Fit: lme.formula(fixed = clicks ~ type, data = ads, random = list((~1 |
ad), (~1 | sub)))

Linear Hypotheses:
            Estimate Std. Error z value Pr(>|z|)
A to B == 0 -0.03000    0.02466  -1.217   0.3778
A to C == 0 -0.13000    0.02466  -5.272   <0.001 ***
B to C == 0 -0.10000    0.04271  -2.341   0.0392 *
---

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)

> summary(glht(lmer4,contrasts))
   Simultaneous Tests for General Linear Hypotheses

Fit: lmer(formula = clicks ~ type + (1 | ad) + (1 | sub), data = ads)
Linear Hypotheses:
            Estimate Std. Error z value Pr(>|z|)
A to B == 0 -0.03000    0.02466  -1.217   0.3777
A to C == 0 -0.13000    0.02466  -5.272   <0.001 ***
B to C == 0 -0.10000    0.04271  -2.341   0.0395 *
---

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

Again, we have highly significant differences between all three advertisement types. But on average, A and B ads were no different, we must have gotten (un)lucky and sampled advertisements that happened to differ a lot. If you run the entire data set again without the set.seed command, you can see how often A and B differ significantly, and it is not very common.

Finally, maybe, we want to be a bit more conservative here and consider the advertisement to be nested within method. Here, we would not link same ad to multiple types. It would treat each advertisement as unique and sampled independently from ads of that type. The type—ad and 1—ad arguments essentially specify that advertisement and the type:advertisement interaction are random effects. Type is a fixed effect, but the particular advertisement we selected is sampled from advertisements of that type.

```
1
am5 <- lme(clicks~type,random=list(~1|ad2,~type|ad2,~1|sub),data=ads) ##both,
    ad nested within type
3 lmer5 <- lmer(clicks~ type + (1|ad2) + (type|ad2) + (1|sub),data=ads)
```

The lme model:

```
1 > summary(am5)
  Linear mixed-effects model fit by REML
3 Data: ads
  AIC      BIC    logLik
5 590.2903 642.9933 -283.1451

7 Random effects:
  Formula: ~1 | ad2
9 (Intercept)
  StdDev:   0.1019386
11
  Formula: ~type | ad2 %in% ad2
13 Structure: General positive-definite, Log-Cholesky parametrization
             StdDev    Corr
15 (Intercept) 0.1964997 (Intr) type1
  type1       0.1408065 0.751
17 type2       0.1855748 0.845  0.563

19 Formula: ~1 | sub %in% ad2 %in% ad2
  (Intercept)  Residual
21 StdDev:   0.3467419 0.1384117

23 Fixed effects: clicks ~ type
              Value  Std.Error  DF   t-value p-value
25 (Intercept)  0.66 0.09026704 588  7.311639  0.0000
  type1       -0.03 0.13259510   9 -0.226253  0.8261
27 type2       -0.13 0.14307455   9 -0.908617  0.3872
  Correlation:
29 (Intr) type1
  type1  0.107
31 type2  0.323 -0.717

33 Standardized Within-Group Residuals:
  Min          Q1         Med          Q3         Max
35 -0.92583689 -0.17037318  0.06716072  0.24262530  0.82262443

37 Number of Observations: 600
  Number of Groups:
39 ad2          ad2.1 %in% ad2 sub %in% ad2.1 %in% ad2
  12                  12                     600
41 > summary(glht(am5,contrasts))

43 Simultaneous Tests for General Linear Hypotheses

45 Fit: lme.formula(fixed = clicks ~ type, data = ads, random = list(~1 |
  ad2, ~type | ad2, ~1 | sub))
47
  Linear Hypotheses:
49 Estimate Std. Error z value Pr(>|z|)
  A to B == 0  -0.0300     0.1326  -0.226    0.955
51 A to C == 0  -0.1300     0.1431  -0.909    0.523
  B to C == 0  -0.1000     0.2555  -0.391    0.874
53 (Adjusted p values reported -- single-step method)
```

the lmer model:

```
> summary(lmer5)
Linear mixed model fit by REML ['lmerMod']
Formula: clicks ~ type + (1 | ad2) + (type | ad2) + (1 | sub)
Data: ads

REML criterion at convergence: 566.3

Scaled residuals:
Min      1Q  Median      3Q     Max
-2.4973 -0.4596  0.1812  0.6544  2.2189

Random effects:
Groups   Name        Variance Std.Dev. Corr
sub      (Intercept) 0.000000 0.00000
ad2      (Intercept) 0.008901 0.09434
type1       0.074154 0.27231   0.29
type2       0.118691 0.34452   0.08 -0.88
ad2.1    (Intercept) 0.012544 0.11200
Residual             0.139388 0.37335
Number of obs: 600, groups:  sub, 50; ad2, 12

Fixed effects:
             Estimate Std. Error t value
(Intercept)  0.66000    0.09026   7.312
type1       -0.03000    0.13259  -0.226
type2       -0.13000    0.14307  -0.909

Correlation of Fixed Effects:
(Intr) type1
type1  0.107
type2  0.323 -0.717
optimizer (nloptwrap) convergence code: 0 (OK)
boundary (singular) fit: see ?isSingular

> summary(glht(lmer5,contrasts))

Simultaneous Tests for General Linear Hypotheses

Fit: lmer(formula = clicks ~ type + (1 | ad2) + (type | ad2) + (1 |
sub), data = ads)

Linear Hypotheses:
Estimate Std. Error z value Pr(>|z|)
A to B == 0  -0.0300     0.1326  -0.226    0.955
A to C == 0  -0.1300     0.1431  -0.909    0.523
B to C == 0  -0.1000     0.2555  -0.391    0.874
(Adjusted p values reported -- single-step method)
```

Again, we see very similar results from the two models. Now, however, none of the post-hoc test were significant. This makes sense because we have only sampled four advertisements from the world of possible advertisements. Some are better than others, and we are likely to sample a few in one condition that are better than in another condition, not because of the condition but just because of the variability in the ad. This nested model takes this into

account. When we did not include it, ad served more like a paired-samples t-test: we had the ability to detect small differences because we tested the same three delivery methods on the same ad content.

21.1.2 Exercise: Chick Weight

Let's consider the chick weight example again. We previously found that the outcome log(wt) was somewhat linear. We can consider the fixed effects (time, diet) and the random effects (chick, and chick x time interaction).

```
1
cw <- ChickWeight
3 cw$logwt <- log(ChickWeight$weight)

5 lmer.cw0 <- lmer(logwt~Time*Diet + (1|Chick), data=cw )
summary(lmer.cw0)
7
 summary(lmer.cw0)
9 Linear mixed model fit by REML ['lmerMod']
Formula: logwt ~ Time * Diet + (1 | Chick)
11    Data: cw

13 REML criterion at convergence: -296

15 Scaled residuals:
     Min      1Q  Median      3Q     Max
17 -4.2030 -0.5479  0.1269  0.6332  2.8654

19 Random effects:
 Groups   Name        Variance Std.Dev.
21 Chick    (Intercept) 0.02580  0.1606
 Residual             0.02585  0.1608
23 Number of obs: 578, groups:  Chick, 50

25 Fixed effects:
            Estimate Std. Error t value
27 (Intercept) 3.768319   0.041194  91.477
Time        0.067537   0.001639  41.215
29 Diet2       0.048496   0.071074   0.682
Diet3       0.024561   0.071074   0.346
31 Diet4       0.104324   0.071125   1.467
Time:Diet2  0.008219   0.002716   3.026
33 Time:Diet3  0.022093   0.002716   8.134
Time:Diet4  0.014737   0.002751   5.356
35
Correlation of Fixed Effects:
37           (Intr) Time   Diet2  Diet3  Diet4  Tm:Dt2 Tm:Dt3
Time       -0.401
39 Diet2      -0.580  0.232
Diet3      -0.580  0.232  0.336
41 Diet4      -0.579  0.232  0.336  0.336
Time:Diet2  0.242 -0.603 -0.405 -0.140 -0.140
43 Time:Diet3  0.242 -0.603 -0.140 -0.405 -0.140  0.364
Time:Diet4  0.239 -0.596 -0.138 -0.138 -0.406  0.359  0.359
```

This model treats chick as a random factor, and this only impacts the intercept–the mean log-weight. If we want to find out whether diet matters, we can do an anova, comparing it to the model we care about. Let's consider the time diet interaction:

```
1
lmer.cw1 <- lmer(logwt~Time+Diet + (1|Chick),data=cw)
3 > summary(lmer.cw1)
Linear mixed model fit by REML ['lmerMod']
5 Formula: logwt ~ Time + Diet + (1 | Chick)
   Data: cw
7
REML criterion at convergence: -257.4
9
Scaled residuals:
11     Min      1Q  Median      3Q     Max
-4.0531 -0.5329  0.1537  0.6429  2.6851
13
Random effects:
15 Groups   Name        Variance Std.Dev.
 Chick    (Intercept) 0.02443  0.1563
17 Residual             0.02938  0.1714
Number of obs: 578, groups:  Chick, 50
19
Fixed effects:
21            Estimate Std. Error t value
(Intercept) 3.673389   0.038580  95.214
23 Time        0.076998   0.001064  72.361
Diet2       0.129867   0.063729   2.038
25 Diet3       0.257389   0.063729   4.039
Diet4       0.255877   0.063765   4.013
27
Correlation of Fixed Effects:
29      (Intr) Time   Diet2  Diet3
Time  -0.278
31 Diet2 -0.555 -0.014
Diet3 -0.555 -0.014  0.338
33 Diet4 -0.555 -0.011  0.338  0.338


35
anova(lmer.cw0,lmer.cw1)
37 refitting model(s) with ML (instead of REML)
Data: cw
39 Models:
lmer.cw1: logwt ~ Time + Diet + (1 | Chick)
41 lmer.cw0: logwt ~ Time * Diet + (1 | Chick)
         Df     AIC     BIC logLik deviance  Chisq Chi Df Pr(>Chisq)
43 lmer.cw1  7 -272.43 -241.91 143.22  -286.43
lmer.cw0 10 -335.24 -291.64 177.62  -355.24 68.807      3  7.684e-15 ***
45 ---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The summary won't tell us directly whether diet is significant, but reports the regression models. This model has a single slope, and main effects for diet. We are comparing it to a model with different slopes for each diet. The resulting comparison (it uses a chi-squared test) is highly significant here, and so we would prefer the more complex model with an interaction.

But, we can model random effects in more precise ways. Here, the (1+Time—Chick) says that the intercept and slope of chick are random effects. That is, we expect the starting weight and overall weight gain to depend on the chick we sampled.

```
lmer.cw2 <- lmer(logwt~Time*Diet + (1+Time|Chick),data=cw)
2 > summary(lmer.cw2)
Linear mixed model fit by REML ['lmerMod']
```

```
Formula: logwt ~ Time * Diet + (1 + Time | Chick)
   Data: cw

REML criterion at convergence: -681.1

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.1774 -0.5638 -0.0403  0.5825  3.0850

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 Chick    (Intercept) 0.003088 0.05557
          Time        0.000349 0.01868  -0.57
 Residual             0.011374 0.10665
Number of obs: 578, groups:  Chick, 50

Fixed effects:
            Estimate Std. Error t value
(Intercept) 3.785030   0.018386 205.861
Time        0.064705   0.004429  14.608
Diet2       0.031786   0.031428   1.011
Diet3       0.007850   0.031428   0.250
Diet4       0.090166   0.031505   2.862
Time:Diet2  0.011051   0.007522   1.469
Time:Diet3  0.024925   0.007522   3.313
Time:Diet4  0.017153   0.007531   2.278

Correlation of Fixed Effects:
           (Intr) Time   Diet2  Diet3  Diet4  Tm:Dt2 Tm:Dt3
Time       -0.535
Diet2      -0.585  0.313
Diet3      -0.585  0.313  0.342
Diet4      -0.584  0.312  0.341  0.341
Time:Diet2  0.315 -0.589 -0.531 -0.184 -0.184
Time:Diet3  0.315 -0.589 -0.184 -0.531 -0.184  0.347
Time:Diet4  0.315 -0.588 -0.184 -0.184 -0.532  0.346  0.346
```

To test differences, we can use simulation to estimate confidence bounds of each parameter with the `profile` function. This is the best approach if you have correlation between fixed effects. Or, we can apply the glht test as well. If we want to know whether diet2 and diet 3 slopes with respect to time differ (the diet:time interactions), we can see that the bootstrapped confidence regions overlap; the glht test shows they are not significantly different:

```
 pr.cw <- profile(lmer.cw2)
confint(pr.cw)
                    2.5 %      97.5 %
.sig01        0.024009039  0.07615894
.sig02       -0.903401709 -0.18451232
.sig03        0.014522474  0.02240651
.sigma        0.100246669  0.11383234
(Intercept)   3.749651469  3.82016268
Time          0.056222048  0.07323108
Diet2        -0.028296122  0.09212537
Diet3        -0.052231601  0.06818989
Diet4         0.029930207  0.15064761
Time:Diet2   -0.003397207  0.02545495
Time:Diet3    0.010476720  0.03932888
Time:Diet4    0.002688380  0.03157406
```

```
contrasts.cw <- rbind("Diet 3 vs diet 2 by time"=c(0,0,0,0, 0,1,-1,0))
summary(glht(lmer.cw2,contrasts.cw))
   Simultaneous Tests for General Linear Hypotheses

Fit: lmer(formula = logwt ~ Time * Diet + (1 + Time | Chick), data = cw)

Linear Hypotheses:
                                 Estimate Std. Error z value Pr(>|z|)
Diet 3 vs diet 2 by time == 0 -0.013874   0.008599  -1.614    0.107
(Adjusted p values reported -- single-step method)
```

21.2 Using `ezMixed`

Just as the `ez` package provides a straightforward way of implementing aov models with Error() terms, it also provides an avenue to creating lmer models.

```
ezm <- ezMixed(data=ads,
                   dv=.(clicks1),
                   random=.(sub,ad2),
                   fixed=.(type))


print(ezm)
$summary
effect errors warnings      bits
1   type  FALSE    FALSE -2.653758

$formulae
$formulae$type
$formulae$type$restricted
[1] "clicks1 ~ (1|sub) + (1|ad2)"

$formulae$type$unrestricted
[1] "clicks1 ~ (1|sub) + (1|ad2) + type"



$errors
$errors$type
named list()


$warnings
$warnings$type
named list()


$models
$models$type
$models$type$restricted
Linear mixed model fit by maximum likelihood  ['lmerMod']
Formula: clicks1 ~ (1 | sub) + (1 | ad2)
Data: this_data
AIC       BIC    logLik  deviance  df.resid
569.8268  587.4145 -280.9134  561.8268       596
Random effects:
```

```
Groups   Name        Std.Dev.
sub      (Intercept) 0.0000
ad2      (Intercept) 0.2916
Residual             0.3733
Number of obs: 600, groups:  sub, 50; ad2, 12
Fixed Effects:
(Intercept)
0.66
optimizer (nloptwrap) convergence code: 0 (OK) ; 0 optimizer warnings; 1 lme4
    warnings

$models$type$unrestricted
Linear mixed model fit by maximum likelihood  ['lmerMod']
Formula: clicks1 ~ (1 | sub) + (1 | ad2) + type
Data: this_data
AIC       BIC    logLik  deviance  df.resid
571.6662  598.0478 -279.8331  559.6662       594
Random effects:
Groups   Name        Std.Dev.
sub      (Intercept) 0.0000
ad2      (Intercept) 0.2656
Residual             0.3733
Number of obs: 600, groups:  sub, 50; ad2, 12
Fixed Effects:
(Intercept)       type.L       type.Q
0.6600       0.1344       0.1592
optimizer (nloptwrap) convergence code: 0 (OK) ; 0 optimizer warnings; 1 lme4
    warnings
```

This prints out two models, the restricted (random only) and unrestricted (complete) models. Notice that it changes the contrasts of the fixed effects to be polynomial to ensure they are orthogonal to the other models–even though they were already polynomial. It might be useful to use ezMixed as a check to make sure you are formatting your lmer model correctly, because it prints out the formula.

21.3 Summary

This chapter is really just the tip of the iceberg in developing mixed effects models. It is probably the minimal needed to frame and test simple mixed effects models we see in social science, with one or two random effects, and simple or no nested structures. They are difficult to verify whether you are correct, and so you must know how they work to be confident in your results. But the advantage is they give you a lot of power to represent sampling from a number of sources, and (unlike `aov` with `Error()`) more easily permit post-hoc testing.