Using ggplot2

Shane T. Mueller shanem@mtu.edu

2021-01-20

(libraries used include ggplot, ggvis, MASS, reshape2,ggthemes, tidyverse)

Using ggplot2 and qplot: Method Overview

The ggplot2 library is a follow-up of the ggplot library, and stands for the 'grammar of graphics'. It produces attractive, professional-looking graphics that are good, especially for presentations. This comes at a cost of some of the flexibility that standard R graphics give, but it is often worthwhile. The ggplot2 library was developed by Hadley Wickham, who also developed reshape2 and dplyr and the rest of the tidyverse. Because of its peculiarities, you may end up using these other libraries to make full use of its power.

There are a lot of basic introductory tutorials for ggplot out there. See links in the resources below.

Finally, ggplot has pretty nice control over the graphics formats you save. It has functions that will create several different formats directly, with specified dimensions and dpi, so you are not at the whim of the RStudio window for creating your figure resolutions.

There is a similar graphics library being developed by the developers of ggplot2 called ggvis, which intends to make interactive web graphics. If you understand ggplot2, ggvis should be easy to pick up and let you do some exciting things.

Overview of Functions

There are two main ways to use ggplot2. The grammar-of-graphics way (using the ggplot() function) can be difficult to get started with but offers a lot of power. The simple way works more like traditional R graphics (using a function called qplot) is more straight-forwrad, but can be a bit more limiting. You can mix the two as well.

The basic idea of ggplot is that it is a composable grammar that has variables represented deliberately in its structure. In core R graphics, you create lines and points and layers, and what you did is lost. This means if you want to add a legend, you have to manually determine what it means. If you want to add additional adornment to a specific variable, again you do it manually. In ggplot, you create a baseline object using the ggplot function, and then you modify it by adding (with the + sign) additional graphical elements that understand the data you specified. This is a bit strange to get used to, so first we will look at a way to access much of the same output, but without the grammar, and instead with just arguments to the qplot function.

The more traditional way: qplot

`qplot` stands for `quickplot`, and either function name can be used. It wraps up all major plotting methods into one function. If we look at the function definitions, some of the arguments are similar to what appear in the normal `plot()` functions: you specify x and y values, labels, limits, etc. The `geom` argument specifies what type of plot you want to create.

```
qplot(x, y = NULL, ..., data, facets = NULL, margins = FALSE,
  geom = "auto", stat = list(NULL), position = list(NULL), xlim = c(NA,
  NA), ylim = c(NA, NA), log = "", main = NULL,
  xlab = deparse(substitute(x)), ylab = deparse(substitute(y)), asp = NA)
```

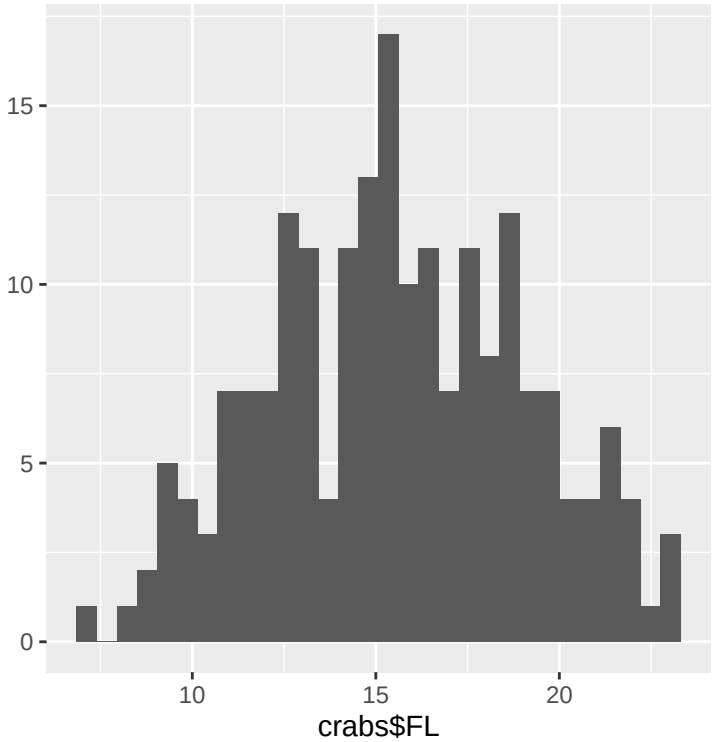
Let's look at the crabs data set in the MASS library. This data set has 200 observations of crabs (100 male and 100 female) on 5 different measures, with two species (Blue or orange). FL=frontal lobe size; RW rear width; CL=carapace length, CW = carapace width, BD = body depth.

```
library(MASS)
library(ggplot2)
library(tidyverse)
head(crabs)
```

```
  sp sex index   FL  RW   CL   CW  BD
1  B   M     1  8.1 6.7 16.1 19.0 7.0
2  B   M     2  8.8 7.7 18.1 20.8 7.4
3  B   M     3  9.2 7.8 19.0 22.4 7.7
4  B   M     4  9.6 7.9 20.1 23.1 8.2
5  B   M     5  9.8 8.0 20.3 23.0 8.2
6  B   M     6 10.8 9.0 23.0 26.5 9.8
```

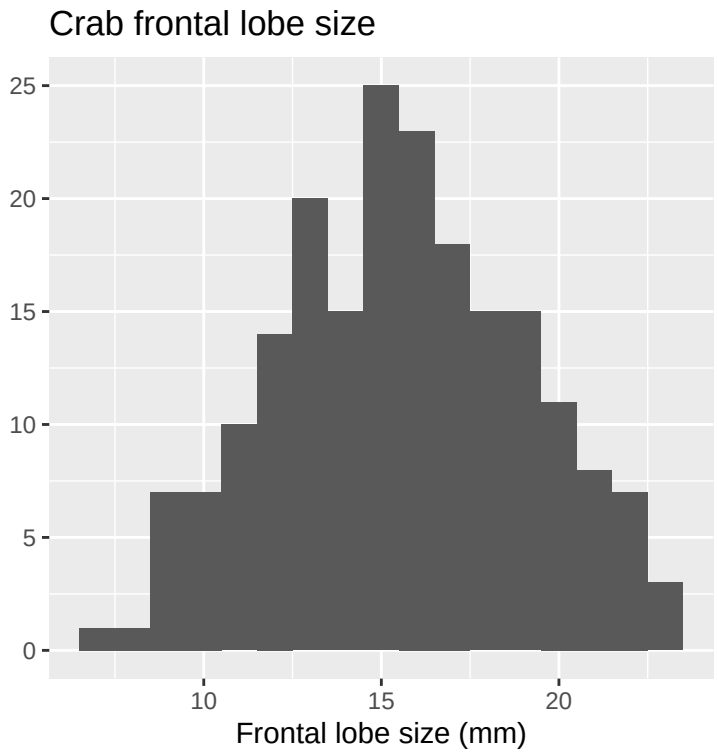
Let's look at the frontal lobe width. Doing qplot with one continuous variable will just plot a histogram.

```
qplot(crabs$FL)
```



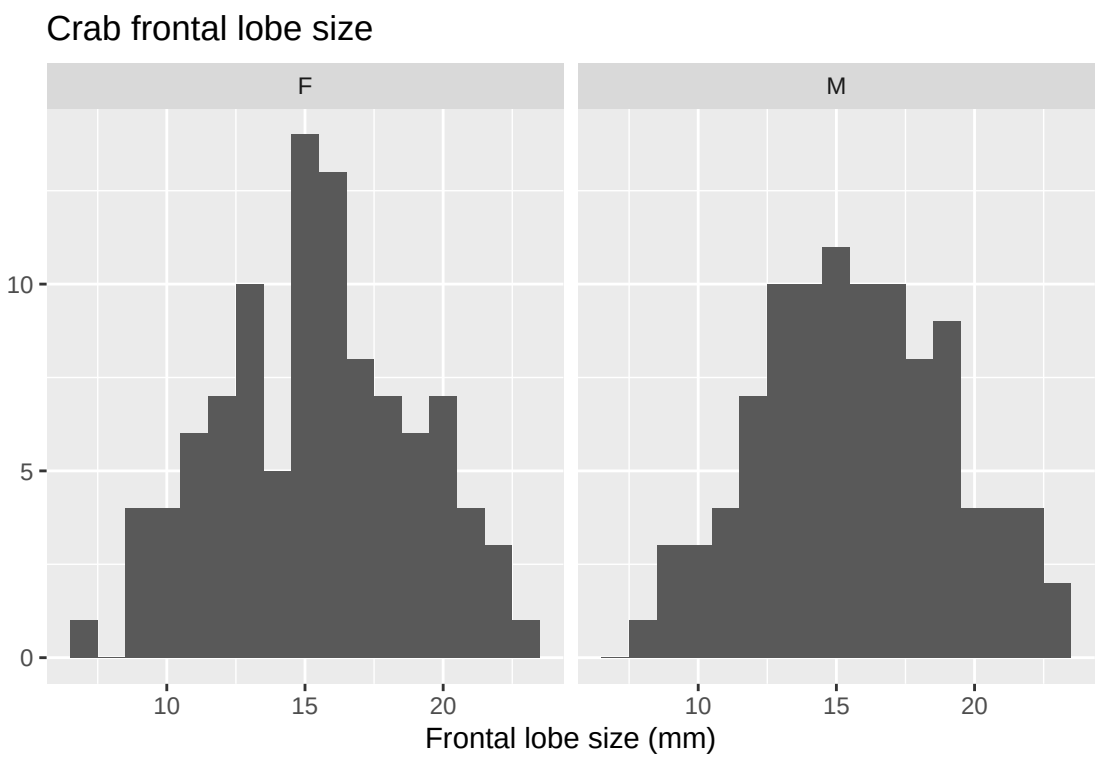
I'm not enamored with how this looks. Let's spruce it up a bit:

```
qplot(FL, data = crabs, xlab = "Frontal lobe size (mm)", main = "Crab frontal lobe size",
    binwidth = 1)
```



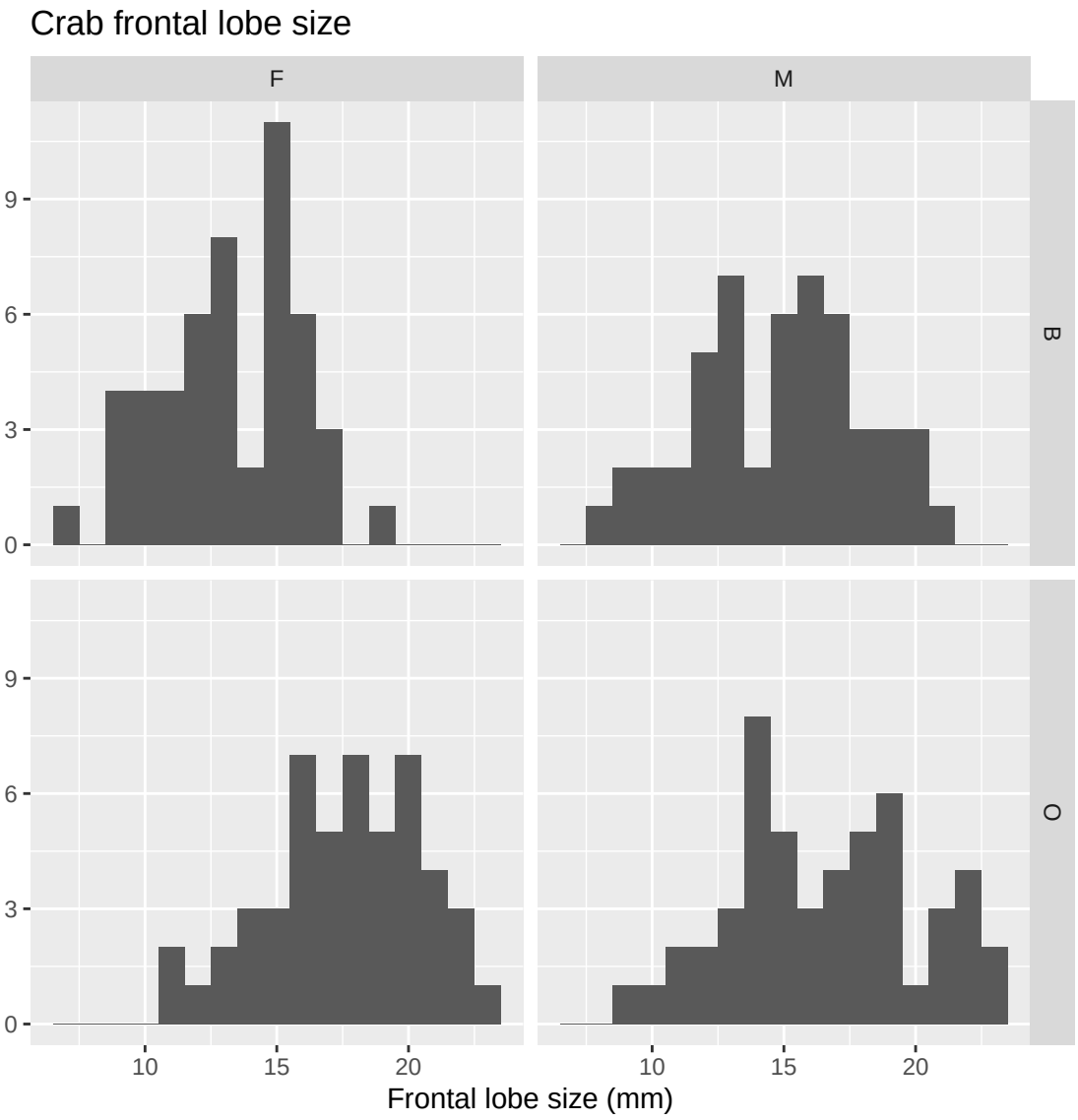
What if we wanted a separate histogram for each species? We can use the facet argument to make separate plots for each level of another variable.

```
qplot(FL, data = crabs, xlab = "Frontal lobe size (mm)", main = "Crab frontal lobe size",
    binwidth = 1, facets = . ~ sex)
```



Or, we can make facets in a grid along two IVs:

```
qplot(FL, data = crabs, xlab = "Frontal lobe size (mm)", main = "Crab frontal lobe size",
    binwidth = 1, facets = sp ~ sex)
```

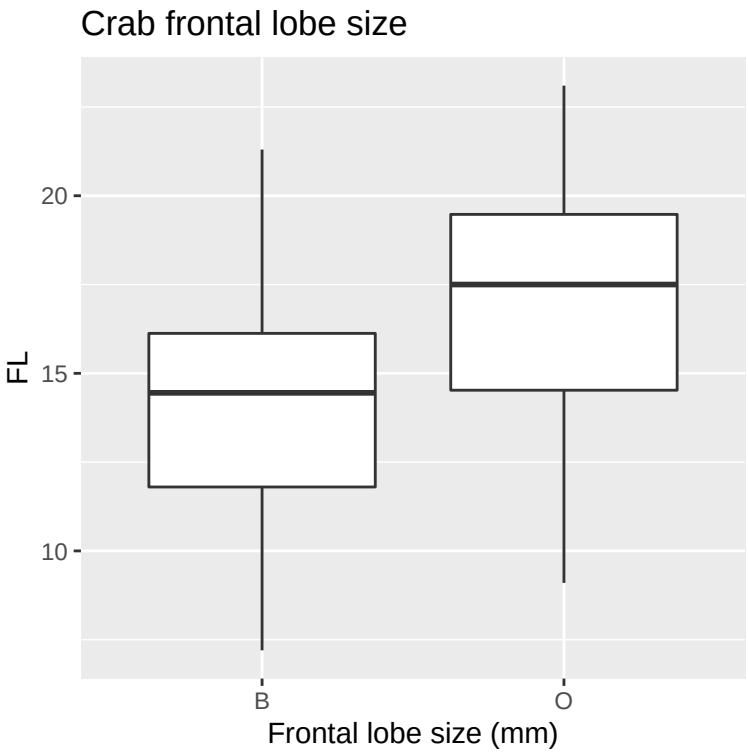


The rows and columns aren't entirely easy to read, but you could change the names of these with dplyr.

boxplots

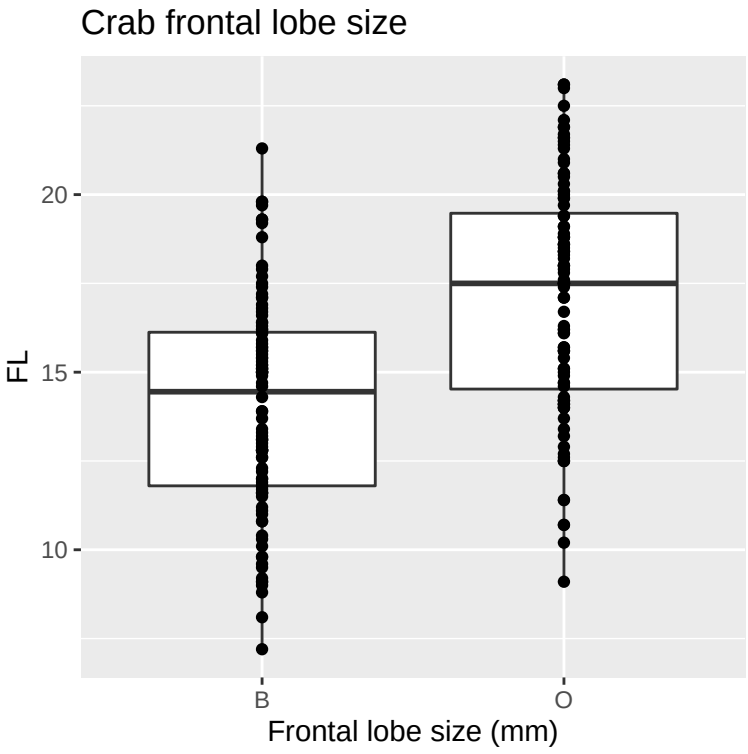
The qplot can create other kinds of graphs by specifying a different 'geom'. For example, a boxplot. We can specify both x and y values here.

```
qplot(y = FL, x = sp, data = crabs, geom = "boxplot", xlab = "Frontal lobe size (mm)",
    main = "Crab frontal lobe size")
```



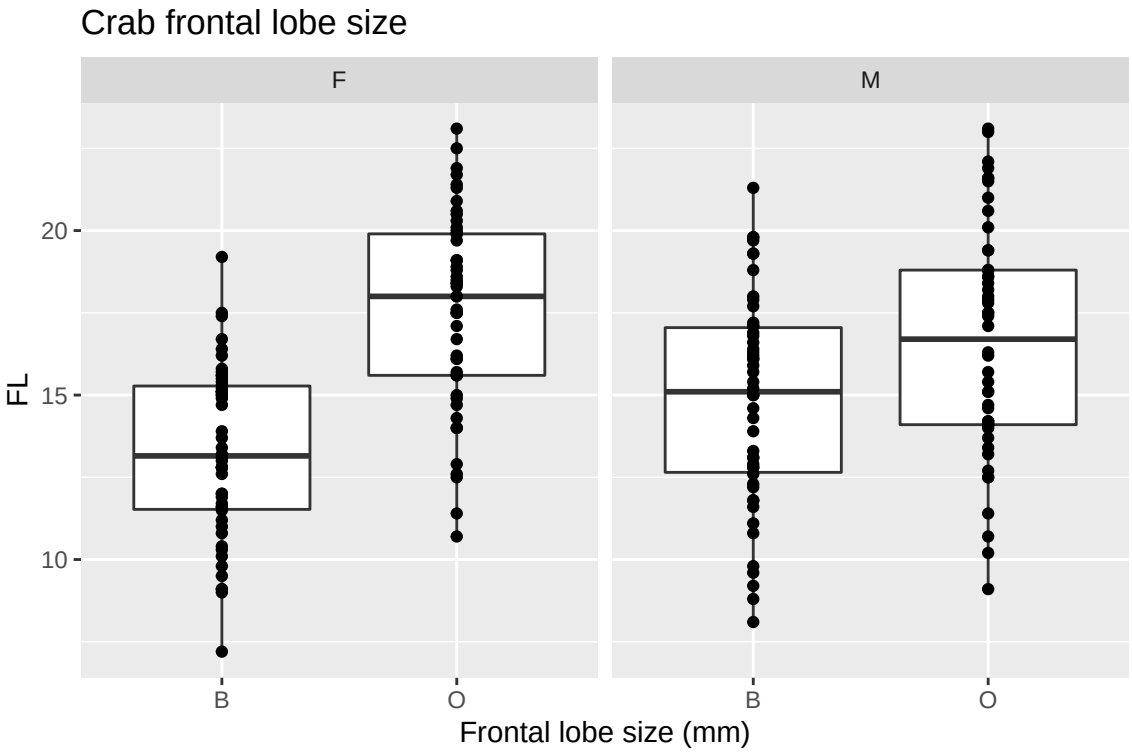
You could add the points on top of these by specifying a list in the goem

```
qplot(y = FL, x = sp, data = crabs, geom = c("boxplot", "point"), xlab = "Frontal lobe size (mm)",
    main = "Crab frontal lobe size")
```



And of course, this could be faceted too:

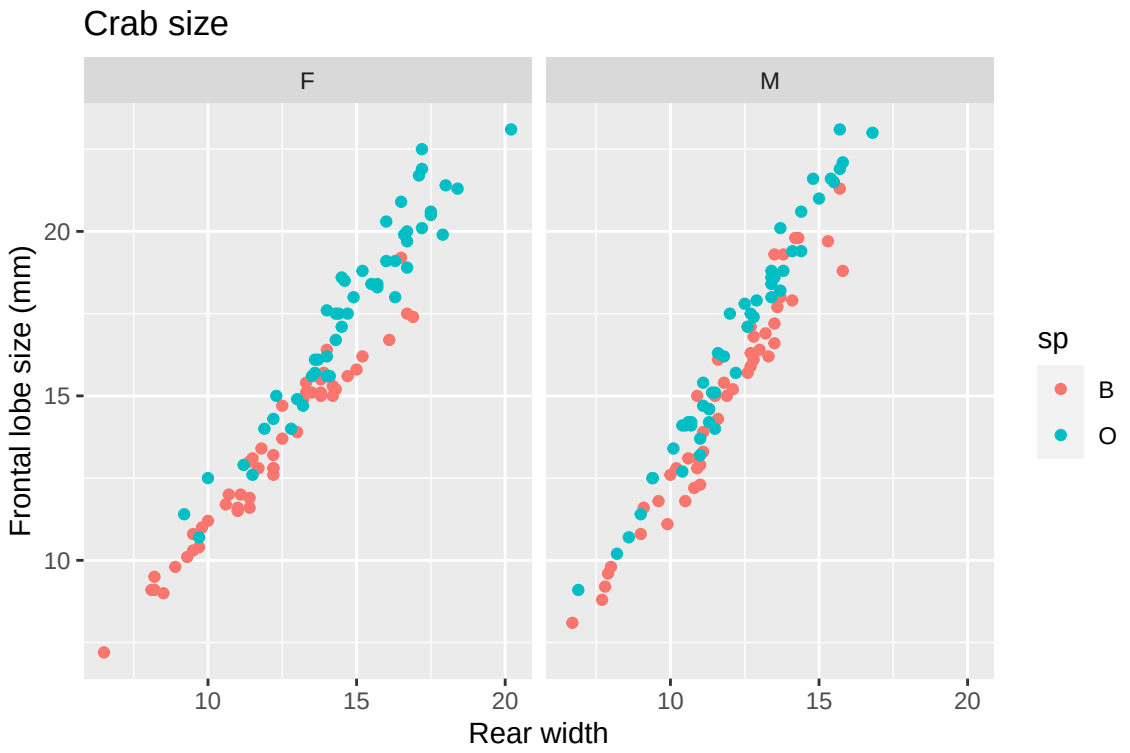
```
qplot(y = FL, x = sp, data = crabs, geom = c("boxplot", "point"), facets = . ~ sex,
    xlab = "Frontal lobe size (mm)", main = "Crab frontal lobe size")
```



Making scatter plots with qplot

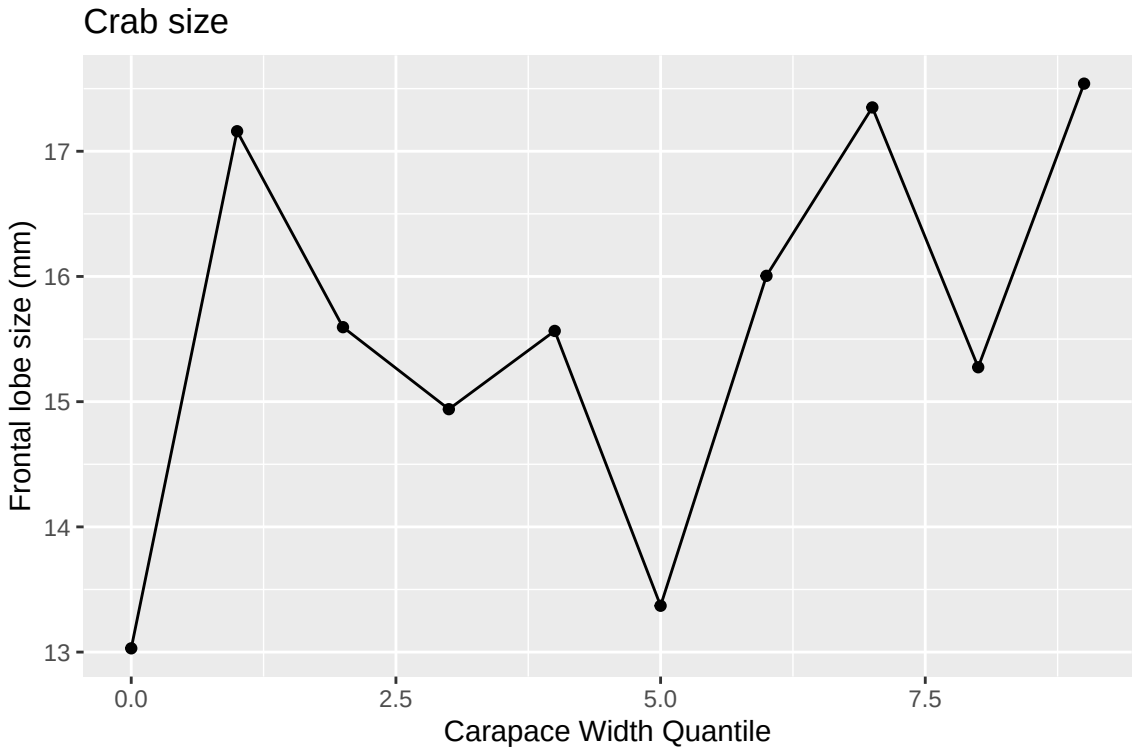
If you want to plot points against one another, you specify an x and y value (again, we can facet by sex). But we will color each one by its species, so we specify that we want colour to use sp (species).

```
qplot(y = FL, x = RW, data = crabs, geom = c("point"), facets = . ~ sex, colour = sp,
    ylab = "Frontal lobe size (mm)", xlab = "Rear width", main = "Crab size")
```



Let's see how frontal size changes as carapace width changes. To do this, let's start by creating quantiles of CW

```
crabs$CW2 <- floor((order(crabs$CW) - 1)/100 * 5)

# agg <- aggregate(crabs$FL,list(crabs$CW2),mean) Instead of using aggregate,
# let's use summarize below

agg <- crabs %>% group_by(CW2) %>% summarize(FL = mean(FL))

qplot(x = CW2, y = FL, data = agg, geom = c("point", "line"), ylab = "Frontal lobe size (mm)",
    xlab = "Carapace Width Quantile", main = "Crab size")
```

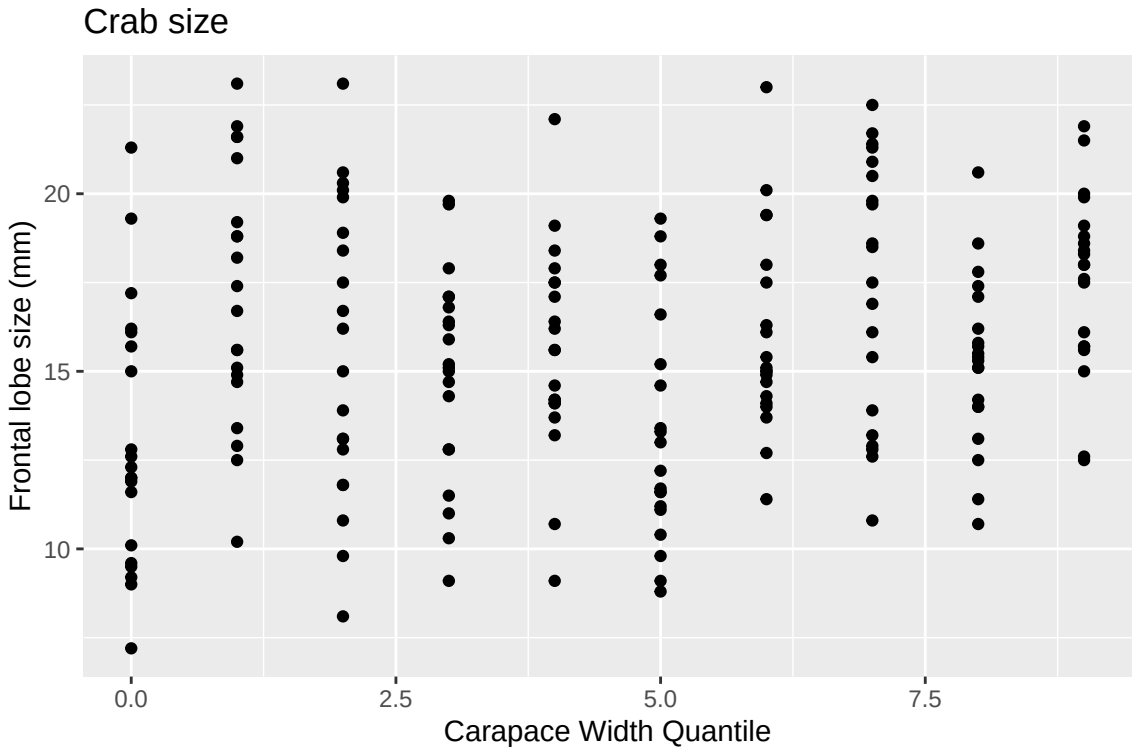


Making barplots with qplot (deprecated)

Bar plots will often want you to create a statistic on the data. Although in the past, qplot could do this from the raw data, qplot can no longer do this (and users are directed to the more complex ggplot()).

```
## Deprecated in ggplot2 2.0: qplot(x=CW2, y=FL,data=crabs, geom='bar')
```

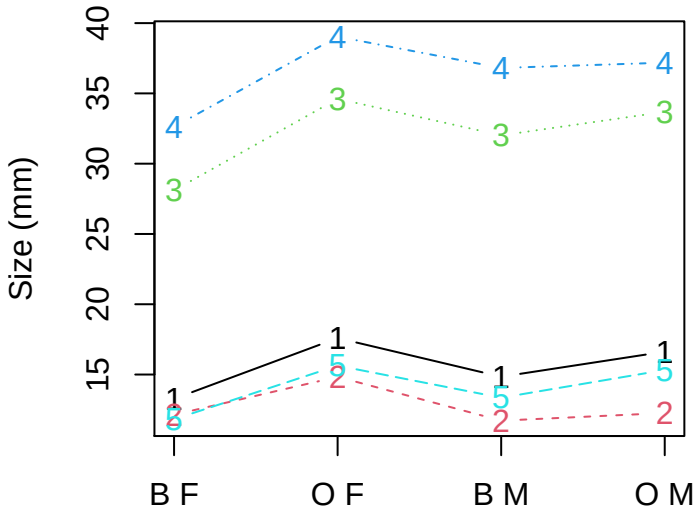
```
qplot(x = CW2, y = FL, data = crabs, geom = c("point"), ylab = "Frontal lobe size (mm)",
    xlab = "Carapace Width Quantile", main = "Crab size")
```



How to make a 'matplot' in ggplot2

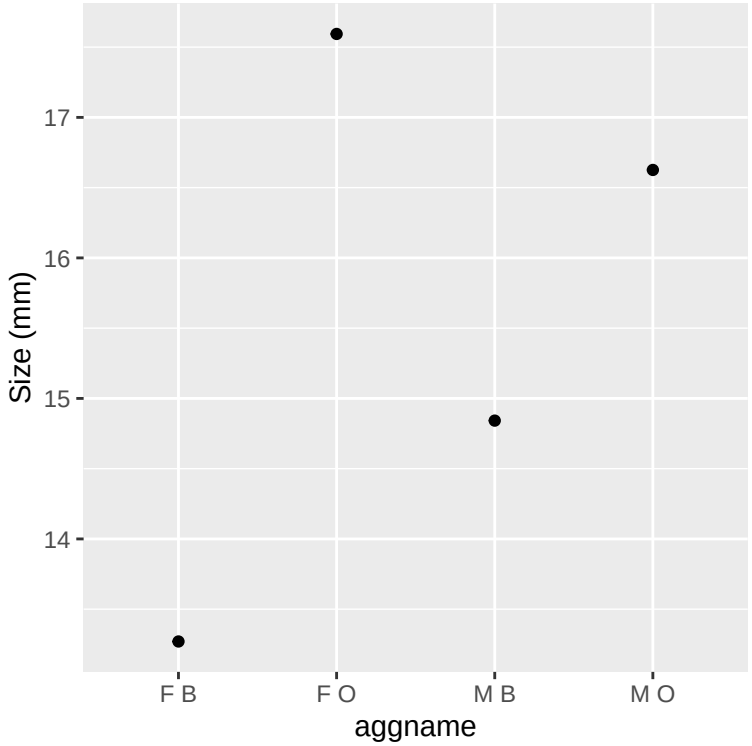
In standard R, the matplot function is handy because it allows you to take a matrix of values and plot several series against each other.

```
library(dplyr)
agg <- aggregate(dplyr::select(crabs, FL, RW, CL, CW, BD, CW), list(species = crabs$sp,
    sex = crabs$sex), mean)
matplot(dplyr::select(agg, FL, RW, CL, CW, BD), type = "b", xaxt = "n", ylab = "Size (mm)")

axis(1, 1:4, labels = paste(agg$species, agg$sex))
```



Let's say we want to do something like this, but with qplot/ggplot. We quickly find out that it is not easy. It doesn't work because ggplot expects a single variable, and you are to tell it how it needs to be plotted. We can do one series, but not multiple:

```
newagg <- dplyr::select(agg, species, sex, FL, RW, CL, CW, BD)
newagg$aggname <- paste(agg$sex, agg$species)
qplot(x = aggname, y = FL, data = newagg, ylab = "Size (mm)")
```



To do multiple series, we can rely on melt in reshape2

```
library(reshape2)
head(melt(newagg))  ##this basically does it right!
```

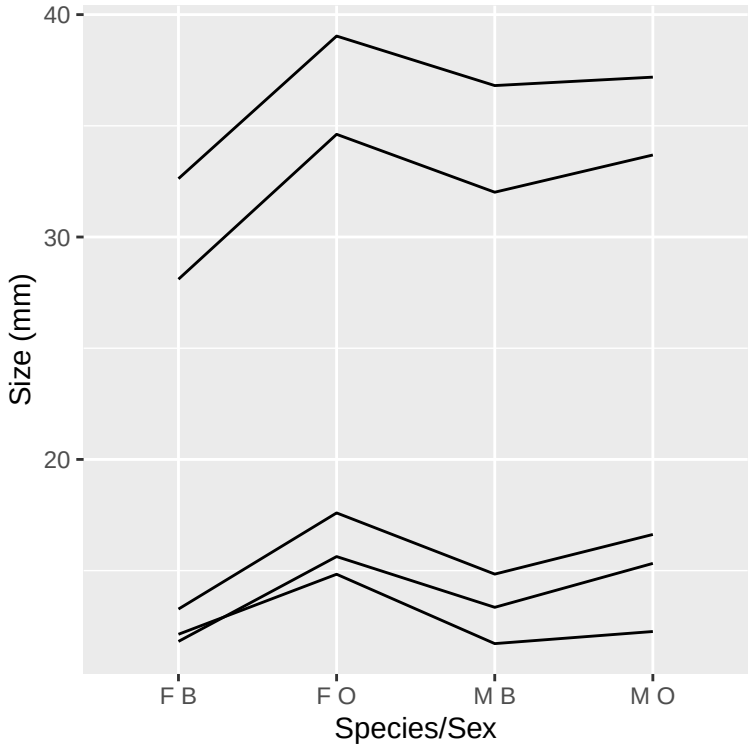
```
  species sex aggname variable  value
1       B   F     F B       FL 13.270
2       O   F     F O       FL 17.594
3       B   M     M B       FL 14.842
4       O   M     M O       FL 16.626
5       B   F     F B       RW 12.138
6       O   F     F O       RW 14.836
```

```
## alternately:
agg2 <- melt(newagg[, 1:6], id.vars = c("species", "sex"))
head(agg2)
```

```
  species sex variable  value
1       B   F       FL 13.270
2       O   F       FL 17.594
3       B   M       FL 14.842
4       O   M       FL 16.626
5       B   F       RW 12.138
6       O   F       RW 14.836
```

```
## or, using tidyr:
library(tidyr)
agg2 <- pivot_longer(newagg, cols = c(FL:BD), names_to = "variable")
```

```
qplot(x = aggname, y = value, group = variable, geom = "line", data = agg2, ylab = "Size (mm)",
    xlab = "Species/Sex")
```

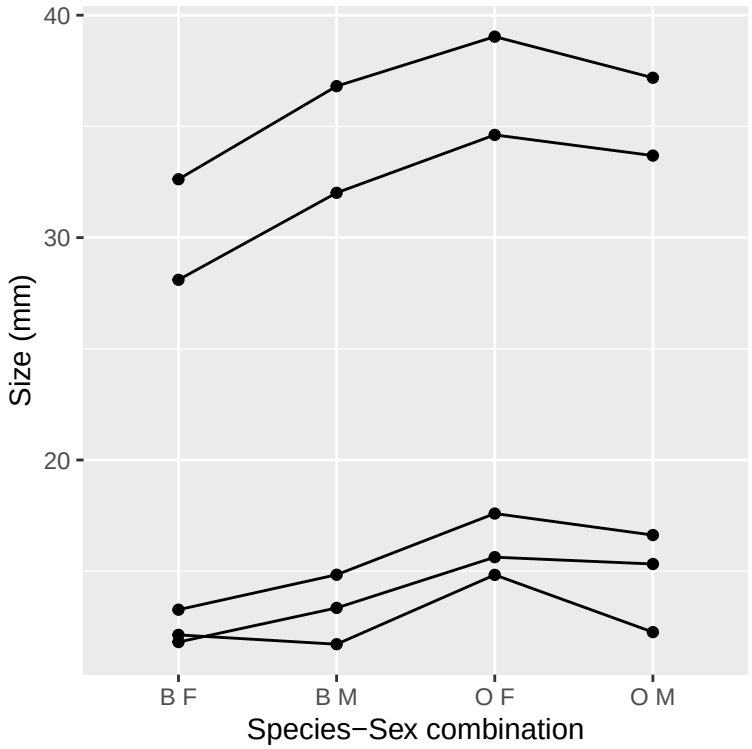


Adding adornments

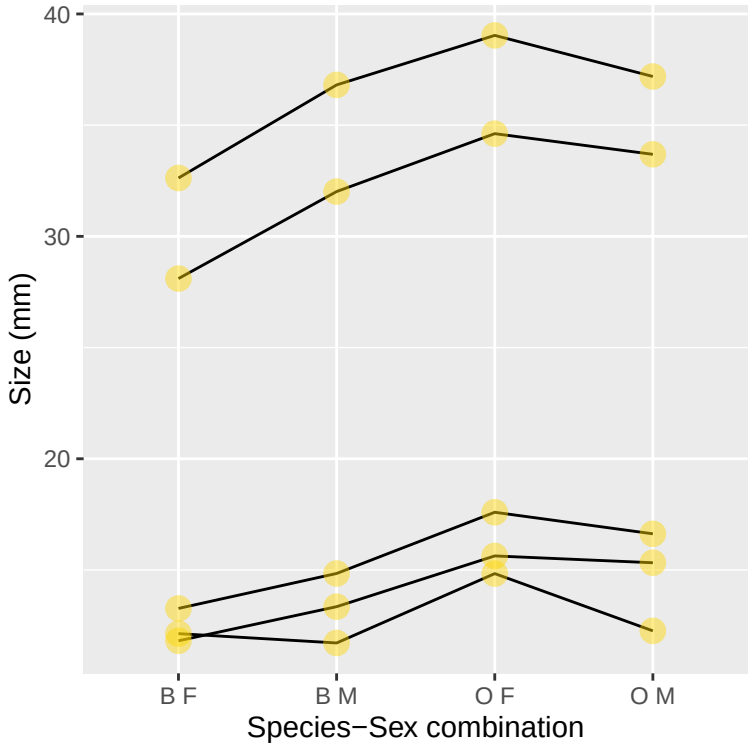
ggplot is called 'grammar of graphics' because it is a true grammar for composing more complex graphics. We do this by adding functions to the baseline plotting function using +. These are essentially like adding multiple arguments into the 'geom' argument' but we have much more control. Note that we can save a plot as a variable, and then change the saved graphic. Sometimes, you may need to print() a saved ggplot object to get it to display.

In the previous example, suppose we wanted to add points to the plot. We can do so with syntax like this:

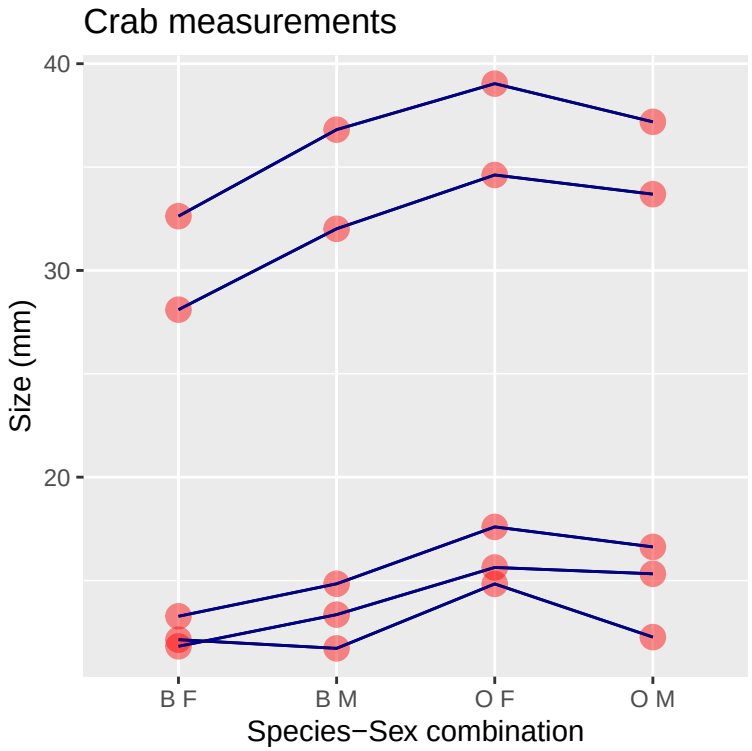
```
## Nothing happens here that we can see::
myplot <- qplot(x = paste(species, sex), y = value, group = variable, geom = "line",
    data = agg2, ylab = "Size (mm)", xlab = "Species-Sex combination")
# add points
myplot + geom_point()
```




```
## add color/alpha transparency to points
myplot + geom_point(size = 4, alpha = 0.45, color = "gold")
```




```
myplot + geom_point(size = 4, alpha = 0.45, color = "red") + geom_line(col = "darkblue") +
    ggtitle("Crab measurements")
```



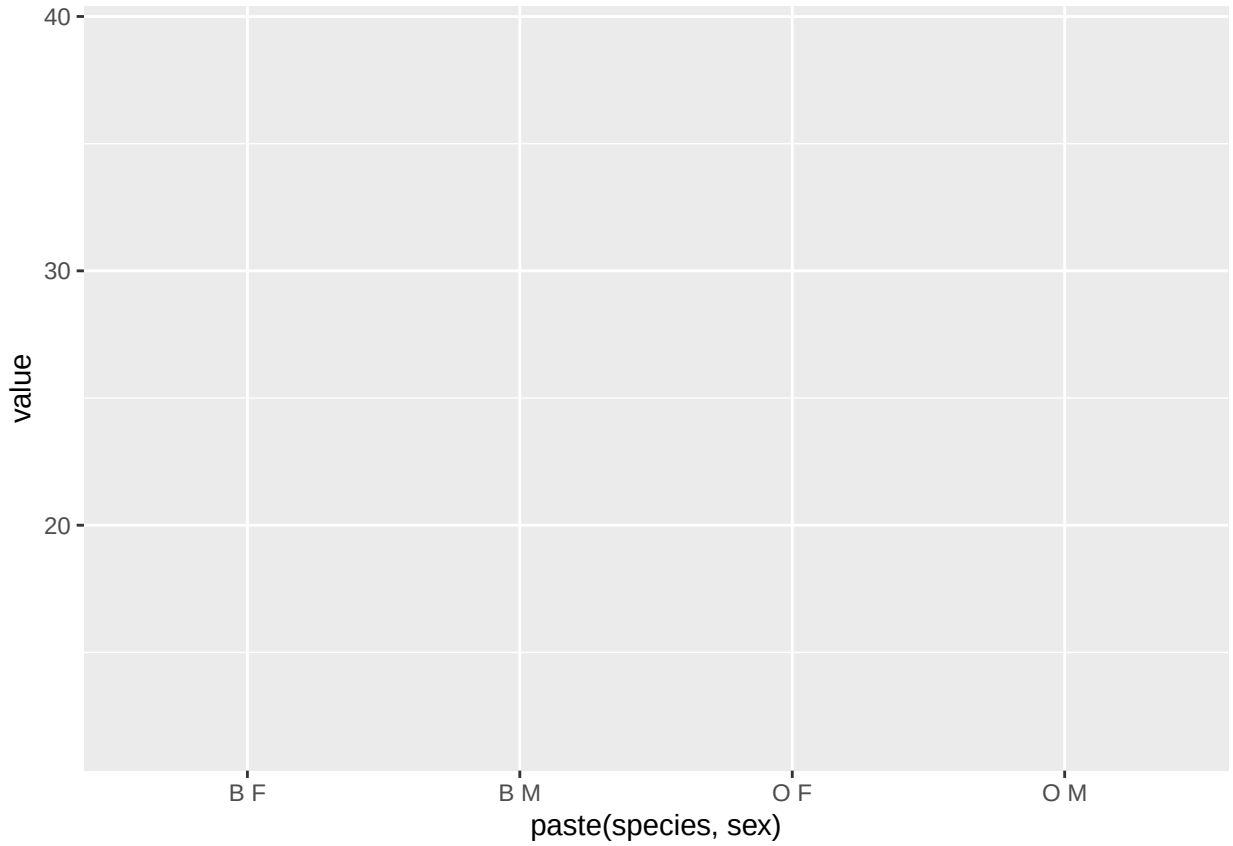
For simple graphs with minimal adornment, this method is fine. But qplot is a shortcut for a more complex set of operations using the ggplot function. If you want to do more complex things, you will need to use `ggplot()`.

The powerful way: ggplot

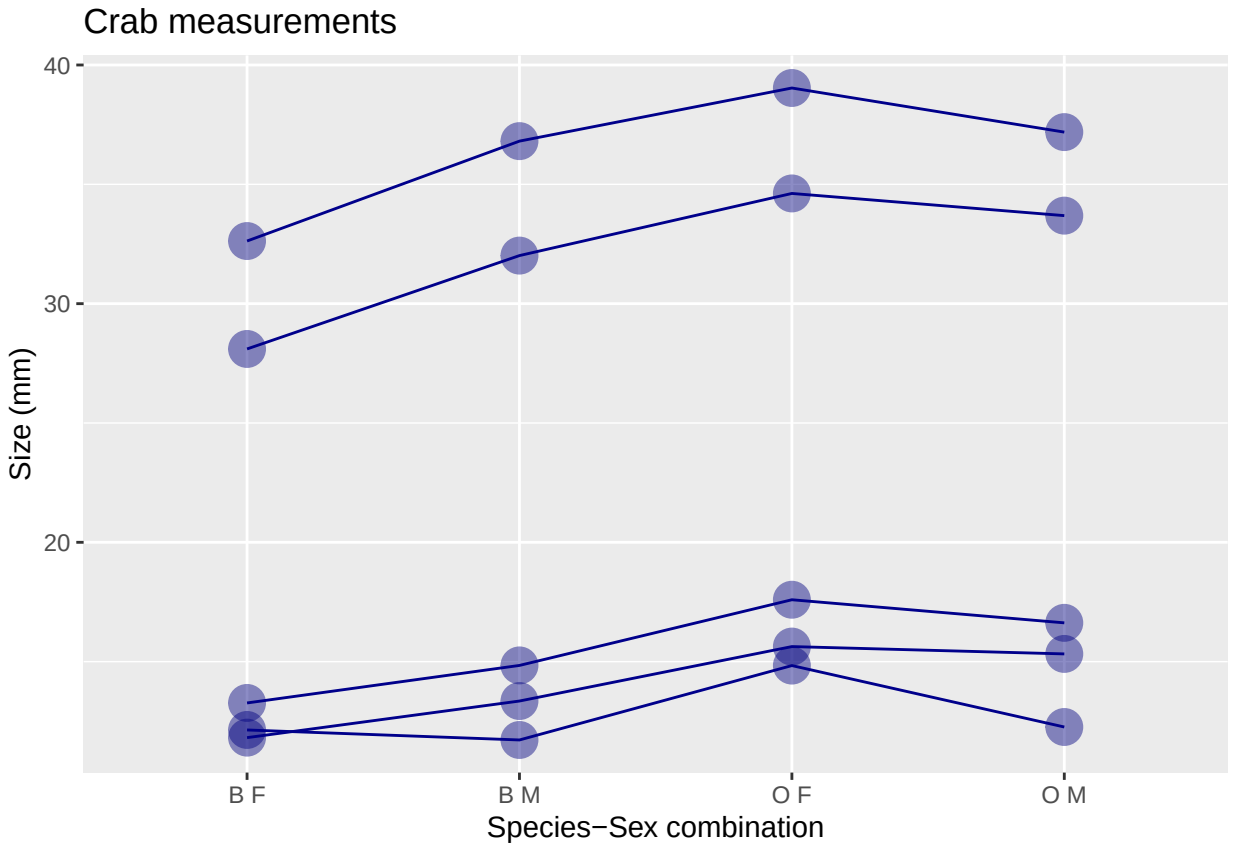
The *ggplot* function provides much more precise control and access over how things are formatted, aggregated, and displayed. ggplot requires you to use a data fram in a 'melted' format (qplot seems to be a bit more forgiving). All of the additional adornments work fine, but now you need to think harder about how to organize your data in the ggplot function. If you do this right, it makes organizing things fairly easy.

Now, instead of qplot, we need to use ggplot. It works almost like the qplot, but instead of specifying x,y directly, we need to add logic that tells how the 'aesthetics' map onto the data colmumns. This is done via the aes() argument. Now, we *need* a `geom` function added to the function to display anything.

```
baseplot <- ggplot(agg2, aes(x = paste(species, sex), y = value, group = variable))
myplot <- baseplot + geom_point(size = 6, alpha = 0.45, color = "navy") + ylab("Size (mm)") +
    xlab("Species-Sex combination") + geom_line(col = "darkblue") + ggtitle("Crab measurements")
print(baseplot)
```

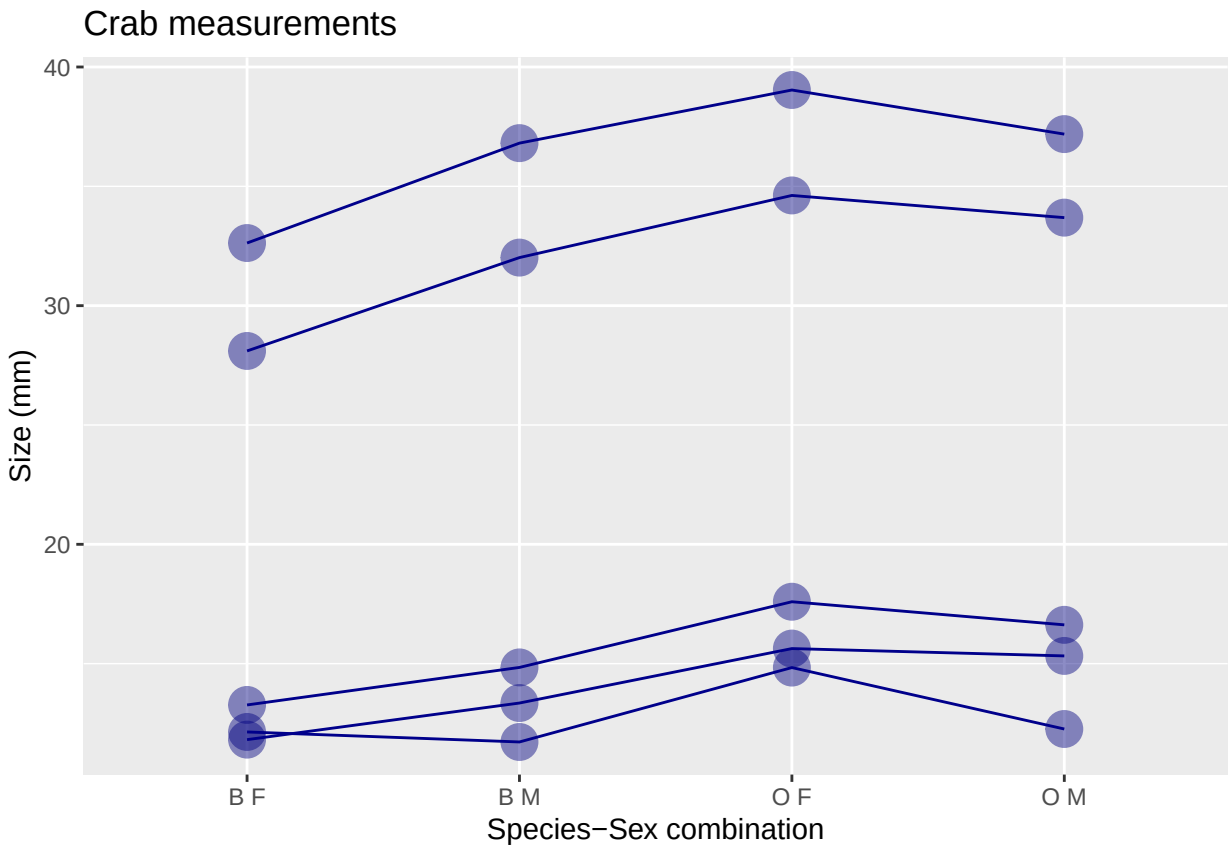



```
print(myplot)
```

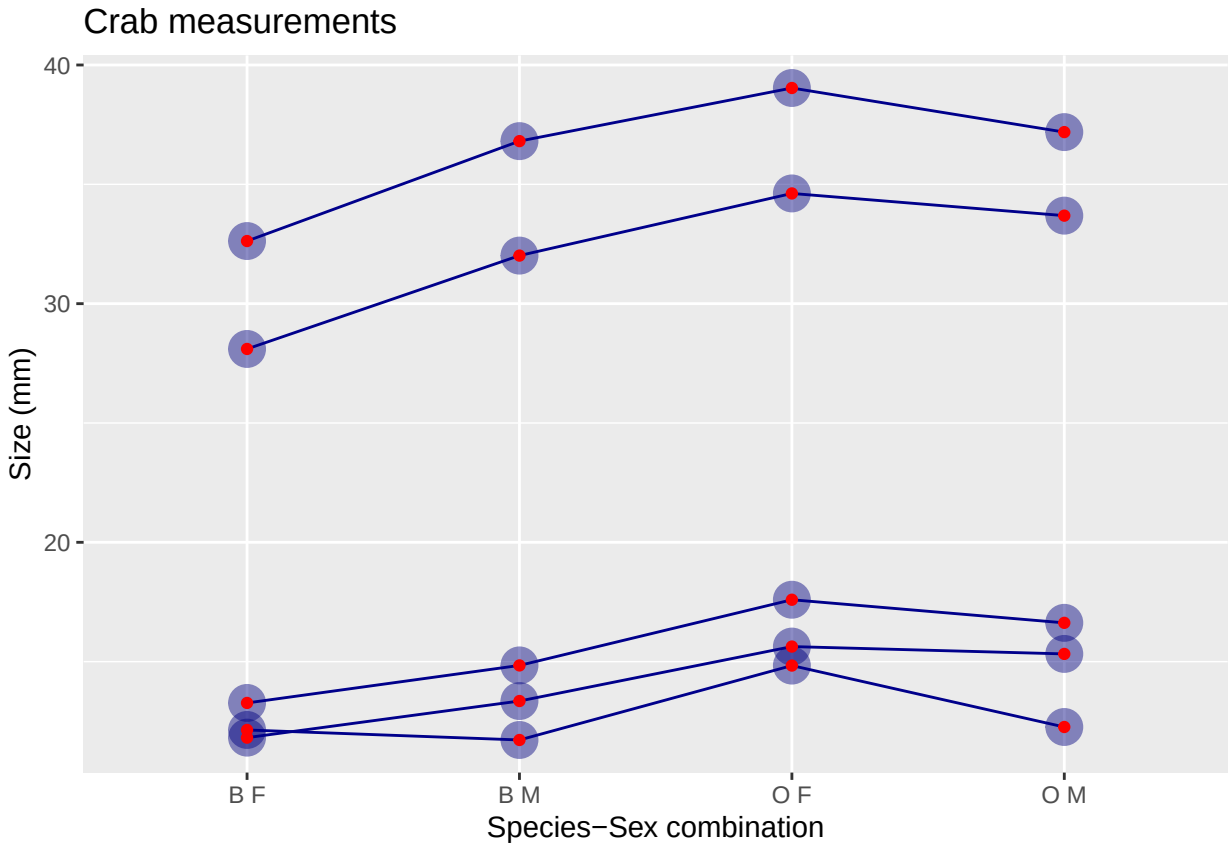


Notice that we have saved two graph objects, but neither showed up until we print()ed them. We can usually just call the object as well, and we don't need to save intermediate variables, like this:

```
myplot
```

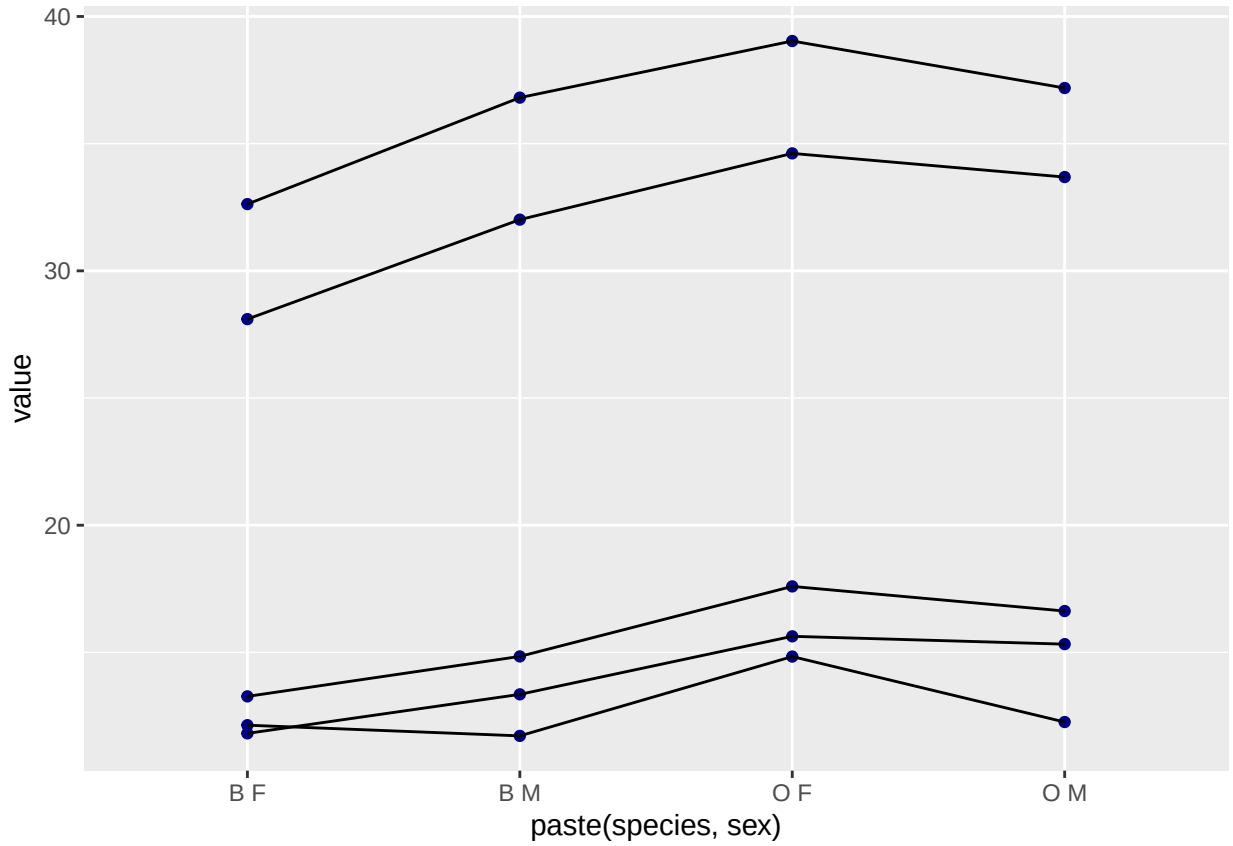



```
myplot + geom_point(color = "red")
```

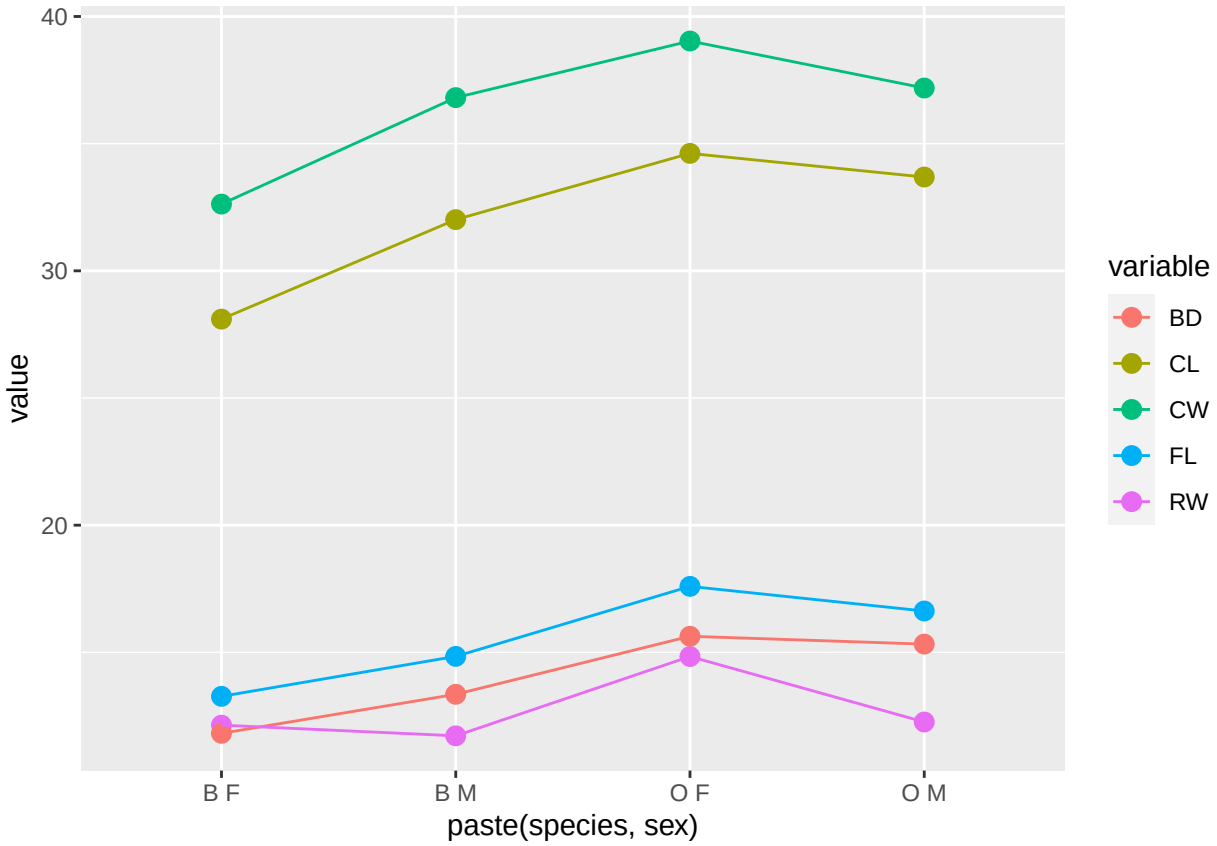


Also notice that no data printed out when we hadn't added the first geom_point() argument. The other thing to note is that you can specify graphical parameters in two ways. When a parameter (shape, color, group, size, etc) is specified within the ggplot() or a geom_() function, it applies universally to the whole aspect. If you want the color/shape/etc mapped onto a meaningful dimension, it needs to be specified within the aes() argument which refers to 'aesthetic'. When speciied in aes, you will usually also automatically get a legend displayed.

```
baseplot <- ggplot(agg2, aes(x = paste(species, sex), y = value, group = variable))
baseplot + geom_point(col = "navy") + geom_line()
```



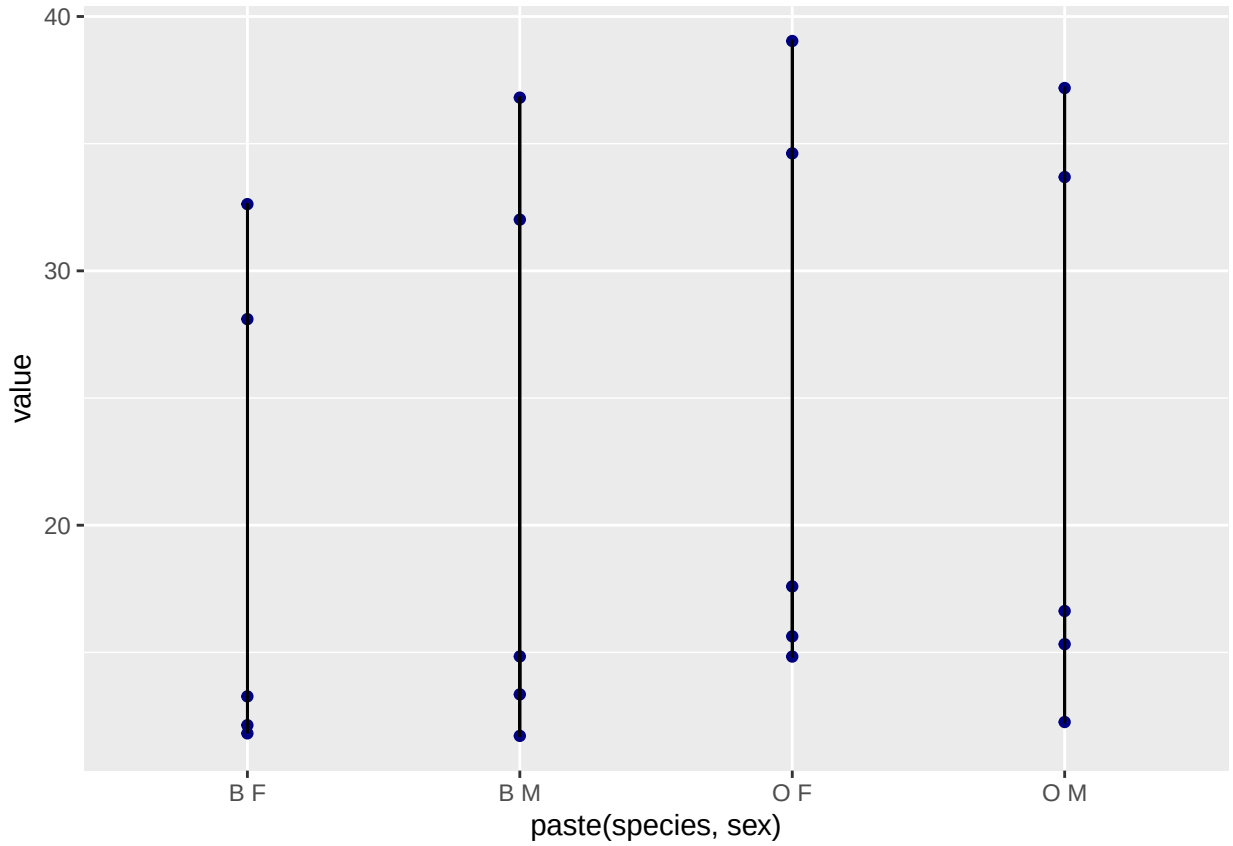

```
baseplot + geom_point(aes(color = variable), size = 3) + geom_line(aes(color = variable))
```



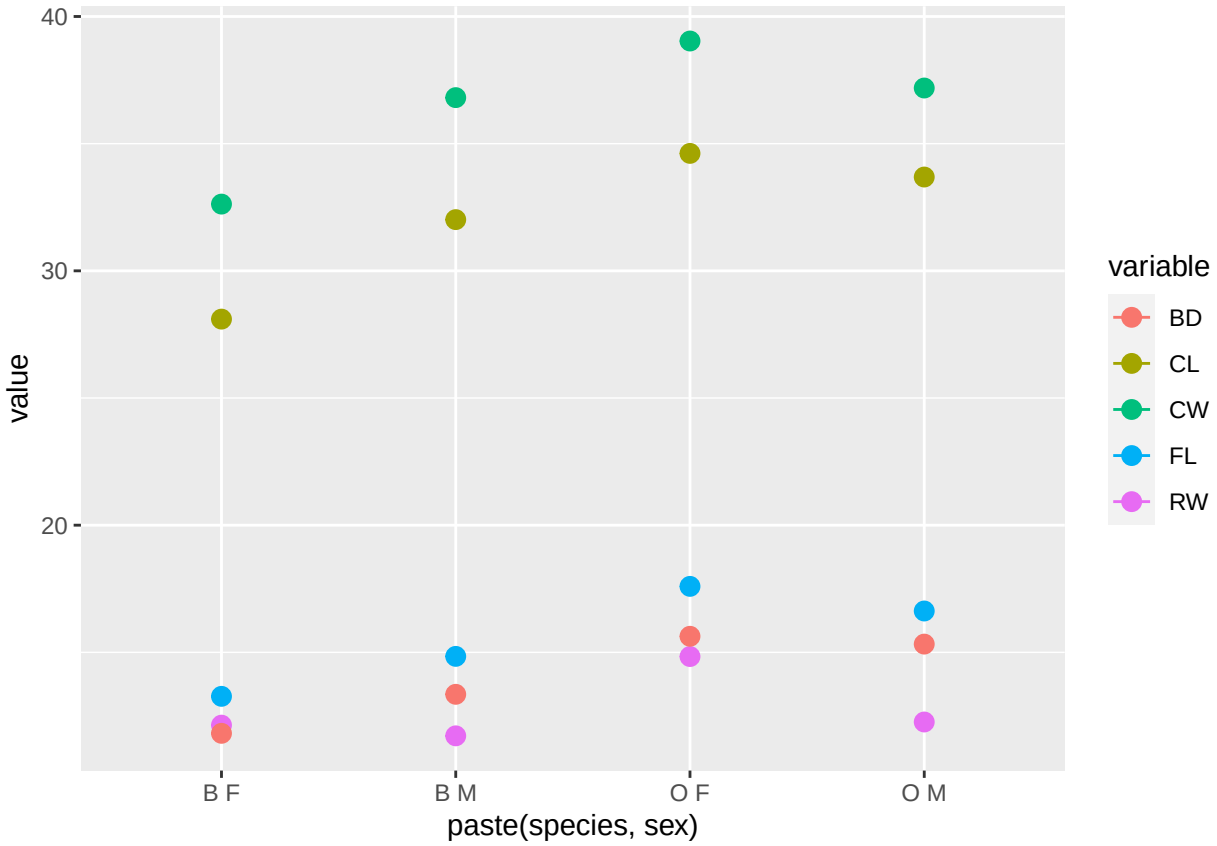
Grouping variables is ggplot

The above seems fairly easy, but what is 'group' doing? Let's see what happens if we remove it:

```
baseplot <- ggplot(agg2, aes(x = paste(species, sex), y = value))
baseplot + geom_point(col = "navy") + geom_line()
```




```
baseplot + geom_point(aes(color = variable), size = 3) + geom_line(aes(color = variable))
```



The group argument in aes binds a series together for the particular graph. The first graph we made is completely strange, and the second doesn't make the lines we ask for because of it. If things are not working out in your ggplot, be sure you have group specified, and also look out that you have done the right level of aggregation/summarizing.

Error bars

Suppose we want to add error bars. We need to first compute standard errors for our data set, which shouldn't be too hard.

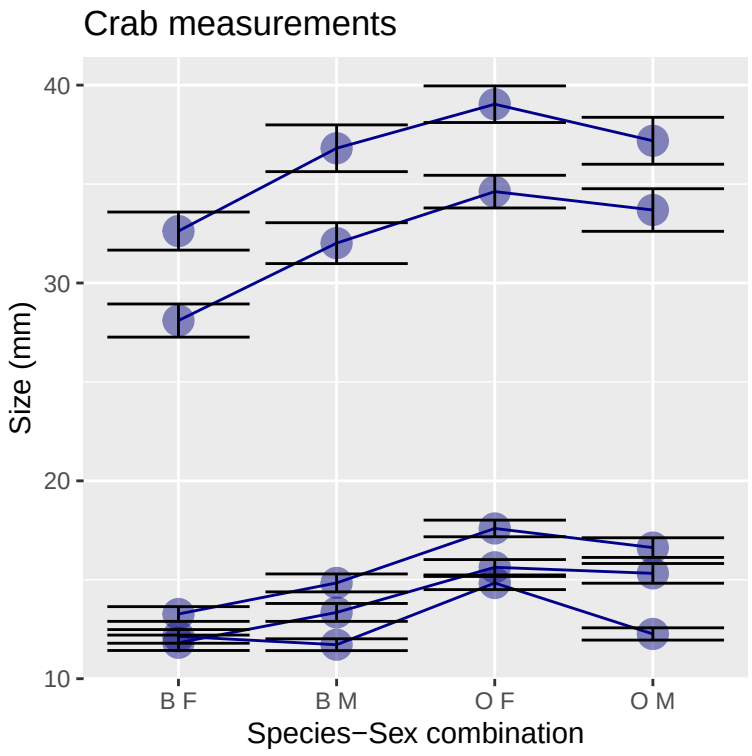
```
library(reshape2)
agg.se <- aggregate(dplyr::select(crabs, FL, RW, CL, CW, BD, CW), list(species = crabs$sp,
    sex = crabs$sex), function(x) {
    sd(x)/sqrt(length(x))
})
newagg.se <- dplyr::select(agg.se, species, sex, FL, RW, CL, CW, BD)
agg2.se <- melt(newagg.se, id.vars = c("species", "sex"))
agg2.a <- left_join(agg2, agg2.se, by = c("species", "sex", "variable"))
agg2.a$ymin <- agg2.a$value.x - agg2.a$value.y
agg2.a$ymax <- agg2.a$value.x + agg2.a$value.y


head(agg2.a)
```

```
# A tibble: 6 x 8
  species sex   aggname variable value.x value.y  ymin  ymax
  <fct>   <fct> <chr>   <chr>      <dbl>   <dbl> <dbl> <dbl>
1 B       F     F B     FL          13.3   0.372  12.9  13.6
```

```
2 B      F     F B     RW          12.1   0.345  11.8  12.5
3 B      F     F B     CL          28.1   0.837  27.3  28.9
4 B      F     F B     CW          32.6   0.962  31.7  33.6
5 B      F     F B     BD          11.8   0.389  11.4  12.2
6 O      F     F O     FL          17.6   0.421  17.2  18.0
```

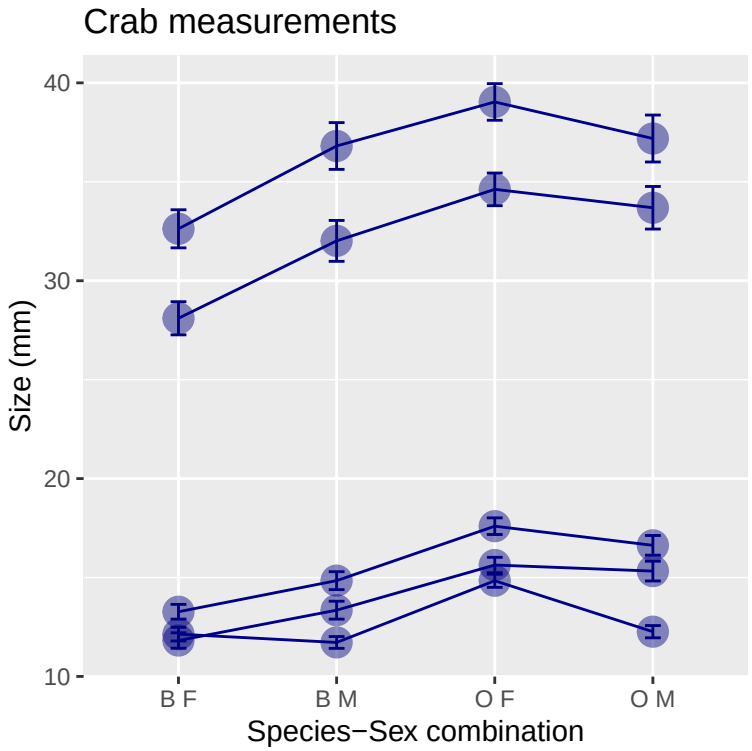
Now, we have a single data frame with value.x which has the mean values, and ymin/ymax which are the top and bottom of the error bars (+/- one s.e.)

```
myplot <- ggplot(agg2.a, aes(x = paste(species, sex), y = value.x, group = variable,
    ymin, ymax)) + geom_point(size = 5, alpha = 0.45, color = "navy") + geom_line(colour = "darkblue") +
    ggtitle("Crab measurements") + ylab("Size (mm)") + xlab("Species-Sex combination")
myplot + geom_errorbar(aes(ymin = ymin, ymax = ymax))
```



This is effective but ugly. We can try a few other tweaks:

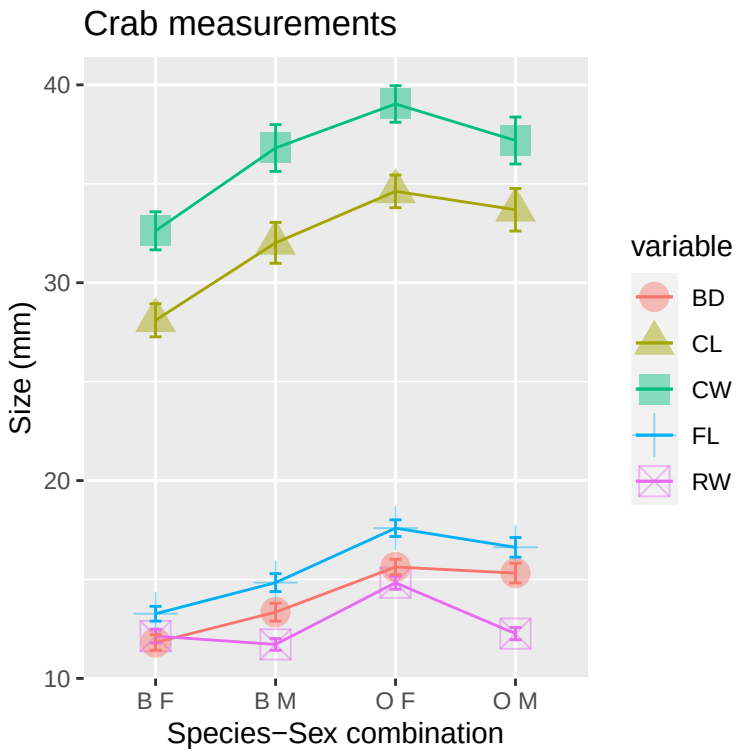
```
myplot + geom_errorbar(aes(ymin = ymin, ymax = ymax), colour = "navy", width = 0.1)
```



Adding a legend

The current figure is not much help, because we don't know what each line means. ggplot will add a legend when it makes sense; sometimes it does not add a legend when you think it also might make sense. In the above figure, although each series was separated with a different line, there was no way of distinguishing the points via a legend. Add this and you will get a legend. Here, we set both colour and shape to depend on a specific value, and the legends will show up.

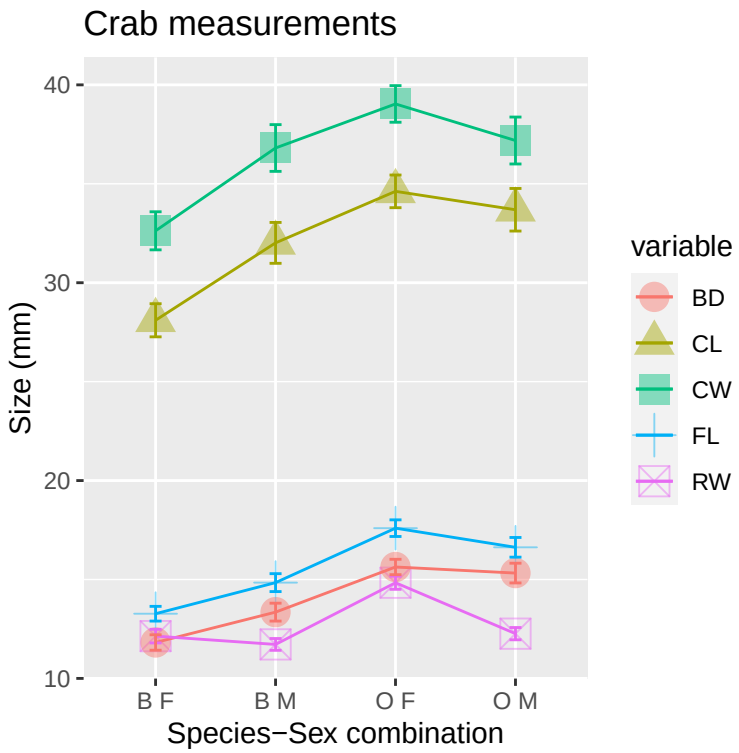
```
myplot <- ggplot(agg2.a, aes(x = paste(species, sex), y = value.x, colour = variable,
    shape = variable, group = variable, ymin, ymax)) + geom_point(size = 5, alpha = 0.45) +
    geom_line() + ggtitle("Crab measurements") + ylab("Size (mm)") + xlab("Species-Sex combination")
myplot + geom_errorbar(aes(ymin = ymin, ymax = ymax), width = 0.1)
```



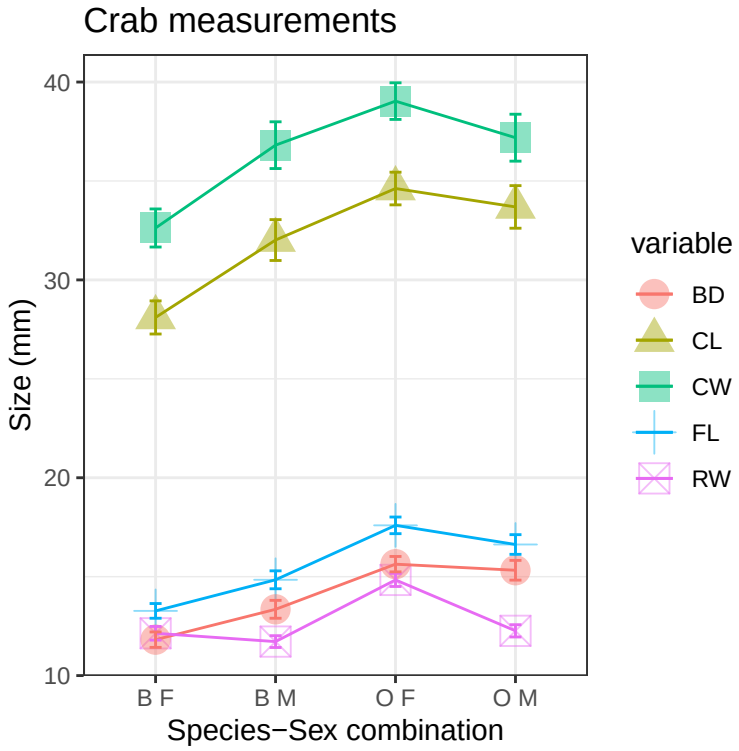
Themes

Changing themes is done by adding various theme_() functions. But it is difficult to get the same level of control you can get with base R graphics. If you are producing a figure for publication, black/white or greyscale is usually preferred. Just add a theme_bw() to the end and you can get this:

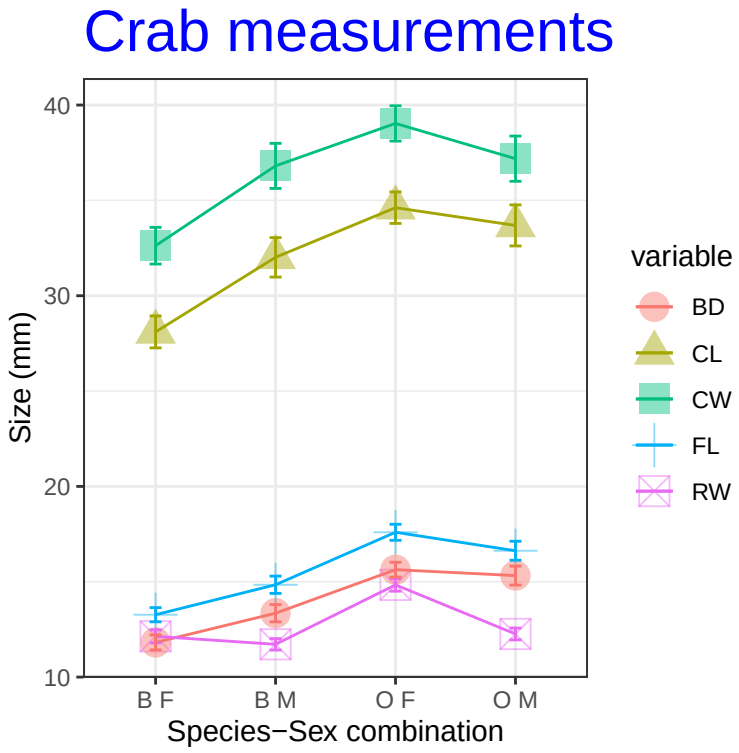
```
# standard theme
myplot + geom_errorbar(aes(ymin = ymin, ymax = ymax), width = 0.1) + theme_grey()
```




```
## black and white theme:
myplot + geom_errorbar(aes(ymin = ymin, ymax = ymax), width = 0.1) + theme_bw()
```

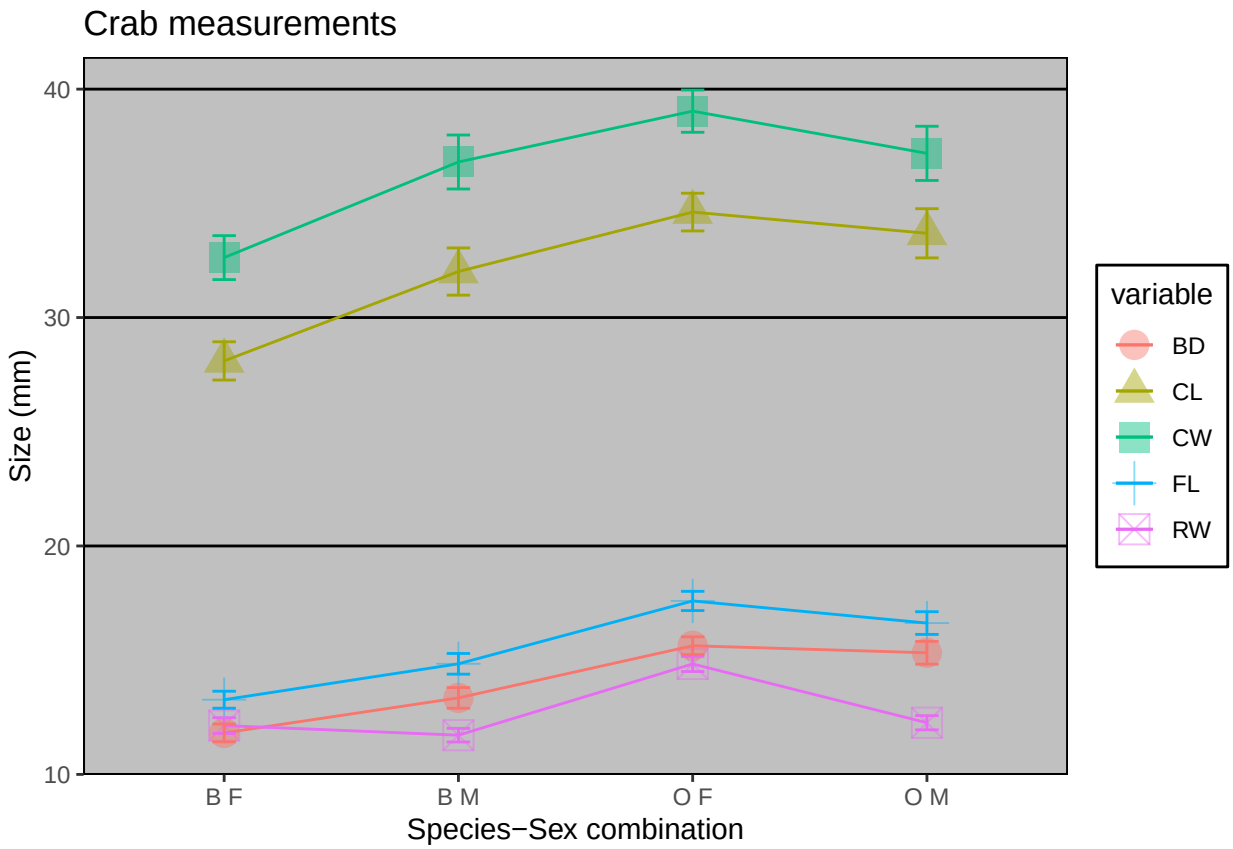



```
myplot + geom_errorbar(aes(ymin = ymin, ymax = ymax), width = 0.1) + theme_bw() +
    theme(plot.title = element_text(size = rel(2), colour = "blue"))
```

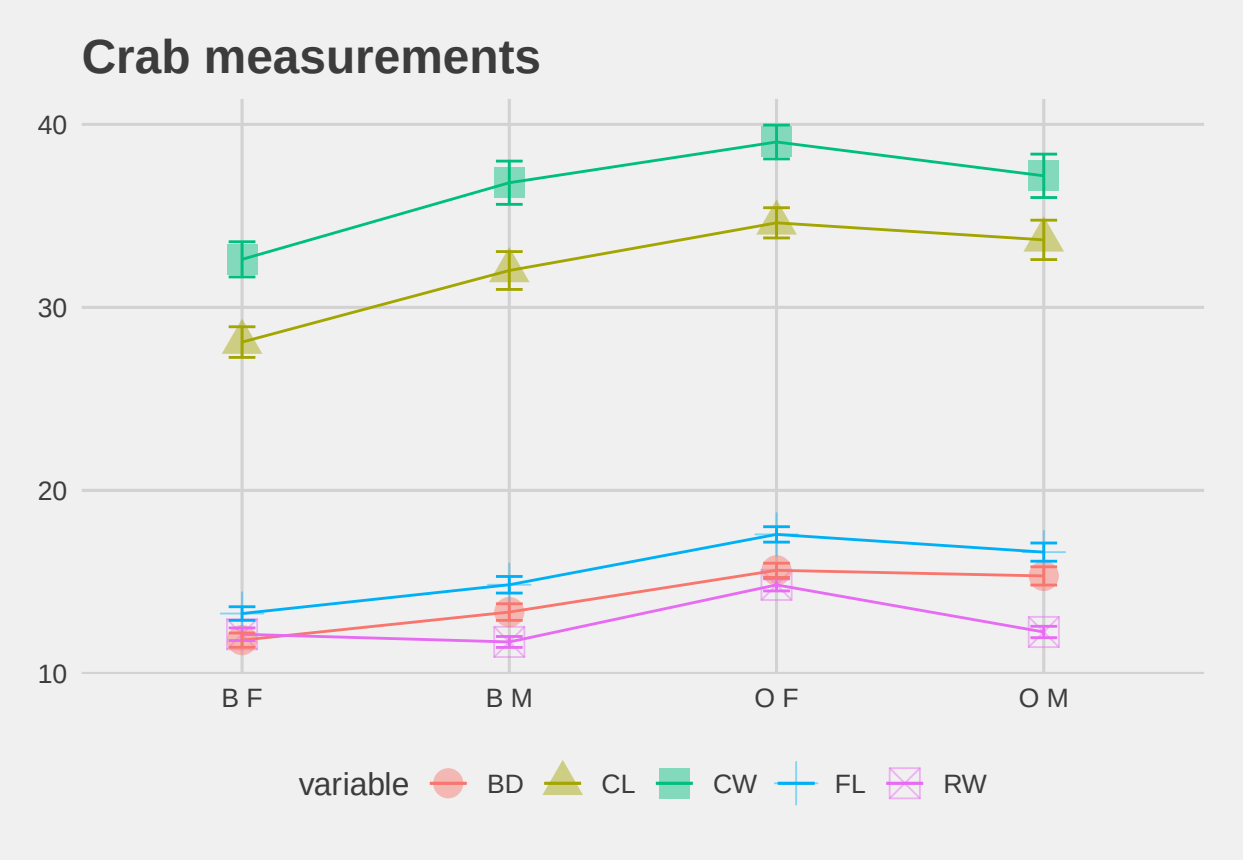


There is a package ggthemes() that hase a lot of popular color/graphic themes, including themes that make your graphics look like excel, fivethirtyeight.com, the economist, stata, and many other built-in and customizeable options.

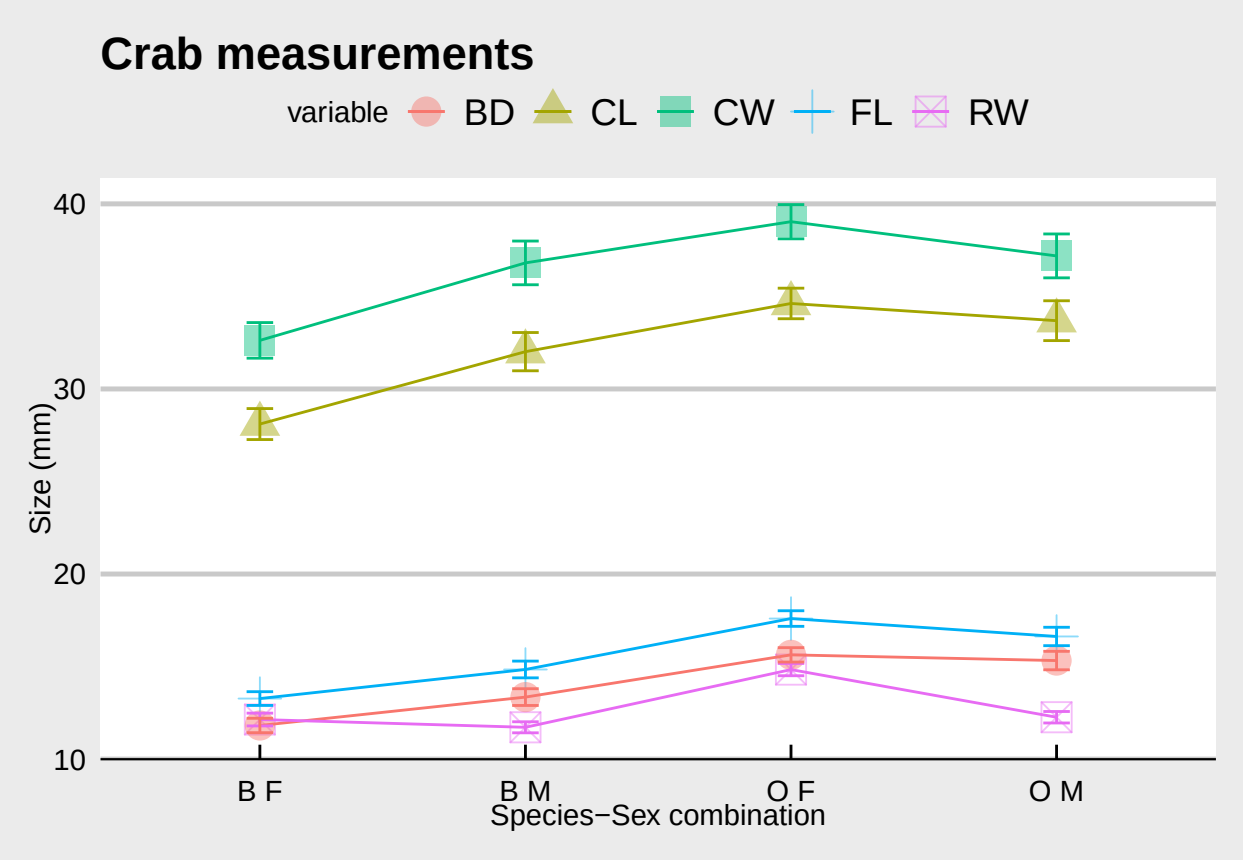
```
library(ggthemes)
myplot + geom_errorbar(aes(ymin = ymin, ymax = ymax), width = 0.1) + theme_excel()
```




```
myplot + geom_errorbar(aes(ymin = ymin, ymax = ymax), width = 0.1) + theme_fivethirtyeight()
```



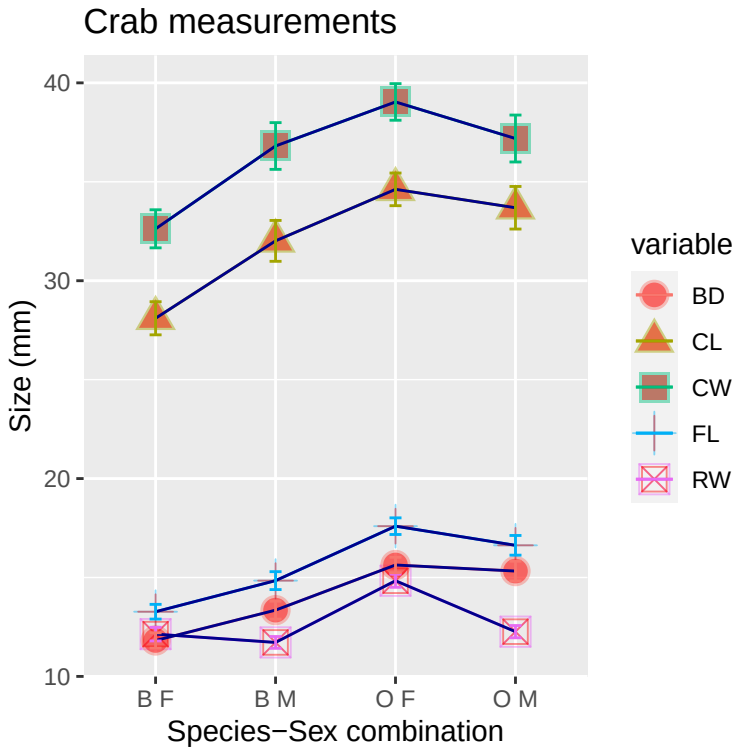

```
myplot + geom_errorbar(aes(ymin = ymin, ymax = ymax), width = 0.1) + theme_economist_white()
```



Saving

You can save graphics just like you always do, but the `ggsave()` functions offers a method for saving the latest graphic you created.

```
myplot + geom_point(size = 4, alpha = 0.45, color = "red") + geom_line(col = "darkblue") +
    ggtitle("Crab measurements") + geom_errorbar(aes(ymin = ymin, ymax = ymax), width = 0.1)
```




```
ggsave(filename = "demo.png", dpi = 600, width = 8, height = 4)
ggsave(filename = "demo.pdf", dpi = 600, width = 8, height = 4)
ggsave(filename = "demo.eps", dpi = 600, width = 8, height = 4)
```

ggsave saves based on the file extension, and currently can save as ps, tex (pictex), pdf, tiff, png, bmp and wmf (windows only).

Other features

The breadth of things you can do with ggplot is truly amazing. The default themes are nice, but you can change these fairly easily if you dig into examples provided in many places.

Reporting

The default settings of ggplot2 are best for on-screen graphics. Some journals prefer black-and-white. Alternate themes are available that will still produce attractive black-and-white images.

Most journals will be a bit picky about the graphic format of images. Many will want .tiff format, which is a compressed graphic format that is lossless. .png works this way as well, but you are best off sticking with something they know and understand.

During pdf creation, a journal sometimes compresses image-based graphics and they can be lossy. With standard R graphics, you can save as a .ps format (rename to .eps), and most will be able to handle this as well. However, ggplot renders graphics to a rasterized image, and so overall you images will probably be larger and there could be quality issues if you don't use a high enough dpi or image size.

Minimal example with ggvis

```
library(ggvis)
mtcars %>% ggvis(~wt, ~mpg) %>% layer_points()
```

Renderer: SVG | Canvas

Download

Exercises and challenges

Here are some things that appear like they should be easy to accomplish, but may not be.

Smoothing envelope

Using geom_smooth() is a popular visualization that creates a loess regression, and also displays some sort of estimate of standard error or quantiles. Plot a smoothed line and the individual points for the relationship between HDI and CPI in the EconomostData.csv file.

```
library(readr)
ec <- read.csv("EconomistData.csv")
```

Bar plots

- Make a histogram of the CPI values
- Make a bar plot/histogram of the of the number of countries in each region
- Make a bar plot showing average CPI per region
- Make a xy plot showing the relationship between CPI and HDI for separate colored series for each region
- Add a geom_smooth to that.
- Use faceting instead for each line.

plotting raw data

Plot the following series in ggplot. * First, plot outcome alone, so yo ucan see the 100 values in sequence. * Then, plot both outcome and outcome2, so you can see separate sequences, using a single data frame. * Do the same thing, but with different data arguments for each line so that you use two separate data frames.

```
outcome <- rnorm(100) + 1:100

outcome2 <- runif(100) + 100:1
```

Group means

- For the crabs data, make a bar plot (not a boxplot) showing mean FL by sex and species.

Layers

Compute bin averages for outcome as follows:

```
binmeans <- aggregate(outcome, list(rep(1:10, each = 10)), mean)
```

Add the mean of each bin to your outcome plot at an appropriate location (i.e., 5,15,25,. . . ,95)

Adding text

- Add text labels to the xy plot of HDI vs. CPI
- Use ggrepel to add text instead of the normal way, using both `geom_text_repel` and `geom_label_repel`.

```
base <- ggplot(data = ec, aes(x = HDI, y = CPI))
```

Assumptions and Limitations

ggplot is fairly flexible, and can produce beautiful graphics, but it can be a little frustrating to use when you are used to the layering and drawing of traditional R graphics. Many additional libraries aim to make ggplot more flexible, produce custom plots, or give alternative (easier) syntax to access the plots.

Resources

- http://had.co.nz/ggplot2/book/qplot.pdf
- http://www.cookbook-r.com/Graphs/Plotting_means_and_error_bars_%28ggplot2%29/
- https://docs.google.com/viewer?url=http%3A%2F%2Fvita.had.co.nz%2Fpapers%2Fggplot2-wires.pdf
- **Main documentation** http://docs.ggplot2.org/current/

Other related libraries

- https://www.ggplot2-exts.org/ a Site describing various extensions to ggplot2
- GGally: A library based on ggplot2 that sometimes makes simple plots easer
- ggpubr: More traditional R syntax, more customization for publication-ready figures.
- plotly: on-line interactive graphics that can embed ggplot graphics.
- ggvis (web visualization)
- ggtech (many themes)
- ggfortify (links to automatic plotting of standard objects in R, such as lm)
- ggpmisc (identifying peaks/valleys; annotating graph with text)
- ggrepel (add text that does not overlap)
- Dozens of other custom libraries!