Data management with dplyr, tidyr, and reshape2

Shane T. Mueller shanem@mtu.edu

2019-01-16

# Data Management Libraries

In recent years, RStudio has spearheaded development of a series of libraries that make data refactoring, selecting, and management simple and fast for large data sets. Many of these tools are equiva lent to what you can do using selection, sorting, aggregate, and tapply of normal data frames. Some of them offer very useful capabilities that are otherwise very difficult to manage. Most of these are developed by Hadley Wickham, who also created ggplot2. Part of the reason for the proliferation of libraries is the philosophy to not break what people rely on, and so when improved functionality is made, a new library is created so that compatibility can be broken without harming anyone relying on certain functionality.

Some relevant libraries include:

## plyr and dplyr

These libraries are sets of tools for splitting, applying, and combining data. The goal is to have a coherent set of tools for breaking down data into smaller pieces, operating on each chunk of data and reassembling them–an idiom called “split-apply-combine”.

dplyr is a successor to plyr, written to be much faster, to integrae with remote databases, but it works only with data frames. The dplyr library seems to be better supported, and tests show it can be more than a hundred times faster than plyr.

## reshape, reshape2 and tidyr

The reshape2 library is a ‘reboot’ of reshape, that is faster and better. These libraries allow easily transforming a data set from ‘long’ to ‘wide’ format and back again. That is, you can take a data set with multiple columns you are treating as distinct DVs, and reframe the data set so they are both in a single DV column, with a separate column specifiying which level of IV a row belongs to. The tidyr library is the newest entry into data management libraries, also by Wickham, and is described as an “evolution” of reshape2.

# Overview of dplyr

The following creates a couple data sets for use in these examples:

dat0 <- data.frame(sub= c(1,1,1,2,2,2,3,3,3,4,4,4),
 question = c("a","b","c","a","b","c","a","b","c","a","b","c"),
 dv = c(5,3,1,2,3,6,4,2,3,1,3,5))

dat <- data.frame(sub = sample(letters,100,replace=T),
 cond = sample(c("A","B","C"),100,replace=T),
 group = sample(1:10,100,replace=T),
 dv1 = runif(100)\*5)

The dplyr library implements a number of functions that are available in one form or another within R, but may be difficult to use, inconsistent, or slow.

The dplyr library does not create side-effects. That is, it always makes a copy of your original data and returns it, rather than altering the form of your original data. Consequently, you need to usually assign the outcome to a new variable. Sometimes, it is acceptable to assign it to its old name, as in the following:

library(dplyr)
data <- dat
filter(data,sub=="b")

 sub cond group dv1
1 b B 8 0.02159992
2 b B 2 3.86862284
3 b A 7 3.55023181
4 b A 10 0.74506808
5 b C 4 1.95993423

data <-filter(data,sub=="b")
head(data)

 sub cond group dv1
1 b B 8 0.02159992
2 b B 2 3.86862284
3 b A 7 3.55023181
4 b A 10 0.74506808
5 b C 4 1.95993423

However, this is often not the best practice, because it means that the data variable depends on whether you have run some code or not.

# slice and filter

The following use dplyr to rearrange and filter rows of a data frame. filter picks out rows based on a boolean vector of the same size (number of rows)

head((dat$sub=="b")) ##shows the first 6 elements of the boolean

[1] FALSE FALSE FALSE FALSE FALSE FALSE

filter(dat,sub=="b") ##use filter to pick out just tho subject B rows

 sub cond group dv1
1 b B 8 0.02159992
2 b B 2 3.86862284
3 b A 7 3.55023181
4 b A 10 0.74506808
5 b C 4 1.95993423

Similarly, slice allows you to do this based on the row index (number)

slice(dat,1) ##first row

 sub cond group dv1
1 p A 6 0.8920465

slice(dat,2:10) ##9 rows after the first

 sub cond group dv1
1 y C 1 1.9234548
2 r C 9 0.5484909
3 v C 3 4.5435177
4 s B 4 2.2748488
5 o B 5 1.2491712
6 j C 3 1.4397083
7 s A 9 1.5833983
8 q C 5 3.3348710
9 g B 7 4.3093207

slice(dat,1:20\*2) ##even rows 2..40

 sub cond group dv1
1 y C 1 1.92345481
2 v C 3 4.54351771
3 o B 5 1.24917123
4 s A 9 1.58339833
5 g B 7 4.30932072
6 z B 6 4.19409553
7 k B 4 3.06690507
8 q A 7 2.21262991
9 h B 5 2.31832159
10 n C 7 1.97875672
11 v A 1 1.28046895
12 r B 1 3.94814875
13 e A 5 1.98548959
14 l B 7 3.02942468
15 y C 7 0.92892031
16 c A 3 2.88657606
17 b B 8 0.02159992
18 z B 5 3.56611500
 [ reached getOption("max.print") -- omitted 2 rows ]

slice(dat,-1)

 sub cond group dv1
1 y C 1 1.92345481
2 r C 9 0.54849086
3 v C 3 4.54351771
4 s B 4 2.27484879
5 o B 5 1.24917123
6 j C 3 1.43970831
7 s A 9 1.58339833
8 q C 5 3.33487098
9 g B 7 4.30932072
10 u B 9 4.21354733
11 z B 6 4.19409553
12 f C 2 3.22020284
13 k B 4 3.06690507
14 r C 2 0.08333187
15 q A 7 2.21262991
16 e A 2 1.42505701
17 h B 5 2.31832159
18 t B 5 4.69740370
 [ reached getOption("max.print") -- omitted 81 rows ]

# arrange()

The arrange function reorders the rows by the levels of a specific factor

arrange(dat,sub)

 sub cond group dv1
1 a A 7 1.75937246
2 a C 3 3.01395102
3 b B 8 0.02159992
4 b B 2 3.86862284
5 b A 7 3.55023181
6 b A 10 0.74506808
7 b C 4 1.95993423
8 c C 8 3.03982364
9 c A 3 2.88657606
10 c A 3 0.75181274
11 c C 6 1.96465752
12 d A 10 0.26949307
13 d A 5 1.06291885
14 d A 8 2.01532086
15 d B 8 4.73926346
16 e A 2 1.42505701
17 e A 5 1.98548959
18 e C 2 4.70717221
 [ reached getOption("max.print") -- omitted 82 rows ]

arrange(dat,sub,group)

 sub cond group dv1
1 a C 3 3.01395102
2 a A 7 1.75937246
3 b B 2 3.86862284
4 b C 4 1.95993423
5 b A 7 3.55023181
6 b B 8 0.02159992
7 b A 10 0.74506808
8 c A 3 2.88657606
9 c A 3 0.75181274
10 c C 6 1.96465752
11 c C 8 3.03982364
12 d A 5 1.06291885
13 d A 8 2.01532086
14 d B 8 4.73926346
15 d A 10 0.26949307
16 e A 2 1.42505701
17 e C 2 4.70717221
18 e A 5 1.98548959
 [ reached getOption("max.print") -- omitted 82 rows ]

# select()

* The select function picks out **columns** by name

select(dat0,sub,dv)

 sub dv
1 1 5
2 1 3
3 1 1
4 2 2
5 2 3
6 2 6
7 3 4
8 3 2
9 3 3
10 4 1
11 4 3
12 4 5

select(dat0,sub:dv)

 sub question dv
1 1 a 5
2 1 b 3
3 1 c 1
4 2 a 2
5 2 b 3
6 2 c 6
7 3 a 4
8 3 b 2
9 3 c 3
10 4 a 1
11 4 b 3
12 4 c 5

select(dat0,-question)

 sub dv
1 1 5
2 1 3
3 1 1
4 2 2
5 2 3
6 2 6
7 3 4
8 3 2
9 3 3
10 4 1
11 4 3
12 4 5

There are a lot of matching functions:

select(dat0,starts\_with("s"))

 sub
1 1
2 1
3 1
4 2
5 2
6 2
7 3
8 3
9 3
10 4
11 4
12 4

This function can be very handy for situations like survey data where you have dozens or hundreds of columns/variables. You may be interested in just a few of these, and select will pick these out.

# rename()

* The rename function renames columns.

rename(dat0,participant=sub)

 participant question dv
1 1 a 5
2 1 b 3
3 1 c 1
4 2 a 2
5 2 b 3
6 2 c 6
7 3 a 4
8 3 b 2
9 3 c 3
10 4 a 1
11 4 b 3
12 4 c 5

# distinct()

* The distinct function finds distinct combinations of values (typically IVs). This is similar to doing a table, or identifying the levels of a factor.

dat2 <- data.frame(a=sample(1:10,20,replace=T),
 b=sample(c(100,200,300),20,replace=T))
distinct(dat2)

 a b
1 1 100
2 8 300
3 3 100
4 9 100
5 2 100
6 8 200
7 5 200
8 2 300
9 2 200
10 9 200
11 6 200
12 5 300
13 9 300
14 6 100
15 3 200
16 3 300

You can also specify specific variables you wish to use:

distinct(dat,sub)

 sub
1 p
2 y
3 r
4 v
5 s
6 o
7 j
8 q
9 g
10 u
11 z
12 f
13 k
14 e
15 h
16 t
17 n
18 c
19 d
20 l
21 w
22 b
23 a
24 x
25 i
26 m

Retain all columns of distinct data:

distinct(dat,sub,.keep\_all=T)

 sub cond group dv1
1 p A 6 0.89204647
2 y C 1 1.92345481
3 r C 9 0.54849086
4 v C 3 4.54351771
5 s B 4 2.27484879
6 o B 5 1.24917123
7 j C 3 1.43970831
8 q C 5 3.33487098
9 g B 7 4.30932072
10 u B 9 4.21354733
11 z B 6 4.19409553
12 f C 2 3.22020284
13 k B 4 3.06690507
14 e A 2 1.42505701
15 h B 5 2.31832159
16 t B 5 4.69740370
17 n C 7 1.97875672
18 c C 8 3.03982364
 [ reached getOption("max.print") -- omitted 8 rows ]

# mutate() and transmute()

* The mutate function adds a column that is a function of other columns. Transmute does the same thing, but returns only the new variable. This can be really useful for creating summarized data, composite values of ratings scales, and the like.

##reverse code a scale
dat1 <- mutate(dat0,newdv=6-dv)

More complex mutations are possible:

mutate(dat1,newdv2 = dv\*newdv)

 sub question dv newdv newdv2
1 1 a 5 1 5
2 1 b 3 3 9
3 1 c 1 5 5
4 2 a 2 4 8
5 2 b 3 3 9
6 2 c 6 0 0
7 3 a 4 2 8
8 3 b 2 4 8
9 3 c 3 3 9
10 4 a 1 5 5
11 4 b 3 3 9
12 4 c 5 1 5

# merging and joining

dplyr has a lot of functions to merge data frames, and these are especially useful when you may not have an exact match between the levels (so you cant just do a cbind)

A <- data.frame(sub=c("A","B","C","E"),data1=1:4)
B <- data.frame(sub=c("A","B","D","F"),data2=11:14)

* left\_join(A,B) Joins everything into A that is in B

left\_join(A,B, by="sub")

 sub data1 data2
1 A 1 11
2 B 2 12
3 C 3 NA
4 E 4 NA

* right\_join(A,B)

right\_join(A,B, by="sub")

 sub data1 data2
1 A 1 11
2 B 2 12
3 D NA 13
4 F NA 14

\*inner\_join(A,B)

inner\_join(A,B, by="sub")

 sub data1 data2
1 A 1 11
2 B 2 12

\*full\_join(A,B) adds all data, incorporating NAs when one or the other are missing.

full\_join(A,B, by="sub")

 sub data1 data2
1 A 1 11
2 B 2 12
3 C 3 NA
4 E 4 NA
5 D NA 13
6 F NA 14

\*``semi\_join picks out just the first argument for variables where both exist; anti\_join picks out the first argument for those where the second doesn’t exist. These can be useful for imputing data and the like–you can choose the values for which the other value is missing.

semi\_join(A,B, by="sub")

 sub data1
1 A 1
2 B 2

anti\_join(A,B,by="sub")

 sub data1
1 C 3
2 E 4

#Combining data frames row-wise The bind\_rows acts like rbind, stacking two data frames on top o fone another.

##This doesn't make any sense, but it works:
bind\_rows(left\_join(A,B,by="sub"),
 right\_join(A,B,by="sub"))

 sub data1 data2
1 A 1 11
2 B 2 12
3 C 3 NA
4 E 4 NA
5 A 1 11
6 B 2 12
7 D NA 13
8 F NA 14

# Advanced exercises

suppose every other item was reverse coded

dat0$coding <- rep(c(-1,1),6)

Recode using mutate and filter:

d1<-mutate(filter(dat0,coding==1),newdv=dv)
d2<-mutate(filter(dat0,coding==-1),newdv=6-dv)
dat0b <- bind\_rows(d1,d2)
arrange(dat0b,sub,question)

 sub question dv coding newdv
1 1 a 5 -1 1
2 1 b 3 1 3
3 1 c 1 -1 5
4 2 a 2 1 2
5 2 b 3 -1 3
6 2 c 6 1 6
7 3 a 4 -1 2
8 3 b 2 1 2
9 3 c 3 -1 3
10 4 a 1 1 1
11 4 b 3 -1 3
12 4 c 5 1 5

# Big five coding

Load the data set using the big five personality questionnaire.

* The Q1..Q44 are the personality questions. Some are reverse coded, so that the proper coding is 6-X instead of X.
* The questions alternate between 5 factors, but at the end they are a bit off.
* Some of them are reverse coded.

big5 <- read.csv("bigfive.csv")
qtype <- c("E","A","C","N","O", "E","A","C","N","O",
 "E","A","C","N","O", "E","A","C","N","O",
 "E","A","C","N","O", "E","A","C","N","O",
 "E","A","C","N","O", "E","A","C","N","O",
 "O","A","C","O")
valence <- c(1,-1,1,1,1, -1,1,-1,-1,1,
 1,-1,1,1,1, 1,1,-1,1,1,
 -1,1,-1,-1,1, 1,-1,1,1,1,
 -1,1,1,-1,-1, 1,-1,1,1,1,
 -1,1,-1,1)

## Exercise:

Use the above data and dplyr to recode the responses by valence, and then select out each of five personality variables as sums of the proper dimension.

## \*\*\*\*\*

# The reshape2 library

The following gives instructions for using the (older) reshape2 library. The tidyr library is its successor, and can also be used (diffenet function names, different arguments) for doing much of the same thing, but instructions for using that will not be covered here.

Load the library and a survey for examples:

library(reshape2)
dat1 <- read.csv("pooled-survey.csv")
head(dat1)

 subcode question timestamp type time answer
1 207 1 Fri Oct 24 14:27:59 2014 inst 88803
2 207 2 Fri Oct 24 14:28:04 2014 short 5172 20
3 207 3 Fri Oct 24 14:28:11 2014 short 6582 english
4 207 4 Fri Oct 24 14:28:29 2014 short 18461 na
5 207 5 Fri Oct 24 14:28:49 2014 multi 19452 1
6 201 1 Mon Oct 20 17:55:59 2014 inst 29450

Notice that here, we have five questions of different types in a survey, across a bunch of respondents. This is ‘long’ format (what Wickham calls ‘tidy’). What if we want “wide”? We can use dcast to reorganize into a data frame (d= data frame):

dat2 <-dcast(dat1,subcode~question,value.var="answer")
dat2

 subcode 1 2 3 4 5
1 101 20 english na 1
2 102 19 english <NA> 1
3 103 20 English <NA> 1
4 104 18 English <NA> 1
5 201 19 english <NA> 1
6 202 19 english na 1
7 203 19 english na 1
8 204 20 English <NA> 1
9 206 19 english <NA> 1
10 207 20 english na 1
11 209 16 english na 3
12 210 22 english <NA> 1
 [ reached getOption("max.print") -- omitted 12 rows ]

This is good, but the variable names are a bit inconvenient.

colnames(dat2) <- c("subcode","q1","q2","q3","q4","q5")

or, use acast for a vector/matrix. This is not appropriate in this case:

dat3 <-acast(dat1,subcode~question,value.var="answer")
dat3[1:5,]

 1 2 3 4 5
101 20 english na 1
102 19 english <NA> 1
103 20 English <NA> 1
104 18 English <NA> 1
201 19 english <NA> 1
Levels: 1 16 18 19 20 22 3 english English na

What if we want a table of timestamps for each question–maybe to look at how long each one took? Specify this as value.var.

dat4 <-dcast(dat1,subcode~question,value.var="timestamp")
dat4

 subcode 1 2
1 101 Fri Oct 24 11:28:24 2014 Fri Oct 24 11:28:33 2014
2 102 Fri Oct 24 13:03:34 2014 Fri Oct 24 13:03:41 2014
3 103 Fri Nov 07 09:53:40 2014 Fri Nov 07 09:54:06 2014
4 104 Fri Nov 07 12:59:11 2014 Fri Nov 07 12:59:23 2014
5 201 Mon Oct 20 17:55:59 2014 Mon Oct 20 17:56:05 2014
6 202 Thu Oct 23 15:58:06 2014 Thu Oct 23 15:58:13 2014
7 203 Fri Oct 24 09:57:43 2014 Fri Oct 24 09:57:51 2014
8 204 Fri Oct 24 11:36:44 2014 Fri Oct 24 11:37:07 2014
9 206 Fri Oct 24 13:04:24 2014 Fri Oct 24 13:04:28 2014
10 207 Fri Oct 24 14:27:59 2014 Fri Oct 24 14:28:04 2014
11 209 Fri Nov 07 09:55:46 2014 Fri Nov 07 09:55:49 2014
12 210 Fri Nov 07 11:31:02 2014 Fri Nov 07 11:31:13 2014
 3 4
1 Fri Oct 24 11:28:40 2014 Fri Oct 24 11:28:54 2014
2 Fri Oct 24 13:03:45 2014 Fri Oct 24 13:03:54 2014
3 Fri Nov 07 09:54:18 2014 Fri Nov 07 09:54:26 2014
4 Fri Nov 07 12:59:31 2014 Fri Nov 07 12:59:37 2014
5 Mon Oct 20 17:56:12 2014 Mon Oct 20 17:56:19 2014
6 Thu Oct 23 15:58:19 2014 Thu Oct 23 15:58:26 2014
7 Fri Oct 24 09:58:02 2014 Fri Oct 24 09:58:13 2014
8 Fri Oct 24 11:37:11 2014 Fri Oct 24 11:37:17 2014
9 Fri Oct 24 13:04:31 2014 Fri Oct 24 13:04:37 2014
10 Fri Oct 24 14:28:11 2014 Fri Oct 24 14:28:29 2014
11 Fri Nov 07 09:56:03 2014 Fri Nov 07 09:56:11 2014
12 Fri Nov 07 11:31:16 2014 Fri Nov 07 11:31:22 2014
 5
1 Fri Oct 24 11:28:57 2014
2 Fri Oct 24 13:03:57 2014
3 Fri Nov 07 09:54:30 2014
4 Fri Nov 07 12:59:41 2014
5 Mon Oct 20 17:56:22 2014
6 Thu Oct 23 15:58:32 2014
7 Fri Oct 24 09:58:18 2014
8 Fri Oct 24 11:37:22 2014
9 Fri Oct 24 13:04:40 2014
10 Fri Oct 24 14:28:49 2014
11 Fri Nov 07 09:56:17 2014
12 Fri Nov 07 11:31:27 2014
 [ reached getOption("max.print") -- omitted 12 rows ]

Now, do the same for time:

dat4 <-dcast(dat1,subcode~question,value.var="time")
dat4

 subcode 1 2 3 4 5
1 101 32764 9226 6762 13743 3104
2 102 20689 7266 4396 8204 2891
3 103 38236 25939 12205 7573 4403
4 104 45862 12164 7875 5612 4136
5 201 29450 5183 7235 6557 3187
6 202 74307 6757 6266 7033 5502
7 203 34879 7859 11528 10525 5120
8 204 37176 22599 4510 5656 5098
9 206 31742 3629 3055 5933 3415
10 207 88803 5172 6582 18461 19452
11 209 30038 3523 13551 7792 6457
12 210 42821 10280 3643 5601 4601
 [ reached getOption("max.print") -- omitted 12 rows ]

# Using melt to re-form wide data frames

The \*cast function take long (tidy) format and make data frames based on a category label. We can do the opposite too, a process referred to as ‘melting’ (in tidyr, you can use ‘gather’). Before, question was used as the label.

this doesn’t work right. It uses q1..q5 as id varaibles, because they are non-numeric.

melt(dat2)

 q1 q2 q3 q4 q5 variable value
1 20 english na 1 subcode 101
2 19 english <NA> 1 subcode 102
3 20 English <NA> 1 subcode 103
4 18 English <NA> 1 subcode 104
5 19 english <NA> 1 subcode 201
6 19 english na 1 subcode 202
7 19 english na 1 subcode 203
8 20 English <NA> 1 subcode 204
9 19 english <NA> 1 subcode 206
10 20 english na 1 subcode 207
 [ reached getOption("max.print") -- omitted 14 rows ]

Instead, we can specify id.vars, which gets us closer

melt(dat2,id.vars = c("subcode"))

 subcode variable value
1 101 q1
2 102 q1
3 103 q1
4 104 q1
5 201 q1
6 202 q1
7 203 q1
8 204 q1
9 206 q1
10 207 q1
11 209 q1
12 210 q1
13 211 q1
14 212 q1
15 301 q1
16 302 q1
17 303 q1
18 304 q1
19 305 q1
20 306 q1
21 307 q1
22 308 q1
23 309 q1
24 310 q1
25 101 q2 20
 [ reached getOption("max.print") -- omitted 95 rows ]

It is a bit puzzling why this works. It uses only subcode as the id variable. Any variable we wanting tagging each row we can move out of the variable set and into the id set, for example, language:

melt(dat2,id.vars = c("subcode","q3"))

 subcode q3 variable value
1 101 english q1
2 102 english q1
3 103 English q1
4 104 English q1
5 201 english q1
6 202 english q1
7 203 english q1
8 204 English q1
9 206 english q1
10 207 english q1
11 209 english q1
12 210 english q1
13 211 English q1
14 212 English q1
15 301 english q1
16 302 English q1
17 303 English q1
18 304 english q1
 [ reached getOption("max.print") -- omitted 78 rows ]

id.vars specify the variables you want to keep and not split on. These appear several times in the new data . Notice that value.name names the value that the matrix is being unfolded to.

we can name the response like this:

melt(dat2,id.vars = c("subcode","q3"),value.name="response",variable.name="Question")

 subcode q3 Question response
1 101 english q1
2 102 english q1
3 103 English q1
4 104 English q1
5 201 english q1
6 202 english q1
7 203 english q1
8 204 English q1
9 206 english q1
10 207 english q1
11 209 english q1
12 210 english q1
13 211 English q1
14 212 English q1
15 301 english q1
16 302 English q1
17 303 English q1
18 304 english q1
 [ reached getOption("max.print") -- omitted 78 rows ]

Notice that q1 was empty, so we can specify just the measure variables we care about:

melt(dat2,id.vars = c("subcode","q3"),
 measure.vars=c("q2","q4","q5"),
 value.name="response",variable.name="Question")

 subcode q3 Question response
1 101 english q2 20
2 102 english q2 19
3 103 English q2 20
4 104 English q2 18
5 201 english q2 19
6 202 english q2 19
7 203 english q2 19
8 204 English q2 20
9 206 english q2 19
10 207 english q2 20
11 209 english q2 16
12 210 english q2 22
13 211 English q2 20
14 212 English q2 20
15 301 english q2 20
16 302 English q2 20
17 303 English q2 19
18 304 english q2 19
 [ reached getOption("max.print") -- omitted 54 rows ]

# Exercises

* Using the big5 data set, add a unique subject code to each row. Then, use ``melt’’ to create a data frame that has the following columns: subject code, gender, question and answer.

big5 <- read.csv("bigfive.csv")
qtype <- c("E","A","C","N","O", "E","A","C","N","O",
 "E","A","C","N","O", "E","A","C","N","O",
 "E","A","C","N","O", "E","A","C","N","O",
 "E","A","C","N","O", "E","A","C","N","O",
 "O","A","C","O")
valence <- c(1,-1,1,1,1, -1,1,-1,-1,1,
 1,-1,1,1,1, 1,1,-1,1,1,
 -1,1,-1,-1,1, 1,-1,1,1,1,
 -1,1,1,-1,-1, 1,-1,1,1,1,
 -1,1,-1,1)
varnames <- colnames(big5)[2:45]

##first, recode the negative codings.
answers <- select(big5,contains("Q"))

##mutate the columns with -1 valence:
recoded <- answers %>% mutate\_if(valence==-1,function(x){6-x})

melted <- melt(mutate(recoded,sub=1:nrow(recoded)),
 id.vars = c("sub")
 )

arrange(melted,sub,variable)

 sub variable value
1 1 Q1 3
2 1 Q2 2
3 1 Q3 4
4 1 Q4 2
5 1 Q5 3
6 1 Q6 2
7 1 Q7 5
8 1 Q8 2
9 1 Q9 1
10 1 Q10 5
11 1 Q11 3
12 1 Q12 4
13 1 Q13 2
14 1 Q14 4
15 1 Q15 4
16 1 Q16 2
17 1 Q17 5
18 1 Q18 2
19 1 Q19 1
20 1 Q20 4
21 1 Q21 4
22 1 Q22 5
23 1 Q23 3
24 1 Q24 1
25 1 Q25 4
 [ reached getOption("max.print") -- omitted 5563 rows ]

# Solution to Exercise 1.

big5 <- read.csv("bigfive.csv")
qtype <- c("E","A","C","N","O", "E","A","C","N","O",
 "E","A","C","N","O", "E","A","C","N","O",
 "E","A","C","N","O", "E","A","C","N","O",
 "E","A","C","N","O", "E","A","C","N","O",
 "O","A","C","O")
valence <- c(1,-1,1,1,1, -1,1,-1,-1,1,
 1,-1,1,1,1, 1,1,-1,1,1,
 -1,1,-1,-1,1, 1,-1,1,1,1,
 -1,1,1,-1,-1, 1,-1,1,1,1,
 -1,1,-1,1)
varnames <- colnames(big5)[2:45]

##first, recode the negative codings.
answers <- select(big5,contains("Q"))

##mutate the columns with -1 valence:
recoded <- answers %>% mutate\_if(valence==-1,function(x){6-x})

##check this. For negative valence, 2 becomes 4 etc.
bind\_rows(recoded[1,],answers[1,])

 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15 Q16 Q17 Q18 Q19 Q20
1 3 2 4 2 3 2 5 2 1 5 3 4 2 4 4 2 5 2 1 4
 Q21 Q22 Q23 Q24 Q25 Q26 Q27 Q28 Q29 Q30 Q31 Q32 Q33 Q34 Q35 Q36 Q37 Q38
1 4 5 3 1 4 4 2 3 4 2 1 3 5 2 1 3 3 2
 Q39 Q40 Q41 Q42 Q43 Q44
1 3 1 2 2 2 4
 [ reached getOption("max.print") -- omitted 1 row ]

##create composite subsets
b5.e <- select(recoded, one\_of(varnames[qtype=="E"]))
b5.a <- select(recoded, one\_of(varnames[qtype=="A"]))
b5.c <- select(recoded, one\_of(varnames[qtype=="C"]))
b5.n <- select(recoded, one\_of(varnames[qtype=="N"]))
b5.o <- select(recoded, one\_of(varnames[qtype=="O"]))

composites1 <- data.frame(e=rowMeans(b5.e,na.rm=T),
 a=rowMeans(b5.a,na.rm=T),
 c=rowMeans(b5.c,na.rm=T),
 n=rowMeans(b5.n,na.rm=T),
 o=rowMeans(b5.o,na.rm=T)
 )