Intro to dplyr

Contents

```r
acitelli <- read.csv("/Users/randigarcia/Desktop/acitelli.csv", header=TRUE)
```

Data Manipulation (data cleaning)

We'll use the package `dplyr`. The `dplyr` package contains the functions for all of the data cleaning varbs: `filter()`, `mutate()`, `rename()`, `arrange()`, `select()`, `summarize()`, and `group_by()`. You can find a cheat sheet for `dplyr` here.

```r
#install.packages("dplyr")
library(dplyr)
```

Filtering cases with filter()

First, let's filter cases. We can make a dataset of men only. Notice that we used a double equal sign, `==`, instead of single, `=`. When you want to ask if something is equal to some value or another variable, that is, you want to use equal in a *logical statement*, you need the double equal. You can also use `>` `<` `>=` `<=` `&`, which means **AND**, and finally, `|` which means **OR**.

```r
menOnly <- filter(acitelli, gender == 1)
```

We could also use the pipe, `%>%`.

```r
menOnly <- acitelli %>%
  filter(gender ==1)
```

We can save this new data set in our files as a csv. Note that the file path conventions are differnt for Windows machines. If copying and pasting from properties, you will need to change backslashes to forward slashes.

```r
write.csv(menOnly,
  file = "/Users/randigarcia/Desktop/Three-day-workshop/R Workshop/Data/men.csv")
```

It's in your working directory, which you can find with the following command. It is an empty function, so it looks weird.

```r
getwd()
```

How about only the men who are above the median for `Yearsmar`. First, find the median years married.

```r
#use a function in the mosaic package
```

Then, filter for men above that cut off point.

```r
mature_hus <- menOnly %>%
  filter(Yearsmar > -1.089)
```

Instead of first finding the median with `favstats()`, we could ask for the median inside of `filter()` with the Base R function, `median()`. Base R has all of the descriptive stats functions you'd expect, `mean()`, `sd()`, `cor()`, but be careful because if you have missing data you'll have to add `rm.na = TRUE` as an argument to the function. The syntax also differs from `mosaic`.

```r
mature_hus <- menOnly %>%
  filter(Yearsmar > median(Yearsmar))
```

Adding new variables with `mutate()`

Let's add a new categorical variable that marks the median split on Yearsmar. After you create it, take a look at it.

```r
menOnly <- menOnly %>%
  mutate(mature_hus = Yearsmar > median(Yearsmar))
```

How would you get the frequncies on this vairable?

Now for a sanity check, how would you get the descriptive stats split by mature and non-mature husbands?

Renaming variable with `rename()`

We copy a variable and give it a new name with a function you already know, `mutate()`.

```r
menOnly <- menOnly %>%
  mutate(old_hus = mature_hus)
```

We can rename a variable without creating a new one with `rename()`. This is handy if you forget to name variables in Qualtrics!

```r
menOnly <- menOnly %>%
  rename(wise_hus = old_hus)
```

We can rename a bunch at the same time. This is handy if you forget to name variables in Qualtrics!

```r
menOnly <- menOnly %>%
  rename(self_positivity = self_pos,
         other_positivity = other_pos)
```

Sorting with `arrange()`

First, you should know that you can sort in the viewer by clicking the (faint) arrows just to the right of each variable name. Give it a try. It's often handy to have a sort command in your code, and/or you might want to sort by more than one variable.

```r
head(acitelli)
```

```
##   cuplid  Yearsmar gender self_pos other_pos satisfaction tension simhob
## 1      3  8.202667     -1      4.8       4.6     4.000000     1.5      0
## 2      3  8.202667      1      3.8       4.0     3.666667     2.5      1
## 3     10 10.452667     -1      4.6       3.8     3.166667     4.0      0
## 4     10 10.452667      1      4.2       4.0     3.666667     2.0      0
## 5     11 -8.297333     -1      5.0       4.4     3.833333     2.5      0
## 6     11 -8.297333      1      4.2       4.8     3.833333     2.5      0
```

Say we want to take a peak at the women with the bottom 6 `self_pos` scores.

```r
acitelli %>%
  arrange(gender, self_pos) %>%
  head()
```

```
##   cuplid   Yearsmar gender self_pos other_pos satisfaction tension simhob
## 1    160  8.7026667     -1      3.2       3.8     3.333333     4.0      0
## 2     52 13.1193333     -1      3.4       3.8     3.833333     2.0      1
## 3    441  0.1193333     -1      3.4       4.4     4.000000     3.0      0
## 4     70 11.3693333     -1      3.6       4.4     3.833333     1.5      0
## 5    116  4.7860000     -1      3.6       4.2     2.333333     4.0      0
## 6    178 -7.0473333     -1      3.6       3.6     2.666667     3.0      0
```

We could also save the arranged dataset.

```
acitelli <- acitelli %>%
  arrange(gender, self_pos)

head(acitelli)
```

```
##   cuplid   Yearsmar gender self_pos other_pos satisfaction tension simhob
## 1    160  8.7026667     -1      3.2       3.8     3.333333     4.0      0
## 2     52 13.1193333     -1      3.4       3.8     3.833333     2.0      1
## 3    441  0.1193333     -1      3.4       4.4     4.000000     3.0      0
## 4     70 11.3693333     -1      3.6       4.4     3.833333     1.5      0
## 5    116  4.7860000     -1      3.6       4.2     2.333333     4.0      0
## 6    178 -7.0473333     -1      3.6       3.6     2.666667     3.0      0
```

What about the top 6? We can use the `desc()` function inside of `arrange()`.

```
acitelli %>%
  arrange(gender, desc(self_pos)) %>%
  head()
```

```
##   cuplid  Yearsmar gender self_pos other_pos satisfaction tension simhob
## 1     11 -8.297333     -1        5       4.4     3.833333     2.5      0
## 2     98 -9.214000     -1        5       4.2     4.000000     2.0      1
## 3    114 12.619333     -1        5       3.4     3.666667     2.5      0
## 4    127  3.619333     -1        5       4.6     3.833333     2.0      0
## 5    135  7.786000     -1        5       5.0     4.000000     1.5      0
## 6    177 11.619333     -1        5       5.0     4.000000     1.0      1
```

Selecting variables with `select()`

Save a smaller subset of variables.

```
small <- acitelli %>%
  select(cuplid, gender, satisfaction, self_pos)
```

We can also save everything but some variable(s).

```
no_tension <- acitelli %>%
  select(-tension)
```

Descriptive statistics with `summarize()`

```
acitelli %>%
  summarize(mean = mean(satisfaction),
            sd = sd(satisfaction),
            min = min(satisfaction))
```

```
##       mean        sd      min
## 1 3.60473 0.4964205 1.166667
```

Grouped descriptives with `group_by()`

We can split the file and view results grouped by some variable.

```
acitelli %>%
  group_by(gender) %>%
  summarize(mean = mean(satisfaction),
            sd = sd(satisfaction),
            min = min(satisfaction))
```

```
## # A tibble: 2 x 4
##   gender     mean        sd      min
##    <int>    <dbl>     <dbl>    <dbl>
## 1     -1 3.591216 0.5300260 1.500000
## 2      1 3.618243 0.4617875 1.166667
```

You can use `group_by()` to create aggregated variables, this is handy if you have nested data. We actually do have married couples here, so let's create a dyad mean tension variable.

```
acitelli <- acitelli %>%
  group_by(cuplid) %>%
  mutate(tension_mean = mean(tension)) %>%
  ungroup()

#this last command is not entirely nessesary, but good practice
```

Pipelines

We now seen our first piplines, using `group_by()`. Now we can make a pipeline of many of the commands I did above. The last thing I do is drop useless `gender` variable, because the resulting dataset if all men.

```
mature_hus2 <- acitelli %>%
  filter(gender == 1) %>%
  mutate(wise_hus = Yearsmar > median(Yearsmar)) %>%
  rename(self_positivity = self_pos,
         other_positivity = other_pos,
         personID = cuplid) %>%
  arrange(wise_hus) %>%
  select(-gender)
```

Save a dataset of women who are perceiving above the mean tension, and drop the `simhob` variable.

What are the couple ID's of the couples with the lowest 3 average satisfaction scores?