Lab 1: Introduction to R and RStudio

Stats and sports class

Fall 2019

Part I: Getting used to R

Step 1. Open RStudio by going to http://r.skidmore.edu/ or using your own personal computer.

Step 2. Open a new R Markdown file (File / New File / R Markdown...). You can create a basic name – Lab1, for example – and that'll set you up with a new file ready to go.

Part II: Understanding R Markdown

We'll start this lab by reviewing the following as a class:

1. How does RMarkdown work?
2. What does it mean to "Knit" and compile?
3. What is a code chunk?
4. How can one change the top line of a code chunk to dictate what does and doesn't show up in a Markdown output?
5. How do headings work?
6. What's the best format to do labs and homeworks?

Part III: Packages

R is open source, and many contributors have built packages that take original R code, repurpose it, and make R more user-friendly, faster, and engaging. We'll need several packages this semester. One example is the `tidyverse`, which contains data wrangling and visualization frameworks.

To install the `tidyverse` package:

```
install.packages("tidyverse")
```

Note It's okay if you see some warnings. R will tell you anytime there's a *major* error.

Hint: You only have to install a package once. Once it's installed, you don't need to install it again

When you've installed a package, you can load it using

```
library(tidyverse)
```

When compiling a Markdown file, do not include code that installs packages. Markdown will not compile.

Part IV: Univariate and bivariate analysis using baseball statistics.

In this lab we'll be looking at data from baseball teams and players.

To get you started, enter the following in one of the code chunks:

```
library(Lahman) ## If you don't have the `Lahman` package, run "install.packages("Lahman")"
library(tidyverse)
```

Let's load up the data. The data set `Teams` contains team-level information for each MLB team in each season. We're going to focus on the 2016 season.

If you want to learn more about the Teams data set, type `?Teams` and scroll through the help window in the bottom right panel.

```
teams_2016 <- Teams %>% filter(yearID == 2016)
teams_2016
```

What you should see are several columns of numbers, each row representing a different team: the first entry in each row is simply the row number (an index we can use to access the data from individual years if we want), the second is the year, and the remaining columns are team-specific metrics.

Note that the row numbers in the first column are not part of the MLB data. R adds them as part of its printout to help you make visual comparisons. You can think of them as the index that you see on the left side of a spreadsheet. In fact, the comparison to a spreadsheet will generally be helpful. R has stored MLB data in a kind of spreadsheet or table called a *data frame*.

You can `x` out of the data – you will likely not need to refer back to it in this Lab.

You can see the dimensions of this data frame by typing:

```
dim(teams_2016)
```

This command should output `[1] 30 48`, indicating that there are 30 rows and 48 columns (we'll get to what the `[1]` means in a bit).

```
names(Teams.2016)
```

You should see that the data frame contains the columns such as `yearID`, `W`, and `BB`, etc.

The tidyverse

The tidyverse is based on the `%>%` symbol, called a pipe, which takes whatever code is written before the pipe and accordingly does something with it. In the following steps, we'll learn several commands that are a part of the tidyverse.

Finding rows that fit certain criterion: filter

The `filter()` command we used above – to find team data from the 2016 season. We'll use it early and often to take larger data sets and trim them such that the units that remain are the ones we need.

Which years show up in the following data code?

```
Teams %>% filter(yearID > 2014, yearID <= 2016)
```

Which teams show up in each line below?

```
Teams %>% filter(yearID == 2016, lgID == "AL")
Teams %>% filter(yearID == 2016, lgID != "AL")
```

Selecting certain variables: select()

Let's start to examine the data a little more closely. We can access the data in a single column of a data frame separately using a command like

```
teams_2016 %>% select(yearID, teamID, W, L) %>% head()
teams_2016 %>% select(yearID, teamID, W, L) %>% tail()
```

This command will only show certain variables for each team. *Note*: The `head()` and `tail()` commands show the first six and last six rows, respectively.

Q1. Add the number of hits (`H`) variable to the code above.

To obtain several variables at once, we can use the same `select()` command, with a colon in between variables.

```
teams_2016 %>% select (yearID:SF) %>% head()
```

To make the rest of the lab a bit easier, let's focus on variables that relate to how well a team hits.

```
teams_2016_batting <- teams_2016 %>% select (yearID:teamID, R:SF)
```

Creating new variables: mutate()

R can do lots of things besides graphs. In fact, R is really just a big calculator. We can type in mathematical expressions like

```
210 + 930
x <- 10
x
x + 5
```

To get R to do calculations for each row in a data set, there is a faster way: `mutate()`

```
teams_2016_batting <- teams_2016_batting %>%
  mutate(extra_base_hits = X2B + X3B + HR)

teams_2016_batting %>% head()
```

Can you see the new variable called `extra_base_hits`?

One important aspect about naming variables: keep them simple, and avoid funny symbols. The above is named `extra_base_hits`, but could have easily been named `total_number_of_extra_base_hits` or something like that, or even `we_love_prof_lopez` if you got antsy.

A bad variable name? `doubles and triples and HRs`. I don't know what that is, but it shouldn't be a variable if it is too long or has any spaces.

In addition to simple mathematical operators like subtraction and division, you can ask R to make comparisons like greater than, `>`, less than, `<`, and equality, `==`. For example, we can ask which home run observations were greater than 200.

```
teams_2016_batting <- teams_2016_batting %>%
  mutate(big_power = HR > 200)

teams_2016_batting %>% head()
```

This output shows a different kind of data than we have considered so far. Here, we've asked R to create *logical* data, data where the values are either `TRUE` or `FALSE`. In general, data analysis will involve many

different kinds of data types, and one reason for using R is that it is able to represent and compute with many of them.

Q2. Which MLB teams had more than 200 home runs in 2016? Use `filter()` and your `big_power` variable to find out.

Finding high's and low's: `arrange()`

The `arrange` command can make identifying the best or worst teams or players quite simple.

```
teams_2016_batting %>% arrange(R)
teams_2016_batting %>% arrange(-R)
```

Q3. What is the difference between each of the two lines of code above?

Q4. Edit the code above. Identify team had the highest *number of strikeouts* (`SO`)? Which team had the lowest?

Graphs in the tidyverse: `ggplot`

R has some powerful functions for making graphics. The centerpiece of the `tidyverse` syntax is the use of the *ggplot* package. In this case, `gg` stands for grammar of graphics. You can learn more about ggplot at https://ggplot2.tidyverse.org/.

Two continuous variables

For example, we can create a simple plot of the number of runs each team scored versus its hits

```
ggplot(data = teams_2016_batting, aes(x = R, y = H)) +
         geom_point()
```

By default, R creates a scatterplot with each x,y pair indicated by an open circle. The plot itself should appear under the *Plots* tab of the lower right panel of RStudio. Notice that the command above again looks like a function, this time with two arguments separated by a comma. The first argument specifies the data, and the two aesthetics (aes) specify the variables.

Q5. Is there an association between runs scored and hits? How would you describe it?

One continuous variable

Now, suppose we want to plot only the total number of runs.

```
ggplot(data = teams_2016_batting, aes(x = R)) +
         geom_histogram()

ggplot(data = teams_2016_batting, aes(x = R)) +
         geom_histogram(binwidth = 30)


ggplot(data = teams_2016_batting, aes(x = R)) +
         geom_density()
```

Q6. Describe the distribution of runs scored among MLB teams in 2016.

Q7. Describe how changing the binwidth impacts the histogram of runs scored. What happens if bins are too big? too small?

Q8. What does the density curve show slightly better than the histogram? What does the histogram show slightly better than the density curve?

Continuous variables by categorical variables.

Perhaps we decide to compare different groups – in this case leagues. Here, we use boxplots to compare the different leagues using the `SB` (stolen base) variable.

```
ggplot(teams_2016_batting, aes(x = lgID, y = SB)) +
  geom_boxplot()

ggplot(teams_2016_batting, aes(x = SB, colour = lgID)) +
  geom_density()
```

Above, note that the

Q9. Describe the differences between stolen bases among MLB teams in the 2016 season, comparing the National league to the American League.

Q10. What features are apparent in the density curves that are not apparent in the boxplots? What features are apparent in the boxplots that aren't apparent in density curves?

Q11. Why would histograms make it difficult to compare several distributions simultaneously on one graph?

On your own

Note: These questions are often continued into the homework.

1. Arrange the `teams_2016_batting` data set by the `SB` variable. Which teams had the most and fewest stolen bases?
2. Make a graph of team wins during this season. Is the distribution of wins skewed left, right, or symmetric?
3. Teams play 162 games. Make a new variable, `win_pct`, which identifies the percent of games won by each team.
4. Describe the center, shape, and spread of the `X3B` variable – split by each league – using an appropriate plot.
5. How can you change the x and y labels on your plots? How can you add a title? Use google to guide you, and update your plot in Question 3. One trick: include `ggplot` in your google search.
6. Moneyball was based on which team-statistics most strongly correlated to runs. Though there are some variables that already exist in the data, the code below creates batting average, on base percentage, and slugging percentage.

```
teams_2016_batting <- teams_2016_batting %>%
  mutate(BA = (H/AB),
         OBP = (H + BB)/(AB + BB),
         SLG = ((H - X2B - X3B - HR)*1 + X2B*2 + X3B*3 + HR*4)/AB)
```

Using visual evidence, find the variable that you think seems to boast the strongest association to runs (`R`).

6. Estimate the correlation between (i) slugging percentage and runs, (ii) on base percentage and runs and (iii) batting average and runs. Which would you prioritize as a coach using these results? Why?
7. Create a new variable for whether or not a team won 85 games or more. You can call this variable whatever name you want. How many teams won 85 games or more? Use your new variable to find out.