

Lab 1: Introduction to R and RStudio using baseball stats

Michael Lopez, Skidmore College

Important note 1

First, we'll open RStudio by going to <http://r.skidmore.edu/>.

We'll be there all semester – don't be afraid to bookmark this page on your web browser.

Important note 2

Second, we'll be using R Markdown for *all* of our class labs and homeworks.

R Markdown simultaneously stores your commands, executes your commands, and generates an output file automatically, we will use R Markdown files to store our code and answers for *all* Lab and Homework assignments.

Open a new R Markdown file (File / New File / R Markdown...). You can create a basic name – Lab0, for example – and that'll set you up with a new file ready to go.

Important note 3

The base RMarkdown file is not particularly useful. Instead, click on the link below:

https://raw.githubusercontent.com/statsbylopez/FYE_18/master/HWs/HW_lab_base.Rmd

Copy and paste the text at the link above into your Markdown document.

Set the output to HTML mode, and click “Knit HTML” at the top (or Command+Shift+K on Mac). This should produce a web page with the knitting procedure executing your code blocks. You can edit this new file to produce your homework submission. You will also print this and hand in for each homework assignment.

Lab 1: Introduction to R and RStudio

The movie Moneyball focuses on the “quest for the secret of success in baseball”. It follows a low-budget team, the Oakland Athletics, who believed that underused statistics, such as a player's ability to get on base, better predict the ability to score runs than typical statistics like home runs, RBIs (runs batted in), and batting average. Obtaining players who excelled in these underused statistics turned out to be much more affordable for the team.

In this lab we'll be looking at data from all 30 Major League Baseball teams and examining the relationships between in-game statistics. Our primary aim, for today, is a familiarity with R and RStudio, which you'll be using throughout the course both to learn the statistical concepts discussed in class and also to analyze real data and come to informed conclusions.

To straighten out which is which: R is the name of the programming language itself and RStudio is a convenient interface.

As the labs progress, you are encouraged to explore beyond what the labs dictate; a willingness to experiment will make you a much better programmer. Before we get to that stage, however, you need to build some basic fluency in R. Today we begin with the fundamental building blocks of R and RStudio: the interface, reading in data, and basic commands.

The panel in the upper right contains your *workspace* as well as a history of the commands that you've previously entered. Any plots that you generate will show up in the panel in the lower right corner.

The panel on the left is where the action happens. It's called the *console*. Everytime you launch RStudio, it will have the same text at the top of the console telling you the version of R that you're running. Below that information is the *prompt*. As its name suggests, this prompt is really a request, a request for a command. Initially, interacting with R is all about typing commands and interpreting the output. These commands and their syntax have evolved over decades (literally) and now provide what many users feel is a fairly natural way to access data and organize, describe, and invoke statistical computations.

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

```
library(Lahman)
library(tidyverse)
```

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

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.

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

There are additional sets of packages called the **tidyverse** that streamline most of the commands that you will need in this course.

To use the package, you will first need to load it.

```
library(tidyverse)
```

Note that you will have to do this *every* time you start a new R session.

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.

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.

1. 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: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(total.HR.K = HR + SO)
Teams.2016.batting %>% head()
```

Can you see the new variable called `total.HR.K`?

One important aspect about naming variables: keep them simple, and avoid funny symbols. The above is named `total.HR.K`, but could have easily been named `total.hr.strikeout` or something like that, or even `total.hr.proflopezrules.strikeout` if you got antsy.

A bad variable name? `strikeouts` and `ks`, or `strikeouts&%ks`, or `ToTaLkS%&sc`. I don't know what that is, but it shouldn't be a variable.

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. In the `mlb11` data frame, most of our values are numerical. 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.

2. Which MLB teams had more than 200 home runs in 2011? *Hint*: Use the `select` command

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)
```

3. What is the difference between each of the two lines of code above?
4. Edit the code above. Identify team had the highest *number of strikeouts* in 2011?

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.

Two continuous variables

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

```
ggplot(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.

5. 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(Teams.2016.batting, aes(x = R)) +
  geom_histogram()

ggplot(Teams.2016.batting, aes(x = R)) +
  geom_histogram(binwidth = 40)
```

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

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()
```

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

On your own

Note: These questions are often continued into the homework.

- Arrange the `Teams.2016.batting` data set by the `SB` variable. Which teams had the most and fewest stolen bases?
- Make a graph of team wins during this season. Is it more appropriate to use a boxplot or a scatter plot? Is the distribution of wins skewed left, right, or symmetric?
- Describe the center, shape, and spread of the `X3B` variable, using an appropriate plot.
- 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.
- 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`).

- Identify 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?
- 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.

Portions of this lab were adapted for OpenIntro by Andrew Bray and Mine Çetinkaya-Rundel from a lab written by Mark Hansen of UCLA Statistics, a product of OpenIntro that is released under a Creative Commons License

R Markdown simultaneously stores your commands, executes your commands, and generates an output file automatically, we will use R Markdown files to store our code and answers for *all* Lab and Homework assignments.

Open a new R Markdown file (File / New File / R Markdown...). You can create a basic name – Lab 1, for example – and that’ll set you up with a new file ready to go.

Set the output to HTML mode, and click “Knit HTML” at the top (or Command+Shift+K on Mac). This should produce a web page with the knitting procedure executing your code blocks. You can edit this new file to produce your homework submission. You will also print this and hand in for each homework assignment.

Notice how the text you write in the `.Rmd` file shows up in the output file each time you click “Knit HTML”.

General instructions for labs:

- Open the R Markdown file (`.Rmd`) from the course Github page
- Change the heading to include your author name
- Save the R Markdown file (named as: `[MikeID]-[Lab01].Rmd` – e.g. “mlopez-Lab01.Rmd”) to somewhere where you’ll be able to access it later (zip drive, My Documents, Dropbox, etc)

- Your file should contain the code/commands to answer each question in its own code block, which will also produce plots that will be automatically embedded in the output file
- Each answer must be supported by written statements (unless otherwise specified) as well as any code used

Additional instructions for HWs:

- Include the names of anyone you collaborated with at the top of the assignment
- For Lab 01 only, do not worry about adhering to a particular style guide. We will cover this eventually

Today's lab will introduce different aspects of R. We will reference a cheatsheet with tips/tricks on how to format things in R Markdown, hosted [here](#). Prior to starting the lab, please download the handout and open it on your computer screen.

Background

To straighten out which is which: R is the name of the programming language itself and RStudio is a convenient interface.

As the labs progress, you are encouraged to explore beyond what the labs dictate; a willingness to experiment will make you a much better programmer. Before we get to that stage, however, you need to build some basic fluency in R. Today we begin with the fundamental building blocks of R and RStudio: the interface, reading in data, and basic commands.

The panel in the upper right contains your *workspace* as well as a history of the commands that you've previously entered. Any plots that you generate will show up in the panel in the lower right corner.

The panel on the left is where the action happens. It's called the *console*. Everytime you launch RStudio, it will have the same text at the top of the console telling you the version of R that you're running. Below that information is the *prompt*. As its name suggests, this prompt is really a request, a request for a command. Initially, interacting with R is all about typing commands and interpreting the output. These commands and their syntax have evolved over decades (literally) and now provide what many users feel is a fairly natural way to access data and organize, describe, and invoke statistical computations.

Storing Your Work and Storing Your R Commands

Code Blocks:

When you open your new R Markdown file, you should see a block of code (beginning with three apostrophes and “{r}” and ending with three more apostrophes). This block should have a darker gray background color.

This is a code block! You can type commands into this block, and they will be executed by R and included in your output.

0. Play around with the following command by printing different things. Each time, knit.

```
x <- 12
2*x
```

```
## [1] 24
```

```
print(x)
```

```
## [1] 12
```

```
x^2
```

```
## [1] 144
```

Code Comments:

Comments should be used frequently when writing code to give insight into what each piece of code is doing. To add a comment to your code, start a new line with the `#` symbol.

1. Add a comment in your first code block that says “I am printing ‘Hello, World!’ in R Markdown”. Add another comment that says what your major is.

```
print("Hello, World!")
```

```
## [1] "Hello, World!"
```

Notice that the text in the comments shows up in the code block in the output file, but not in the output block when you click “Knit HTML”. Comments exist to help you and others who read your code.

Where is my output file?

Find where you stored your [MikeID]-Lab01.Rmd file on your computer.

In the same directory, there should be a file called [MikeID]-Lab01.html.

Open it. It should automatically open in a browser, and it should contain the output.

In the future, you’ll print and submit these .html files for homeworks and labs.

2. Create a single code block that contains R commands to do all of the following:

- a. $(6+3)*4 - 5$
- b. 4^2
- c. e^{-5} (hint: type `help(exp)` at the command line in R Studio)
- d. Let $y=8$. Let $x = 5-6y$. Print x .
- e. Repeat (d) for $y=5/6$.

Built-in Help and Documentation

3. Using the `help()` and `help.search()` functions at the command line, use R to do the following:

- a. Find the help documentation for the `quantile` function by entering `help(quantile)` or `?quantile`. This function takes a vector of numbers and computes quantiles for the vectors. What is the description of the `probs` argument?
- b. Find the help documentation for the `mean` function. This function takes a vector of numbers and computes their average. What is the example code at the bottom of the help page?
- c. Use the help pages to find the name of the function in R that finds the standard deviation of a vector.

Note that because everything is online, you can use online search engines to achieve many of these same goals. Feel free to do so for all your future assignments.

Loading a library in R

4. In R, there are several libraries or packages/groups of programs that are not used enough to have them permanently stored in R. We can load the library into R by typing `library(library-name)` at the command line. (Sometimes we need to download the library first; more on this later.)

- a. Load the `MASS` library into R by entering `library(MASS)`. Open the help documentation for the `MASS` package. What is the official name of this `MASS` package? It may help to use `library(help=MASS)` to solve this problem.
- b. Load the `datasets` library. Find the help documentation for its `trees` dataset. Describe this dataset using information from the help pages.

- c. Load the **graphics** library into R and open its help documentation. This library is full of graphics/visualization functions that we may use in this class. Find a function that creates a 1-D Scatterplot. Describe its argument `x`.

Reading in Data

We will start by looking at the airline delay data set, part of an online package ‘nycflights13’, created by Hadley Wickham, which contains on-time data for all flights departing NYC in 2013, as well as data on airlines, airports, weather, and planes.

To get the package on your computer, follow the code below.

```
install.packages("nycflights13")
library(nycflights13)
flights <- na.omit(flights)
```

Note that you only need to install an R packages **once** per computer. It’ll be there again.

5. Complete the following

- Start a new code block. Type `flights %>% head(nrows = 10)`. What does the `head` function do?
- How many rows does `flights` have? Hint: `help(nrow)`
- What happens when you add `, nrows = 10` inside the parentheses in the line of code in (a)?
- How many columns does `flights` have? Hint: `help(ncol)`
- How can you get the number of rows and columns simultaneously? Hint: `help(dim)`

Data Management

6. Data management questions

- The `c()` function concatenates numbers into a vector. For example, `myvector <- c(1,3,6,9)` creates a vector that has four elements. Use `c()` to create a vector called `myvector` that contains all the perfect squares (1,4,9,16, ...) less than 100.
- We can access specific rows of a matrix with `[,]`. For example, `flights[2,]` will present the second row of the matrix, and `flights[c(2,4),]` will yield a new matrix with the second and fourth rows of the original only. What was the destination of the 4th flight in this data set?
- We can use the same operator to access columns of a matrix – for example, `flights[,2]` will present the second column of the matrix, and `flights[,c(2,4)]` will yield a new matrix with the second and fourth columns of the original only. Assign the sixth column of `flights` to a variable named `column6`. Use the `min()` and `max()` functions to find the minimum and maximum of this column.
- To get a single element in a matrix, we use row and column numbers, such as `flights[r,c]`. This will also extract a submatrix if multiple entries are used for each. Obtain and display the element in `flights`’s 5th row, 3rd column. Then obtain and display the submatrix `flights[c(1,2),c(3,4)]`.
- We can also ask questions about our data/variables. The `==` operator is a True/False test of equality. For example, `column6 == 0` returns `TRUE` for every element in `column6` that is equal to 0, and `FALSE` for elements different than 1. We can do similar things with the symbols `<`, `<=`, `>`, `>=`. You can also combine two conditions with the AND (`&`) and OR (`|`) operators:
 - `column6 >= -5 & column6 < 5`: `column6` is greater than or equal to -5 AND `column6` is less than 5
 - `sum(column6 == 0)`: counts how many 0s there are in `column6` (while ignoring missing numbers).
 - `which(column6 == 1)`: tells you the positions of the elements in `row4` that are equal to 1.

How many items in `column6` are greater than 0? What does this represent?

Markdown syntax

Return to the cheatsheet for Markdown. By adding different comments to the “{r, }” heading of a code-box, we can provide Markdown alternative ways of printing code.

7. Per the cheatsheet, learn the `eval`, `echo`, and `warning` messages do.

```
x <- 12
```

```
2*x
```

```
## [1] 24
```

```
print(x)
```

```
## [1] 12
```

```
x^2
```

```
## [1] 144
```