Introduction to R and RStudio

Michael Lopez, Skidmore College

Lab 0: Introduction to R and RStudio

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.

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.

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.

It may be helpful to use the LabXX.Rmd file on Github as a template for your submission. Feel free to do so.

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

Todays 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.

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

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

```r
print(x)
```

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

```r
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.

```r
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, aiports, weather, and planes.

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

```r
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

a. Start a new code block. Type `flights %>% head(nrows = 10)`. What does the `head` function do?

b. How many rows does `flights` have? Hint: `help(nrow)`

c. What happens when you add `, nrows = 10` inside the parentheses in the line of code in (a)?

d. How many columns does `flights` have? Hint: `help(ncol)`

e. How can you get the number of rows and columns simultaneously? Hint: `help(dim)`

Data Management

6. Data management questions

a. 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.

b. 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?

c. 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.

d. 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)]`.

e. 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 commends 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
```