Exercise #0: Learning the basics of R

Patrick Applegate, patrick.applegate@psu.edu

7 September 2015

Learning Goals

After completing this exercise, you should be able to

- install R, RStudio, and `swirl` on your computer
- use `swirl` to learn the basics of R, including
  - assigning values to variables in R
  - the differences between vectors, matrices, and data frames
  - using indexing to extract individual values, or ranges of values, from variables
- modify an R script to produce a desired result
- get help by
  - searching R's built-in documentation
  - searching the Web
  - asking questions in a way that increases your chance of receiving a helpful response

Introduction

The computing environment and programming language R is a powerful tool for risk analysis. This exercise will guide you through installing R and its integrated development environment RStudio. You will also learn the basics of using R at the command line using `swirl`, plus how to run scripts in R. Finally, you'll become familiar with different ways of getting help when you encounter problems with your R code.

Tutorial

Installing R and RStudio

To install R, navigate to https://cran.rstudio.com and look for the link that says, "Download R for *[name of your operating system]*," near the top of the page. Click on this link and follow the instructions for installing the software. If you run into problems, look at the FAQ, which is linked to from the same page.

R's basic development environment is functional, but somewhat crude. RStudio provides a much cleaner environment for working with R. There are presently two versions of RStudio, one for companies (which costs money) and one for private individuals and academics (which is free). There are also versions for servers, as well as desktop computers. The free, desktop version works perfectly well for our purposes. **To install RStudio**, navigate to https://www.rstudio.com/products/RStudio/ and download the Open Source Edition of RStudio Desktop.

The RStudio working environment

Open RStudio. You should see a window that looks something like Figure 1. The RStudio Web page provides documentation on how to use RStudio, but we provide a short description here for the sake of completeness.

RStudio divides its window up into four different sub-windows, most of which have tabs. These sub-windows, and their tabs, are as follows.

- The **upper left** sub-window is where you write R scripts (series of R commands that are stored in text files). You can have multiple scripts open at the same time, and clicking on the different tabs lets you switch between these scripts.
- The **lower left** sub-window is the Console window, where you enter R commands one at a time and the output appears immediately in the same window. Some output from running scripts appears here, too. (The other tab, R Markdown, isn't important for our purposes here.)
- The **upper right** sub-window usually shows the variables that presently exist in R's working memory (if the Environment tab is visible). The other tab, History, shows the commands that have been entered in the Console window.
- The **lower right** sub-window is usually set to show either Plots or Help. The other tabs, Files, Packages, and Viewer, are less important for our purposes.

Most often, you will work in the Script sub-window, with occasional jumps to the Console, Plots, and Help sub-windows.

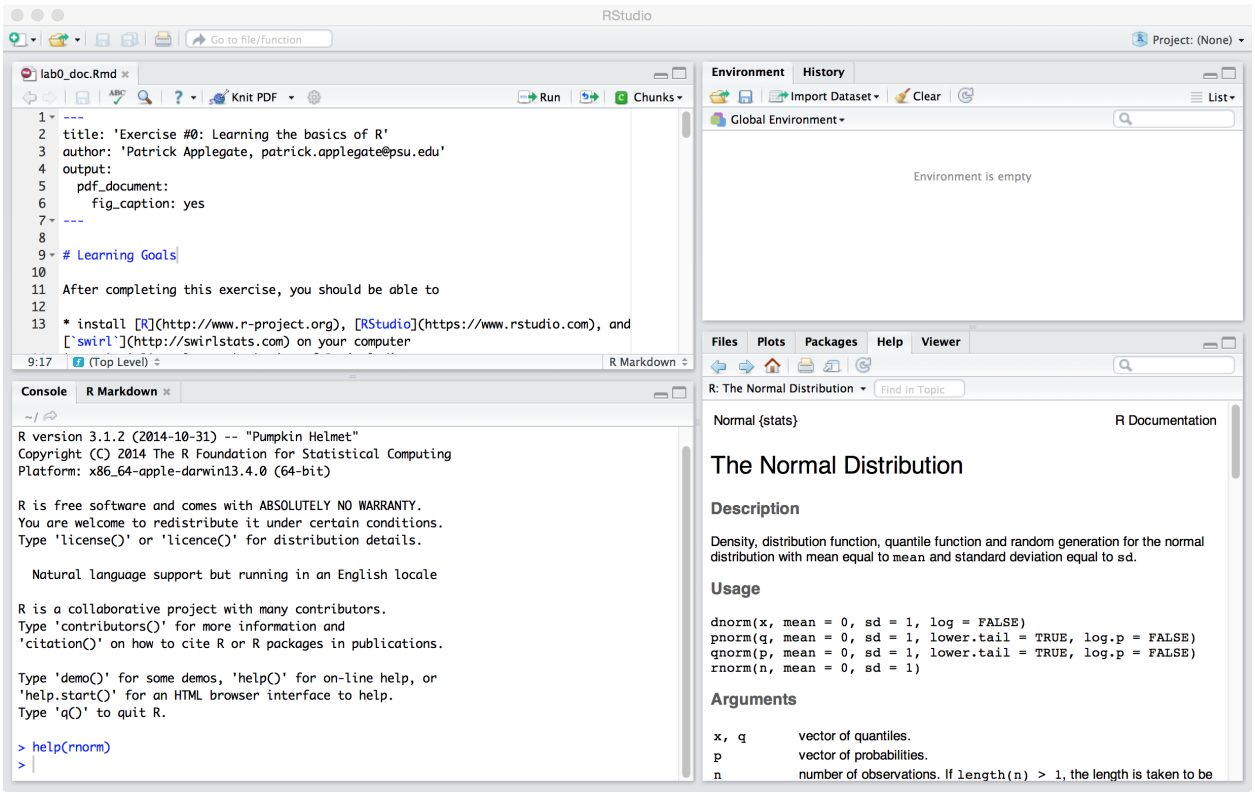

Figure 1: A representative screen shot from RStudio on a Mac. See text for discussion.

Installing `swirl`

`swirl` is a software library that will teach you to use R. The installation process for `swirl` is quite different

from the process you followed for installing R or RStudio. **To install `swirl`**,

- start RStudio, if you haven't already
- click in the Console sub-window in the lower left-hand corner of the RStudio window so that the cursor appears next to the prompt (a greater-than sign, `>`)
- carefully enter (or copy-paste) the following commands in the Console sub-window, pressing Enter after each one:

```
install.packages('swirl')
require('swirl')
install_from_swirl('R Programming Alt')
```

In the block of commands above, the first command installs the `swirl` package, the second loads the package into memory, and the third command installs materials for a basic course in using R.

When you enter the second command above, you should see a message that reads, `| Hi! Type swirl() when you are ready to begin.` Ignore this message. You'll need to enter this command each time you open RStudio if you plan to use `swirl` in that RStudio session.

When you enter the third command above, a small, red STOP sign should appear in the upper right-hand corner of the Console window. If all goes well, you should receive a message saying, `| Course installed successfully!` after a few minutes.

Learning the basics of R with `swirl`

Close and reopen RStudio. Enter each of the following commands, again pressing Enter after each one.

```
require('swirl')
swirl()
```

Follow the prompts and read the messages that appear. When you're given a choice of courses, enter the number corresponding to `R Programming Alt`, the course you installed previously. You'll be offered a list of 12 lessons.

Work through lessons 1-6 of the `R Programming Alt` course in order. Take notes on the material presented during each lesson. For example, Lesson 1 (`Basic Building Blocks`) introduces you to the commands `c`, `?`, `sqrt`, and `abs`. Note what each new command does.

Each `swirl` lesson should take you 10-20 minutes to work through, meaning that you'll need one to two hours (possibly more) to work through the first six exercises in the `R Programming Alt` course. If you stop working and come back, you'll need to enter both commands above each time you open RStudio and want to use `swirl`.

Scripting in RStudio

In the `swirl` tutorial, you worked mostly in the Console sub-window in RStudio, entering commands one at a time. However, real scientific analyses should be *reproducible*; that is, there should be a list of steps that anyone can follow to get the same results as the person who originally did the analysis. R provides the capability of saving a group of commands as a script, which can then be run later just as if the commands had been typed into the Console sub-window in RStudio.

Open the script `lab0_sample.R` in RStudio by clicking on File > Open File... and then navigating to the location of the file on your hard drive. The file should open in the Script sub-window in the upper left-hand corner of the RStudio window.

Take a few moments to look at this file. RStudio automatically colors parts of the code depending on their function.

- Black text represents commands.
- Blue text represents numerical values.
- Green text that begins with a pound sign (`#`) represent comments, notes to the programmer that aren't used by the computer when making calculations.
- Green text surrounded by quotation marks are strings (for example, `'string'` or `"string"`), groups of non-numeric characters that (unlike comments) *are* part of the calculation being performed.

Click on Session > Set Working Directory > To Source File Location and then click on the Source button in the upper right-hand corner of the Script window. When you do this, several things happen:

- A new line appears in the Console window, something like `source('some_path/lab0_sample.R')`, indicating that the script has been executed.
- Several new lines appear in the Environment window, showing the values of variables `mu`, `num`, `sigma`, and `x`.
- A new file, `lab0_fig1.pdf`, appears in the same directory as `lab0_sample.R`.

Open the new `.pdf` file. It should look like Figure 2, which shows a histogram and a red, vertical line near the apex of the histogram. The histogram represents 1000 random samples from a normal distribution with a mean of 1 and a standard deviation of 1.

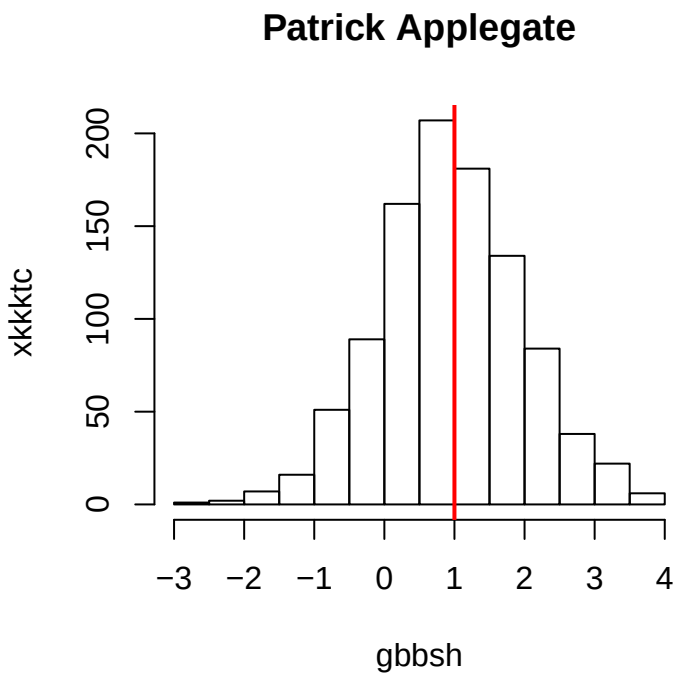

Figure 2: Contents of `lab0_fig1.pdf` after executing `lab0_sample.R`. See text for discussion.

In general, you will want to use the Source button to execute your scripts, rather than Run. To see what the Run button does, place the cursor at the top of `lab0_sample.R` and repeatedly click the Run button. The commands in the script will appear in the Console window. As you might expect, R executes the commands in a script one after another, in order from the top of the script file to the bottom.

Getting `help()`

You might be wondering what the commands in `lab0_sample.R` do. Looking at lines 14-16 in this file, we have

```
# Sample randomly from a normal distribution and assign these values to a
# variable called x.
x <- rnorm(n = num, mean = mu, sd = sigma)
```

The first two lines are comments, but the third line is a piece of code that starts with `x <-`. Based on your experiences with `swirl`, you probably realize that the results of any calculations on the right-hand side of the assignment symbol `<-` will be stored in the variable `x`. The right-hand side of this third line starts with `rnorm` and then has some parentheses surrounding some odd groups of characters separated by equals signs and commas. What is that all about?

As you know from your experiences with `swirl`, R functions are generally a string of characters followed by parentheses that contain the function's arguments. In the third line in the code block above, the function is `rnorm()`, which takes the arguments `n`, `mean`, and `sd`. These arguments are passed values stored in variables that were defined near the top of the script, `num`, `mu`, and `sigma`:

```
# Set some values.
num <- 10^3    # number of random draws
mu <- 1        # mean of normal distribution to draw from
sigma <- 1     # standard deviation of normal distribution
```

Try searching for the `rnorm` command with `help(rnorm)`. The help file that comes up is titled, "The Normal Distribution." It describes not just `rnorm()`, but also three other commands. Careful inspection of the help file suggests that `rnorm()` handles "random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`." Now we know what two of the arguments we mentioned above do, but what about the third? Looking at the list of arguments under **Arguments**, we see that `n` is the "number of observations," or the number of random values to generate.

You'll follow this procedure often. When you see a function that you don't understand, try searching for it in R's documentation using the `help()` command. Careful inspection of the relevant help files will usually give you important clues about what an unknown function does and how its arguments affect the output from the function.

However, sometimes R's documentation can be cryptic. Further down in the script, you'll see the line

```
hist(x, main = 'Patrick Applegate', xlab = 'gbbsh', ylab = 'xkkktc')
```

What do the arguments `main`, `xlab`, and `ylab` do? Here, `help(hist)` is unhelpful; the help file says only that "these arguments to `title` have useful defaults here," which conveys nothing to a beginning user of R. In this case, a Google search on "r hist" may be more useful; the fourth hit leads to an article on the "Basics of Histograms."

When the documentation and searching the Internet don't help

When you encounter problems in writing R code that you can't solve by looking at the documentation or searching the Internet, you'll want to ask someone who knows more about R than you do. When you do, you should try to ask your questions in a way that maximizes your chances of getting the help you need.

A document called "How to Ask Questions the Smart Way" suggests a strategy for asking questions. This document is long and its tone is curt, so we don't recommend that you read it, at least not right away.

Consider coming back to it after finishing the exercises in this lab manual. A shorter document is online here on the Stack Overflow Web site.

Before asking any questions, you should use the resources available to you: search the documentation and the Web. If you still need to ask someone, your question should reflect

1. what you are trying to do (the goal of your program)
2. what your program does that addresses the problem
3. what you were expecting to have happen when you ran your program
4. what actually happened (did you get an error message? if so, what did it say?)
5. what you did to try and solve your problem before asking your question

For an example of a well-written question, have a look at this post on Stack Overflow. Note that the original poster (user tjnel) addressed all of the five points above in his/her question; he/she

- says explicitly that he/she is trying to reproduce the results from a tutorial (question #1, above),
- gives the code that is being used, allowing others to fully understand the problem (#2)
- provides the expected output from running the code in the tutorial (#3),
- shows the error message that he/she obtained when running the tutorial code (#4),
- shows that he/she tried to reproduce a different part of the tutorial exercise before asking a question (#5).

Exercise

Modify `lab0_sample.R` so that it produces a .pdf file that contains a histogram of 10^4 random samples from a normal distribution with mean = 5 and standard deviation = 2. Indicate the mean of the distribution with a vertical, red line. Show the mean minus the standard deviation and the mean plus the standard deviation with two vertical, blue lines. The x-axis of the histogram should be labeled "Variable," and the y-axis should be labeled "Frequency (n = 10^4)." The title of the plot should be your name.