Lab 2: Subsetting Vectors and using the code editor

1 Introduction

The goal of this lab is to introduce you to how to extract certain elements from a data vector in R. You will complete one Swirl lesson and some additional exercises specified in this lab. You can stop and restart the `swirl()` lesson at any time, just type `bye()` to save your progress before exiting. Be sure to use the same name each time so you pick up where you left off. At the end of the lesson you will have the option to send me an email letting me know you have completed the lesson. **Do this so I can give you credit!**

1.1 Subsetting vectors using positional indicators and the bracket `[]` notation.

Complete the **Subsetting Vectors** Swirl lesson.

1.1.1 Practice

1. Create a vector of 100 random normal values by typing `rnorm(100)` and assigning it to a vector.
2. Print out the first 10 values.
3. Print out the values that are greater than 3. Assign this to a new vector object named `y`.

1.2 Subsetting vectors using `subset()`.

As you start to learn `R` you will quickly realize that there are multiple ways to write code that achieves the desired result. In general it comes down to experience, preference, and style. In this lab we will cover another way to subset the data using the function `subset()`.

1. Read the help file for subset by typing `?subset`. Don't worry if it seems cryptic.

1.2.1 Practice

1. Repeat number 3 from the prior practice but use the `subset()` function. Assign this to a new vector object named `z`.
2. Prove that these two methods produce the exact same subset by typing `identical(y,z)`.

Additional Resources

- MartinStatsLectures: R Tutorial 1.2 - Getting started with R, Part II
- MartinStatsLectures: R Tutorial 1.6 - Subsetting Data in R with Square Brackets and Logic Statements

1.3 Using Script Files.

During the Swirl lessons you have been typing commands directly into the interactive `console` window. This is not commonly done and not the way any analysis should be performed. Once you submit that command it's gone, and multi-line commands are difficult to type correctly "on the fly" like that. Instead code is written in a separate file, a `code` or `script` file. With a few exceptions (such as the swirl lessons) from here on out you will be writing all your commands in a code file. Let's get started.

1. In RStudio, go to `File` –> `New File` –> `R Script`. This will open a new `code editor` window in your R Studio, creating a new box in the top left corner. This is the typical format that R Studio is used (with all 4 windows open).
2. Click the save icon, and save this file in your `RBootcamp` folder as `Lab2.R`.

This new window is where you will type all of your commands. This gives you a chance to fix typos and make small adjustments before you submit your command. Let's test out both of the ways we can submit commands from the editor window, into the console window.

Type `2+2` in your script file.

1. With your cursor still on the line of code you want to submit, click `Run` in the top left corner of the window. Notice it passes the command on that line to the console window, and executes that command.
2. With your cursor still on the line of code, press `Ctrl+Enter`. This is simply a keyboard shortcut for the `Run` button.

If you want to submit multiple lines of code at the same time, you need to highlight all rows to submit before hitting `Run` or using Ctrl + Enter.

1.3.1 Practice

Type the following function in your editor window. Highlight the entire thing and submit it. You won't see any output if you did it correctly.

```r
add_for_me <- function(a,b){
  a + b
}
```

Test it out by calling the function. Make sure you see the same output as is displayed below. R output is denoted by the ## in front of each line.

```r
add_for_me(5,6)
```

```
## [1] 11
```

What does this function do? We will not discuss user-defined functions in this class but as you can see they can be pretty easy and useful!