Homework 0: Introduction to R and RStudio

STUDENT NAME

DUE DATE

Introduction

R is a statistical computing language and RStudio is a Graphical User Interface for using R. This assignment will introduce you to the basics of using R, which will be important for later in the semester when we use R to do econometric analyses.

Using RMarkdown

This file is an RMarkdown file (file extension .rmd), as opposed to an R file (file extension .R). An R file would contain only code and, when run in its entirety, would execute that code. An RMarkdown file is a mix of code and "markdown," i.e. text. When an RMarkdown file is run in its entirety, the file executes the code "chunks" within the file and then collates them with the text to produce a pdf document. RMarkdown is ideal for social scientists because it allows you to manipulate data, analyze it, report results, generate figures, and write your report all within the same file. We will be using RMarkdown throughout the course to do assignments.

Coding in RMarkdown takes place within code "chunks." These chunks have options which affect how they appear in the final pdf. The below code chunk prints the text "Hello World!" A chunk is defined by coding between two sets of three left apostrophes (the button right under the Escape key on most keyboards). Right next to the top set of these apostrophes, the name and options for the chunk are defined. For the below chunk this is `{r helloworld, include = TRUE}`– the "r" says that we are coding using the language R (there are a few other options which we will not be using), the "helloworld" is the name of the chunk (this part can be skipped if desired), and the "include = TRUE" is a chunk option. The "include" option being TRUE tells the RMarkdown file to include both the code chunk itself and its output in the pdf. Setting "include = FALSE" would prevent either from being included; you may want to set "include = FALSE" when writing reports for chunks that only clean the data. Include is set to TRUE by default, it is only included as an option here for demonstration purposes. To include the chunk output but not the code itself (for example, when you have code that generates a figure), you can set "echo = FALSE" instead.

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

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

RMarkdown has a few other nice qualities. The text portion of an RMarkdown file is capable of reading LaTeX commands. For example, you could write `$\frac{1}{n}\sum_i x_i=\bar{x}$` in order to include the expression $\frac{1}{n}\sum_i x_i=\bar{x}$ in your pdf.

When you are finished and want to create your pdf, you click "Knit" in the RStudio interface to generate the pdf. I recommend doing this periodically throughout working on the assignment, as errors can prevent the pdf from knitting at all and RMarkdown knitting errors are notoriously unhelpful in diagnosing the problem.

R Basics

R is an object-oriented programming language, which means that it stores information within objects that can then be manipulated by later code. Each object has a class which determines what operations can be done on it. The class of an object `a` can be determined by running the code `class(a)`. This is something you probably want to do in the console (the box below where this text is written in the RStudio interface).

Let's create our first object: a numeric vector with the first few prime numbers.

```
# This line creates an object names "first5primes" which is a numeric vector
# Note that object assignment in R is done using "<-"; newer versions of R also
# allow the use of "=" like most other coding languages but this is still not
# the norm in most R code
# Also note that you can write comments in R code using a preceding "#" (hashtag)
# The c() function generates a vector, which is by default a column vector
first5primes <- c(2, 3, 5, 7, 11)
```

When you run this code chunk (which you can do by clicking the green play button that appears at the top right of the chunk), R will create the `first5primes` object. In the box at the top right of the RStudio interface (labeled "Environment") you will see `first5primes` listed. Try typing `class(first5primes)` into the console, you should see the result "numeric." Next, try copy and pasting each of the following lines into the console to see how they work.

```
# Subsetting is done with square brackets, note that unlike many other coding
# languages there is no "0th" position in R, the first element is in position 1
first5primes[1]
```

```
## [1] 2
```

```
## BASIC MATH
# Addition
5 + 5
```

```
## [1] 10
```

```
# Subraction
5 - 4
```

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

```
# Multiplication
5 * 5
```

```
## [1] 25
```

```
# Division
5/5
```

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

```
# Exponentiation
5^5
```

```
## [1] 3125
```

```
# Logarithms
# By default these are base e, not 10! This is more useful for econometrics anyway
log(5)
```

```
## [1] 1.609438
```

```
## Logic
# True
5 > 3
```

```
## [1] TRUE
```

```r
# False
5 >= 4
```

```
## [1] TRUE
```

```r
# Equality
5 == 5 # Note the DOUBLE equal sign!
```

```
## [1] TRUE
```

```r
# "and"
5 > 3 & 3 >= 2
```

```
## [1] TRUE
```

```r
# "or"
5 > 3 | 3 > 4
```

```
## [1] TRUE
```

```r
# Negation
!(5 > 6)
```

```
## [1] TRUE
```

```r
# "in"
6 %in% c(5, 6, 7)
```

```
## [1] TRUE
```

Before continuing, I want to point out a potentially confusing aspect of using computers do to math. What do you expect will result if you type `0.1+0.2==0.3` to the console? Try it.

It should return false, despite the fact that the statement is actually true. This is due to the way that computers store numbers. Without getting into the details, this is because computers store numbers in floating point binary, which means there is limited precision (i.e. it cannot literally store the infinite 0.333333. . . . that represents the number 1/3). Floating point precision is precise down to quite a few decimal places, so doesn't matter at all for most applications. However, it can cause serious problems when evaluating logical statements like `0.1+0.2==0.3`, because very small errors in numerical representation make the numbers no longer "equal" each other. In some cases, this can be solved by working only with integers in the integer class rather than the numeric class.

With that aside, let's move on to the different types of objects you can create and manipulate in R.

```r
# Vectors (numeric)
# The following are all (almost) equivalent (check their classes!)
c(1, 2, 3, 4, 5)
```

```
## [1] 1 2 3 4 5
```

```r
1:5
```

```
## [1] 1 2 3 4 5
```

```r
seq(from = 1, to = 5, by = 1)
```

```
## [1] 1 2 3 4 5
```

```r
# Vectors (character)
c("a", "b", "c")
```

```
## [1] "a" "b" "c"
```

```r
# What happens here?
c("a", 2, "c")
```

```
## [1] "a" "2" "c"
```

```r
# Vectors cannot have different kinds of elements, so the "2" gets coerced
# into a character
# Instead...

# Lists
list("a", 2, "c")
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] "c"
```

```r
# Matrices
testmatrix <- matrix(data = c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3)
testmatrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```r
# You can subset a matrix like so:
testmatrix[2, 3] # element in 2nd row, 3rd column
```

```
## [1] 8
```

```r
# Dataframes
testdf <- data.frame(a = c(1, 2, 3), b = c(1, 4, 9))
testdf
```

```
##   a b
## 1 1 1
## 2 2 4
## 3 3 9
```

```r
# The following are all ways of seeing the column "a" in this dataframe
testdf$a
```

```
## [1] 1 2 3
```

```r
testdf[,1]
```

```
## [1] 1 2 3
```

```r
testdf[,"a"]
```

```
## [1] 1 2 3
```

```r
# Subsetting can also use logic; this will show the rows of the dataframe where
# the value for "a" is greater than 1
testdf[testdf$a > 1,]
```

```
##   a b
## 2 2 4
## 3 3 9
```

```
# We can add another column later
testdf$c <- c(1, 8, 24)

# Functions
# Functions take inputs and return outputs and are extremely powerful tools
isFirstLetterVowel <- function(word){
  if (is.character(word) == FALSE){
    stop("Function input must be class character")
  }
  firstletter <- substr(word, 1, 1)
  return(firstletter %in% c("a", "e", "i", "o", "u"))
}
isFirstLetterVowel("alphabet")
```

```
## [1] TRUE
```

In the last part of the previous chunk, I wrote a function called `isFirstLetterVowel` which takes characters as an input and returns TRUE or FALSE depending on whether the first letter is a vowel. Note that the first part of the function is a test to make sure the input is actually a character (try running `isFirstLetterVowel(5)` to see how the code handles this). Including tests like this is a good way to debug functions. I then create a new object `firstletter` using the function `substr`. If you want to know what to know what `substr` does, you should type `?substr` into the console. Note that `firstletter` does not show up in the Environment section on the right of RStudio—this is because this object is created only in the function environment. This means we won't be able to use the object `firstletter` anywhere other than within the function `isFirstLetterVowel`. (There are ways to get `firstletter` to show up in the Global Environment, but usually this is a nice feature because it leave the Global Environment clean.)

Finally, it is worth knowing about for and while loops. These can be used to do repetitive tasks automatically. Say that we don't just want to know whether the first letter of a word is a vowel, we want to know if *every* letter is a vowel. We could do that with this code.

```
letterVowels <- function(word){
  if (is.character(word) == FALSE){
    stop("Function input must be class character")
  }
  result <- c() # start with an empty vector
  for (i in 1:nchar(word)){
    letter <- substr(word, i, i)
    result[i] <- letter %in% c("a", "e", "i", "o", "u")
  }
  return(result)
}
letterVowels("alphabet")
```

```
## [1]  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
```

A for loop like the one shown above iterates over a vector or list. In the code above, it iterates over the list of integers starting at 1 and ending at the length of the word, `nchar(word)`. When $i = 1$, it looks at the first letter of the word, tests whether it is a vowel, and adds the result to the ith place in the result vector.

A while loop instead looks like `while(claim){code}`. So long as the claim is true, the code will keep running. For example, setting `a <- 5` and then running `while(a<10){a <- a+1}` would eventually stop when `a==10`. Be careful using while loops—if their claim never evaluates to TRUE, they will never stop running.

Even the best coders in the world generally cannot code blindfolded—coding involves constantly looking up functions and googling to problem solve. Some of the problems below will require functions that I have not told you about yet, but everything is possible without downloading any libraries (something we will discuss later). Use Google and problem solving to figure out how to do them!

Get Familiar With R

Problem 1

Generate a sample of 100 random draws from a standard normal distribution (Hint: use the function `rnorm`). Manually calculate and print the mean of this sample (i.e. without using the `mean` function).

Problem 2

Create a $10x10$ matrix using the the same random sample of 100 numbers from the previous problem. Then replace the 3rd column with zeroes and round the entire matrix to a single decimal place. Print the matrix.

Problem 3

Write a function that does the following: Takes a word (we'll use the word "cake" as an example), considers each letter ("c", "a", "k", "e"), converts that letter into a number corresponding to that letter's order in the alphabet (3, 1, 11, 5) and then adds them up (3+1+11+5=20) and returns that sum. Test the function you write on the word "alphabet"and print the result.//

(To make this easier, type `LETTERS` into the console to see a convenient, already-existing object. You may also want to check out the function `tolower`!)