R Tutorial

Andie Creel

January, 2023

Special thanks to Sam Maher who wrote an R Tutorial in 2018 that heavily influenced this.

Goal

This is R Studio. The goal of this tutorial is to see some basics about R so that we can dive into more exciting things in the future.

Overview of R Studio

Layout of RStudio: Script, Environment/History, Files/Plots/Packages/Help/Viewer, Console

- Script is where you will be writing your own programs.
- Environment/. . . /Git (mostly just environment) shows you which data and objects you have in memory. Also where we do version control.
- Files/Plots/Packages/Help/Viewer: helps you locate packages and other files to load. It's also the default window if you're trying to get help on something
- Console is where the commands will actually run.
  - The script will execute in the console, but the script saves your commands, whereas they disappear in the console. If you want to run something in the console, just type it in and hit "enter".

```r
# this is a comment, you can use '#' to write notes to yourself in your code

# Type the following in the console, then press enter
a <- 2
b <- 3

a + b
```

```
## [1] 5
```

How to write and execute a script

I wrote this pdf using an R Markdown. I love these because I'm able to write lots of notes to myself while I'm coding, and then it produces a nice shareable file where I can share my notes, code and results with others. I will ask you to use an Rmarkdown for pset so I can grade them easily.

- New Script: File > New File > R Script (or R Markdown, which is what we're using now), or just the New Document script in the upper left hand corner of the screen > R Script (or R Markdown)

- Keyboard shortcut to run a section of code: highlight or put your cursor on that line and hit Ctrl + Enter (Windows) or Command + Enter (Mac)
- Executing code: the run button at the top right corner of the script

For R Markdowns, we also have the Knit button at the top of the script. That will run all code and “knit” it together into a pdf or HTML or doc (which is specified at the top of the R Markdown file)

You can “clean up” the Environment after you’ve executed code by clicking the broom icon. **This will delete everything in your environment**.

Basic Data Types

```
# Numeric -- integer: no decimal points
myInt <- 1

# Numeric -- double: decimal points
myNum <- 2.4

# logical (Boolean): a true/false statement
myBool_1 <- (3 < 4)
myBool_2 <- (3 > 4)

# string
myStr <- "a"
```

Ways to store datatypes

```
# vector: can only be a vector of one data type (numeric, logical, string)
myVec_n <- c(1, 2, 3, 4, 5)
myVec_s <- c(str, "b", "c")

# matrix: can only be a matrix of one data type
myMat_n <- matrix(c(myVec_n,
                    6, 7, 8, 9, 10),
                  nrow = 2,
                  ncol = 5)

# Lists: Very powerful, but somewhat confusing. For now, just know they exist
myList <- list(2, "c", myMat_n)
myList[[1]] # returns numeric
myList[[2]] # returns string
myList [[3]] # returns matrx

# data frame: can have multiple data types
myDF <- as.data.frame(myMat_n)
colnames(myDF) # these don't mean anything to me
colnames(myDF) <- c("age_yr", "weight_lb", "income_$", "hieght_ft", "height_in")
```

Tables are like matrices, but we can have different data types in each column. We can also reference specific columns using the “$” operator, followed by the name of the column we want to look at.

```r
# investigate one column
myDF$age_yr

#create a new column
myDF$nonsense <- myDF$age_yr + myDF$weight_lb
```

For the most part, you'll be loading new data in as tables, but you might have to create one at some point. By looking at how they're created, we can get a better sense of what goes into them.

```r
# Create the data frame
BMI <-  data.frame(
   gender = c("Male", "Male","Female"),
   male = c(T, T, F),
   height = c(152, 171.5, 165),
   weight = c(81,93, 78),
   Age = c(42,38,26)
)

# Try referencing one column
BMI$male # version 1
BMI[,2] #version 2

# Try referencing one row
BMI[1,]

# Try referencing one cell
BMI$height[1] # version 1
BMI[1,3] # version 2
```

Functions, loops, and if statement

Functions: once you have initialized them, they take in a set of data, perform a set of operations on them, and then give you some return value

```r
myF <- function(x){
  y <- x - x^2
  return(y)
}

myF(.5)
myF(.25)
myF(.7)
```

for loops: iterates through a task for a set number of times

```r
#simple
for (i in 1:4){
  print(i)
}

# combining loop and function
for (i in 1:4){
  y = myF(i/4)

  print(y)
}


#manipulating a data frame (follows version one of referencing one cell above)
for (i in 1:length(BMI$Age)) {
  BMI$Age[i] <- i
}
```

If statements: sometimes you want to execute a task ONLY if a certain condition is met.

```
# goes through each row and changes age if someone is male
for (i in 1:length(BMI$male)) {

   if (BMI$male[i] == TRUE) {
    BMI$Age_new[i] <- BMI$Age[i] - 3
  }else{
    BMI$Age_new[i] <- BMI$Age[i]
  }

}
```

Other R Tutorials

Princeton Getting Started with R

UCLA Getting Started with R

Some specific packages

We haven't covered packages yet, but a few good resources for the future (beyong this class)

ggplot

dplyr and tidyr

R and GIS

R Shiny (interactive plots)