Day 1, Lecture 2: Base R

Andie Creel

January, 2024

1 Goal

This is R Studio. The goal of this lecture is to see some basics about R so that we can dive into more exciting things tomorrow.

2 Overview of R Studio

Layout of RStudio:

- Script
  - Where you will be writing your own programs
- Environment/../Git
  - Mostly just environment and Git
  - Show which data objects you have loaded in memory
  - Eventually where you'll do version control with GitHub (last day)
- Files/Plots/Packages/Help/Viewer
  - Helps you load packages and other files to load
  - Default window when you're trying to get with a function from a package (we will get to this)
- Console/Terminal
  - where the code actually run
  - Scripts executes in console
  - Code disappear in the console, whereas a script saves your code
  - To run something in the console, type it in and hit "enter"
  - Important to know there is a "terminal" in R studio. Again, just know it's there.

3 Console

- We can type code straight into the console and run it.
- The console in R Studio knows your running R code.

```
# Type the following in the console, then press enter
2+3
```

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

```
# variable assignment happens with an arrow (on a mac can do option -)
a <- 2
b <- 3

a + b
```

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

4 How to write and execute a script

Writing Scripts and Running Code

- New Script: File > New File > R Script (or R Markdown, which is what I used to make the lecture notes), 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 HTML or pdf or doc
  - File type is specified at the top of the R Markdown file
  - You can also run code in a R Markdown by highlighting it and hitting Ctrl + Enter (Windows) or Command + Enter (Mac)

R Markdown

- I wrote this pdf using an R Markdown (**show them quickly**).
- I like because I'm able to write lots of notes to myself while I'm coding
  - it produces a nice shareable file
  - can share my notes, code and results with others.

Problem Sets

- If you have no experience with R, start with a script.
- If you want to do a R Markdown and are used to them, that's fine.

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

5 Comments

```
# This is a comment, you can use '#' to write notes to yourself in your code
# - Comments are what make or break good coders
# - Good comments also create coders who can collaborate with others.
# - If you ever think you're writing "too" many comments, you are not
# - The things you think are obvious in your code won't be to others
# - (nor yourself in a year when you get back to a project)

# Comments are written in the same spot as code
# But they are ignored by the computer
# Let's run some code
print("Hello, World!")
```

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

6 Basic Data Types

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

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

```r
# logical (Boolean/Indicator variable): a true/false statement. Use () to evaluate if something is true
myBool_1 <- (3 < 4)
myBool_2 <- (3 > 4)

# character (string)
myChar_a <- "a"
myChar_b <- 'b'
```

7 Ways to store datatypes

```r
# vector: can only be a vector of one data type (numeric, logical, string)
myVec_n <- c(1, 2, 3, 4, 5)
myVec_s <- c(myChar_a, "b", "c")
myVec_string <- c(1, "b", "c")
myVec_string # notice the 1 has been make char bc of the "
```

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

```r
# matrix: should 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
```

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

```r
myList[[2]] # returns string
```

```
## [1] "c"
```

```r
myList [[3]] # returns matrix
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

7.1 Data Frames

- Like matrices
- Can have different data types in each column
- Reference specific columns using the "$" operator, followed by the name of the column
- For the most part, you'll be loading new data by reading a csv
- 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
# data frame: can have multiple data types
myDF <- as.data.frame(myMat_n)
colnames(myDF) # these don't mean anything to me
```

```
## [1] "V1" "V2" "V3" "V4" "V5"
```

```r
colnames(myDF) <- c("age_yr", "weight_lb", "income_$", "height_ft", "height_in")
```

```r
# investigate one column
myDF$age_yr
```

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

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

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

# Try referencing one column
myPpl$male # version 1
```

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

```r
myPpl[,2] #version 2
```

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

```r
# Try referencing one row
myPpl[1]
```

```
##       gender
## 1       Male
## 2 non-binary
## 3     Female
```

```r
# Try referencing one cell
myPpl$height[1] # version 1
```

```
## [1] 152
```

```r
myPpl[1,3] # version 2
```

```
## [1] 152
```

8 A word of caution

- Make sure you don't over write your variables by accident.

```r
# assigning new value to same variable (something to do carefully)
a <- 5
a <- a + 1 # If you run this line more than once, you will NOT get six
a
```

```
## [1] 6
```

```r
# assigning new value to new variable
a <- 5
```

```
a_new <- a + 1 # If you run this line more than one, you WILL get six
a_new
```

```
## [1] 6
```

9 Functions

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

Example on board

- consider the function: myF(x) { y <- x + 3; return(y)}
- what does myF(3) return? 6

Points:

- These are helpful when you have something that you do often
- Rule of thumb: if you're copying and pasting code 3 times or more, make function
- (I say if you are going to copy past ever, because even if you think it'll only be twice it'll probably be more)
- Recent example for me:
  - wrote a function to take a date and return the season
  - Wrote a function to get kelvin and return Fahrenheit

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

myF(.5)
```

```
## [1] 0.25
```

```
myF(.25)
```

```
## [1] 0.1875
```

```
myF(.7)
```

```
## [1] 0.21
```

10 Loops

- for loops: iterates through a task for a set number of times
- Consider these loops (psuedo code):
  - For (i in 1 through 4) { print i }
  - For (i in 1 through 4) { print i / 4}
- Can be helpful when
  - Iterating through a column of data and do something to each row
  - Construct a new column and want to construct each row by scratch

```
# Complicated code that is simplified by the loop
print(1)
```

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

```r
print(2)
```

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

```r
print(3)
```

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

```r
print(4)
```

```
## [1] 4
```

```r
# The following loop does the exact same thing
for (i in 1:4){
  print(i)
}
```

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

```r
# more involved
for (i in 1:4){
  print(i/4)
}
```

```
## [1] 0.25
## [1] 0.5
## [1] 0.75
## [1] 1
```

```r
# combining loop and function
for (i in 1:4){

  y = myF(i/4)

  print(y)
}
```

```
## [1] 0.1875
## [1] 0.25
## [1] 0.1875
## [1] 0
```

```r
# Example

# Making a new column
# Version one: Line by line
myPpl$age_new_a[1] <- myPpl$age[1] + 1
myPpl$age_new_a[2] <- myPpl$age[2] + 1
myPpl$age_new_a[3] <- myPpl$age[3] + 1

# Version two: loop
for (i in 1:length(myPpl$age)) {
  myPpl$age_new_b[i] <- myPpl$age[i] + 1 # everyone aged one year
}
```

11 If statements

- sometimes you want to execute a task ONLY if a certain condition is met.
- Open the myPpl df:
  - Our RA did not record men's ages right
  - All men are actually 3 years older than what's recorded
  - What would the correct DF look like?
- If statements let you fix a mistake like this
- Also demonstrates why the Boolean (true/false or indicator) variable is so powerful

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

   if (myPpl$male[i] == TRUE) {
    myPpl$age_new_m[i] <- myPpl$age[i] - 3
  }else{
    myPpl$age_new_m[i] <- myPpl$age[i]
  }
}
```

12 Other R Tutorials

UCLA Getting Started with R

git lab intro

13 Some specific packages

We haven't covered packages yet, but a few good resources for the future

ggplot

dplyr and tidyr