IMT 573: Lab 1 - Introduction to R and RStudio Server

Alex Davis

October 06 2020

Objectives

In this demo we will get a first look at writing R code for data science. We will review basic R syntax and take a look at the different R data structures we will use throughout the course. We will learn how to find and access existing datasets, and even make our first visualization of data!

This demonstration also give you a glimpse of writing reproducible data science reports using Rmarkdown. We will talk more about this next week!

A little background on R

Everything in is an object - data, functions, everything! When you type in commands what happens:

- **R** tries to interpret what you've asked it to do (evaluation)
- If it understands what you've told it to do, no problem
- If it does not understand, it will likely give you an error or warning message

Some commands trigger **R** to print something to the screen, others don't.

If you type in an incomplete command, will usually respond by changing the command prompt to the + character to demonstrate it is waiting for something more. A > indicated the beginning of a line. You shouldn't have to consider this too much because you should always write your code in a *script* rather than the Console.

Getting started

Welcome to RStudio! RStudio is an integrated development environment (IDE) for R. It comes with a lot of nifty functionality to make it easier for us to do data science! Take a tour of RStudio using the online learning center or just play around with it after class today.

First, let's consider setting up our environment. In this report, we will be able to write text, as we have done already, but we will also be able to write code! Code should be contained in a *code chunk*. Code chunks are marked as follows:

```
# Hello R!
# This is a code comment.
# Code comments help you document your coding proces!

#This is a code chunk named "our first code chunk", which is the name of the code chu
nk;
# Including code chunk name is optional, but this practice will help you create well
documented code
```

Introduction to basic R syntax

Let's take a look at some basic R syntax. Remember everything in R is an object! We also want to follow the `tidyverse` style guide for writing code. Variable and function names should use only lowercase letters, numbers, and _.

```
1 + 3       # evaluation
```

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

```
a <- 3  # assignment. <- is the assignment symbol
a               # evaluation
```

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

```
a<-3        # spacing does not matter
a    <-    3        # spacing does not matter

sqrt(a)         # use the square root function
```

```
## [1] 1.732051
```

```
b <- sqrt(a)        # use function and save result
b
```

```
## [1] 1.732051
```

```
# x             # evaluate something that is not there

a == b          # is a equal to b?
```

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

```
a != b          # is a not equal to b?
```

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

```
# list objects in the R environment
# (same as viewing the "Workspace" pane in RStudio)
ls()
```

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

```
rm(a)           # remove a single object
rm(list=ls())       # remove everything from the environment
```

Getting help in R

```
# get help with R generally
# (same as viewing the "Help" pane in RStudio)
help.start()
```

```
## starting httpd help server ... done
```

```
## If the browser launched by '/usr/bin/open' is already running, it is
##     *not* restarted, and you must switch to its window.
## Otherwise, be patient ...
```

```
# More targeted help
?sqrt           # get specific help for a function

apropos("sq")       # regular expression match. What do you do when you can't really
recall the exact function name
```

```
## [1] "chisq.test" "dchisq"     "pchisq"     "qchisq"     "rchisq"
## [6] "sqrt"       "sQuote"
```

Data Types in R

There are numerous data types in R that store various kinds of data. The four main types of data most likely to be used are numeric, character (string), Date (time-based) and logical (TRUE/FALSE).

```
# Check the type of data contained in a variable with the class function.
x <- 3
x
```

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

```
class(x)
```

```
## [1] "numeric"
```

```
# Numeric data type -- Testing whether a variable is numeric
a <- 45
a
```

```
## [1] 45
```

```
class(a)
```

```
## [1] "numeric"
```

```
# Character data
b <- "b"
b
```

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

```
class(b)
```

```
## [1] "character"
```

```
# Date type -- Dealing with dates and times can be difficult in any language, and to
further complicate matters R has numerous different types of dates.
c <- Sys.Date()
c
```

```
## [1] "2020-10-06"
```

```
class(c)
```

```
## [1] "Date"
```

```
# Logical data type: True/False
d <- FALSE
d
```

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

```
class(d)
```

```
## [1] "logical"
```

```
# Factor vectors: Ideal for representing categorical variables (More on that later)
vect <- c(1:10)
class(vect)
```

```
## [1] "integer"
```

Vectors and matrices in R

Vectors in R: A collection of elements all of the same data type

```
# Creating vectors using c() function or the "combine" operator
a <- c(1,3,5,7)
a
```

```
## [1] 1 3 5 7
```

```
# select the second element
a[2]
```

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

```
# also works with strings. let's see how.
b <- c("red","green","blue","purple")
b
```

```
## [1] "red"    "green"  "blue"   "purple"
```

```
# all colors except blue
b <- c("red","green","blue","purple")
b[-3]
```

```
## [1] "red"    "green"  "purple"
```

```
#all numbers less than 5
a <- c(1,3,5,7)
a[-3]
```

```
## [1] 1 3 7
```

```
#add a new element
b[5] <- "yellow"


#change the first element
b[1] <- "gold"


# combine by applying recursively
a <- c(a,a)


# mixing types---what happens?
c <- c(a,b)
c
```

```
##  [1] "1"      "3"      "5"      "7"      "1"      "3"      "5"      "7"
##  [9] "gold"   "green"  "blue"   "purple" "yellow"
```

```
# Sequences and replication

#creating a vector using sequence. sequence from 1 to 5
a <- seq(from=1,to=5,by=1)
b <- 1:5                    # a shortcut!

#creating a vector using sequence. sequence from 1 to 10, steps of 2
a <- seq(from=1,to=10,by=2)

# replicate elements of a vector
rep(1,times=3)
```

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

```
# Any, all, and which (with vectors)
rep(1:10,times=3)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5
## [26]  6  7  8  9 10
```

```
# How long is the vector?
length(a)
```

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

Element-wise operations on vectors

```
# Most arithmetic operators are applied element-wise:
a <- 1:5        # create a vector
a + 1       # addition
```

```
## [1] 2 3 4 5 6
```

vectorized functions. Transforming vectors by applying functions

```
#vector of numbers
nums <- c(3.98, 8.2, 10.5, 3.6, 5.5)
round(nums, 1) # round to nearest whole number or number of decimal places, if specif
ied
```

```
## [1]  4.0  8.2 10.5  3.6  5.5
```

From vectors (1D collection of data) to matrices (2D collection of data)

```
# create a matrix the "formal" way...
a <- matrix(1:25, nrow=5, ncol = 5)
a
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

Element-wise operations on matrices (same principles as vectors)

Factor Variable

Factors consist of a finite set of categories (primarily used for categorical variables). Factors also optimize for space. Instead of storing each of the character strings, example ‘small’, ‘medium’, it will store a number and R will remember the relationship between the label and the string. Example: 1 for ‘small’, 2 for ‘medium’, etc. Let’s see with an example

```
# A character vector of shirt sizes
shirt_sizes <- c('small', 'medium', 'small', 'large', 'medium', 'medium')


# YOUR TURN:
# Create a factor variable education that has four categories: "High School", "Colleg
e", "Masters", "Doctorate"
education <- c("High School", "College", "Masters", "Doctorate")
degree <- factor(education, ordered = TRUE)
degree
```

```
## [1] High School College     Masters     Doctorate
## Levels: College < Doctorate < High School < Masters
```

R functions

```
# call the sqrt() function, passing it an argument of 25
sqrt(25)
```

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

```
# call the print function, passing it "IMT 573" as an argument
print("IMT 573")
```

```
## [1] "IMT 573"
```

```
#printing using cat function
cat("value = ", a)
```

```
## value =  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
#min and max function taking multiple arguments
min(a)
```

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

```
max(a)
```

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

```
#function to return upper case
toupper("s")
```

```
## [1] "S"
```

```
#Write a function of your name. Let's see how it works through an example
#write a function to combine first and last name

make_fullname <- function(firstname, lastname) {
  # function body
  fullname <- paste(firstname, lastname)

  #return the value
  return(fullname)
}

#call the function
some_name = make_fullname('John', 'Doe')

### R functions: YOUR TURN - TODO
### Write a function to calculate area of a rectangle
rectangle_area <- function(x, y) {
area <- x * y
return(area)
}

some_area = rectangle_area(2, 2)
some_area
```

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

Data frames - act like tables where data is organized into rows and columns

```
# creating a dataframe by passing vectors to the `data.frame()` function

# a vector of names
name <- c("Alice", "Bob", "Chris", "Diya", "Emma")
# A vector of heights
heights <- c(5.5, 6, 5.3, 5.8, 5.9)
weights <- c(100, 170, 150, 120, 155)

#Combine the vectors into a data frame
# Note the names of the variables become the names of the columns in the dataframe
people <- data.frame(name, heights, weights)

# to create row.names
people2 <- data.frame(name, heights, weights, row.names = 1)

# YOUR TURN
# build an employee data frame of 5 employees with 3 columns: income, manager (T/F), empid
name2 <- c("Alex", "Anna", "Jean", "Zack", "Daniel")
income <- c(2200, 2300, 5400, 6800, 10000)
manager <- c(TRUE, FALSE, FALSE, TRUE, FALSE)
empid <- c(1:5)

employees <- data.frame(name2, income, manager, empid)
employees
```

name2 <chr>	income <dbl>	manager <lgl>	empid <int>
Alex	2200	TRUE	1
Anna	2300	FALSE	2
Jean	5400	FALSE	3
Zack	6800	TRUE	4
Daniel	10000	FALSE	5

5 rows

```
# elements by row and column name
people2['Alice', 'heights']
```

```
## [1] 5.5
```

Read and Write data

Working Directory When working with .csv files, the read.csv() function takes as an argument a path to a file. You need to make sure you have the correct path. To check your current working directory using the R function

```
#read titanic.csv data. Download data from Canvas
data <- read.csv('titanic.csv')

#check the type of data
typeof(data)
```

```
## [1] "list"
```

```
#check additional structure and type in the data


# inspect the data - look at top and bottom
head(data)
```

	pclass <int>	survived <int>	name <chr>	sex <chr>	age <dbl>
1	1	1	Allen, Miss. Elisabeth Walton	female	29.0000
2	1	1	Allison, Master. Hudson Trevor	male	0.9167
3	1	0	Allison, Miss. Helen Loraine	female	2.0000
4	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0000
5	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000
6	1	1	Anderson, Mr. Harry	male	48.0000

6 rows | 1-8 of 15 columns

```
tail(data)
```

	pclass <int>	survived <int>	name <chr>	sex <chr>	age <dbl>	sib... <int>	pa... <int>	ticket <chr>	fare <dbl>
1304	3	0	Yousseff, Mr. Gerious	male	*NA*	0	0	2627	14.4583
1305	3	0	Zabour, Miss. Hileni	female	14.5	1	0	2665	14.4542

1306	3	0	Zabour, Miss. Thamine	female	*NA*	1	0	2665	14.4542
1307	3	0	Zakarian, Mr. Mapriededer	male	26.5	0	0	2656	7.2250
1308	3	0	Zakarian, Mr. Ortin	male	27.0	0	0	2670	7.2250
1309	3	0	Zimmerman, Mr. Leo	male	29.0	0	0	315082	7.8750

6 rows | 1-10 of 15 columns

Finding built-in data sets

Elementary visualization