

Introduction to R

MSMI Bootcamp

Lucas De Oliveira

Data Science:

Data Scientist at Capgemini

Formerly D.S. at Nextracker

Education:

M.S. in Data Science from USF

B.A. in Economics from UVA



Bootcamp Files

All files can be found in the following repository:

https://github.com/lbdeoliveira/MSMI_Bootcamp

Getting started

Install R and RStudio

<https://www.rstudio.com/products/rstudio/download/>

- Select **RStudio Desktop Free**
- Click on link to download required version of R
- Download RStudio

The RStudio Console

Let's take a look!

A Little Pep Talk

- Learning to program can be intimidating
- You learn by trial and error
- Making mistakes makes you a better programmer
- Perseverance is key

“Never let the computer win” - Terence Parr



Getting Help

- Documentation (? in R)
- Online resources:
 - Stackoverflow
 - RBloggers
 - Data Camp
 - Coursera
 - Google!
- Your peers
- Your professors
- Anyone/anything that's willing to listen!



R Programming Basics

Hello, world!

```
# First program:  
print("Hello, world!")
```

Variables

- A variable is an object in our code that can store information for later use and operation
- We have two ways of assigning a value to a variable: using `=` or using `<-`
- Variable names:
 - Variables can contain letters, digits, underscores, or periods
 - It is good form to name your variables as words that are easily interpreted by whoever is looking at your code
 - Should start with letters, use a decimal or underscore to separate words
 - Case sensitive

```
10 # Assigns text to variable msg
11 msg = "Hello, world!"
12 print(msg)
13
14 # Reassign text to variable msg
15 msg <- "Hola, mundo!"
16 print(msg)
```

What is the value of msg at the print statement?

```
19 msg <- "Good morning everyone"  
20 msg <- "It's nice to be here!"  
21 msg <- "Who wants some coffee?"  
22 print(msg)
```

What is the value of num at the 2nd print statement?

```
24 # Assign number 1 variable num
25 num = 1
26 print(num)
27
28 # Add 1 to existing value of num and reassign to num
29 num = num + 1
30 print(num)
```

And now?

```
33 num = 1
34 num = num * 2
35 num = num + 8
36 num = num / 2
37 print(num)
```

Data Types

- Character (text data)
- Numeric (decimals)
- Integer (integers)
- Logical (true/false)
- Vector (list of items)
- Data Frame (like a table, more on this later)

Use **class(x)** to get data type of x.

Character data

The **character** data type is what we refer to as **text data** or **strings** in other programming languages.

```
39 # Character data type
40 msg = "This is in fact text data. In R it's called character data. How nice!"
41 class(msg)
```

Numeric data

The **numeric** data type refers to what we call **decimals** (or **floats** in other programming languages).

```
43 # Numeric data type
44 pi = 3.14
45 class(pi)
```

Integer data

- Self-explanatory
- R defaults to **numeric** data type, have to manually specify integer type

```
51 # How to declare integer type
52 # option 1:
53 my.int = 3
54 my.int = as.integer(my.int) # convert to integer
55 print(class(my.int))
56
57 # option 2:
58 my.int = 3L
59 print(class(my.int))
```

Logical data

- True or False
- Very important in control structures and filtering

```
61 # Logical data type
62 true = TRUE
63 class(true)
```

Pop quiz!

Guess the value of variables a, b, c, and d below:

```
65 # Pop quiz: guess the value of a, b, c, and d:
66 a = 3 < 4
67 b = 5 >= 8
68 c = "a" == "a"
69 d = "a" != "b"
```

Vectors

- The backbone of modern data computation
- “List” of items of the **same data type**
- **class()** function returns the data type of the **items** in the vector

```
76 # Vectors
77 num.vec <- c(1, 2, 3)
78 print(class(num.vec))
79 print(num.vec)
```

Shortcut: sequential integers

- Use “:” to construct a vector of sequential integers

```
86 # Vector of sequential integers shortcut
87 sequential.vec <- 1:10
88 print(class(sequential.vec))
89 print(sequential.vec)
```

Accessing elements in vector

Take an element of the vector by passing its index to the “[]” operator:

```
91 # Accessing elements in an vector
92 # Define and print vector
93 my.vec <- 1:10
94 print(my.vec)
95
96 # Extract and print first item in vector
97 first = my.vec[1]
98 print('First item:')
99 print(first)
100 print(my.vec[1])
```


Slicing the vector

Take a range of values (or a slice of the vector) between two indices using the “:” operator:

```
106 # Slice vector
107 # Take 3rd through 6th elements of vector
108 three.thru.six <- my.vec[2:6]
109 print(three.thru.six)
```

Mathematical operations

+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation

Logical operations

==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
&	and
	or
%in%	in

Guess the output of each print function below:

```
145 # Logical operation example
146 x = 5
147 print(x > 3)
148 print(x != 4)
149 print((x > 3) & (x < 4))
150 print((x > 3) | (x < 4))
```

Functions

Functions are ways in which we can create "mini-programs", or organize code that takes in an input and (usually) returns an output.

We won't get into creating functions much, although I show you an example below. The important thing, is knowing that we can call a function, feed it some information, and it will do something pre-defined.

```
155 # Define the function
156 ▾ add.one <- function(x) {
157     return(x+1)
158 ▲ }
```

```
160 # Example
161 x = 5
162 y = add.one(x)
163
164 # Print results
165 print(x)
166 print(y)
```

Reading data & working directories

Reading files

- In the real world, we will usually have to read in data from a file, the internet, or a database.
- We will cover how to read a CSV file into a data frame in R and how to write an R data frame to a CSV file.

Working directory: dude where are my files?

- The working directory refers to the folder in your computer file system that you are currently working in
- R can automatically see all files in your working directory (as shown in the Files tab of the bottom right panel in RStudio)
- If you wish to load data from outside your current working directory, you will have to tell R where to look
- Check your current working directory by using the **getwd()** function
- Set a new working directory by using the **setwd()** function

Loading a CSV file

- Use the **read.csv()** function to read in a CSV file into a data frame
- Use the **head()** function to display the first 6 rows
- Use the **tail()** function to display the last 6 rows

```
182 # Reading in CSV file
183 iris.data = read.csv("data/iris.csv")
184
185 # Display first 6 rows of data
186 head(iris.data)
187
188 # Display last 6 rows of data
189 tail(iris.data)
```

Dataframes

Manually create a data frame

```
191 # Creating a data frame manually
192 ex.df = data.frame(name=c("Betty", "Harry", "Susie", "Barry"),
193                    fav.food=c("Eggs", "Dairy", "Limes", "Berries"),
194                    age=c(14, 40, 24, 60))
195 print(ex.df)
```

Check dimension of data frame

- Use **nrow()** function to get number of rows
- Use **ncol()** function to get number of columns

```
197 # Check dimensions (number of rows, number of columns) of data frame
198 print(nrow(iris.data))
199 print(ncol(iris.data))
```

Extracting column names

- Use the **names()** function to extract column names

```
201 # Extracting column names
202 iris.names <- names(iris.data)
203 print(iris.names)
```

Extracting columns

- Use the \$ operator to access columns by name

```
205 # Extracting columns
206 iris.variety <- iris.data$variety
207 print(iris.variety)
```

Subsetting multiple columns

- Create a vector with the names of columns you would like to keep
- Subset data frame by passing that vector to the `[]` operator

```
209 # Subsetting multiple columns
210 cols.of.interest = c("petal.length", "petal.width", "variety")
211 iris.subset <- iris.data[cols.of.interest]
212 head(iris.subset)
```

Selecting rows

We can filter a dataframe by certain logical conditions. This is most easily done using the **subset()** function but there are many, many ways of doing this.

```
215 # Selecting rows
216 iris.virginica <- subset(iris.data,
217                           variety == "Virginica")
218 head(iris.virginica)
219 tail(iris.virginica)
```

We can also subset on multiple filtering conditions

```
222 # Subset on multiple filtering conditions
223 iris.subset <- subset(iris.data,
224                       (variety == "Virginica") & (petal.length >= 6))
225 head(iris.subset)
226 tail(iris.subset)
```


Subsetting rows and columns simultaneously

In addition to filtering logic seen in the **subset()** function, we can pass a vector with the column names we would like to keep to the **select** argument.

```
229 # Subsetting both rows and columns simultaneously
230 iris.subset <- subset(iris.data,
231                      (variety == "Virginica") & (petal.width < 2),
232                      select = c("petal.length", "petal.width", "variety"))
233 head(iris.subset)
```

Creating new columns

Use the `$` operator followed by the new column name to create a new column in the existing dataframe

```
236 # Create a new variable called `petal.width.mean` that contains
237 # the average value of the `petal.width` column.
238 iris.data$petal.width.mean <- mean(iris.data$petal.width)
239 print(head(iris.data))
240
241
242 # Create a column called `sepal.minus.petal` that is defined as
243 # the `sepal.length` minus the `petal.length`.
244 iris.data$sepal.minus.petal <- iris.data$sepal.length - iris.data$petal.length
245 head(iris.data)
```

Example: mean-centering a column

```
248 # Mean-center the `petal.length` column; that is, define a new column called
249 # `petal.length.centered` that is defined as `petal.length` minus the mean
250 # of all values in the `petal.length` column
251 mean.petal.length <- mean(iris.data$petal.length)
252 iris.data$petal.length.centered <- iris.data$petal.length - mean.petal.length
253 head(iris.data)
```

Write a dataframe to CSV

Use the **write.csv()** function to write your dataframe to a csv file.

```
256 # Saving a data frame to a CSV File
257 write.csv(ex.df, "data/example_df.csv", row.names=FALSE)
```

Knowledge check

1. Load the ``cars.csv`` file found in the data folder into a data frame variable called ``cars``.
2. Inspect the first 6 rows of the ``cars`` data frame. What is the third value under the column ``cyl``?
3. Reassign the variable ``cars`` to a data frame containing all rows and columns of the ``cars`` data frame **except for** the ``model`` column.
4. Using only one line of code, print the value of the 12th observation in the ``cyl`` column.
5. Calculate the mean of the ``mpg`` column. Print it out.
6. Create a new column called ``mpg.centered`` which is a mean-centered version of the ``mpg`` column.
7. Subset the data frame for observations where the **mpg is above the mean**.
8. Repeat **7** but only return the ``mpg``, ``cyl``, ``hp``, and ``wt`` columns.
9. Repeat **8** but include the additional filtering condition that ``hp`` must be greater than or equal to 100. Save this subset to a variable called ``vroom_vroom``.
10. Write the data frame ``vroom_vroom`` to a CSV file with a name of your choosing.