**Data Analytics (CMP330)**

# Practical 1 – R Basics

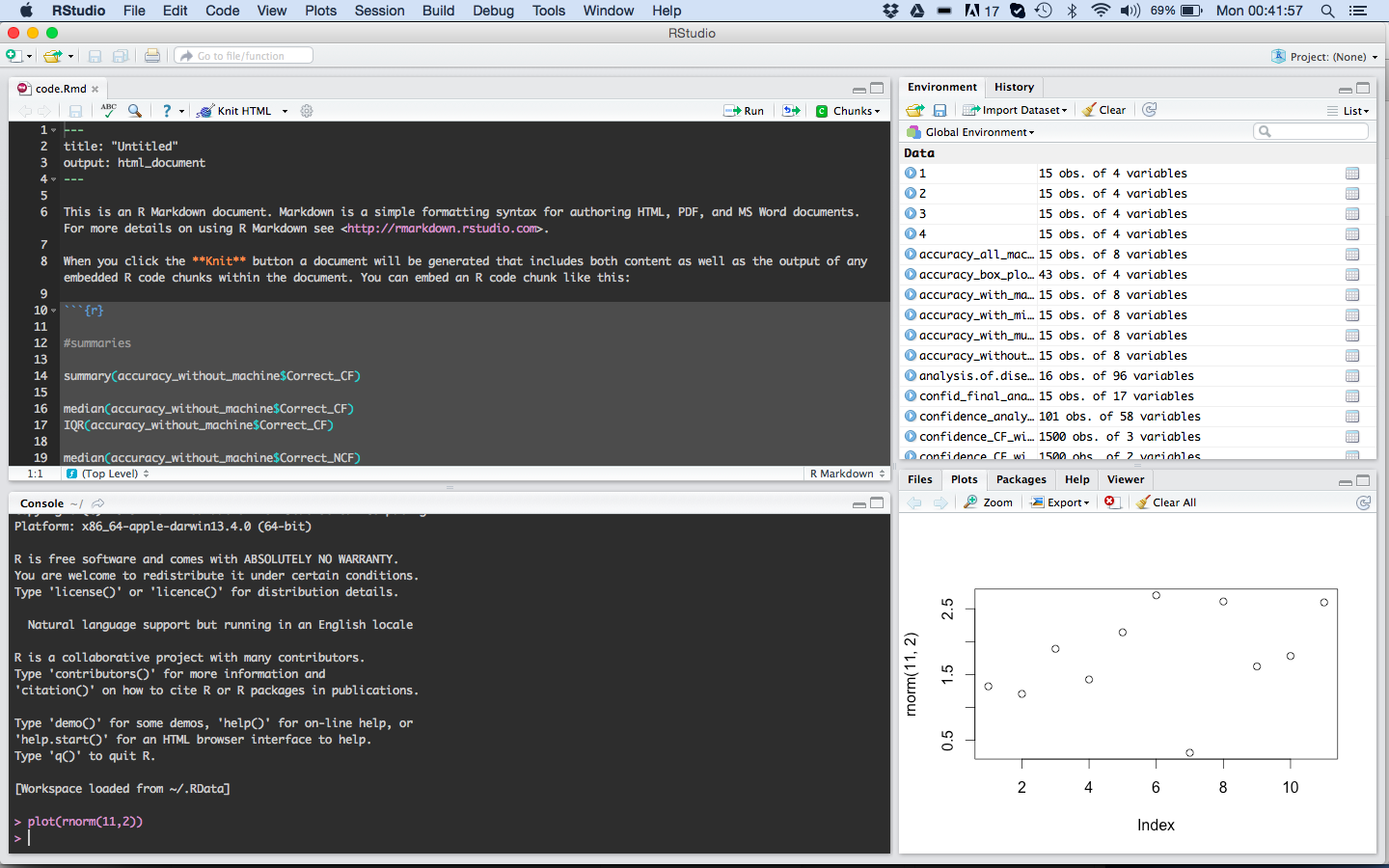
1. **Get Orientated with R Studio, identify the following:**

1. Console.

2. Workspace and history.

3. Files, plots, packages and help.

4. The R script(s) and data view.

****

1. **Variables**

Create a numerical variable in the console and display it:

x <- 12.7389

x

You can also display it using the print function:

print(x)

To format it to 2 decimal places (digits specifies the full number of digits):

print(format(x,digits=4))

We can also concatenate output:

cat("Value of x:", x)

Standard operations apply, e.g.

x <- (25 - 4.5\*\*2) / 3

x

If we assign a value as follows, it is treated as a numerical variable:

x <- 14

class(x)

If we want to represent it as an integer:

x <- 14L

class(x)

There are also logical or Boolean variables, e.g.

b <- TRUE

So if x is a numerical value we could assign the result of a Boolean expression:

b <- (x > 2)

or combine various expressions. What does this expression mean?

b <- (x > 2 && x %% 2 == 0)

Of course, there are also character/string variables:

str <- “apple”

We can find the number of characters in a string:

nchar(str)

and concatenate strings. Try out the following after initialising str1 and str2:

str = paste(str1,str2)

**Exercise 1. You should spend some time exploring mathematical, logical and comparison operators in R to make sure you understand them. See** [**https://www.w3schools.com/r/r\_operators.asp**](https://www.w3schools.com/r/r_operators.asp)

**You should also explore built-in mathematical functions. See** [**https://www.w3schools.com/r/r\_math.asp**](https://www.w3schools.com/r/r_math.asp)

1. **Create a Script File and Save it**

So far we have just been using the console. To work with script files, first set the working directory by using the following commands (note that # is used for comments):

# find the current working directory

getwd()

# Changes the wd – choose a directory that you wish to work in (specify as

# appropriate)

setwd(“C:/myfolder/data")

Now go to ***File – New File – R Script*** and then type in:

a <- 2

b <- -7

c <- -15

x = (-b + sqrt(b\*\*2 - 4\*a\*c)) / 2\*a

print(x)

To run the code, highlight all the code, then click **Run** or **Ctrl-Enter** or simply click **Source** to run all code in script. (Note: if you don’t highlight the code and click **Run**, then it will only run the line of code the cursor is on.)

**Exercise 2. Hopefully the formula in this code looks familiar to you. It is an attempt to get one of the solutions to the quadratic equation:**

**However, there is a problem with the code. Can you find it and correct it? Make sure to check the answer. Can you also modify the code to find the other solution?**

1. **Writing Functions**

Create a new Script file with the following code to define and call a function:

add <- function(a,b){

y <- a+b

return(y)

}

add(2,3)

**Exercise 3. Rewrite the code from Exercise 2 as a function with the coefficients a, b and c (2, -7 and -15 in this example) as arguments so that your function can find one of the solutions for quadratic equations in general, e.g. it should give the solution if you have different values of a, b, c such as a = 1, b = -6, c = 5.**

**Extra Exercise. Following the approach from Exercise 2 will give you one of the solutions, but there are a couple of complications for a full solution. First, you would need to think about how you would return the two solutions to the problem. You could work out the two solutions, save them as a vector (see the section on vectors below) and return the vector. Second, you would need to think about what happens in cases where b2 < 4ac. For example, what happens if you set a = 2, b = -7, c = 15? You could use an if statement (see below) to ensure b2 ≥ 4ac. (Or alternatively, you could use complex numbers, see** [**https://www.w3schools.com/r/r\_numbers.asp**](https://www.w3schools.com/r/r_numbers.asp)**)**

1. **Branching and Iteration**

These are very similar to other languages such as Python. For example, try out the following code by assigning strings to str1 and str2:

if (str1 == str2) {

print("The strings are the same.")

} else {

print("The strings are different.")

}

Are strings considered to be equal if one is lower case and the other is upper case? Does the code work if you remove the braces {}? Or if you move “else” on to the next line?

Try out the following code to see what it does:

f <- function(n){

sum <- 0

for (i in 1:n) {

sum <- sum + i

}

return(sum)

}

f(5)

Alternatively, we could create a vector using:

v <- 1:n # or v <- c(1:n)

and rewrite the for loop as:

for (i in v) {

sum <- sum + i

}

**Exercise 4. You should spend some time exploring for-loops further as well as while loops, see** [**https://www.w3schools.com/r/r\_while\_loop.asp**](https://www.w3schools.com/r/r_while_loop.asp)

**Exercise 5. Write a function that checks for each number from 1 to 100 whether it is divisible by 3 or 5 and then outputs the number and states whether it is divisible by 3 or 5:**

**1. Not divisible by 3 or 5.**

**2. Divisible by 3 or 5.**

**…**

**(Hint: use the paste0 function (a version of paste function we used earlier) to format the output.)**

1. **Vectors**

**Creating vectors**

Vectors are essentially the same as lists in Python. We’ve already seen an example of how to create a vector:

v <- c(1:10)

Or we can specify particular values:

v <- c(7, 6, -3, 4)

Or uses previously defined variables:

a <- 4

b <- 3

v <- (a,b)

Or we can create an empty vector and then append values as needed:

v <- c()

v <- append(v, 5)

Or we can create a vector of a specified length with all the values being the same:

v <- rep(0, 10)

Or we can use the sequence function:

v <- seq(1, 5, by = 0.2)

# specifies a step size of 2, i.e. (1.0, 1.2, 1.4, …, 5.0)

v <- seq(1, 10, length.out = 5)

# specifies the length of 5, i.e. (1.00, 3.25, 5.50, 7.75, 10.00)

**Accessing vectors**

We can access an element of a vector by specifying its index:

v <- c(3, 7, -4, 6, 9, 13)

v[3] # accesses the value -4

Note that the index of the first element is 1 (not 0 as it is in Python), so:

v[1] # accesses the value 3

We can access multiple specified values, e.g.

v[c(2,4)] # accesses the 2nd and 4th elements, i.e. 7 and 6.

Or a sequence of elements, e.g.

v[3:5] # accesses the values -4, 6 and 9

**Exercise 6. Try out the following statements to determine what they do:**

any(v > 8)

any(v < 8)

all(v > 50)

v > 4

which(v > 4)

**Now try to work out the output from the following statementbefore running code.**

y <- v[v > 4]

y <- v[v > 5 | v == 3]

(Note: while “||” is the logical OR operator, “|” is the elementwise logical OR operator, i.e. it is applied to each element of the vector in turn.)

**Modifying vectors**

We can of course modify the elements of a vector:

v[4] <- 1

Or modify multiple elements based on a condition:

v[v > 4] <- 0

Try out some arithmetic with vectors:

2 + c(6,0,9,20,22)

3 \* c(6,0,9,20,22)

**Applying functions to vectors**

Find the length of the vector:

length(v)

Calculate the mean and standard deviation:

mean(v)

sd(v)

Use the summary function:

summary(v)

Take the square root:

sqrt(v)

The round function:

v <- c(9.5,16.32,25.16)

round(v) # round(v,1) to round to 1 d.p.

**Recycling vectors**

R automatically repeats or recycles vectors when necessary. Consider

v <- c(3,4,2)

v + 2

R converts this to c(3,4,2) + c(2,2,2).

c(1,2,4) + c(6,0,9,20,22,11)

becomes

c(1,2,4,1,2,4) + c(6,0,9,20,22,11)

**Key value pairs**

v <- c(1,2,3)

names(v) <- c("a", "b", "c")

v["b"] # displays the value 2

**Explicit coercion or casting**

Try each line of code and understand what is happening:

v <- c(0:6)

class(v)

as.numeric(v)

as.logical(v)

as.character(v)

v <- as.character(v)

**Nonsensical Coercion**

x <- c("a", "b", "c")

as.numeric(x)

prints out: [1] NA NA NA

Warning message:

NAs introduced by coercion

as.logical(x)

prints out: [1] NA NA NA

**Exercise 7. Write a function that determines how many elements in a vector lie within a specified range, i.e. between two specified values. Your function should have three arguments: one for the vector and for each of the two values. Make sure to test your function thoroughly with different input vectors.**

1. **Simulating Data in R**

Try each line of code. The ‘?’ is a help function that will give you details about the built-in function that you specify. In this case the **rnorm** or random normalization function which provides a vector of numbers that are randomly drawn from a probability distribution with a particular mean and SD.

?rnorm

x <- rnorm(10)

x <- rnorm(10, 20, 2)

summary(x)

**Setting the seed**

This ensures that you can get the same set of random numbers. Ensures reproducibility.

set.seed(1)

rnorm(5)

rnorm(5)

set.seed(1)

rnorm(5)

**Bootstrapping**

The sample() function allows for bootstrapping, where you can draw a sample from a vector. You specify what vector you want to sample from and the number of samples. You can choose to bootstrap/sample with and without replacement. You can read more online - <https://en.wikipedia.org/wiki/Bootstrapping_(statistics)>

set.seed(1)

?sample

sample(c(1:10), 4)

sample(c(1:10), 10, replace = TRUE)

**Exercise 8. Create a vector with 1000 elements drawn randomly from a normal distribution with a mean of 50 and standard deviation of 10. Use your function from Exercise 7 to determine what proportion of your data lie within one standard deviation of the mean, i.e. between 40 and 60. Is your result consistent with the empirical rule?** [**https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7\_rule**](https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule)

**Now generate a sample consisting of 100 numbers drawn randomly with replacement from your vector of 1000 elements using the sample function. Determine the mean value of this sample and the proportion of its values lying between 40 and 60. Finally, use a for loop to repeat this process for 100 samples and then find the average of all their mean values and the average proportion of values lying between 40 and 60.**

1. **Read article to compare R and Python as data science tools**
   1. <https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis#gs.kWTKL0A>
   2. Carry out your own Google searches to compare R and Python
2. **Watch video by John Cook on R – the good, the bad and the ugly**

[**https://www.youtube.com/watch?v=6S9r\_YbqHy8**](https://www.youtube.com/watch?v=6S9r_YbqHy8)