Lab 1 – Base R

Section 1 - Introduction

Before starting … remove objects from the working environment

```
rm(list = ls()) # clears environment
```

Section 1.1 Comments

Good practice in writing code is to provide notes to yourself or others who may review it. This is achieved by using the # symbol:

```
##########################################################
#     COMMENTS
##########################################################
#Comments begin with a hash symbol
```

Note: if I enter a comment beginning with a # symbol, then the system simply ignores the text which follows on the same line. If I enter the same text without a # symbol, R will throw out an error.

Exercise 1.1: Make a comment in R.

Exercise 1.2: Enter a comment into R without a # and note the error generated.

Section 1.2 R Help

```
##########################################################
#     R HELP
##########################################################
#to access R help type
help.start()
# You can then use this to search for a desired topic or browse the
documentation.
#To get help on a particular function, use the question mark
?plot
```

Exercise 1.3: Access the R help.

Exercise 1.4: Access the plot help.

Exercise 1.5: Generate two vectors, one being the numbers 1,2,3,…10 and the other being 2,4,6,…20.

Exercise 1.6: Using the R help plot these.

Exercise 1.7: Using the R help add a main title, subtitle and label the axes.

Section 1.3 Basic Calculations

```
##########################################################
#     GETTING USED TO CODING - BASIC CALCULATIONS
##########################################################
```

Exercise 1.8: Run the following code

```
a=2
a=3
b<-3
a*b # note here the value of a*b is printed and then lost
c=a*b # now the value of a*b is stored as c
b/a
3*6+234
exp(log(10))

a = 2;  is.logical(a); is.list(a); is.numeric(a)
# Note that semicolons (;) let you run several calculations in the
same line.
# You could write the above as 4 separate lines if you prefer
```

Exercise 1.9: Save the code from Exercise 8 in a script file. Add appropriate comments to the code. Run the code again to ensure you have not generated any errors

Exercise 1.10: A manufacturing site are running a process; one of the unit operations in this process is a filtration step. There was 1000L of material entering the filter with a concentration of 2mg/L. 997L are recovered with a concentration of 1.95mg/L. Determine the yield. Note

$$\text{Yield} = \frac{\text{amount}_{\text{out}}}{\text{amount}_{\text{in}}} = \frac{\text{Vol}_{\text{out}} * \text{Conc}_{\text{out}}}{\text{Vol}_{\text{in}} * \text{Conc}_{\text{in}}}$$

Section 1.4 Sourcing Data

```
##########################################################
#     LOAD AN AVAILABLE DATASET
##########################################################
```

Exercise 1.11: Run the following code:

```
data() # gives list of available built-in datasets
mtcars # prints the Motor Trend Car Road Tests dataset
?mtcars # gives info on the mtcars dataset
```

Exercise 1.12: Use the plot function to plot mpg against quarter mile sprint time. You may find dataframe indexing useful here.

```
#Most built-in functions use these built-in datasets in the examples,
# so it is good to be familiar with them and how to access them.
```

Section 1.5 Data Types

```
#########################################################
#     VECTORS
#########################################################
```

We were introduced to vectors in this week's lecture.

Exercise 1.13: Run the following code which generates a vector and then selects different entries in that vector, also finally measuring the length of that vector.

```
x = c(0.1, 2, 4.3, 3.1, 5)
x[2] # returns the second value in that vector
x[1:3] # returns the first three values
x[-(1:3)] # gives all but the first three elements
x[x>3] # selects the subset of values in x greater than 3
N = length(x) # stores the length of x
```

Exercise 1.14: Generate a new vector *y* where *y=2*x+1.*

Exercise 1.15: Calculate the mean, median and standard deviation of both x and y.

Exercise 1.16: Use the *seq()* function to generate a vector containing the following values:
-5, -4, -3, …, 3, 4, 5

Exercise 1.17: Use the rep function to repeat the vector x twice.

Exercise 1.18: Run the following code. Note the relationship between the different fruits and their values.

```
#Vectors may also contain characters or strings:
fruit <- c(5, 10, 4)
names(fruit) <- c("orange", "banana", "apple")
lunch <- fruit[c("apple","orange")]
```

Section 1.6 Matrices, Arrays & Lists

```
#########################################################
#     MATRICES
#########################################################
```

Exercise 1.19: Below is some code showing how to use matrices. You should run this code at home and make sure it works and you understand it. Ask any questions you have at the next lab.

```
# Create a vector
x = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
# Now make it a 3×3 matrix:
matrix(x, ncol = 3) # note how this is arranged.
#If we prefer arranging by rows:
M = matrix(x, ncol = 3, byrow = TRUE)
#The matrix now has dimensions:
dim(M)
ncol(M)
nrow(M)
#What do the following do?
t(M); aperm(M, c(2,1))
diag(M)
det(M)

#define two matrices A and B as below:
A <- M; B <- t(M)
#Element-by-element product:
A * B #(if A and B are square)
#Matrix product:
A %*% B
# note how these are considerably different!



#########################################################
#     ARRAYS
#########################################################
```

Exercise 1.20: Below is some code showing how to use arrays. You should run this code at home and make sure it works and you understand it. Ask any questions you have at the next lab.

```
h = seq(1, 24, by=1)
Z <- array(h, dim=c(3,4,2))
Z[1,1,1]; Z[1,2,2]
```

```
#########################################################
#     LISTS
#########################################################
```

Exercise 1.21: Below is some code showing how to use lists. You should run this code at home and make sure it works and you understand it. Ask any questions you have at the next class.

```
Lst <- list(name="Fred", wife="Mary", no.children=3,
            child.ages=c(4,7,9))
Lst[[2]]
Lst[[4]][1]
#It is important to be familiar and comfortable with indexing in this
way
#notice the difference in the following two outputs
Lst[4]
Lst[[4]]
#You can also index into a list (or dataframe) using $.
# If you type Lst$, you should get a prompt from RStudio of the
# available sublists/variables/
Lst$child.ages
```

Section 2 – Reading & Writing Files, Loops & Functions, Dataframes

Before starting … remove objects from the working environment

```
rm(list = ls()) # clears environment
```

Section 2.1 – Reading files

```
#########################################################
# READ IN A CSV FILE
#########################################################
```

The Book1.csv file is available on Canvas; download and save this to a folder named Lab2.

Exercise 2.1

The following is code that reads a csv file into R and saves it in a data structure called *data.* Modify the directories in this code and then run this code.

```
#to make directory "C:\Users\YourName\STAT8010\Lab2"
getwd() # check current directory in R
setwd("C:\\Users\\YourName\\STAT8010\\Lab2")
## you will need to change this to where you saved the files
getwd()  # check folder you are working with now


#read in file
#1. observe what happens is using "read.table" first
data1=read.table("Book1.csv", header=F)
data1

#Now use read.csv with correct options/arguments
data=read.csv("Book1.csv", col.names=c("a","b","c"),header=F)
#notice the options for col.names & header;

# see what happens if you remove these
data2=read.csv("Book1.csv", header=F) # note no column names
data2 # note that header=F is the default if left blank, see
?read.csv


#Now try with header = TRUE
data3=read.csv("Book1.csv", header=T)
# Column names incorrectly set to first data row
data3

#Note you can also use scan() to read this file. The good thing
# about this function is that it can read mostly everything, though
# you must give the data the desired format manually.
```

```
data4 = scan("Book1.csv", sep=",")  # read observations
# define variable names separately
names = c(”a”,”b”,”c”)
data;names


#Turn this into a matrix to give structure
M = matrix(data4, ncol=3);M   # give matrix structure
DF = data.frame(M); DF     # give data.frame structure
colnames(DF) = names;      # give variable names to the data.frame
DF
##
#Note that you can create the directory using the command in R
# “dir.create()”. This is particularly useful you need to safe files
# in different folders while you are analysing a dataset
# (e.g. you have a dataset with different patients, each patient in a
# different folder, and you need to safe the regression output for
# each patient).
#Some other useful functions for dealing with files and directories
list.files()  # Check the files in the folder
dir() # same as above
which(dir() == "Book1.csv")
#gives the position of the file in the folder
```

Section 2.2 Creating dataframes and writing to a file

```
############################################################
#Creating a dataframe and writing to a file
############################################################
```

Exercise 2.2: Use the following code to define the following dataframe in R:

```
df <- dataframe(x=1:5,
                y=seq(3,11,2),
                z=c(6,2,22,7,15)
      )
```

X	Y	Z
1	3	6
2	5	2
3	7	22
4	9	7
5	11	15

Then write this dataframe to the following file formats:

- Xlsx
- Txt
- Csv

Note the options in the writing functions and resulting outputs in the files.

Section 2.3 If-Statements

```
#########################################################
#LAB IF STATEMENT EXERCISE
#########################################################
```

If statements were briefly mentioned in class. Run the following block of code and explore which lines of code execute and why they execute. Next change "hello" to "one" and run this code, then change "hello" to "two" and run the code again. Finally, try running with x=1 and then with x=2 in the first line. What do you notice?

```
x="hello"
if(x==1){
         print("one")
        }else if(x==2){
                      print("two")
                     }else{
                           print("not one or two")

                          }
```

Exercise 2.3

You should run the following code, which was discussed in the lecture earlier:

```
i=1
if(i==1){
         print("one")
        }

if(i==2){
         print("two")
        }

i=2
if(i==2){
         print("two")
        }
```

Exercise 2.4

You should run the following code (one line at a time), which asks the user to provide their name, and their age and then prints out a sentence using this information.

```
my.name <- readline(prompt="Enter name: ")
my.age <- readline(prompt="Enter age: ")
# convert character into integer
my.age <- as.integer(my.age)
print(paste("Hi,", my.name, "next year you will be", my.age+1,
            "years old."))
```

Use the help function in R to investigate any functions you are unfamiliar with.

Exercise 2.5

You should modify the code in **Exercise 2.3** with a readline command to allow the user to provide a value of i, rather than it being hardcoded.

Exercise 2.6

You should now modify the code in **Exercise 2.5** to be an `if…else…if` statement rather than two separate if statements.

Exercise 2.7

Change the code from **Exercise 2.6** to include not just if and else if but also an else statement which should output "not one or two"

Section 2.4 Loops & Functions

Exercise 2.8

Try running the following lines of code and noting the output.

```
###Example loop print the numbers one to ten on the screen
for (i in 1:10){
                print(i)
               }
############
#The following define various functions.  Try playing around with
these functions, with different inputs, to get different outputs.

#Example function to print "Hello World on the screen"
helloworld <- function(){ print("Hello World!") }
#Write a funciton to square a number
squared <- function(x){ x*x }
```

```
EuToD = function(e){ e*1.174391 }

#function to convert temperature (approximately) from
# Celsius to Fahrenheit
TOf =function(c){ c*9/5+32 }
```

Exercise 2.9

Write a line of code to assign y a random uniformly distributed number between 1 and 10 [investigate the `runif()` function]. Write a nested if else statement which will print the value of y and state whether it is

- Less than 2
- Between 2 and 5 (not including 5)
- Between 5 and 9 (not including 9)
- Greater than or equal to 9

The output should be something like "`y is 5.535017, which is between 5 and 9`" [investigate the `cat()` command for printing to console].

Exercise 2.10

Write a for loop which assigns a random number between 1 and 100 to z. The for loop should do this 200 times; each time the loop should print the number, for example "The random number is 45.658616".

Exercise 2.11

Modify the above for loop to end if a random number generated is greater than 90. If this happens, the code should also print "A number >90 found – Exiting loop"

Exercise 2.12

Write a nested for loop to generate the following matrix: $M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 6 & 8 & 10 \\ 3 & 6 & 9 & 12 & 15 \\ 4 & 8 & 12 & 16 & 20 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix}$

Section 2.5 Filtering & Summarising Data

Exercise 2.13

Load the Edu_salary.csv file into a data frame in R. Now create a smaller data set containing all data relating to observations with salaries greater than or equal 45 [investigate the subset() function for this]. Then write this smaller data set to a new .csv file called High_Salary.csv

Exercise 2.14

Load the brainsize.txt file into a data frame in R. Display the structure & summary statistics of the data set. Calculate the sum of all variables except the gender variable.

Exercise 2.15

Add an id number variable to the dataframe and give each person a unique id number.

Exercise 2.16

Write code which will give the details of each person with an MRI count above 1,000,000.