**R Basics**

**General Tips**

* R is case sensitive ( ‘x’ is not equal to ‘X’)
* To assign values use <- or =
* == is used for equality tests (e.g.: *which(variable == min(variable))* )
* To assign characters to a variable you must use quotation marks (e.g. x = “blue”)
* Objects’ name may include letters, numbers or periods (e.g.: x.1), but the number must never come first in the name
* It’s possible to perform operations in R: +, - , \* , /, ^ , sqrt(), log (), exp(), log2(), abs ()
* If you enter an incomplete command, R will follow that up with a plus sign in the line below
* To enter comments, just initialize the sentence with hash #
* To list rows that have missing values on a dataset: *data[!complete.cases(data),]*

**R Help**

* Access R help by entering the command *help(topic)* or, equivalently, entering a question mark followed by the topic you want to search *?read.csv*

**Working Place, Saving and Quitting:**

* Save through the command *save.image(“ProjectName.Rdata”)*or go to Session > Save Workspace as… . As a reminder, if you choose to save through code, the project will be saved in your current Working Directory.
* To change the Working Directory you can also do it manually by Session > Set Working Directory > Choose Directory or do it manually by inserting the command *setwd(“/Users/Mariana/Desktop/Project”)* or simply *setwd(“~/Desktop/Project”)*
* To know the current Working Place enter *getwd()*
* To quit you can enter *q()*or manually go to RStudio > Quit RStudio
* Notice that R gets confused when you \ to write a path’s name, instead you can use / or \\ (e.g.: “c:\\mydocuments\\rdata.txt” )

**Scripts Initialization**

* Any script can be loaded if it has the extension .R through File > Open File…
* To create a new Script, go to File > New > R Script

**Importing Data**

* You may import files from excel by saving the spreadsheet as .csv (recommended) or .txt
* You can save and import the data by creating an object using one of the following options:
* For .csv files: data <- *read.csv(file.choose(), header = TRUE, sep = “,”)* or if the file is already in the directory use *read.csv( ‘Filename.csv’, header = TRUE, sep = “,”)*
* Second option (more general): *data <- read.table(file.choose(), header = TRUE, sep=”,”)*. In the sep command (which is also optional) you should put the character by which the elements will be separated into quotation marks (“,” for .csv or “\t” for .txt)
* Third option for .txt files: *data <- read.delim(file.choose(), header = T)*
* Fourth option (do it manually): select Import Dataset from the Workspace: Import Dataset > From text file…

**Shortcuts**

* CTRL + L clears the console
* Using Arrow Up Key on the console will bring the last command used
* To run current line/selection press Command + Enter (for iOS users) or Ctrl + Enter (Windows)
* To Interrupt currently executing command press Esc
* Press tab when writing an expression to get a list of all possible commands from R
* CTRL + L clears the console
* For a complete list, go to Tools > Keyboard Shortcuts Help…

**General Useful Commands**

* *print(variable)* R will return the value for the variable in ()
* *ls(variable)* to see what variables are stored in the workspace (memory)
* *rm(variable)* to remove the object/variable from the workspace. To remove all use the expression *rm(list=ls())*
* *c(sequence)* used to concatenate or create a vector of values, e.g. *x <- c(0,1,2,3,4,5,6)*, *gender <- c(“male”, “female”)*.
* To create sequences of integer values, simply use colon, e.g.: 2:7 will generate the sequence 2 3 4 5 6 7. For general sequences, use the seq command, e.g.: *seq(from=1, to=7, by=1),* where by corresponds to the pace of the increment and can be any fraction of number.
* To count the number of a specific element in a matrix or dataframe : *data[names(data[which(variable==0)])]*
* To create a vector of repeated numbers or characters use the rep command, e.g. *rep(1, times=10)* to generate the sequence 1 1 1 1 1 1 1 1 1 1 or *rep(c(“m”, “f”), times =5))*
* To make an operation to each element of a vector simply enter the vector and the desired operation (e.g.: *x + 10*)
* To extract elements from a vector call it on brackets, e.g. *x[1]* corresponds to the first element of the vector x. A negative sign will extract all the elements but that one in the brackets, e.g. *x[-3]* will extract all except the third element. Other forms: *x[1:3]*, *x[c(1, 5)]*, *x[x<6]*, etc.
* To create a matrix, use the command *matrix(c(numbers), nrow = number of rows, ncol = number of columns, byrow=TRUE)*, where the byrow is a logical that when FALSE (default) the matrix is filled by columns, otherwise the matrix is filled by rows. The first argument c(numbers) is optional and could also be an expression.
* To extract elements from any matrix, remember to add information about the row and column, e.g.: x[1,2] as row 1 and column 2, or x[2,] to extract all elements in row 2 independently of the column, x[c(1,3), 2], etc.
* Use the dim command to know the dimensions *dim(x)*
* Use *head(x)* to see the first 6 rows or *tail(x)* to see the last 6 rows
* Use the command *names()* to check the names of the variables in a dataset (headers)
* In the case R does not recognize a header’s name as a variable, you can force it by adding the dollar sign $ to the dataset, e.g. *data$gender*. However, this will not make “gender” a variable, as you will always have to add $. Alternatively, use the command *attach(dataset)* which will transform all headers in variables. By using *detach(dataset)* you can delete the variables previously put in the workspace.
* To know the type of each variable, simply enter the command *class(variable.name)*. Possible values: integer, numeric, factor (categorical), etc. To change from one to another, enter the expression *variable.name <- as.newclass(variable.name),* e.g., *x <- as.factor(x)*
* To know the different types of categories, enter the command *levels(variable.name)* which will return the possible values in the dataset.
* Attach vectors in a column-wise fashion using the *cbind(dataset, new.column name)* command or in a row-wise fashion using *rbind(dataset, new.row.name)*
* Extract data with a particular value = many forms, check some below:
* *variable <- dataset.name[gender==”female” & age>15, ]*, where the whitespace after comma means all columns should appear. You can check if it worked by asking for dim(variable) or summary(variable) or variable[1:4, ], etc.
* *variable <- Age>15* will check in the database column Age each element, returning TRUE for the ones >15 and FALSE for the other ones. To transform to binary numbers, you can use variable <- as.numeric(Age>15)
* *variable* <- which(data$column1== min(data$column1))

**Packages**

* To learn more about, you can access the help menu through *help(install.packages)*
* To install any packages use the command *install.packages(“Package.name”).* R will then ask you to select a mirror, which is simply the location the file will be downloaded from.
* If you don’t know the name of the package, you can leave in blank and R will return a list with all the packages available for your location *install.packages(“”)*
* After installed, the package will be available forever, unless you uninstall through *remove.packages(“package.name”)***.**
* The commands will only be available if you load the library for the package through the function *library(package.name)*. This must be done every time you reset R.

**Descriptive statistics**

* Use the command *summary(dataset)* to get a brief summary of the dataset, with basic statistics information.
* *summaryBy* available at package DoBy allows to display summary by many variables, e.g. *summaryBy( data$x ~ data$QueryDate, data = data, FUN = function(x) { c( n=length(x),sum= sum(x), mean = mean(x), median=median(x), std = sd(x), range=range(x)) } )* to get info about length, total sum, mean, median, std and range of the variable x according to QueryDate
* *sd(variable):* to get population standard deviation . To get sample sd, you could use: sqrt(var(x)/length(x)).
* *mean(variable):* to get mean (ex: mean(x[height==0])
* *var(variable):* returns population variance

**Basic Plots**

* *dev.new()* to create a new blank figure (to create a new graphic after doing one)
* *par(mfrow=c(2,1))* to divide the figure into 2 panels. Change c(x,y) to increase/decrease number of panels.
* *qqnorm(data)* creates a Q.Q. norm
* *qqplot(data)* creates a Q.Q. plot. Complex example *qqplot(variances, (rchisq(500, df = (100-1))*, where rchisq is the function used for the sampling distribution of variances.
* *qqline(data)* add normality line to the Q Q Plot created
* *abline(linearmodel)* add straight line to the regression model stored in the variable linearmodel
* *boxplot( y ~ as.factor(x), main = “Title”, xlab = “X-axis Title”, ylab= “Y-axis title”)***:** creates a boxplot. Can also use comparisons to split x (ex: x==1)
* See function plot help, many possibilities!
* Also many possibilities using the library ggplot2, but it can get really complex, please check the “cheat sheet” available at:

[*https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf*](https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf)

**Random Number Generation**

* **Random Normal**: to generate a sample of 100 random N(0,1) observations use the command *x = rnorm(100, mean=0, sd=1)*. You can vary any of the parameters (number, mean and sd).
* **Random Uniform:** to generate a sample of 100 random U(0,1) observations use the command *x = runif(100, min=0, max=1)*. You can vary any of the parameters.
* **Many others available (check R help)**

**Hypothesis Tests:**

* *t.test(data, alternative, mu, conf.level):* performs a t.test. Default is 0.95 confidence level, two-sided test, mu is the mean we want to test. Alternative can be “greater”, “less” for a one-sided test or “two.sided” (default). Ex: *t.test(data,alternative = c("greater"), mu = 30, conf.level=0.95)*; or it’s possible to use only the two variables *t.test(diffhr[height==0],diffhr[height==1])* to test whether their mean is different or not, null hypothesis is that they’re equal (no difference between means or To do a paired t-test, just add paired= TRUE (not default).
* Many others available, see help!

**Loops and If**

* *For(i in 1:S) {command}*
* *Apply(array, margin, fun):* apply a function (fun) to the array (x) over a vector (margin). Margin can be a row (1) or column(2), so the result will be expressed in a row/columns with the same number of row/columns from the array (x). Examples: *apply(x, 1, mean, trim = .2)*, *apply(x, 2, sort)*. Use “which.function” so apply will return the index of the row (or column) that corresponds to that function value (ex: which.max will return the index of the maximum value of the row).
* *While(response != “No”) {command}*
* *Repeat {command}*, example:*repeat {*

*response <- as.integer(readinteger());*

*if (response == 1) {*

*print("Finish!");*

*break*

*} else print("Try again");*

*}*

* *If (test expression) { statement 1} else {statement 2}*