

## **Modelling for data analysis – FIT2086**

### **Studio 1 – Getting started with R and descriptive statistics**

The following topics will be covered in this session:


<b>1. GETTING STARTED IN R/RSTUDIO</b>	<b>2</b>
<b>2. USING SCRIPT FILES</b>	<b>3</b>
<b>3. ACCESSING DATA</b>	<b>4</b>
<b>4. FLOW CONTROL AND FUNCTIONS</b>	<b>6</b>
<b>5. FURTHER READING</b>	<b>8</b>
<b>6. BASIC DESCRIPTIVE STATISTICS IN R</b>	<b>8</b>
<b>7. CATEGORICAL DATA</b>	<b>8</b>
<b>8. MORE DESCRIPTIVE STATISTICS FOR NUMERIC VARIABLES</b>	<b>9</b>

## 1. Getting started in R/RStudio

The primary aim of this first studio is to obtain some familiarity in using R, and compute some basic descriptive statistics of dataset using R.

We will start by getting familiar with RStudio, its layout and functions.

- i. Open the dataset `chol_bmieig.csv` by using the menus (*File -> Import Dataset -> From CSV -> select `chol_bmieig.csv` and click "Import"*).
- ii. Become familiar with RStudio's layout by finding the following windows: Console, Environment, History, Help, and Files. What is displayed in each window?
- iii. Browse the dataset that you have just opened by clicking on `chol_bmieig` in the Environment window.
  - a. What does each row and column of the spreadsheet represent?
  - b. What other things do you notice while browsing the data?

You can also get information on the data set by clicking on the small arrow to the left of '`chol_bmieig`' in the Environment window (i.e.  `chol_bmieig` ).

- a. What information does this give you?
- b. What is different about the **Sex**, **bmi\_WHO** and **smoke** variables?

There are two ways of running most commands in RStudio: either typing commands in the console window, or typing and running commands in a script file (more about these later) and executing it using R. The former is more interactive, and is a good way of beginning to design analyses, explore data and get experience with R.

We are going to use the command window to open the `chol_bmieig.csv` file.

- iv. First, let us see what directory RStudio currently has set as the working directory; do this by typing: `getwd()`
- v. Change the directory to the folder which contains the `chol_bmieig.csv` file; type, e.g., `setwd("~/Documents/FIT2086/Studio 1")` (or whichever folder you have put the `chol_bmieig.csv` file in)
- vi. Clear the current environment of all variables/datasets.  
Type `rm(list=ls())`
- vii. Open the `chol_bmieig.dta` dataset. Type:  
`chol_bmieig <- read.csv("chol_bmieig.csv")`
- viii. Browse the `chol_bmieig` dataset.

```
Type View(chol_bmieg)
```

## 2. Using script files

Using script-files will ensure reproducible data management and analysis. Log files are used to make a full record of your RStudio session, or of the output of your script files.

R script files are text files which contain a series of commands. You can tell R to execute (run) these script files, and they will essentially issue these commands to the R console as if you were typing them, one after another.

To create an R script, go to File -> New File -> R Script. This will create an empty script file that you can begin editing using the R editor. When you are done editing a script file, you can save it using File -> Save (or Ctrl-S). Save this script file as "studio1.R"

It's a good idea to open your script file with some header information about what it does. This can be done using *comments*. Comments are pieces of text that begin with a "#" symbol, and are ignored by R when executing a script. For example, a good header to use might be:

```
• #####
• # R script:      studio1.R
• # Project:      Studio 5
• #
• # Date:         19/07/2017
• # Author:       D. Schmidt
• #
• # Purpose:      Introductory script for Studio 1
• #####
```

Opening headers can be as detailed as required, and in general more information is better. Comments should be used *throughout* a script file to annotate the script and make it clear, both to you, and to others what the script is doing. Whitespace (lines, extra spaces) is also ignored by R when running a script, so you can, and *should*, use this liberally to improve readability.

To execute your script file, from the command console type  
`source("studio1.R")`

This runs your script by telling R to treat the contents of your script file as if they were literally being typed into the command console. You can also issue this command by clicking the “source” button in the top-right of your editor.

For the rest of this studio, and all future studios, you should use the script file editor to record the commands that you use.

### 3. Accessing data

Once we have our dataset in R, we will usually want to explore the data to ensure that the data has been imported correctly, and that we understand the dataset that we are using.

- i. To get a summary of the structure of your dataset, type `str(chol_bmie)`. What does this tell you about each of the variables in the dataset?
- ii. Get a summary of only the **age**, height (**hgt**) and weight (**wgt**) variables in the dataset `str(chol_bmie[,c("age", "hgt", "wgt")])`

This line demonstrates a very important aspect of R, as it shows you how to access particular columns (variables) of your dataset. You can use as many variable names in quotes inside the `c()` function (called the “combine” function) to access as many of the variables as you like. If you omit the `c()` function, then all data columns will be selected, e.g., `str(chol_bmie[,])` is the same as `str(chol_bmie)`. Try them both to confirm.

The `c()` function takes all of its arguments and collects them into a single list. You can also access variables by using the column number in place of the column name inside the call to `c()`. For example:

```
str( chol_bmie[,c("age", "hgt", "wgt")] )
```

is the same as

```
str( chol_bmie[,c(3,4,5)] )
```

in our dataset, as variable **age** is column number 3, variable **hgt** is column number 4, and variable **wgt** is column number 5.

R also lets you use the shorthand notation A:B to specify all the numbers from A to B, e.g.,

```
str( chol_bmie[,c(3:5)] )
```

is the same as the above two commands. Verify this.

- iii. To display just the names of the variables in the dataset we can use the command  
`names( chol_bmie )`

You can modify the names of your variables by assigning the new name for the specific variable you want to rename, e.g.,

```
names( chol_bmie )[3] <- "Age"
```

will rename the **age** variable to have the new name **Age**.

You can rename multiple variables by using the combine function `c()`, e.g.,

```
names( chol_bmie )[c(4,5,6,8)] <- c("Hgt", "Wgt", "BMI",  
"Smoke")
```

- iv. To display some (or all) of the data in the results window, we simply specify our data set as above, with the required columns selected. Typing

```
chol_bmie[, c("Age", "Hgt", "Wgt")]
```

will display the **Age**, **Hgt** and **Wgt** variables from our dataset. In addition to controlling the columns that are selected, we can also control the *rows* that we want. One way to do this is to specify the particular row numbers that we want. Typing

```
chol_bmie[1:5, c("Age", "Hgt", "Wgt")]
```

will select (and display) the first 5 rows of the **Age**, **Hgt** and **Wgt** columns. How would you select rows 50 through 60 of the variables **Sex** and **bmi**?

- v. One way to summarise variables in R is to use the `summary()` function. Summarise the **Age**, **Hgt** and **Wgt** variables by typing

```
summary( chol_bmie[,c("Age", "Hgt", "Wgt")] )
```

What does the output from summarise tell you? Now use `summary()` to summarize the entire dataset

- vi. Summarise the age, weight and height of the participants by Sex. We do this by putting a logical expression in place of the rows to use. This will then select those rows for which the logical expression is true from the specified columns. To do this, type:

```
summary( chol_bmie[ chol_bmie$Sex == "male"
,c("Age", "Hgt", "Wgt")] )
```

```
summary( chol_bmie[ chol_bmie$Sex == "female"
,c("Age", "Hgt", "Wgt")] )
```

These lines of code also introduce another important way of accessing columns of data. If you are only interested in a *single* column, then the notation `dataset$varname` can be used to access variable “varname” in the data set “dataset”.

You can use logical expressions to access subsets of rows. See if you can work out how to summarise **BMI** and **Smoke** for people who have a total cholesterol value of less than 5, and those who have a total cholesterol value greater than or equal to 5.

- vii. The `summary()` function can also be used to summarise individual categorical variables, as seen above. However, the `table()` function can be used to produce basic cross-tabulation of several categorical variables. To do this, call `table()` on your dataset, selecting out only **Sex** and **bmi\_WHO** variables. What does the output tell you?

An important thing to note is that the `summary()`, `table()` and `str()` functions we have used are just one way of summarising and dealing with the data. There are a large number of extra R packages available that provide a whole range of extra functionality and alternatives to these built in functions. This is one of the strengths of R – for almost any task you can think of there are likely two or more packages out there that offer alternative ways to approach the task. We will deal with extra packages in later practical sessions.

#### 4. Flow control and functions

It is important that you familiarise yourself with the flow control commands used in R. The important flow control commands implemented in R are for loops, while loops and “if” statements. Use the command

```
help("for")
```

to bring up the R documentation regarding these commands.

When executing a loop or a function, R suppresses the output from the functions you are calling. The `cat()` function can be used to display text and numbers to the console. The special symbol `\n` can be used in a string to force a newline when printing using `cat()`, i.e.,

```
i = 45
cat("The value of i is:", i, "\n")
```

- i. Implement a loop that prints out all the odd numbers from 1 to 15.

The other key programming tool required to use R effectively are user-defined functions. The `function()` function is used to define a function; the `return()` function is used to return values from the function. For example

```
mydiv <- function(a, b)
{
  return(a/b)
}

mydiv(10, 3)
```

The important thing to note with R is that the function will not be defined (and usable) until you have executed the code that defines the function. If this is in a script file, you can simply run it using `source`. Redefining a function will overwrite the previous function, so there is no need to remove it before re-running your script if you are updating your function.

- ii. Write a function that takes an argument  $n$ , calculates the factorial of  $n$ , and returns it.

Returning multiple return values can be done in R using R lists, or R “structures”. These objects are placeholders for multiple fields of data. To create a list use the `list()` function, i.e.,

```
retval = list()
```

Then, once the list has been created you can access fields in the list (or add new fields) using the “\$” operator; for example:

```
retval$a = 1
retval$b = "you can store strings too"
retval$c = list()
retval$c$g = "You can store lists inside lists"

retval
```

- iii. Write a function that takes a vector of numbers and returns the minimum and maximum number in the vector.  
*(hint: the `length()` function lets you find the length of a vector).*  
*(hint: remember to return multiple return values using a list as described above).*

## 5. Further reading

You should download and read the documents [R0001.pdf](#) through to [R0007.pdf](#) from Moodle before Studio 2. These offer more introductory information on how to use R and will serve as a useful reference material. The file [Basic commands in R.R](#) is available on Moodle and summarises a number of useful R commands.

## 6. Basic descriptive statistics in R

We will now calculate some basic descriptive statistics in R.

- i. Load the in-built tooth growth dataset using

```
data("ToothGrowth")
```
- ii. Identify and calculate the appropriate statistics for central tendency and for dispersion for the three variables.
- iii. Compute a box plot and histogram for the **len** variable. The functions `boxplot()` and `hist()` can be used to produce these plots. Use the `help()` function to learn how to use `boxplot()` and `hist()`.
- iv. Calculate the 5<sup>th</sup>, 25<sup>th</sup>, 50<sup>th</sup>, 75<sup>th</sup> and 95<sup>th</sup> quantiles of the **len** variable using the `quantile()` function. (*hint: the combine function `c()` can be used to build vectors of numbers, for example, for the `probs` parameter of the `quantile()` function*).
- v. Scatterplot and calculate the correlation coefficient between the **dose** and **len** variables:

```
plot(ToothGrowth$len, ToothGrowth$dose)
cor(ToothGrowth$len, ToothGrowth$dose)
```

## 7. Categorical data

In this section we will explore some descriptive statistics when variables are categorical.

- i. Copy the mushroom dataset Mushroom.csv from the Moodle page and load it into R.

```
mush <- read.csv("Mushroom.csv")
```
- ii. Create a pie chart for each variable to show the relative frequencies of the values.  
Example:

```
tab <- table(mush$cap.shape)
pie(tab)
```



- iii. Cross-tabulate the `mush$class` variable against each of the other variables. Which variable is most strongly correlated with `class`?

The following code fragment shows how to cross-tabulate `class` with `cap.shape`:

```
table(mush$class, mush$cap.shape)
```

## 8. More descriptive statistics for numeric variables

Load the `wine.csv` dataset. This dataset contains measurements of various quantities related to wines, along with a score (**quality**) determining the overall quality of the wine. Clearly, it is of interest to winemakers to know which attributes of a wine contribute to quality, and this dataset is an excellent example of the wide range of applicability of data science concepts.

We will learn more advanced procedures to model the relationship between the wine attributes and wine quality in the coming weeks, but for now we can measure the strength of association using correlation coefficient.

- i. Write a loop that computes the correlation coefficient between each of the wine attributes and the wine quality variable. Your loop should display the information to the console in the form

```
Correlation between fixed.acidity and quality = -0.1136628
```

for each of the variables. (*hint: you can use the command `names(wine)[i]` to access the name of the *i*-th variable in a dataset*).

- ii. Which of the attributes is most strongly associated? Which ones appear to have no association?