Dr Isabella Gollini
Jose Camarena Brenes

Probability and Statistics

In this `R` lab you will:

- Learn how to install `R` and RStudio.
- Learn how to install and load new `R` packages.
- Access datasets available in `R` packages.
- Write/load data to/from a file.
- Calculate the most commonly used summary statistics.
- Produce boxplots.
- Produce and customise histograms.

1 Introduction to `R`

Skip Sections 1.1 and 1.2 if you are working on a university computer.

1.1 Install `R`

In order to install `R` follow the following commands:

1. go to `http://www.r-project.org/`
2. click `download CRAN` in the left bar
3. choose a download site
4. choose your target operating system
5. click base
6. choose Download R 3.2.2 and
7. choose default answers for all questions

1.2 Install RStudio

I suggest to use RStudio which is an easy-to-use free interface for `R`. More info: `www.rstudio.org`.

In order to install RStudio follow the following commands:

1. go to `http://www.rstudio.org/`
2. click `Products - RStudio`
3. click `RStudio Desktop`
4. click `Download RStudio Desktop`
5. click `Recommended For Your System`
6. choose default answers for all questions

1.3 Starting `R`

In order to start `R`, you can either click on the `R` icon or use RStudio.

Depending on the operating system the layout will change a little bit, but in any case it will open the *Console* window, that always contains something along the lines of (the first few lines change depending on the version of `R` and operating system in use):

```
R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

1.4 Important symbols and useful commands

`>` When the symbol `>` appears at the start of a line is called *prompt*. It means that `R` is ready to receive a command.

`+` When the symbol `+` appears at the start of a line is called *continuation prompt*. It appears when the expression is written in multiple lines. If it appears inadvertently it is possible to stop it either completing the command, or pressing ESC (Windows and Mac) or Ctrl-C (Unix).

`<-` The combination of the two symbols `<-` (with no space between the `<` and the `-`) is called *arrow*. The arrow is used to assign a value to a variable.

`#` The symbol `#` introduces a comment in `R`. The rest of the line is skipped.

Names are case-sensitive: `pluto`, `PLUTO` and `Pluto` do not refer to the same variable.

`getwd()` Working directory: `getwd()`.

`setwd` With the function `setwd` it is possible to set the working directory. The path in the working directory must be specified. This is equivalent of doing:

In `R` for Windows: File - Change dir...
In `R` for Mac: Misc - Change Working Directory
In RStudio: Session - Set Working Directory - Choose directory...

and then browse for the folder you want to set as working directory

`list.files()` `list.files()` shows all the files in the working directory

`file.choose()` `file.choose()` allows you to choose a file interactively. It is useful if you want to read the path to a file (and avoid typing mistakes!). It can also be used inside a function: e.g. `read.table(file.choose())`

1.5 `R` Library

A library in `R` is a collection of function and/or datasets. When you use a library for the first time, you have to install the package. You can do it by typing: `install.packages("...")` (where `...` is the name of the package you want to install), or clicking on the following:

`R` for Windows: packages - install package(s)
`R` for Mac: Packages & Data - Package Installer - Get List
RStudio: Tools - install packages

and select the package you want to install.

1.6 Where to work and what to save!

In order to be able to save time in case of mistakes or if we want to be able to reproduce the analysis, it is important not to write the command directly on the console.

It is possible to do copy-and-paste from any text editor and save the `.txt` file. An alternative to the text editor is the *Script Editor* window in `R` and RStudio. This is very useful as it also gives suggestions while writing the code. Also in this case you have to copy-and-paste the code into the console, or to press CTRL + R on Windows or CMD + ENTER on Mac.

1.7 Objects

An object can be a single number, character, or word. If you want to define an object called `x` with the value `7`, you type:

```
x <- 7
```

and press enter. This simple statement asks `R` to assign the value `7` to an object with the name `x`. Notice that the code which you type appears on your screen in blue text, whereas any output `R` prints to the screen is in black coloured text.

To see the value of any object, type its name and press enter. To check the value of `x` simply type:

```
x
```

You can make objects whose values are letters instead of numbers. If you want an object called `y` with the value `"word"`, you would type:

```
y <- "word"
```

Notice that letters go inside quotation marks; numbers don't.

1.8 Operations and functions

Usually you want to perform an 'operation' on the objects you've created. Suppose you want to perform the following operation on the object `x`: create a new object called `z` which has a value `2` greater than `x`. To perform this operation you could simply type:

```
z <- x + 2
```

In other words you want to perform the operation where you sum the value of the object `x` and the number `2`. Equivalently, you could have typed:

```
z <- sum(x, 2)
```

Here, `sum` is an example of an `R` function which performs an operation. It tells `R` to perform the sum operation (i.e. add together) the things inside the brackets. `R` then gives the result of this the name `z`.

Type:

```
z
```

and press enter to display the value of `z`. Is the value displayed what you expected it to be?

You can find more information about the `sum` function by typing:

```
help(sum)
```

or

```
?sum
```

This help option works for any of `R`'s inbuilt functions. Most operations have intuitive names like sum;

> What does the function `prod` do?

1.9 Creating vectors

We aren't limited to working with one number or word at a time. Another type of object we will use a lot is a vector. A vector is simply a list of words or a list of numbers. If you want to create a vector called `v` containing the values:

1
3
6

we use a function called `c`. (The name of this function is short for "combine values into a vector".) This function combines a list of objects into one vector or a single column. Again, you can find information about it by typing `help(c)`.

To create `v` type:

```
v <- c(1, 3, 6)
```

That is, take the three numbers in brackets (notice that they are separated by commas), perform the `c` function on them (i.e. turn them into a vector) and save the result as an object called `v`.

To see the vector, just type its name and press enter.

> What does the command `sum(v)` do?

If you wanted another vector, called `v2`, to contain these numbers and also the values of `x` and `z` which you created earlier, you have two choices. The first is to include `x` and `z` in the brackets following the function name `c` and type out all the numbers again, like this:

```
v2 <- c(1, 3, 6, x, z)
```

To see the vector and make sure that worked type:

```
v2
```

The other option is to ask `R` to combine the objects `v`, `x` and `z` (that's one vector and two numbers) into a new vector:

```
v2 <- c(v, x, z)
```

You can combine two or more vectors that way too! Try combining the vectors you have just made, `v` and `v2`, into a new vector called `v3`.

```
v3 <- c(v, v2)
```

Notice that the `c` function always arranges things in the order you type them. So `c(1, 3, 6)` and `c(3, 6, 1)` ask `R` for different things.

What if you want a vector of words instead of numbers? The `c` function can do that as well. Type:

```
v4 <- c("one", "three", "six", "seven", "nine")
```

This is just the same as making a vector with numbers in it, except that the words in the brackets are in quotation marks.

To make sure that worked type:

```
v4
```

1.10 Creating data frames

The third type of object we will use is called a data frame. You can think of a data frame as several (column) vectors put together, side by side, so that a data frame is like a matrix or a table of data. The function to do this is called `data.frame`.

To make a new object (a data frame) called `data1` which contains the vectors `v2` and `v4` as columns type:

```
data1 <- data.frame(v2, v4)
```

To see what it looks like type

```
data1
```

Notice that when `R` displays the data frame, the columns are labelled with the names of the original vectors, and the rows are numbered.

As before, you can find more information on this function by typing:

```
help(data.frame)
```

You may need to access particular entries in your data frame. Data frames are dealt with in the following way: `data1[2,1]`, for example, denotes the entry in the second row and first column of the data frame called `data1`. (Notice that square brackets are used here.) To see what value is in the second row and first column of your data frame type:

```
data1[2,1]
```

To display the entry in the first row and second column type:

```
data1[1,2]
```

To display the second row only, type:

```
data1[2,]
```

To display the second column only, type:

```
data1[,2]
```

2 Illustrative example: exam marks

Now we have some basic functions to work with, let's deal with an example!

A teacher has the following data available on five students (both exams are marked out of 100):

Name	Exam 1 Mark	Exam 2 Mark
John	92	82
Anne	54	96
Terry	98	60
Fred	62	55
Maria	79	72

We will begin by making a table (i.e. a data frame) in `R` of this data. To do that, we first create a vector in `R` of each of the three variables, using the `c` function.

Remember to write all the commands into the Editor, and regularly save the `R` script while you are working!

First let's create a vector which contains the names of the five students. At the prompt (remember, the prompt is the "greater than sign" `>` which starts every line in `R`), type:

```
studentname <- c("John", "Anne", "Terry", "Fred", "Maria")
```

Next, create a list of the Exam 1 marks:

```
exam1 <- c(92, 54, 98, 62, 79)
```

Because these are numbers, there are no quotation marks. Make a third list of numbers called `exam2` containing the marks for Exam 2.

(Notice that we never put spaces in the names of objects!)

What code creates the list of `exam2` marks?

Now, we want to create a single table (or data frame) with all this information in it. Recall that the `data.frame` function does that! Type:

```
results <- data.frame(studentname, exam1, exam2)
```

Next, the teacher wants to make a new column in the table which records each student's average mark for the two exams.

When an object is a column within a particular data frame, we refer to it by the name of the data frame, followed by a dollar sign `$`, followed by the name of the column. So

```
results$exam1
```

will display the column called `exam1` in the data frame called `results`. We're going to use this to make a new column in the data frame.

```
results$avg <- (results$exam1 + results$exam2) / 2
```

This creates a new object, a column called `avg` in the data frame called `results`. Each number in this new column is the average of the two numbers in the `exam1` and `exam2` columns of the corresponding row in the data frame called `results`.

Notice that the division symbol is `/` (to multiply use `*`).

2.1 The `cut` function

Next, the teacher wants to assign letter grades to the students. The bands are as follows:

< 50	D
50 - 69	C
70 - 84	B
85 - 100	A

The function we can use to convert the average grades to letter grades is called `cut`.

This function creates a new column according to three parameters:

1. a column of values to be "recoded" (in this case the students' average scores)
2. a column listing the endpoints of the bands (in this case, 0, 49, 69, 84 and 100), and
3. a column listing the new letter values (D, C, B and A).

The first thing we have to do is make the second two columns listed above. Remember, we use the `c` function to do that.

```
bands <- c(0, 49, 69, 84, 100)
lettergrades <- c("D", "C", "B", "A")
```

Now we are ready to use the `cut` function:

```
results$grade <- cut(results$avg,
                     breaks = bands,
                     labels = lettergrades)
```

Are you clear on what is being done here? We're making a new object called `results$grade`, which contains the first value in `lettergrades` if the number in `results$avg` is between the first and second numbers in `bands`. It contains the second value in `lettergrades` if the number in `results$avg` is between the second and third numbers in `bands`, and so on.

Print the data frame called `results` again to see if the `cut` function worked.

> Who has a better grade, Maria or Terry?

Lastly, suppose we want to create a table which only contains the students' names and letter grades. We therefore need to make a new data frame with only those two columns in it. To do this we could type:

```
results2 <- data.frame(results$studentname, results$grade)
```

and then view it on the screen:

```
results2
```

> Suppose you wanted to print out only the names and exam 1 marks of the students. What code would you use?

Now, we want to save the table to a `.txt` file.

The function `write.table` allows us to do that. It needs at least two elements: the table, and a file name:

```
write.table(results, file = "examsresults.txt", col.names = TRUE)
```

The option `col.names = TRUE` allows to save in the first row of the file the variable names (in this case `studentname`, `exam1`, `exam2`, `avg`, and `grade`).

If you go to your working directory you can find the new file `examsresults.txt`. Type `getwd()` to get the filepath to the working directory. If you want to save the file into another location you have to specify the file path: `file = "E:/ProbAndStats/examsresults.txt"`, change `E:/ProbAndStats/` to the correct path for your computer.

Note: always use `/` in the file path (Do not use \).

A similar command is available to read a table from a file:

```
res <- read.table(file = "examsresults.txt", header = TRUE)
res
```

In the option `header = TRUE` is needed to read the first line of the file as titles of the columns. If you type `?read.table` you can see other arguments that can be added in order to read files in different format.

Note that in this case you need to create a new object (`res`), and `res` is identical to `results`.

3 Example: 1974 car fuel consumption

This dataset is contained in `R base` package (installed by default in both `R` and RStudio). Type `?mtcars` in order to access the description of the dataset.

```
data(mtcars)
mtcars
```

We focus on the variable `mpg` that is the car fuel consumption.

```
consumption <- mtcars$mpg
consumption
```

Obtain the default descriptive statistics that `R` provides:

```
summary(consumption)
```

or the select the ones you want:

```
min(consumption)
max(consumption)
mean(consumption)
median(consumption)
quantile(consumption, p = c(0.25, 0.75))
```

We can also calculate the variance:

```
var(consumption)
```

What was the interquartile range?

Check that your answer corresponds to the one given by the function `IQR` (Use `?IQR` if you wonder what it is . . .)

```
IQR(consumption)
```

Obtain a boxplot of the `consumption` data, including the title "car fuel consumption - Miles / (US) gallon".

```
boxplot(consumption,
  main = "car fuel consumption - Miles / (US) gallon")
```

Obtain a histogram of the `consumption` data, including the title "car fuel consumption - Miles / (US) gallon".

```
hist(consumption,
  main = "car fuel consumption - Miles / (US) gallon")
```

Obtain a new histogram of the `consumption` data, using class boundaries 10 to 34 in steps of 2.

```
cutpoint <- seq(from = 10, to = 34, by = 2)

hist(consumption,
  breaks = cutpoint,
  main = "car fuel consumption - Miles / (US) gallon")
```

Are there any outliers?

Is the distribution skewed? If so, positively or negatively?

4 Example: Prices of almost 54,000 round cut diamonds

The dataset `diamonds` is available in the `R` package called `ggplot2`.

Install and load the package `ggplot2`, and load the dataset:

```
library(ggplot2)
data(diamonds)
```

Type `?diamonds` in order to access the description of the dataset.

In case you have any problem downloading the package in the labs, you can download the data `diamonds.csv` from Moodle. If you downloaded the data into the working directory you can read the dataset by using:

```
diamonds <- read.csv("diamonds.csv", header = TRUE)
```

otherwise you have to specify all the path to the file `diamonds.csv`, or use the function `file.choose()`.

You can also read the data directly from the website:
`https://raw.githubusercontent.com/hadley/ggplot2/master/data-raw/diamonds.csv`
and create the new object `diamonds` by using the website address as a path:

```
diamonds <- read.csv("https://raw.githubusercontent.com/hadley/ggplot2/master/data-raw/diamonds.csv",
                     header = TRUE)
```

The main difference of reading the dataset with respect to using the data in the package, is that now there isn't an help page with the description of the data.

But we can have a look at the data:

```
dim(diamonds)
head(diamonds)
tail(diamonds)
```

If you type `diamonds` it shows only the first 1000 rows, since our dataset is huge, it has 53940 rows!

We focus on the variable `price` that is the price in US dollars.

Calculate the following summary statistics for the variable `price`:

- Sample Minimum and Maximum
- Sample Mean
- Sample Standard Deviation
- Sample Median
- Sample Lower Quartile
- Sample Upper Quartile
- Sample Range
- Sample Interquartile Range

Produce the boxplot for the variable `price`. Are there any outliers?

Produce the histogram, with an appropriate main title, and appropriate class boundaries.

Is the distribution skewed? If so, positively or negatively?