MA26620: Practical 1

Introduction to R and RStudio: Exploring Data

1 Introduction: R and RStudio

This practical will introduce some of the facilities for summarising and displaying data sets in R, a powerful and widely used Statistical Package available on all Aberystwyth University computers. To access R in a particularly convenient way we will be using RStudio, which is also available on AU computers.

R and RStudio are free, open-source and cross-platform software, so if you wish you can also install them on your own computer; for example the Windows version can be downloaded from `https://cran.r-project.org/bin/windows/base` while RStudio can be downloaded from `https://www.rstudio.com`. I've also put an installation guide on the module webpages: `https://stats.vellender.com/installing-software.html`.

R is the name of the statistical programming language, while RStudio is an 'integrated development environment' (IDE) which acts as a frontend for R and makes interacting with R fairly straightforward.

2 Getting Started

Begin by logging on. The easiest way to launch RStudio is probably to click on the Windows icon at the bottom of your screen (Start) then type "RStudio" (all one word) and click on the resulting icon. Alternatively, click on the "All apps" button and RStudio should appear in the resulting alphabetical list. You may then be asked to choose which R installation you want to use, from an *extensive* list of options – oh wait, actually, just the one option. So you can't really go wrong at this stage - just click 'OK'!

RStudio will open, initially showing three panes. Along the top are buttons and menus; below these sits a toolbar. Hover over the buttons to see what they do and check out the menus but don't worry if some of the commands seem obscure at present. The left pane is called the R-console. We can directly type in commands in the R-console and get numerical results.

3 First steps in R: the overgrown calculator

At the > prompt, type in 2+3 and press the Enter key. Similarly try the following (again type at the > prompts and press Enter after each):

```
(5-12)*-3      5/3-1      5^4      (7-3)^-2
```

R also has inbuilt functions so try

```
sqrt(2)      pi      exp(1)      log(exp(2))      sin(2*pi)
```

(Note the rounding error!)

3.1 Warnings and errors

Let's briefly become anarchists and try to break RStudio (though please don't try too hard!). Frequently you'll forget a bracket in a command or similar, and so it's good to understand what R looks like when you do this. Type `sqrt(2` without closing the brackets and press Enter. You'll find that you actually have to delete the close-bracket to make this error as RStudio doesn't like anarchists and types a '`)`' for you every time you type '`(`' to try to stop you from going astray. You'll see that R starts a new line that starts with a `+` rather than the usual `>`. This is R's way of telling you it doesn't think you've finished your command. Typing a single close-bracket and pressing Enter will tell you what the square root of 2 is and will let you get back to normal.

The main message of the above is that if you get the `+` symbol at the command prompt, it's normally RStudio's way of telling you you've missed a bracket somewhere. Normally you can get rid of it by typing `)` and then Enter, possibly a few times.

Before we move on from trying to break things, let's have a quick look at what happens if we make other mistakes. Try `squareroot(2)` and note the error message. How about `Sqrt(2)`? And `sqrt(-2)`?

3.2 Assigning variables

Right, let's go back to being well behaved citizens (your fresher days are behind you) and stop trying to break things.

Even on a calculator you will quickly need some way to store intermediate results so you don't have to type them in over and over again. R, like other computer languages has *symbolic variables*: names that can be used to represent values. To assign the value 42, say, to the variable `x`, you can enter `x <- 42`. The left-pointing arrow here is just the 'less than' symbol followed by a hyphen. `42 -> x` will also work; in both cases note that the 'arrow' points to the side of the assignment that the name is on (in this case `x`).

There is no immediately visible result, but from now on, `x` has the value of 42 and can be used in subsequent expressions. To see the value of `x`, simply type `x` at a command prompt and press Enter. We can also do calculations, e.g. try `x + x`. These can be given a name too, so try for example `y <- 3*x+2` and then type `y` to see the result (pressing Enter after each)[1].

We were not exactly groundbreakingly original with our choice of variable names (`x` and `y`). In general we have freedom to assign nearly whatever name we like, built from letters, digits and the dot (full-stop) symbol. However, names mustn't start with a digit, nor a period followed by a digit. It's a bad idea to call something the same as a function, e.g. don't give something the name `sqrt`. Names are case sensitive so we could have variables called `WT` and `wt` which were different. For obvious reasons of near-inevitable confusion, it's very rarely wise to do so. In fact, it's a good idea to have some convention about when you will use capital letters, for instance keep the first word of a variable name in lowercase but subsequently capitalise each new word, e.g. `dolphinWeight` (programmers call this *camel case*), or maybe always use only lowercase. However, this is entirely your choice and not something that R insists upon.

3.3 Removing variables

Suppose we don't need `x` and `y` any more. We can remove them using the `rm` command:

```
rm(x) #this removes x
rm(y) #this removes y
```

[1]A note for those familiar with other programming languages, e.g. people taking MA25220 Numerical Analysis where you'll get good at using Python: you might find this syntax for assigning a value to a name unusual (most programming languages just use `=` instead of `<-`. In many cases, you can use `=` in R to assign a value, but there are some differences between `<-` and `=`, plus people who use R lots will be much more familiar with `<-`, so it's safest to get into the habit of using the left-pointed arrow for assignment.

We can use the # (octothorpe symbol…a much better name than 'hashtag'!) to add comments which don't interfere with the function. R will ignore anything on a line that is typed after the #, so don't comment in the middle of a command! Use this whenever you feel a reminder might be useful in future, since comments are easier to read/understand than lines of code.

In these practicals, you don't need to type the comments written on the worksheets (i.e. everything after #) into R if you don't want to, but do read them and understand what they're saying.

4 Working smarter

So far we've been running each command one-at-a-time in the Console window. You can perhaps imagine that if you were dealing with lots of commands, say several lines of code, it might be easier to be able to play around with them in a notepad, and then run them from there.

Thankfully, this is straightforward to do in RStudio and is usually a smarter way of working than directly running commands in the console.

Let's have a play. Click File > New File > R Script (or equivalently click the leftmost icon in the bar of icons and choose R Script). A pane appears, usually in the top left. This is a little notepad into which you can essentially draft your commands.

Try typing something like the following four lines into this new pane:

```
a <- 1
b <- 2
c <- a+b
c
```

Unlike the Console, when you press Enter between each line, nothing happens. The commands do not run. When we're ready to actually run them, select all four lines (eg by clicking and dragging) and click the "Run" button at the top of that pane. Equivalently, you can press Ctrl+Enter, which does the same as clicking Run. You'll see that this runs the lines you have highlighted in the console; the commands and their outputs appear in the Console.

It's a good habit to get into to always use this top-left R Script notepad, and then highlighting and clicking Run (or Ctrl+Enter) - it will make life easier than using just the console as we move through our RStudio learning journey. I therefore suggest you use this pane for the rest of the practical!

5 Vectorised arithmetic

We can't do much statistical analysis on single numbers! Rather, we'll look at data from a group of patients for example. A data vector is simply a list of numbers, and can be constructed like this:

```
weight <- c(60, 72, 57, 90, 95, 72)
```

After entering this command, type `weight` and R will return the vector of weights you've just inputted. The important thing to remember here is that the command `c(1,2,3,...)` joins numbers together into vectors (where `c` stands for 'concatenate' (meaning 'join')).

We can conduct calculations with vectors just like ordinary numbers, as long as they are of the same length. Suppose we also have the heights (m) that correspond to the weights (kg) we've just inputted:

```
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
```

The body mass index (BMI) is defined for each person as the weight in kg divided by the square of the height in meters. This could be calculated as follows:

```
bmi <- weight/height^2
bmi
```

Notice that the operation is carried out elementwise (that is the first value of `bmi` is $60/1.75^2$ and so on, and that the ^ operator is used for raising a value to a power.

Suppose we want to determine the mean weight, $\bar{x} = \sum x_i/n$:

```
sum(weight)
length(weight)
```

respectively give $\sum x_i$ and n (the latter being the length of the vector of weights, i.e. how many people there are in our sample). Thus $\bar{x}$ can be determined by

```
xbar <- sum(weight)/length(weight)
xbar
```

Of course, finding means is a very common thing to do, so handily R has a built-in `mean` function which is quicker than having to type the above commands. Compare your value of `xbar` to `mean(weight)`. Try also `sd(weight)`, `median(weight)` and `summary(weight)`.

6 Loading data from a file

Frequently the data we will use will be in a comma-separated file (.csv) such as could be exported from LibreOffice Calc, Excel or some other spreadsheet program. Such files are just a list of numbers, possibly with column headers, separated by commas.

Using your web browser of choice (Firefox/Chrome etc.), go to the MA26620 webpages. You can either do this through Blackboard (log on to MA26620's Blackboard and click the link) or go to `https://stats.vellender.com`. Either way, in the *Practical worksheets* area, you'll find a file called `cherrytrees.csv`. Download this into a suitably named folder (e.g. Applied Statistics) on your `M:` drive. Throughout these practicals, saving to the `M:` drive is the best idea as then you can access it in future weeks.

`cherrytrees.csv` contains information about 30 black cherry trees, namely their Diameter, Height, Volume. Each row refers to the dimensions of the trunk of one tree. Note that the diameter is measured in inches at 4.5 feet above ground level, height is in feet (12 inches) and volume in cubic feet. The data was gathered with a view to estimating the amount of wood in the trunk without, in future, having to cut down the tree.

We need to import the data into R. In the Environment pane (top right), click Import dataset $\rightarrow$ From Text (base). Find the folder with `cherrytrees.csv` and select it. You will be shown the form of the original Input File and the dataframe it will become in R. Notice that Heading is set as Yes (because in the original file the first row is the names of the columns) and Separator as Comma. Click on Import and notice that the top left pane now displays the data. Note also the structure of the data. Each tree has a row with 3 column entries giving its diameter, height and volume. All the columns are equal in length so it has a matrix-like structure which R calls a *dataframe*.

7 Descriptive statistics – the five number summary

Type the following command in the R console:

```
summary(cherrytrees$Volume)
```

The same information can be presented graphically in a Box and Whisker plot (which we will discuss in the next lecture) as follows:

```
boxplot(cherrytrees$Volume)
```

The plot will appear in the bottom right pane. This plot is a visual representation of the summary statistics we just obtained. The top and bottom lines of the 'box' respectively show the position of the upper and lower quartiles, while the line through the middle of the box shows the position of the median (NB: **NOT** the mean). The 'whiskers' (i.e. the lines protruding from either end of the box) show the range of the data, reaching down to the minimum value in the data, and up to the maximum. If, however, any values lie more than 1.5x the interquartile range (often abbreviated IQR, calculated by taking the difference between the upper and lower quartiles) from the quartile lines, they are deemed outliers and denoted by a dot. You should see signs of skewness in this data. What are they?

It can get a bit tedious retyping `cherrytrees$` to identify the dataframe, so the following command is very useful:

```
attach(cherrytrees) # From now on R will know where to look!
```

Now try

```
summary(log(Volume)) # log here means natural logarithm - not tree related!
boxplot(log(Volume)) # it's easier to type Volume than cherrytrees$Volume
```

Before moving on try

```
boxplot(Height, Diameter) # For a multiple boxplot, add extra variables.
```

7.1 Titles and axis labels

A good plot should have titles and axis labels. Let's make a well-labelled boxplot of the height data. Try the following commands:

```
boxplot(Height) # No labels so far
boxplot(Height, main="Height of Cherry Trees") # Adds a title
boxplot(Height, main="Height of Cherry Trees", ylab="Height (ft)")
                                               # adds vertical axis label
```

You can also change the colours of your labels and titles, for example:

```
boxplot(Height, main="Height of Cherry Trees", ylab="Height (ft)", col.main="blue",
  col.lab="red")
```

8 Scatterplots

When variables refer to the same object it makes good sense to plot them in a scatterplot. Try

```
plot(Diameter, Volume) # NB: plot(x,y) gives you y versus x
plot(Height, Volume)
plot(Volume) # For one variable you get a plot vs the index; rarely useful
```

We now create a new variable.

```
treeFormula <- pi*((Diameter/24)^2)*Height #Remember the units?!
```

Think about what you have just calculated. Why might the collectors of this data find this useful? Now produce a scatterplot showing the volume of the trunk versus `treeFormula`. Add a title and give the axes more meaningful labels using `main`, `xlab` and `ylab` commands.

Click on Export → Copy Plot to Clipboard → Copy Plot. Checking the box for *Copy as Metafile* gives a better quality, more resizable plot than *Copy as Bitmap*. Paste this into a Word document, explain what it shows and write a comment on what you learn from it. **Save your Word document somewhere that you'll be able to access it in future (e.g. somewhere on your M: drive) as this may form part of what you hand in for the first assignment.**

9 Tidy up and finish

```
detach(cherrytrees) # So R doesn't keep looking in cherrytrees!
```

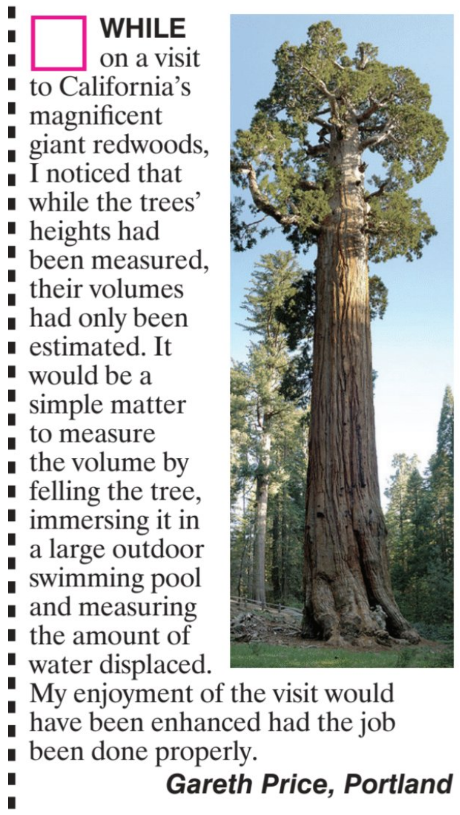
One of the things that new users can find a bit confusing about R/RStudio is how/what to save at the end of a session. RStudio doesn't have one big Save function that saves everything into one file. Instead, it allows you to save two files: one that contains the actual *data* (i.e. all your dataframes you've created that you can see listed in the Environment pane, like cherrytrees, weight, height) and one that contains the commands you've run (this is called the *command history*).

From these two files, it's easy to reconstruct anything you've done during a session... you'd just load the data and run the relevant command(s) from the command history file to recreate a plot or output or whatever you'd been doing.

Save these files in a suitably named folder on your M: drive by selecting the Environment tab and clicking the "Save" icon adjacent to "Import Dataset", calling your file prac1.RData for example, and then select the History tab and click on the Save button adjacent to "To Console", perhaps calling your file prac1.RHistory. Close RStudio and you're done until Practical 2.

Off-topic post-script

Spotted the following on Viz magazine's Twitter account – had the writer done this practical in the past?! Whoever said Statistics wasn't a rich source of comedy?



WHILE on a visit to California's magnificent giant redwoods, I noticed that while the trees' heights had been measured, their volumes had only been estimated. It would be a simple matter to measure the volume by felling the tree, immersing it in a large outdoor swimming pool and measuring the amount of water displaced. My enjoyment of the visit would have been enhanced had the job been done properly.

Gareth Price, Portland