STAT0023 Workshop 1: R revision and graphics

In this workshop we will revise some simple use of the R language and will explore some of its graphical capabilities. It is assumed that all students are familiar with the contents of the "Introduction to R" guide that has been posted on the course Moodle page (it can be found on the 'Course overview and useful materials' tab there).

1 Getting started

We will start by getting set up for this course in the cluster rooms, using the central UCL computing system. You may also want to use your own computers for self-study, or to do some of the assessed work for the course. In this case, you will need to set up a directory for your STAT0023 work (see below) on your own computer as well as on the UCL Filestore. You can do this in your own time.

If you want a 'walk-through' guide to the material in this section, look at the Lecturecast recording for the lecture of 13th January (the link is on the Moodle page, under the 'Week 1: R revision' tab): the relevant part of the lecture starts after around 30 minutes.

1.1 Setting up a directory

Whenever you use a computer, it is good practice to organise your files in such a way that you can find them again later. This is accomplished by keeping related groups of files together in *directories* (a 'directory' is the same as a 'folder' — a major software company confused everything a few years back by changing the vocabulary they used).

In the UCL cluster rooms, we will use R and SAS on the *UCL Desktop*. When you log in, your personal files are stored in your filestore on drive `N:`. You should keep your STAT0023 work in a separate directory on drive `N:` — for example, `N:\STAT0023` — so that you can find it easily later on. Your first task, therefore, is to log in and create this directory. **Do NOT proceed further until you have done this!** If you don't know how to do it, ask.[1]

You may want to create subdirectories to keep each week's work separate — for example `N:\STAT0023\Workshop1`, `N:\STAT0023\Workshop2` etc. This is entirely up to you: the important thing is that you have some system for organising your files and that you know where to find them.

[1] NB we assume throughout this course that you will let the workshop organiser or demonstrators know if you need help. This enables us to help you learn more effectively. Please don't be afraid to ask if you're stuck!

1.2 Moodle enrolment

Throughout the course, material will be made available to you via Moodle. All students who are registered for STAT0023 on Portico should automatically have access to the course Moodle page: if you cannot see it in your list of Moodle courses, contact the workshop organiser.

1.3 Files for this workshop

After creating your STAT0023 directory, you should download all of the material that you need for this week's workshop. Log on to Moodle, find the STAT0023 page, go to the 'Week 1: R revision' tab and download the files `Workshop1_Galapagos.r`, `Workshop1_Galapagos_Clean.r`, `galapagos.dat`, `Workshop1_Iris.r` and `Workshop1_Iris_Clean.r`. The best way to do this is probably by right-clicking on the relevant links: your web browser should then give you a menu with an option to save the files (e.g. in Firefox or Google Chrome, choose '`Save Link As`'; in Internet Explorer or Microsoft Edge, choose '`Save Target As`'; in , choose ' `). Make sure you save the files to the directory that you created in Section \ref{dirse` unsure.

Be careful to ensure that Windows doesn't append any unnecessary suffices (e.g. `.txt`) extensions to the filenames when you download them. If you follow the instructions above, you should be OK but if you use any other method then you may find yourself in trouble later!

1.4 Starting RStudio

We will run R from within RStudio, with which you should all be familiar (if you're not, it means that you haven't studied the "Introduction to R" document as instructed). To start RStudio on the *UCL Desktop*, type `RStudio` in the search box at the bottom left-hand corner of the screen, and then click on 'RStudio 1.2.1335' when it appears. It may take 20 seconds or so for the application to start.

1.5 Setting the working directory

Having started RStudio, you need to set the working directory to the place where you saved all of the files for this workshop. The recommended way to do this is via the RStudio `Session` menu at the top of the window: from here, choose `Set Working Directory` and then `Choose Directory`. This opens up a dialogue box from which you can navigate to the appropriate place and then click `Select Folder`.

After doing this, click the 'Files' tab in the lower right-hand pane of the RStudio window. You should see the files that you downloaded from the Moodle page there. If not, you've done

something wrong! Also, at this point it is worth checking that the file names are correct and haven't been altered during the download process — if they *have* been altered, you can click the box(es) next to the offending file(s) in the 'Files' tab and then use the `Rename` button to correct the mistake.

Now you're ready to start!

2 The Galapagos biodiversity data

In the lecture, we looked at analysis of biodiversity data from the Galapagos islands. The first exercise of this workshop is to work through the script that was used to carry out this analysis. This is the file `Workshop1_Galapagos.r`. Open it by clicking on it in the 'Files' tab of the lower right-hand pane of the RStudio window. It will display in the top left-hand pane.

The first 49 lines of the script are comments. They explain what the data represent. Read through them carefully, then take a look at the data file `galapagos.dat`, by clicking on it in the 'Files' tab. Like all the other files here, it's just a text file. Do **NOT** change the file in any way! Just take a quick look and make sure you understand how the contents of the file correspond to the description given in the R script. When you are satisfied with this, close the data file — you don't need to look at it any more.

You should now return to the R script and work through it slowly and carefully, one line at a time, at each stage making sure you understand exactly what the commands are doing. You might want to use the `Run` button in RStudio, to run each command once you understand what it's supposed to do and how it works. Then study the output of each command (if there is any) and make sure that you know how to interpret it. The script is very well commented, so you should be able to follow the logic as you work through it. Make a note of the commands that are used, and what they do. If you don't understand anything, ask.

If you work effectively, this exercise should take you between 30 and 45 minutes. At the end of it, you should understand the following:

- How to use the hash symbol # for commenting: this is needed to ensure that the code is clear and readable. Notice also how the code is spaced out to enhance readability. You will be expected to emulate this kind of style when you write your own code.
- How to use the '`<-`' symbol to assign the result of an operation to an object.
- How to read data from a file using `read.table()`.
- How a data frame (`species.data` in this case) is used to store collection of variables. The variables are all numeric or integer here but could also include character, logical or factor variables.
- How to find information about an R object e.g. using commands such as `names()`, `str()`, `summary()` and `class()`.

- How to use square brackets `[]` to extract parts of an object according to a logical condition (`big.island` in this example).
- How to use the dollar sign `$` to work with named components of an object.
- How to plot different types of R object, with control over labels and formatting.
- How to save graphics to files in different formats (`PDF`, `JPEG`, `PNG`, ...)

When you have finished this, press the `Source` button at the top of the RStudio script pane, to run all of the commands in a single batch. Do you notice a problem here?

2.1 Running commands in batch mode

The problem with `Source` above is that hardly any of the output appears. This is not very helpful. In fact, for output to appear when you use `Source`, you need to use `print` statements in the script. This is illustrated in the `Workshop1_Galapagos_Clean.r` which is a 'clean' version of the original script. Open this by clicking in the RStudio 'Files' tab, and take a quick look through it. Do **NOT** run the commands line by line!

As you read through the script, notice the use of the `sink` command to send the output to a results file called `Galapagos_Results.txt` (and the use of the command `sink()` at the end of the script to stop diverting the output to this results file — without this, you'd never see anything on the screen again during this RStudio session!). Notice also the use of commands `cat` and `print` throughout the script, whenever it is required to output any information. The `cat` command outputs exactly what is given to it: the `\n` symbol is needed whenever we want to move on to a new line of output. On the other hand, the `print` command tries to make some intelligent decisions about how to format the output. Usually, as here, a mixture of `cat` and `print` is needed to produce output that looks nice.

When you have read through this script, click the `Source` button. You will see nothing at all in the console window this time. However, if you look at your RStudio 'Files' tab you will see that you now have a file called `Galapagos_Results.txt` there (you may need to refresh the 'Files' tab to see this — there's a 'refresh' button in the top right-hand corner). Click on the results file to see what's in it, and notice how each of the `cat` and `print` statements has produced nicely formatted output.

In addition to the basic R commands you learned before, you should now understand the following:

- How to use `source()` and `sink()` to run an entire script and send output to a text file.
- How to use `cat()` and `print()` to format output and display it when using `source()`.

(a) (b)

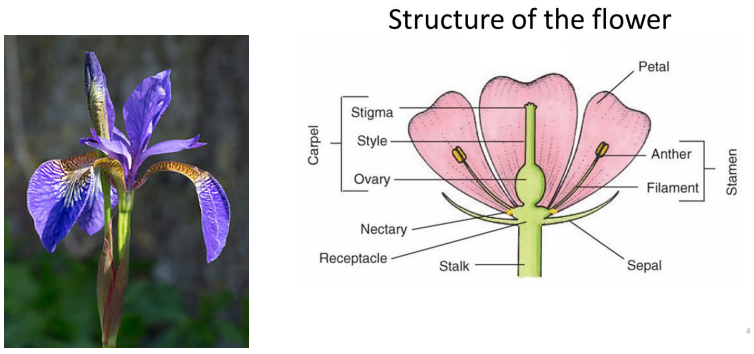

Figure 1: (a) A purple bearded iris (source: Wikipedia) (b) Diagram of a flower showing (among other things) the petals and sepals (http://images.slideplayer.com/18/5664385/slides/slide_4.jpg)

.

3 The Fisher-Anderson iris data

The second exercise introduces some basic statistical procedures and some more advanced graphics capabilities. It uses a classic dataset relating to measurements taken on 50 specimens each of three species of iris (a type of flower: see Figure 1(a)). The measurements are the widths and lengths, in centimetres, of the petals and sepals (see Figure 1(b)).

This exercise serves two purposes: first, to review some further capabilities of R, and second to illustrate how one might start to analyse a dataset using simple — and then slightly more sophisticated — exploratory tools. For this dataset, it may be of particular interest to examine potential differences between the species, and this has motivated some of the analyses in the script.

Among the further capabilities of R in this exercise is the use of add-on libraries, which allow you to use commands that aren't available in the basic R installation. The libraries used in the exercise are:

`ggplot2`: this provides an alternative to conventional R graphics for producing plots and so forth. It is popular in the Data Science community, although it is quite aggressive and (for example) redefines some of the commands in basic R — you may see a warning message about this when you load the library. It's nothing to worry about, but it illustrates a point about libraries: don't use them unless you need them!

`rgl`: this allows interactive data visualisation, including the ability to produce 3-dimensional graphs and spin them round to look at the data from different angles.

`RColorBrewer`: this provides a variety of carefully constructed colour scales that are useful for ensuring that plots are easy to read and interpret.

These libraries are not installed on your UCL Desktop system by default: the comments at the top of the script (see below) explain how to download and install them. Read the comments carefully, therefore!

The script for this analysis is called `Workshop1_Iris.r`. Open this in RStudio (you should know how to do this by now) and, once again, work through it line-by-line and make sure that you understand what's going on. Also try to understand the logic of the analysis: *why* is it sensible to perform this particular set of analyses and in this order? Remember the main aims of an exploratory analysis:

1. To gain a preliminary understanding of structure in a dataset;
2. To look for possible outliers or data quality problems;
3. To suggest some initial assumptions that may be reasonable as a starting point in subsequent modelling and analysis.

As well as using add-on libraries, this script introduces some additional commands, some of which are a bit more advanced than the ones in the Galapagos example. There are also some 'cunning tricks', of the type that experienced programmers will often use to write clean, elegant and efficient code. The `tapply()` command on line 111 of the script is one line that may not be entirely obvious, as is the command to open a graphics window on line 132; and the arguments to the `plot()` command on lines 266 and 267 will also require some perseverance if you want to understand them fully (the example towards the bottom of page 7 of the "Introduction to R" document may be helpful here).

Once again, when you have finished working through this script line-by-line, open the 'clean' version `Workshop1_Iris_Clean.r`, note the use of `print` and `cat` statements throughout, and use `source` to generate a nicely-formatted set of output.[2] **Note** that there are no `sink` commands in the `Workshop1_Iris_Clean.r` script, so if you want to divert the output to a file then you'll have to do it yourself. Don't forget the final `sink()` command when you've finished, though!

This exercise should take you about an hour. At the end of it, you should understand the following in addition to what you learned from the Galapagos example:

- How to make commands from 'add-on' package available to your R session, and how to use the '`package :: command`' syntax to ensure if necessary that you're using the command from the right package.
- How to use `round()` for finer control of numeric output from your script.

[2] If you're concentrating, you will also note that there is a print() statement enclosing the `ggplot() + ... + ...` command. Without this, the resulting plot doesn't appear when the script is source()d. This is a 'feature' of the ggplot2 library, along with some other add-on libraries for graphics in R. It isn't needed for any 'standard' R graphics (i.e. those produced without using commands from any add-on libraries).

- How to use `tapply()` to calculate summary statistics for subgroups of observations in a data frame.
- How to open a graphics window of a specific size, and to save your graphics subsequently to a file of the same size in order to avoid any distortion when saving.
- How to produce boxplots to compare the distributions of a variable between groups, with control over features like the colour of the boxes as well as axis labels and title.
- How to use a variety of other graphical displays, with fine control over plotting symbols and colours and including the ability to use different colours and symbols to distinguish different subgroups of observations.
- How to use graphics to inform a subsequent statistical analysis (e.g. by noticing that the boxplots show a roughly symmetric distribution so that assumptions of normality might be reasonable).
- How to carry out, and interpret the results of, standard two-sample tests for equality of mean and variance using `t.test()` and `var.test()`.

You have also seen an example of interactive graphics, which hopefully convinces you that the petal and sepal measurements are separated into almost completely distinct groups depending on the species. This information could be used by botanists to classify a plant of unknown species, based solely on these quantities that can be measured easily. The statistical technique required to develop such a classification rule is called *discriminant analysis*, but we won't cover this formally in the course. There is, of course, nothing to stop you from trying to find out whether R can carry out a discriminant analysis, by searching for 'discriminant' in the help system (remember the '`??`' command ...).

4 Moodle quizzes

At the end of each week's workshop materials, you will find one or more Moodle quizzes that are designed to help you discover whether you have understood everything that is expected of you. Some of the questions relate directly to material that has been provided directly to you; others require you to adapt some of the R commands that you have seen. You can attempt these quizzes as many times as you like — you'll get different questions every time, so you have plenty of opportunity to practise. **It is strongly recommended that you spend some time on these quizzes**, both during the workshop and in your own time afterwards: apart from anything else, each of the two assessments for the course will have Moodle quiz element, and the questions for these assessed quizzes will be all be taken from the weekly quizzes. This is an opportunity for you to get some free marks therefore — and to learn something in the process!

5 A final point

Although you have seen a lot of commands in this workshop, you should not expect us to give you every single command that you will ever need. R is a language just like any other and, when you learn a foreign language, you don't usually expect your teacher to tell you every single word! Rather, your teacher gives you a framework so that you can start to understand what other people are saying, and then you increase your vocabulary with experience. It's the same with a programming language: we will use the STAT0023 workshops to get you started and to give you some confidence, but then it's your responsibility to experiment a little for yourselves. For example, if you want to understand more about how the `ggplot2` graphics library works then you could experiment with the relevant code from the `Workshop1_Iris.r` script. The original code is as follows:

```
ggplot(data=iris, mapping=aes(x=Petal.Length, y=Sepal.Length) ) +
  geom_point() +
  facet_wrap(~ Species) +
  labs(x="Petal length (cm)", y="Sepal length (cm)",
       title="Sepal length versus petal length for specimens of three iris species") +
  theme(plot.title=element_text(hjust=0.5))
```

What do you think will be the effect of the following modifications to the above code? Try each one of them, and find out:

1. Remove (or comment out) the `facet_wrap` line.
2. Remove (or comment out) the `theme` line.
3. Remove (or comment out) the `labs` lines.
4. Add the argument `colour=Species` to the aesthetics in the original plot specification.
5. Both (1) and (4) above.
6. Add the argument `nrow=2` to the `facet_wrap` line.
7. Add another layer to the plot, defined by `geom_smooth(method="lm")` (you should be able to guess what `method="lm"` means, because you used the `lm` command in your analysis of the Galapagos data).
8. Add `theme_light()` to the plot commands. Does it make any difference whether you add this line before or after the existing `theme()` command?

Those are just a few ideas, to help you understand how to use `ggplot2`. You could try many other things as well, to learn more about the use of 'standard' R commands as well as `ggplot2`. Be creative! (within reason).