Chapter 2 - Getting started

1 Aim of this chapter

This is a first look at R. By the end of this chapter you should know:

- how to open R;
- how to get help;
- how to run commands (functions);
- how to use a script file;
- how to format code consistently and how to annotate it;
- the difference between .R files and .RData files;
- about the Workspace/Global Environment;
- how to download and install packages.

We'll actually use RStudio, a nicer interface than the standard R one.

2 Starting and quitting RStudio

2.1 Starting from University terminals

RStudio may already be installed on the machine you use. If so click the button at the bottom left of the screen and then scroll down to RStudio.

If RStudio is not yet installed, you will need to double-click the *Software Center* icon on the Managed Desktop, select RStudio from the list and click the install button. When the installation is complete, close the *Software Center* window and run RStudio as above.

If RStudio is not available in the list that comes up when you start the *Software Center*, then it is probably installed already; you can check by looking at the *Installed Software* tab within the *Software Center*.

2.2 Installing and starting RStudio on your own computer

You need to install R version 2.11.1 or higher before installing RStudio (if you are new to R then just install the latest version, which will be the default choice). Once installed, double click on the RStudio icon, or select *Start* → *All Programs* → RStudio.

2.3 Quitting RStudio

When you are ready to leave R click on the usual × in the top right hand corner.

3 Using RStudio

3.1 Entering code (the R Console window)

From now on when we refer to R we really mean RStudio. When you open R the Console window will appear in the bottom left of the screen. Commands can be typed, pasted or edited here. Press *Enter* or *Return* to execute a command. For example, type the following in the Console window and press return

```
x <- 2
```

This command just creates an object called `x` taking a value 2. You can read `<-` as *takes the value*. Objects get overwritten if you assign them a new value. For example

```
num <- 2
num <- 6
num
```

```
## [1] 6
```

creates an object called `num` and gives it a value 2, then changes the value of `num` to 6 and then prints the `num` object to the screen (ignore the [1] for the moment).

The command prompt symbol `>` is displayed in the Console window whenever R is waiting for instructions. If R thinks that a command you have typed and returned is incomplete, the prompt will change from `>` to `+`. You can choose to complete the command, or press *Esc* to start again. For example suppose you type the following and press return too early by mistake

```
y <- x -
```

You'll see the `+` symbol which means R is waiting for you to complete the code. Either complete it by typing a number (say 2), or press *Esc* to start again.

3.2 Re-running code

When typing code in the Console window you often need to re-run the same command, possibly after editing. There are a couple of ways to do this:

- in the Console window you can use the up-arrow ↑ on the keyboard to scroll through past commands;
- alternatively find the command in the *History* tab (top right window), click on it, then click *To Console*.

In the Console window enter the following commands (you need to press Enter after each line)

```
x <- 2
y <- 6
z <- 8
```

> Try this
>
> - Modify the first command to `x <- 78` using the ↑ key.
> - Modify the second command to `y <- 9` using the *History* tab.

When you leave R the commands that you have issued during the session will be recorded in a file and will available in the *History* tab.

3.3 Other windows in RStudio

R also provides two other windows: the *Environment*/*History* window and the *Files*/*Plots*/*Packages*/*Help*/*Viewer* window. There are 2 tabs in the *Environment*/*History* window:

- the *Environment* tab shows data/objects you have stored in memory and the names of any functions you have created. You should see the objects `x`, `y`, `z` and `num` that you created and the values they take.
- the *History* tab, just mentioned, shows any previous commands that you have run.

There are 5 tabs in the *Files*/*Plots*/*Packages*/*Help*/*Viewer* window:

- the *Files* tab shows your files and folder structure
- the *Plots* tab shows any plots you have created which can then be exported in various formats (pdf, png etc) for incorporation into other documents
- the *Packages* tab shows packages available to you (you can install other packages from here)
- the *Help* tab shows any packages/commands you have requested more information about
- the *Viewer* tab may not show and you can ignore it for the moment

4 Creating and running script files

The Console window is fine for simple coding tasks but isn't for anything more substantial. The code can be retrieved from the *History* tab but really we want to be able to keep all our code together in a single file.

If you are just writing R code, the usual way of working is to create a *script* file which contains all your code for a particular task. To open a new script window, select *File* → *New File* → *R Script*. You type commands in the window and run them by using the buttons at the top right of the window.

- The *Run* button runs the line containing the cursor (the cursor can be anywhere on the line)
- The cursor then moves onto the next line so that repeated use of *Run* will run a script one line at a time
- If you want to run the same line of code repeatedly use the button to the right of *Run*.
- You can run multiple lines of code by highlighting all the code you want to run and using the *Run* button
- Similarly you can run part of a line by highlighting it and using *Run*.
- You can run the whole script file using the *Source* button.

> Try this:
>
> - Open a new script file.
> - Create an object called z taking the value 7.
> - Run the line of code from the script file using the *Run* button.
> - Check it appears in the Global Environment (Workspace).
> - Change the code in the script file so that z now takes the value 10.
> - Re-run the code.
> - Check it has updated the value of z in the Workspace.

4.1 Keyboard shortcut (run code in a script file)

In a script file you can run the line of code containing the cursor by pressing *Ctrl* and *Enter* simultaneously. From now on we abbreviate this action to *Ctrl* + *Enter*.

4.2 Opening and saving script files (.R files)

Commands in a script may be saved in a file by selecting *Save* from the *File* menu. Similarly an existing script file may be opened using *File* → *Open File*. Note that R script files are automatically saved with file extension .R (You don't need to add it to the file name).

4.3 Annotating R code in a script file

It is good practice to include comments explaining what your R code does. Suppose you write some complicated code and then come back to it several months later. If you didn't annotate your code you will find it difficult to understand what the code does, which just wastes your time. Annotation in R is simple. Including the character `#` in a line tells R to ignore anything on the line after the `#`, so comments preceded by `#` can be added anywhere without affecting processing.

```
###  Some code to calculate y for a given x value
x <- 8 # Declare the value of x
y <- (x - 1) * (x + 3) # Declare a quadratic relationship between y and x
y # Find the value of y
```

```
## [1] 77
```

> Try this:
>
> Annotate the code you created in your script file. Create a folder for this module on your U drive and call it MAS6002 or MAS6024 or MAS468. Create a subfolder called 'Chapter 2' and save your script file in this folder (give it a meaningful name).

5 Saving objects, functions and data (.RData files)

We have seen how to save R script files (code files) but how do we save objects in the Workspace that we have created?

5.1 The Working Directory

When you save objects in R they are stored in R data files. The default location where these data files are written to and read from is called the *Working Directory*. This is just some directory in your file structure.

You can see which directory is the Working Directory by typing *getwd()*. You can change the location of the Working Directory by selecting *Session* → *Set Working Directory* → *Choose Directory*.

5.2 Saving objects in the Working Directory

When you quit R a pop-up box appears offering you the opportunity to `Save the work space image'. If you do save then the objects in your workspace (objects, functions and data) will be saved in an *.RData* file. R will automatically give the Workspace files a *.RData* extension so you don't need to.

You can also save the Workspace at any time in the session using *Session* → *Save Workspace As*. The *.RData* file will be saved in your Working Directory.

5.3 Saving objects elsewhere

To save the Workspace somewhere other than the Working Directory you need to add the path of the directory you want to save the Workspace in.

```
save(x, y, file = "u:/Project1/mydat.Rdata")
```

would save the `x` and `y` objects in a Workspace called *mydat.Rdata* in the *Project1* folder on the U drive.

5.4 Loading *.RData* files

If R **is not open** and you double click on a file with a *.RData* extension you open up a new R session. The objects in the Workspace will then automatically be put in Global Environment and the working directory will be set to the path of the *.RData* file.

If R **is open** and you double click on a file with a *.RData* extension the objects from this *.RData* file can be added to your current Workspace.

6 A typical R session

A typical session will involve

- thinking about where you are going to store your various R files (R script files and .RData files) and creating directories as necessary
- opening R (possibly by double clicking on an existing *.RData* file)
- opening (or creating) an *.R* script file
- editing it and running the code
- annotating the code so you will understand it next time
- saving the *.R* script file
- saving the *.RData* file (if you have created new objects you want to save)
- quitting *R*

7 Exercises A

1.
   - The `Turkeys.RDdata` workspace is in the *Using R* folder on *Blackboard*. Download this file to the *Chapter 2* folder on your U: drive (right click and *Save link As*).
   - Using Windows Explorer or My Computer locate the file and double click on it. When R opens up check that a data frame *turkeys_df* is present in the workspace.
   - What has the working directory been set to (check using *getwd()*)?

8 Good housekeeping (keeping your files tidy)

It is good practice to establish different working directories for distinct pieces of work. You may then choose to store separate *.RData* and *.R* files in each directory if necessary. This allows you to continue quickly from where you finished previously. You are encouraged to create different .RData files for each of the R assignments in this course.

8.1 Saving .R script files and .RData files in Intro Week

During Intro Week we recommended you save all your work on your *U:* drive. You can access this drive from anywhere through the *unidrive* service on *MUSE*. If you do this you will be able to access your Intro Week work when you are not on campus.

9 Packages and environments

9.1 Installing packages

One of the strengths of R is that it is easy for the scientific community to make newly developed functions available for everyone else to use for free.

- The way you access these functions is via *packages* (sometimes called *libraries*)
- The currently available packages on your computer can be seen on the *Packages* tab (several are available by default)
- If the package you need isn't there, you need to install it via *packages* → *install*
- Type the name of the package in the search box.

> Try this:
>
> Install the `magrittr` package. It should now be visible in the Packages window (if not try refreshing it).

9.2 Objects and environments

Objects are just things stored in memory. Examples of objects include character strings, vectors, lists, functions or dataframes. Whenever an R command refers to an object, R needs to locate that object. It does so by searching in a sequence of environments which make up the search path. These environments are just collections of objects kept together in the computer's virtual memory. The search path can be seen by typing `search()`.

9.3 The Global Environment (Workspace)

The first environment is .GlobalEnv (the Global Environment or Workspace). Generally when you create something in R it will appear here. Other environments (usually packages) contain other R functions and dataframes.

You can see the contents of a package by clicking on it in the Packages tab in RStudio.

> Try this:
>
> Click on the `magrittr` package in the Packages tab and inspect the documentation.

9.4 Loading packages

We have seen how to install a package (i.e. to make it available in the Packages window). But to actually make the objects in the package available directly we need to add it to the search path. This is done by ticking the box to the left of the package in the Package window.

Try this:

- Look at your current search path using the function `search()`.
- Add the `magrittr` package to the search path by ticking the box next to it in the Packages window.
- Look at the search path again and observe that `magrittr` is now the second environment.
- Remove the `magrittr` package from the search path by ticking the box next to it in the Packages window.
- Look at the search path again; `magrittr` is no longer in the search path.

10 Getting help

10.1 Help within R

Help is available online within R. For example, help on the function *mean*, which not surprisingly calculates the mean of a set of numbers, can be found by typing *help(mean)*. Information will appear in the *Help* tab.

Whenever *anyone* uses a new function they really should look at the *help* file for it (this includes experienced R programmers). The *help* file tells the user what format (vector, list, array, …) the function arguments are allowed to take. It also tells the user about optional arguments and importantly the default values for arguments. If you use functions without knowing all of these details you are likely to run in to trouble. So get in the habit of using *help* from the start.

Further general help is available via the Help menu on the R Console. *Help → R Help* will open clickable links in the *Help* tab that provide help on all aspects of R.

10.2 Help from the R community

Stack Overflow (SO) is a useful forum where you can post questions for the R community. Use the *search* facility to see if a question has already been asked before - it usually has. The url is here (http://stackoverflow.com/questions). Obviously never post questions relating directly to assignments here.

11 Style Guide

Reading R code is much easier if everyone uses the same coding style. In these notes we use the style proposed by the R guru Hadley Wickham (Wickham 2015). There will also be further a *Style Guide* section once we start to talk about writing functions in R. Wickham's suggestions for good style are:

1. put spaces around all operators (etc). The exceptions are : and ::
2. put a space after a comma, but not before it
3. use `<-` not `=` to assign a name
4. variable names should
    - be lowercase
    - use underscores not hyphens and not spaces to join words together
    - be concise
    - be meaningful
    - be a noun, not a verb.

5. no more than 80 characters per line
6. put a space before a (unless you are calling a function
7. put a space after the comment character `#`

Naming variables and functions in a meaningful way is surprisingly difficult. Here is some code that is in this format (if you are a new R user just concentrate on the layout for now, not what the code does)

```
x <- 2
x_squared <- x ^ 2
y_sequence <- 1:x_squared
even_sequence <- seq(2, 10, by = 2)
```

> Try this:
>
> Check the code above adheres to the rules in the Style guide.

We are now in a position to write some R code. We start with a really common use of R in statistics: working with probability distributions.

12 Working with probability distributions in R

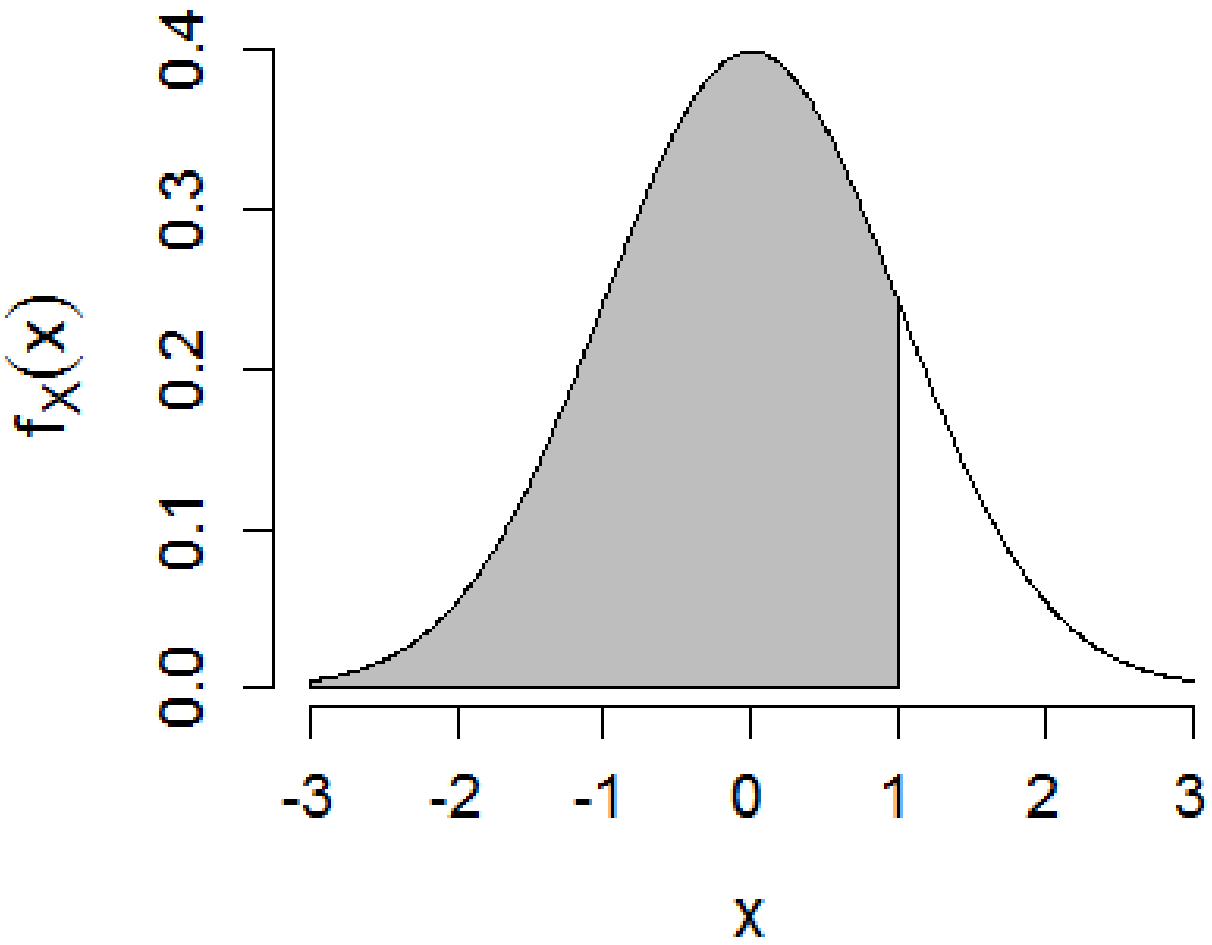

Figure 12.1: A standard normal probability density function illustrating the relationships between *dnorm*, *qnorm* and *pnorm*

There are three numerical values related to probability distributions that we need to be able to calculate easily, these are

1. probability densities (for continuous random variables), e.g. $f_X(X = 3)$ or equivalently probabilities (for discrete random variables), e.g. $Pr(X = 3)$
2. cumulative probabilities, e.g. $Pr(X \leq 3)$
3. quantiles of the random variable, e.g. the value x of the random variable X such that $Pr(X \leq x) = h$ where h is some specified probability bewtween 0 and 1 (x is the quantile)

Consider the standard normal distribution in Figure 12.1. We have

1. the probability density at $X = 1$ (height of the curve) is given by *dnorm(1, mean = 0, sd = 1)* giving a value 0.241
2. the cumulative probability $Pr(X \leq 1)$ (shaded area) is given by *pnorm(1, mean = 0, sd = 1)* giving a value 0.84 (the total area is of course 1)
3. the 84$^{\text{th}}$ percentile (value of x such that $Pr(X \leq x) = 0.84$) is given by *qnorm(0.84, mean = 0, sd = 1)* giving a value of approximately 1

Probability densities/probabilities for random variables with the Poisson, binomial, chi-squared and t distributions are obtained using *dpois, dbinom, dchisq, dt* etc. Similarly quantiles are obtained using *qpois, qbinom, qchisq* and *qt*.

So, for example, *qchisq(p = 0.95,df = 5)* returns the 95th percentile of a random variable with a chi-squared distribution on 5 degrees of freedom.

It is important to look at the help files for further details of how to specify the parameters of these distributions. For example, R uses the **standard deviation** rather than the **variance** in the dnorm function. You can only find this out by looking at the help files.

Another important function is *rnorm* (for the normal distribution). This simulates a specified number of independent realisations of a random variable having a normal distribution. So to simulate 10 realisations of a random variable with a $N(1, 9)$ probability distribution we can use

```
rnorm(n = 10, mean = 1,sd = 3)
```

```
##  [1]  0.6256729 -0.2105271 -1.4080566  5.7130615  1.2624545  1.1035634
##  [7]  2.7638890  4.8046035  1.0519547  9.0556406
```

13 Exercises B

Answer these questions using a script file.

1.
   - Look at the default R search path from the Console Window.
   - Add a library (the MASS library for example) and look at how the path has changed.
   - View the contents of the MASS library.
2. Create some folders in your U: drive that you can use to store your work for this module. As a minimum you will need folders for
   - solutions to chapter exercises
   - assessed work
3. Create a new script file. Save it somewhere sensible in your newly created folders and give it a suitable name. Use this script file to write your code to answer the following questions.
4. Suppose we want to plot the probability density function of the standard normal distribution. We can do this by
   - specifying a sequence of x co-ordinates
   - calculating the y co-ordinates for these values of x

- plotting them

This can be done in R using the code below. Copy and paste this code into your script file.

```
x <- seq(-3, 3, by = 0.01)
y <- dnorm(x)
plot(x,y, type="l")
```

5. Find at least three ways to run this code.
6. Use the *help* system to find out about the *plot* function. Hence modify the plot so that the y-axis label is *probability density* rather than *y* (you may need to look at the examples provided in the help file).
7. Consider a discrete random variable X with a Poisson probability distribution with a mean of 3 ($X \sim \text{Po}(3)$). How could you use *dpois* in R to calculate $P(X = 7)$? (use the *help* system to find out about the *dpois* function if you are unsure).
8. By using the probability mass function of the Poisson distribution verify the value of $P(X = 7)$ in R without using *dpois* (look at the Wikipedia entry (https://en.wikipedia.org/wiki/Poisson_distribution) for an expression for the probability mass function if you have forgotten it). You will need the following operators:
    - `*` for 'multiply' (e.g. `8 * 4`)
    - `\` for divide (e.g. `8 \ 4`)
    - `^` for raising one number to a power (e.g. `2 ^ 3`)
    - `exp()` : the exponential function (e.g. `exp(4)`)
    - `factorial()` : the factorial function (e.g. `factorial(3)`)
9. By modifying the code in Question 3, plot the probability mass function of a random variable with a $Po(3)$ distribution.
10. Below is some badly formatted R code (so bad the syntax actually causes a common error and it won't run). Copy it and improve the syntax/style:

```
normal.quantile <- qnorm(0.4, mean=0,sd= 1)
squared value = normal.quantile ^2
```

11. Save the objects in your Workspace into a suitably named *.RData* file and close RStudio.
12. Re-load these objects into RStudio by double clicking on the *.RData* file you created.

References

Wickham, H. 2015. *Advanced R*. CRC Press.