Navigating R and RStudio

NES2303

# 1 Aims and objectives of Practical 0

The overall aim of this practical is to help you become more familiar with RStudio, what its main panels are for, how to download data from Canvas, and import it into R. Specific objectives include:

1. explain the concept of RStudio projects to allow you to store data, files containing R commands (known as scripts), and the four main panels in RStudio
2. demonstrate the value of .csv files (comma separated files) as these can be read / written by both R and Microsoft Excel, and the functions to do this.
3. understand free R packages to extend the power of R. How to install and load packages for use.
4. how to calculate simple summary statistics of the data you have imported into R

## 1.1 R vs RStudio

R is the basic underlying computer software you use to undertake analyses of biological data. RStudio is a separate programme that creates an “Integrated Development Environment” (IDE) that makes it easier to do data science. Think of the differences between doing arithmetic on a hand-calculator vs Microsoft Excel. RStudio defaults to automatically uses R, but can handle other computer languages such as C++, HTML and Python.

## 1.2 Starting RStudio

Campus computers have the software installed and NES2303 assumes you will be using an NUIT-managed PC. Both R and RStudio are free download from [download the R software](https://cran.r-project.org/) and [Download the R Studio desktop software](https://rstudio.com/products/rstudio/#rstudio-desktop) respectively. However, if you run it on your own laptop or MacBook we **will not provide you with any technical support** as there is too much variation between operating systems, PC configuration etc.

You should have a ‘search’ bar at the bottom left of your NUIT Windows PC, labelled *Type here to search*. In it type ***RStudio*** and the launch icon should be displayed for you to click on:



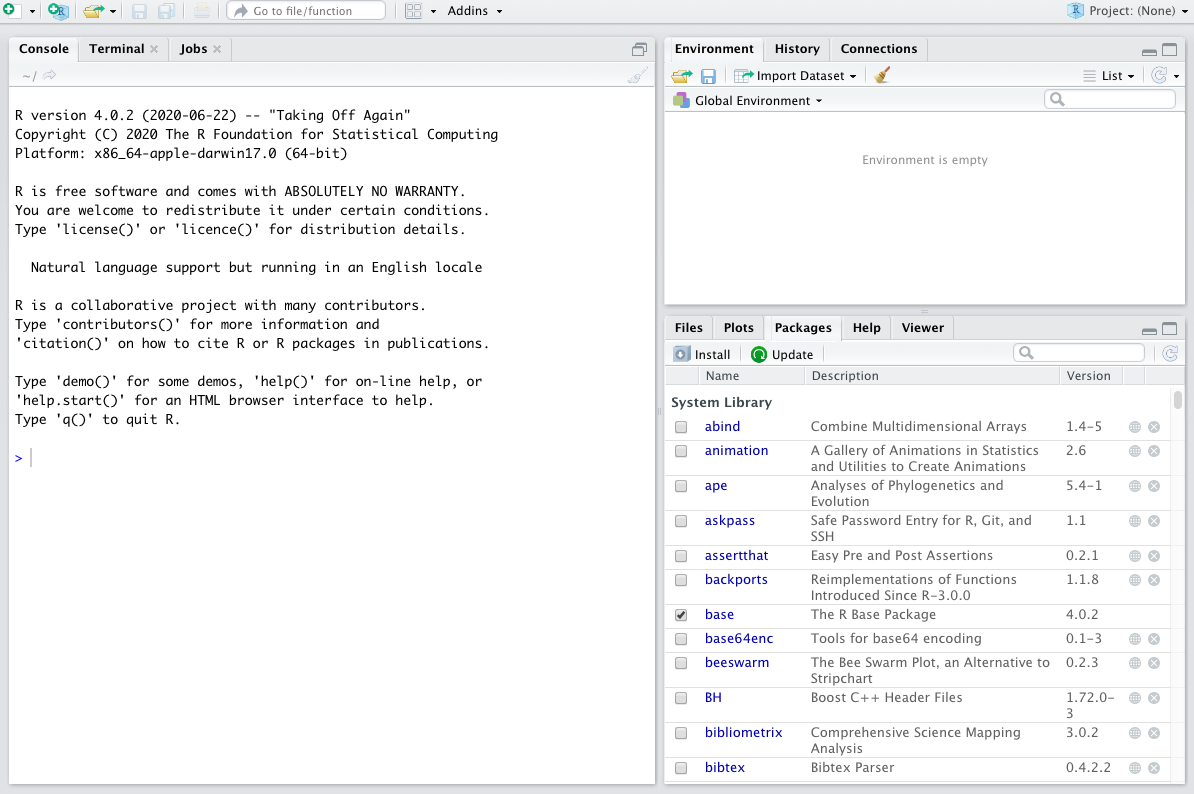
(If you do not have a search bar, click on the Windows icon, navigate to *Programming and Databases* or *RStudio* and click on the *RStudio* icon. Do **not** click on the icon labelled *R i386* or *R x64* )

**IMPORTANT**: If a popup window appears when you start RStudio click “Ignore Update”

Once RStudio has started up you can get to work.

# 2 Modify RStudio default settings

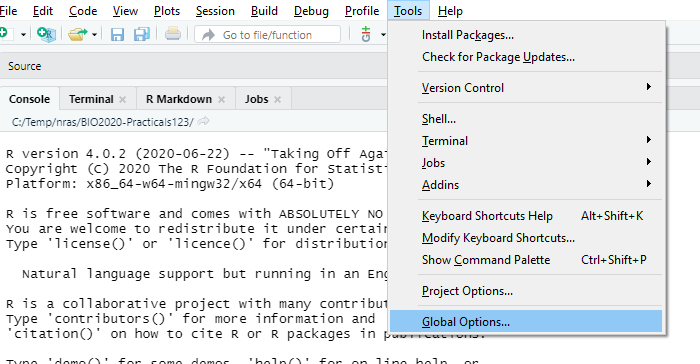
On opening R Studio you should see something like this



Unfortunately RStudio has a number of default settings which from past experience is confusing for students. If you suffer from dyslexia or have poor eyesight there are several other settings you may want to change.

## 2.1 Startup and close defaults

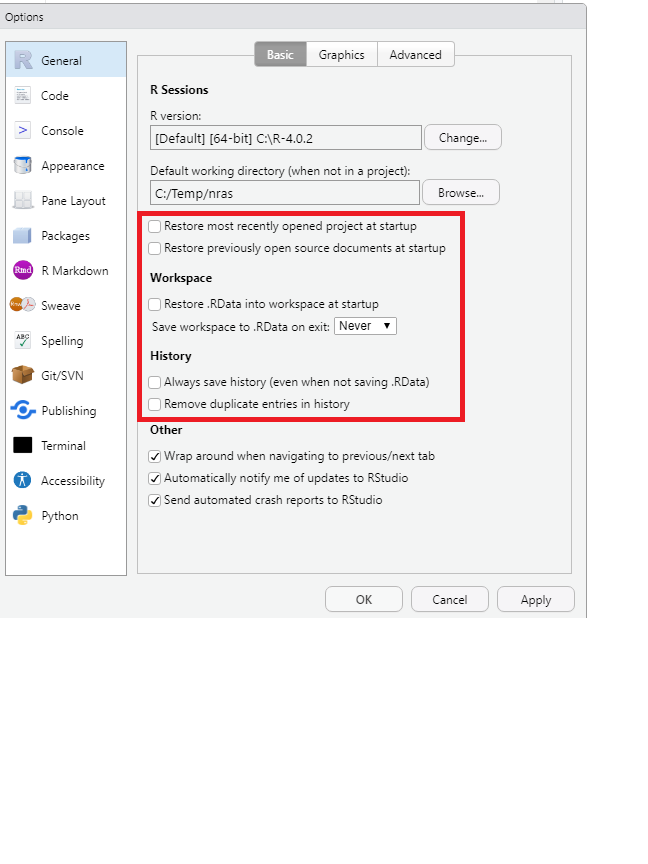
By default RStudio tries to re-open everything you last worked with (even if you are analysing new data), prompts you to store data (even when you have it stored already) etc. Let’s switch off those annoying defaults. From the main menu click on **Tools -> Global options**



Make sure that the following boxes are **unticked** or changed (see below):

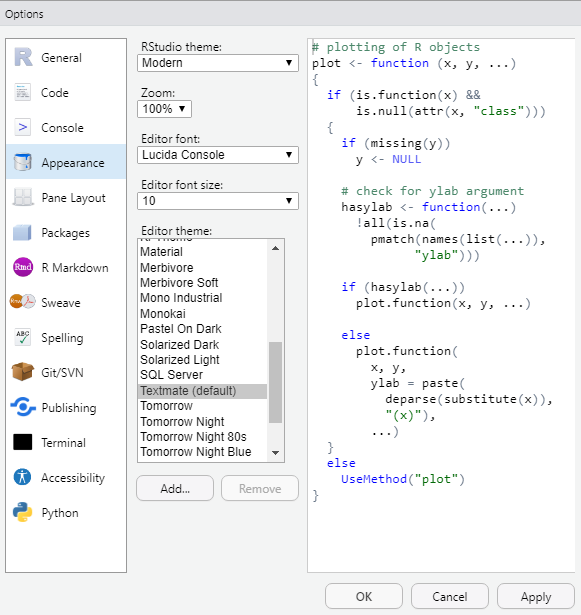
* Restore most recently opened project at startup
* Restore previously opened source files at startup
* Restore .RData into workspace at startup
* Save workspace to .RData on exit. **Change to never**
* Always save history (even when not saving .RData)
* Remove duplicate entries in history

Click on ‘OK’ once you have done this.



## 2.2 Change appearance (optional)

If like me you have poor eyesight, or e.g. have dyslexia and find some fonts difficult to read, you can modify the RStudio appearance. Again click on **Tools -> Global options** but click on the **Appearance** icon on the left. You should see something similar to below:



The default ‘Editor theme’ appearance is called Textmate, which I find rather harsh on the eyes. Preview other appearance options by clicking on them (personally I like the ‘Tomorrow Night bright’ theme which has a black background). You can also change the font size to make it bigger, or the Zoom ratio.

# 3 R packages

A package is an additional ‘plug-in’ for R that can extend its capabilities. When R was originally created there were only about 10 add-on packages, but now there are over 20,000. This is one of the reasons why it is so popular amongst scientists: it is easy to adapt, modify and improve. Here are some of the things I’ve seen people use packages for:

* create maps of satellite data
* analyse GPS animal movement data
* visualise genome sequences
* analyse next-generation sequence MinIon data
* create phylogenetic trees
* plot citizen science species records
* model flood risk
* artificial intelligence on wildlife camera trap
* analyse songbird audio
* model disease spread in humans and wildlife
* scrape data from Twitter for ‘sentiment’ analysis
* write books and reports
* create websites

Packages are hosted on the Comprehensive R Archive or CRAN. As there are now so many R packages they have been organised into [Task Views](https://cran.r-project.org/web/views/) on CRAN to help you navigate them.

## 3.1 How to install and activate a package from CRAN

Using a package is a 2-step process:

* Install a package. This downloads software from the internet onto your computer. **It only has to be done once**. If you use NUIT cluster room PCs most of the packages you need for NES2303 should already be available so you can skip this step. Use install.packages("package\_name") to install a package.
* Activate a package. This makes the extra R commands available for you to use. **It has to be done every time you use R**. Use the command library(package\_name) to activate a package.

A common mistake is to forget to use the library() command. I try to remember to use it first in any R session.

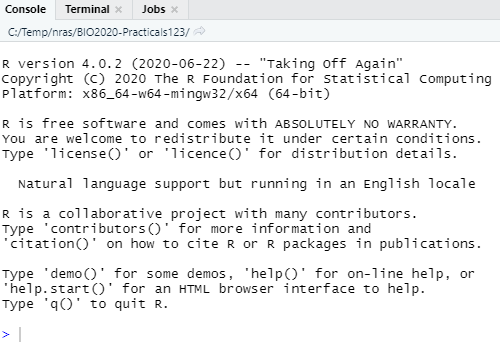
## 3.2 Installing the R packages needed for NES2303 (**Needed for this practical**)

You need three R packages installed initially for this module. You should only need to do the installation once on an NUIT networked PC. The three packages are:

* mosaic : this does a goal-oriented approach for plots and models
* car : this makes the output of some analyses easier to understand
* tidyr : useful for tidying up date

You will also use the vegan package in the last two practical classes.

On the left of your RStudio screen you should see a large window headed **Console** which will contain introductory text about the R version you are usin etc. On my PC it shows as version 4.0.2 but you may have a more recent version:



Let’s install mosaic, car, tidyr and vegan now so you are ready to use them for later and they are already installed. In the Console window type the text below followed by the Enter key:

install.packages("mosaic")  
install.packages("car")  
install.packages("tidyr")  
install.packages("vegan")

Normally lots of output is generated as packages are downloaded from the internet and stored on your computer. You will often see other packages being installed at the same time. This is nothing to worry about.

## 3.3 Activating an R package (**Needed for every practical**)

Use the library() function to activate an R package. Installing an R package simply puts the extra software on your PC, but it does not make it available for use. This is because you may have many packages eventually installed, that might have commands with identical names but which do different things. To take a simple example, in some packages the plot command will create a map, whereas in other packages it might display a genome phylogeny, and you would not want to confuse them. Therefore, in any R session you generally only make available the packages you need. To make the commands from a package available to use, enter the library command followed by the package name:

library(mosaic)  
library(car)  
library(tidyr)

The vegan package is needed for the last two practicals later in the Semester.

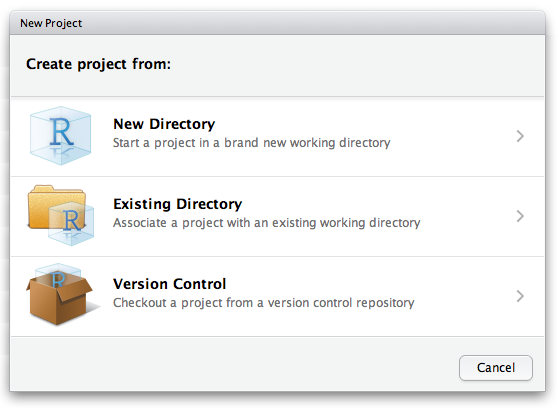
# 4 RStudio projects

An RStudio project is a special folder where you can safely keep all the commands for your analyses and your data files together. I generally create a separate RStudio project folder for each experiment or survey that I am involved in.

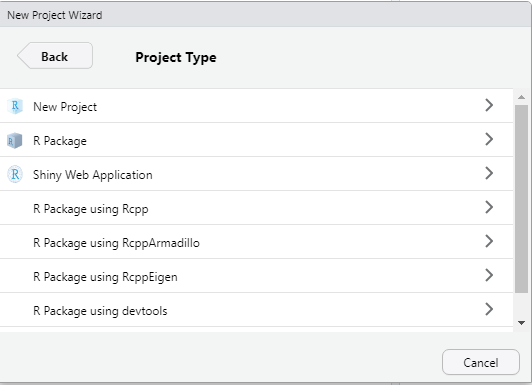
## 4.1 Create an RStudio project (Just needed once in this first practical)

We are going to start by creating an RStudio Project called ‘NES2303’ to keep all of our work relating to the ‘NES2303’ course together in one place. Later in your degree you will want to create new RStudio projects for other experiments or work.

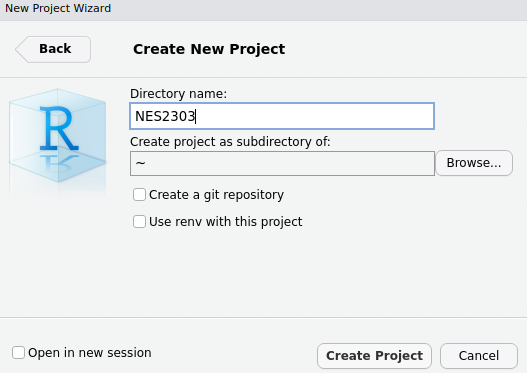
To create a new project click on the main menu on File > New Project  
You should now see this popup window, on which you should select **New directory**



then a **Project Type** window like the following should appear, from which you should select **New Project**:



Finally, the following window, entitled **Create New Project** should appear:



In this window complete two sections:

* The **Directory name**. This is the name of the folder to store your files containing records of your analyses and your data . For convenience, call it **NES2303** as shown above.
* The section **Create project as subdirectory of:** Click on the **Browser** button and navigate to **Documents** or **My Documents**. The exact title will differ depending on your NUIT settings. This wants to be where you normally store all your folders and files. On my own PC, the “My Documents” area has a “shortcut” name of **~** which you can see displayed above. The setup might be slightly different on your PC.

Finally, click on the **Create project** button at the bottom and you should now see the name of your project displayed in the top right hand side of your R Studio window.

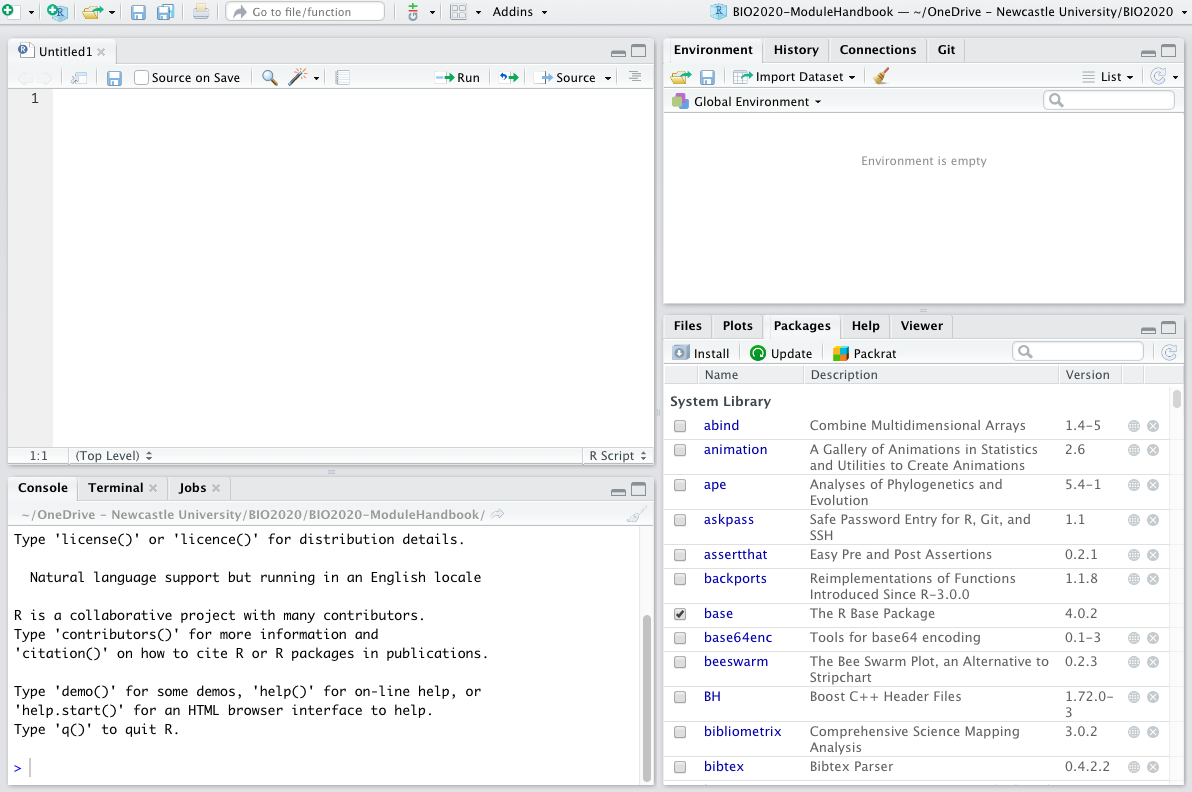
## 4.2 How to start RStudio with your project (all future practicals in NES2303)

Once you have created a project, all you need to do in is:

* Start RStudio
* Click on **File -> Recent projects…**
* Select **NES2303** from the list

# 5 Navigating RStudio

In the top left click file > new file > R script and now you should have four windows that look something like this:



You should now have Four panes in your R Studio window, you might notice that some of them have multiple tabs as well. Let’s run through the meaning of each of the panes, and the tabs that we are going to be using:

## 5.1 Scripts

The pane in the **top left** contains your scripts. A script is a plain text file that should be saved with the extension .R and it contains a list of all the commands you use to import, summarise, display, analyse, plot or export data.

One of the key reasons that R is so popular is because it allows us to make our work reproducible. By writing down each bit of our analysis and saving it in a script we can reproduce our work with ease. Future you will thank past you for saving your code in scripts when you are making last minute edits to your dissertation and you need to remember exactly what the analysis methods you used were!

## 5.2 Environment

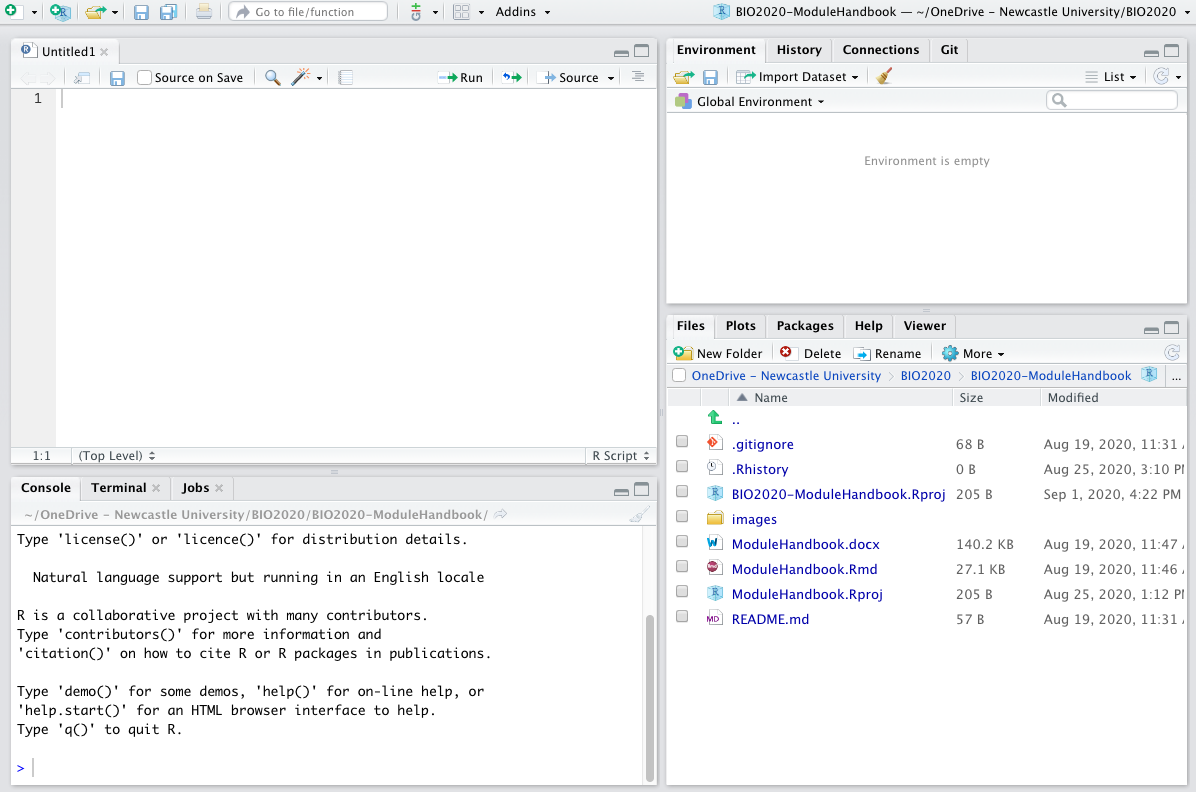
The pane in the **top right** has a couple of tabs but the one that we need to know about is the Environment tab. The Environment pane tells us what objects R has loaded in to its memory (we’ll touch on this a bit more later on), these might be data files that we have imported or the results of some analyses. This pane shows us the objects that R has available for doing useful stuff.

## 5.3 Console

The pane in the **bottom left** also has several tabs and the one that we use is called the console. This pane is the ‘doing pane’ when we want to run bits of code that we have saved in our script we send them to the console to be executed - which means R does what our code tells it to do and we get some output. You have already directly typed commands into the Console to setup some intial packages. In future **strongly advise against doing this**, it is much better to write your code in a script that you can save and come back to for future reference.

## 5.4 Files/Plots/Packages

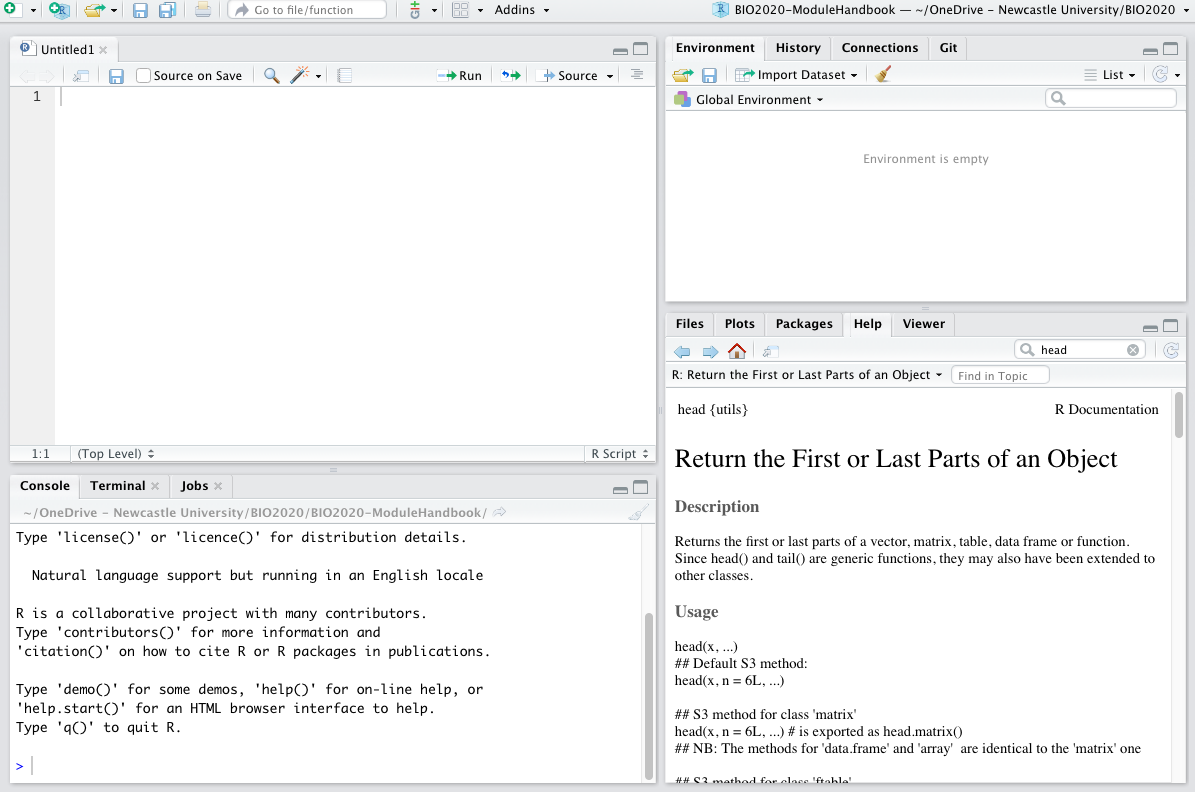
**Bottom right pane** has several tabs that give the pane different functions that can be of use.

The ***Files*** tab can be used a bit like windows explorer/finder to navigate through the files on your computer. 

The ***Plots*** tab is where the plots that you create using the code written in your script will appear. It has an export button that you can use to save your plots or to copy them to your clipboard so that you can then insert them into documents or presentations.

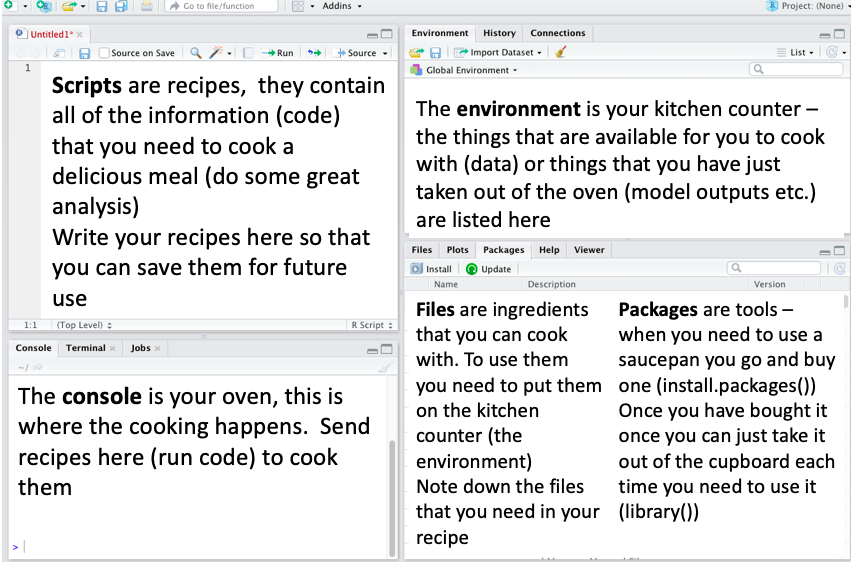
The ***Packages*** tab shows you the packages that are installed on your computer, the ones with ticks in the boxes are ones that are loaded and ready for you to use. Packages are collections of code and/or data that other people have written that we can use. Packages mostly contain bits of code called *functions* which we will cover in a bit more depth in the ‘working with data’ section.

The ***Help*** tab offers you an interactive window through which you can use the help files for R functions. You can use the search bar at the top of the pane to search for the help files associated with a given package or function. We will cover the help files in more detail once we have got a bit more of an understanding of some of the R lingo



**Think of R as your Kitchen**

Using R can be thought of similar to using a kitchen to cook.



There are loads of really great chefs (coders) in the world who have already figured out a lot of techniques and gadgets that can be used to make some great food. You don’t always have to cook everything from scratch - you can use the information and tools that other people have provided to make your cooking experience easier and more enjoyable.

# 6 Download files from Canvas and import data

In NES2303 all of the data will be provided for you on Canvas for import into R. When you work independently, you will probably be entering data in Microsoft Excel. Either way there are a couple of ‘gotchas’ to look out for. Download the Excel File yield.xlsx from Canvas - you can open it in Excel and look at it, but don’t edit it or make any changes to it. The file contains crop growth in three different soil types. Now, in your Script window (top left) type the following text:

library(NES2303) # or library(mosaic) if NES2303 did not install  
pract0\_dat<-read.csv("yield.csv")  
summary(pract0\_dat)  
mean(yield~soil,data=pract0\_dat)

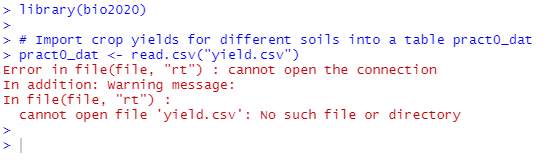
A few key points:

* Whenever you start RStudio you will need to activate the relevant packages, so have a library command near the top
* The second line reads in your data file and saves it in a table called pract0\_dat
* The third line will produce a short summary of your data
* But **it is really hard to understand**! Imagine **iftextiswrittenlikethis** then it would be hard to understand. Whereas **if text is written like this** it now makes sense. So we will add spaces, blank lines and **Comments**. A comment is anything that follows the # symbol and is ignored by R, but helps you understand what you have done. So before going further, edit your script to make it more readable, with comment lines, blank lines, and a space around the <- symbol, and in the mean line:

# Example script to check setup is working  
  
# Activate the NES2303 package so extra commands available  
library(NES2303) # or library(mosaic) if NES2303 did not install  
  
# Import crop yields for different soils into a table pract0\_dat  
pract0\_dat <- read.csv("yield.csv")  
  
# Print out a summary of the data held in pract0\_dat  
summary(pract0\_dat)  
  
# Obtain the mean yield for each soil type  
mean(yield ~ soil, data = pract0\_dat)

Now save your R script. Click on **File -> Save** from the RStudio main menu, or if you prefer the save icon, and save the script as practical0.R. Next run each line of your script. Click on the library(NES2303) line and hit the **Ctl-Shift-Enter** keys simultaneously. This key combination sends a line from your script to the Console window where it is executed. The command should run without displaying anything as NES2303 is probably already activated. Your cursor should jump to the read.csv line. (If you had problems with the NES2303 package, simply type library(mosaic) instead).

However, when you hit Ctl-Shift-Enter on the read.csv line you should get the following error:



What has gone wrong? We’ve just downloaded the file from Canvas yet R claims it is missing? The problem is that when you downloaded the data file from Canvas, it placed it into your **Downloads** folder. You need to move it to the **NES2303** folder, that already contains your project settings file, and your practical0.R file:

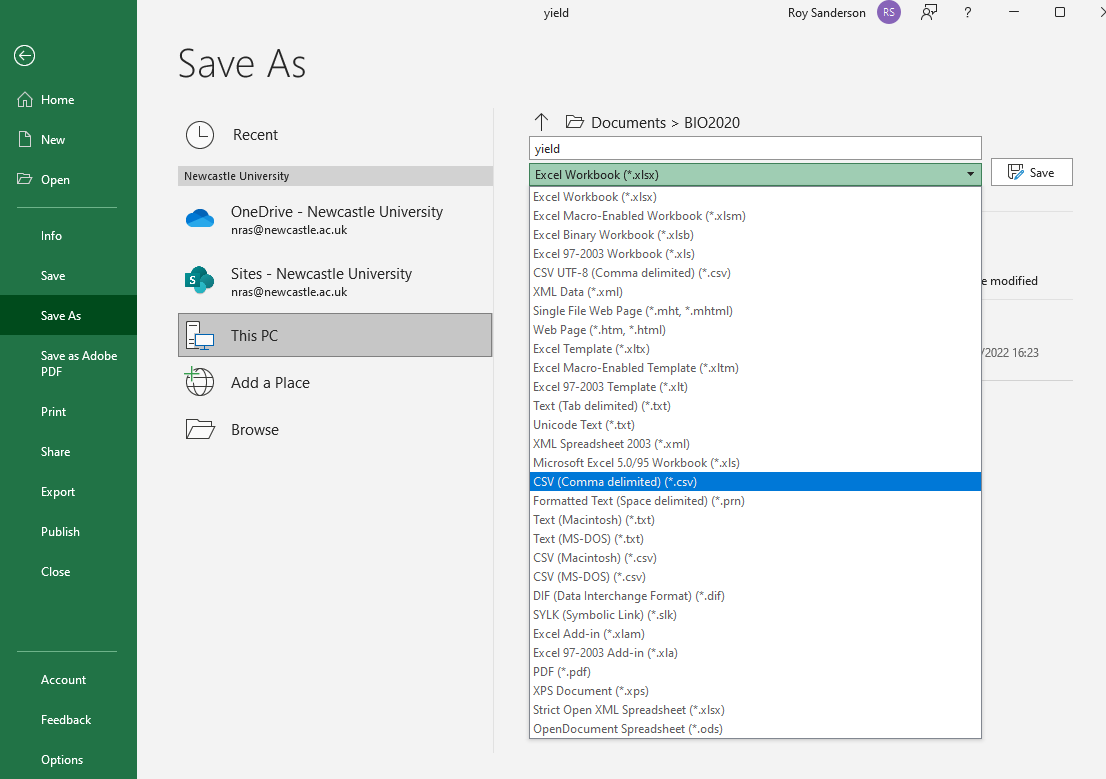
* Start up the File Manager either from the main Windows menu, or by entering “File Manager” in the search bar
* Navigate to your Downloads folder
* Locate the yield file in your Downloads folder
* Right-click on your yield file and select **Copy**
* Navigate to your NES2303 folder, Right-click and select **Paste**
* Go back to RStudio, and in the Files tab (bottom right) click on the “refresh” button (a small circlur arrow on the right of the files tab). You should see the yield file listed.

Now go back to your R script and try re-running the read.csv command by clicking on the line, and hitting the Ctrl-Shift-Enter key combination again. BUT **it does not work**. You get the same error message!! Why??

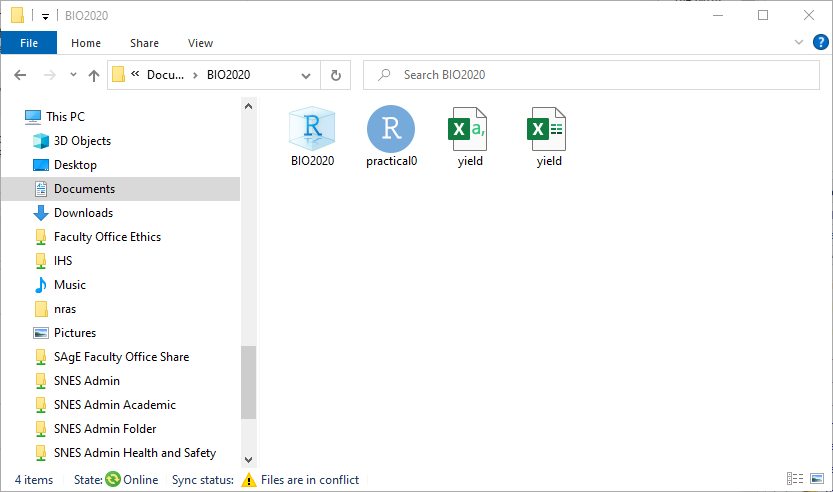
This is such a common error, that I thought it worth emphasising it. Some of you may even have already worked out what has gone wrong. The clue is in the name of the command read.csv which is to read a CSV file. What is this:

* CSV stands for “comma separated value” and is a plain text, simple file, that can be read by hundreds of different programs and apps
* A standard Microsoft Excel file is a commercial-format file, that can be imported by a more limited range of software. It is much more complex as it can contain multiple tables, forumlae, graphs etc.

So, finally we need to convert our Microsoft Excel file. Go back to your File Manager, navigate to the NES2303 folder (it might already be open) and double-click on the yield file to open it in Microsoft Excel. Then click on the **File -> Save As** option, and in the drop-box below the filename, select the CSV option:



Now close Excel; you may get a warning claiming you haven’t saved your file (you have!), but Microsoft gets cross if you don’t use its commercial file formats. Go back to your File Manager and you should see something similar to this:



There are four files listed, although the default settings in File Manager does not give their full names (last few letters are omitted) which can cause confusion

* NES2303 which is actually NES2303.RProj and contains your RStudio project settings
* practical0 which is actuallry practical0.R and contains your script with R commands
* yield which is actually yield.csv in Comma Separated Value format
* yield which is actuall yield.xlsx in Microsoft Excel format

Look very carefully and you can see that the icons for the two yield files are slightly different, as one has **,a** in it. You can change the settings in File Manager so that full file names with the final “extension” are showing. For the rest of the NES2303 course all the files will be CSV format, so you will not have this problem. When you enter your own data into Excel for use in RStudio, you need to be aware of it.

Go back to RStudio. Now you should be able to re-run your read.csv line. If it is successful, nothing will be displayed in the console, but in your Environment window (top right panel) you should see the table (data frame) pract0\_dat listed, and when you run the summary(pract0\_dat) and mean(yield ~ soil, data = pract0\_dat) commands you should see:

## soil yield   
## Length:30 Min. : 3.00   
## Class :character 1st Qu.: 9.25   
## Mode :character Median :12.00   
## Mean :11.90   
## 3rd Qu.:14.75   
## Max. :18.00

## clay loam sand   
## 11.5 14.3 9.9

Success! Please let myself or one of the demonstrators know if you have problems with this step.

**Where to store your CSV files?**

You can store them either in your main project folder. Sometimes if you have lots of data files this can get confusing, so I generally recommend creating a folder called **Data** inside your NES2303 project folder, and putting your CSV files there. The only difference is you need to change the read.csv command from:

pract0\_dat <- read.csv("yield.csv")

to

pract0\_dat <- read.csv("Data/yield.csv")

My preference (and recommendation to you) is to put all your CSV files into a Data subfolder in your NES2303 folder, in which case you’ll need to remember to add Data/ to any read.csv commands. That way your NES2303 folder is less likely to get cluttered as you go through the module.

# 7 Working with data

## 7.1 Using R as a calculator

You probably won’t use this very often, but you can easily enter values into the Console window if you need to do some quick arithmetic:

7 \* 3

## [1] 21

log(24)

## [1] 3.178054

4^2

## [1] 16

7 / 3

## [1] 2.333333

pi \* 3 ^ 2

## [1] 28.27433

Individual numbers, columns, rows, or tables of data, the results from calculations, statistical analyses and graphs are all **R objects** and the ones you have created in your current RStudio are shown in the top right **Environment** panel.

As mentioned before R is a programming language, as with any language there is some vocab that needs to be covered in order to help your understanding of what is being described.

## 7.2 Creating Objects

Although you have done some arithmetic above, it is much more useful to **assign** the results to an R object, so you can re-use them later. You do this using the assignment symbol <- which looks like a left arrow. If you are having trouble finding the < and - keys, a handy shortcut is to press

typing Alt + - (push Alt at the same time as the - key) will write <- in a single keystroke.

Now you can *assign* a value(s) to an *object* so that R remembers it. You need to come up with a name for your object. Come up with a good name!

Good names:

* my\_height
* bird\_weights
* bacteria\_growth

Poor names:

* my\_dat You will soon forget what this contains
* Bird\_wght You will forget the upper-case B and whether it is wght, Wght, weight etc.
* bacterialgrowth Rather long and difficult to read

For example, if you are 175 cm tall, you can assign your height to an R object simply by the following line:

my\_height <- 175

It can be assigned and displayed:

my\_height <- 175  
  
# check the new value  
my\_height

## [1] 175

Now that you have stored your height in the my\_height *object* we can do some arithmetic with it. Run the below code to convert your height from cm to inches

my\_height <- 175   
  
# Convert to inches  
my\_height/2.54

## [1] 68.89764

In the following code *value* of my\_height is unchanged at 175

my\_height <- 175   
  
# Convert to inches  
my\_height/2.54

## [1] 68.89764

# Note that my\_height is still 175  
my\_height

## [1] 175

You will notice that the *value* of my\_height is still in cm, why? Because we didn’t store your height in inches as an object, let’s do that now:

# Assign height in cm  
my\_height <- 175  
  
# Calculate height in inches and assign to new R object  
my\_height\_inch <- my\_height/2.54  
  
# check the value of the object   
my\_height\_inch

## [1] 68.89764

### Comments

The comment character in R is #, anything to the right of a # in a script will be ignored by R. It is very useful to leave notes and explanations in your scripts, we highly recommend that you get into the habit of commenting your code as it will help you to remember what you have done AND it will help us to help you if you encounter problems. R Studio makes it easy to comment or uncomment a paragraph: after selecting the lines you want to comment, press at the same time on your keyboard Ctrl + Shift + C. If you only want to comment out one line, you can put the cursor at any location of that line (i.e. no need to select the whole line), then press Ctrl + Shift + C. **Comments are the most important lines in any R script!!**

## 7.3 Functions (or “commands”)

Almost all or our R coding uses functions, which are essentially bits of code that have been written to do something. When we refer to functions we will generally do so following the convention functionname() the brackets are a giveaway that we are referring to a function. Functions take *arguments*, the arguments (inputs) are specified within the brackets. We have already used the read.csv function, and gave it the argument “yield.csv” to import some data into R.

Lets use the function sqrt() as an example. The argument to this function is a value (or an object that contains a value) for which we want to calculate the square root. When we run a function it returns an output that is the square root of the input value.

# Run the function on a number, to find square-root of 10  
 sqrt(10)

## [1] 3.162278

# Run the function on an object that contains a numeric value  
 sqrt(my\_height)

## [1] 13.22876

Functions may “return” numbers (e.g. sqrt()), tables of data (data.frames), summaries of statistical analyses and models, and even graphs. Functions generally have “arguments”. For the sqrt() function you had a single argument containing a number. Arguments can be anything, not only numbers or filenames, but also other objects. Exactly what each arguments are needed and mean differs per function, and can be further investigated using the help files (Remember the help pane, bottom left) which we will explain further later on.

## 7.4 Data types

R can deal with many different types of data and where it recognizes differences in data it may treat it differently. It is helpful to understand the different types of data that you will most commonly encounter in R:

**Numeric**

Numeric data are numbers

1, 2, 3, 4.5, 6.78

**Character**

Character data contain non numeric characters and are most often words. R understands character data to be everything enclosed within the quotation marks ““.

"Lion", "tiger", "bear"

When we use this type of data in R we have to remember the “” otherwise R will assume that we are referring to objects (that may not have been created) of the same name and it likely to return an error.

**Factors**

These are useful in data analysis. If your data are already in character format, then R will automatically treat them as factors. However, you might have experimental treatments coded “1”, “2”, “3” and you would want to force R to recognise that they are categorical factors, not continuous numbers. R has a special as.factor() function to do this.

**Logical**

R understands logical data as TRUE FALSE, different from character data so no “” required and note that it is case sensitive.

TRUE, FALSE

| **Case sensitivity in R** |
| --- |
| R is case sensitive, we created an object called `my\_height`, if we were to ask R for the value of the object `My\_height` with a capital M we would see R return an error (unless we also have a variable called My\_height stored in our environment) |

## 7.5 Data structures

We have already learned how to *assign* a *value* to an *object* but to do more complicated things in R we often need to store more than one value in a single object.

### 7.5.1 Vectors

A vector is the most common and basic data type in R, a vector is composed by a series of values, which can be either numbers or characters. We can assign a series of values to a vector using the c() function. For example we can create a vector of animal weights and assign it to a new object weight\_kg

weight\_kg <- c(190, 310, 500)

We could also create a vector containing animal names

animals <- c("lion", "tiger", "bear")

We can inspect the contents of a vector, using functions such as length() and class() by giving these functions the name of the vector that we want to inspect.

Using length() and class() inspect the vectors weight\_kg and animals that we have just created to see if you can work out what these functions do. Edit the following two lines to put the correct text inside the () brackets:

# Weights of three animals  
weight\_kg <- c(190, 310, 500)  
  
# Species of three animals  
animals <- c("lion", "tiger", "bear")  
  
# The length() and class() function help us understand their contents  
length(weight\_kg)  
class(weight\_kg)  
  
length(animals)  
class(animals)

As we can see the above examples are vectors that contain a single data type, lets have a look at what happens when we have multiple data types.

### 7.5.2 Data frames

This is the best R object for storing your data in. It is basically a table of data, with columns of numbers or text. Another very common data structure in R is the Data frame, these may be more useful when we have different data types that we want to keep together.

Rather than combining animals and weight\_kg into a single vector we can convert them into a data frame.

# create an object in which to store the data frame  
animals\_df <- data.frame(animals, weight\_kg)  
  
# look at the data frame  
animals\_df

## animals weight\_kg  
## 1 lion 190  
## 2 tiger 310  
## 3 bear 500

We now have something that might look more familiar to those of us that are used to working with data in excel.

We can use the str() function to look at the structure of the data frame.

# Animals data  
weight\_kg <- c(190, 310, 500)  
animals <- c("lion", "tiger", "bear")  
  
# Put into a data.frame  
animals\_df <- data.frame(animals, weight\_kg)  
  
# Understand contents of data.frame with str()  
str(animals\_df)

## 'data.frame': 3 obs. of 2 variables:  
## $ animals : chr "lion" "tiger" "bear"  
## $ weight\_kg: num 190 310 500

From this we can see that the data frame has 3 observations (rows) and 2 variables(columns), one called animals which contains character data and one called weight\_kg which contains numeric data.

If we want to extract a specific column of a data frame we can do so using the $ operator. To extract a specific value we can *index* it using the square brackets [] to specify the position by location using dataframe[rownumber,columnnumber]

# Single column  
animals\_df$animals  
  
# Use the [row, column] suffix  
# Display first column  
animals\_df[, 1]  
  
# Display second row  
animals\_df[2, ]  
  
# Display single elements  
animals\_df[1,2] # First row, second column  
animals\_df[2,1] # Second row, first column  
  
# Enter the name of the data.frame to display its contents  
animals\_df

These operations can be useful when we are dealing with small data frames but often when we are working with data we have large datasets, perhaps contained in spreadsheets on your computer or downloaded from the net. Such information needs to be imported into R before we can start to interrogate the data frame.

Remember the kitchen analogy - Data contained in a file on your computer is like an ingredient in your cupboard, to be able to use it you first need to put it on the kitchen counter, which means we need to import it in to R and assign it to an object so that it appears in our environment pane.

# 8 Importing and exporting data

To be able to analyse our data in R we first have to import/read it in to R so that it is available to us in our environment (Taking it out of the cupboard and placing it on the counter). How we go about this varies slightly depending on the file that we are going to be using. In all cases we need to assign the information in the file to an object so our code will start with the name of the object we are creating and the <- symbol

## 8.1 Importing tables of data via CSV files

We have already looked at CSV files, which you can export from Microsoft Excel using Excel’s “Save As” option. This makes them the most useful “general” format for storing data.

my\_data\_object <- read.csv(path = "Path\_to\_the\_file.csv")

## 8.2 Saving (exporting) outputs

### 8.2.1 Saving data objects

Once we have manipulated our data we may want to save it as a file that we can then share with others. Data can be saved to .csv files using the write.csv function.

The code is similar to that used to read data in, it requires the filepath specifying where you want to save the file which includes the name that you want to give to the file, and it requires the name of the object that is being written (saved).

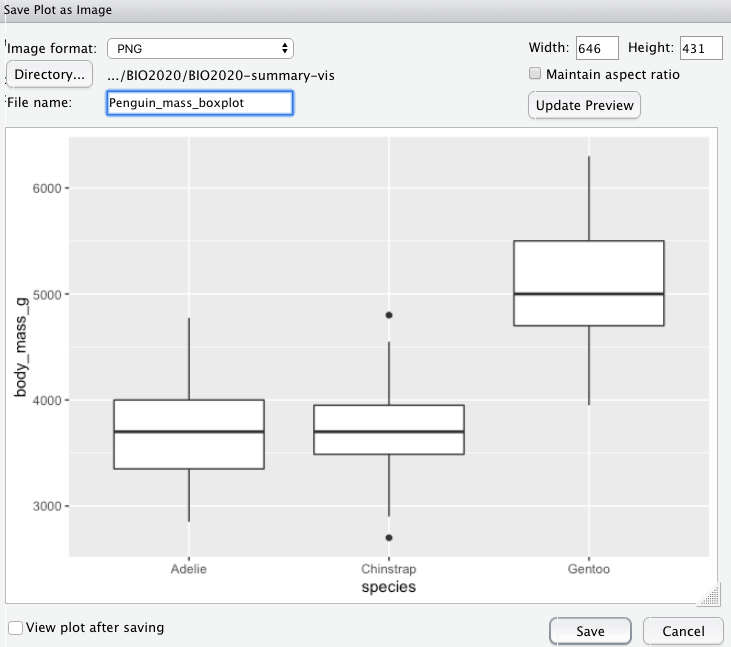
# writing my\_data\_object to a file called my\_output\_data  
  
write.csv(my\_data\_object, path = "Data/my\_output\_data.csv")

### 8.2.2 Saving plots

We will create various graphs of data in the next practical. Once you have created beautiful plots that support your analysis you will want to be able to copy them into your reports. The easiest way to do this is to use the dropdown Export menu in the plots pane.

You can copy plots to the clipboard or save them as a file (specifying the file type, .png/.jpg/.tiff etc.)

The popup preview that appears when you choose to copy a plot to the clipboard or to save it as an image allows you to specify the size that you want the image to be. Altering the size of your plot here will give you a much clearer plot than if you re-size a copied or saved plot after adding it to your destination document.

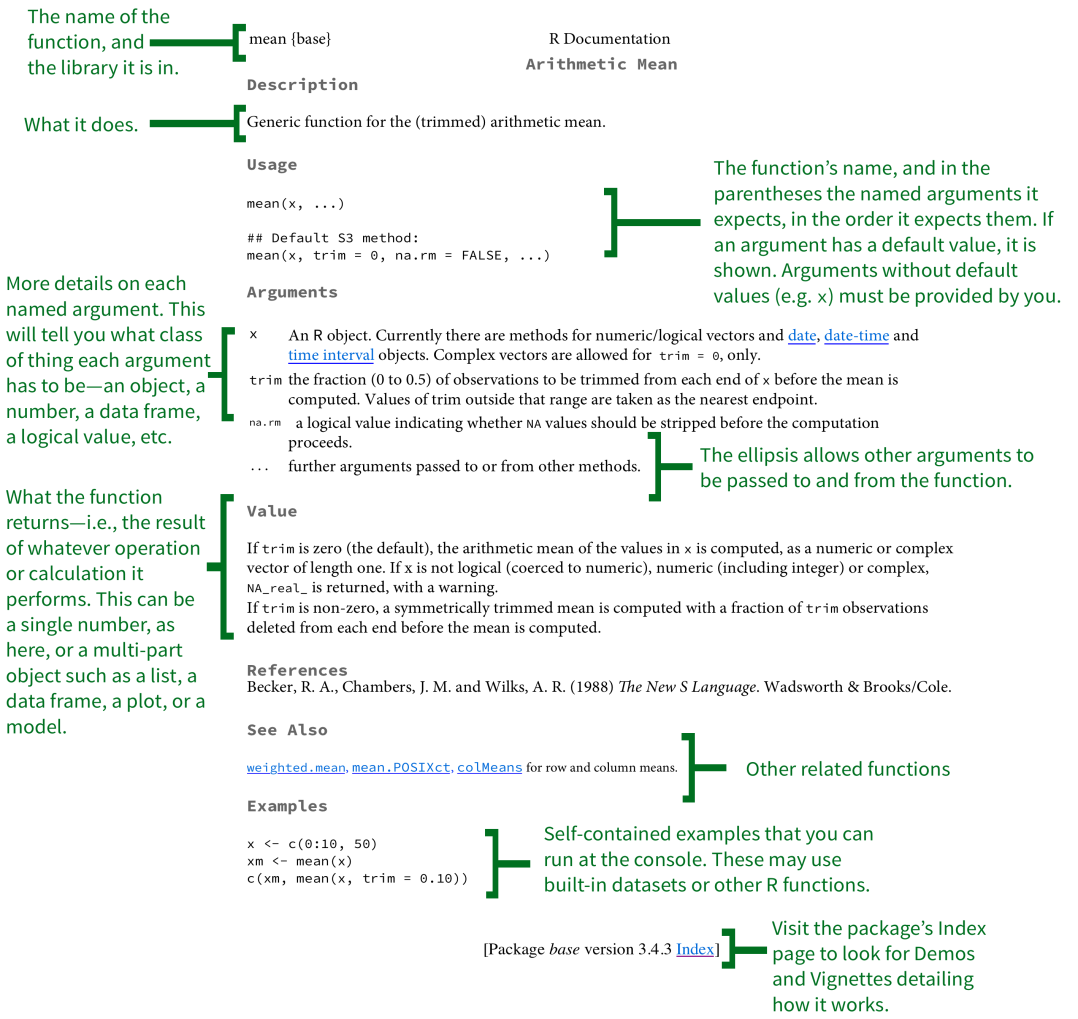


You can specify the image size using the numbers or you can grab the plot by the triangle in the bottom right corner of the preview pane and drag the plot to the desired size.

# 9 Getting help

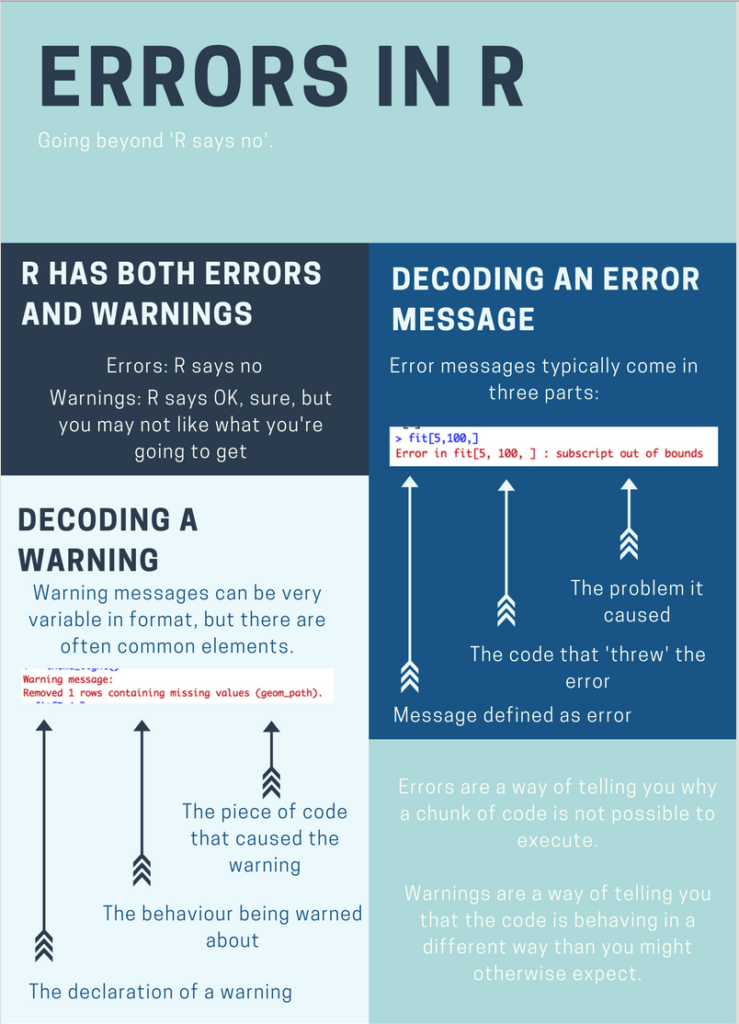
**Using the help function** Functions are documented in R’s help system, you can access the help file for a function by typing in the search bar of the help pane.

These help files can be quite daunting (and difficult to understand) but they do all follow the same structure. The image below breaks this down a bit to try and make the it a bit easier for you to actually understand how to use the help files to actually help!



More detail about the sections of the help files can be found [here](https://socviz.co/appendix.html#a-little-more-about-r)

**Error messages**

Error messages can be difficult themselves to decode, the below graphic brakes things down a bit 

you can try googling them or searching [Rseek](https://rseek.org/) a search engine specifically for R related materials.

**Still stuck?** Please do not worry. R has a steep initial learning curve, so expect to hit problems.

* Ask your classmates, they might have encountered the same or similar problem!
* Make use of **Microsoft Teams NES2303**. Ask questions of staff and **answer questions from other students**
* Check out the [#rstats](https://twitter.com/hashtag/rstats?lang=en) on twitter for useful advice.

The community or R users are extremely welcoming and supportive so just remember you’re not alone!

# 10 Further resources

## 10.1 Background theory

Please look at the **Ten-minute videos** (TMV) that provide clear background information **before** you attend each set of practicals. Understanding R requires you to understand what it is doing, and how to read its often cryptic outputs.

## 10.2 Books, online guides

See the Canvas reading list for NES2303. I have updated this with new resources for 2022, with most books in electronic format. There are also lots of excellent resources out there for learning R, some of those that we recommend are listed here and should be enough to keep you going

[learnR4free](http://learnR4free.com)

[Data science box](http://datasciencebox.org)

[Data carpentry](https://datacarpentry.org/R-ecology-lesson/index.html)

[Our Coding Club - written by students for students](https://ourcodingclub.github.io/)

These sites are for reference only.