The RStudio How-To Manual

This chapter introduces you to the statistics program RStudio, which is 'jazzed up' version of the basic but powerful R statistical programming language. In fact RStudio is the statistics package that was used to construct most of the graphs in your lecture notes! Detailed instructions on how to use RStudio to perform each of the important data analysis tasks you will encounter during this course are provided. RStudio is already installed in the Red Centre computer labs for you to use, however RStudio is FREE to download and use on your own computer, and simple instructions to install it on your own computer are given at the start of this manual.

This manual gives you the skills required to complete the weekly MATH1041 online lab exercises using RStudio. Furthermore, it will provide a useful guide when it comes to completing the computing components of your MATH1041 assessments, as well as any statistical analysis that you will do in your future careers!

In MATH1041, we will use RStudio (or just plain R if really wanted) for the assessment and the online lab exercises through Maple TA. There are many benefits to learning how to use RStudio for data analysis. Many of the data analysis tasks you are required to do in this course are much easier and much faster to do on RStudio than other software like Excel. Furthermore, in later week there will be some advanced statistical methods you will learn which take maybe 10–15 mins to perform in Excel, whereas they yet take a couple of commands in RStudio with often more useful output.

This chapter provides details on how to complete a range of tasks, as described below.

R1 General introduction to RStudio

R1.1 How to install RStudio on your home computer

RStudio is already installed in the Red Centre computer labs.

If you want to also install it on your own computer so that you can work at home, then it is easy to do: Either watch the short youtube video http://www.youtube.com/watch?v=9m3Q-gIJh4M for visual instructions, or follow the steps below

1. Go to CRAN http://cran.r-project.org/, the official R website.
2. In the section at the top of the page, titled "Download and install R", select the link corresponding to the type of computer you want to install R on.
3. Mac users: simply download the appropriate .pkg file and open it to start installing.

   Windows users: click on "base" and download the file with a name of the form R-...exe. This is the installation file, and opening it will start installing.

   Linux users: you will need to know what operating system you are running. Navigate to the directory describing your operating system, and download an appropriate R-base file. (or use your appropriate package manager commands to locate and install R).
4. Once you open the file you just need to follow the installation instructions.
5. Go to http://www.rstudio.com/products/rstudio/download/ to choose your version of RStudio appropriate for your computer, download and follow the installation instructions.

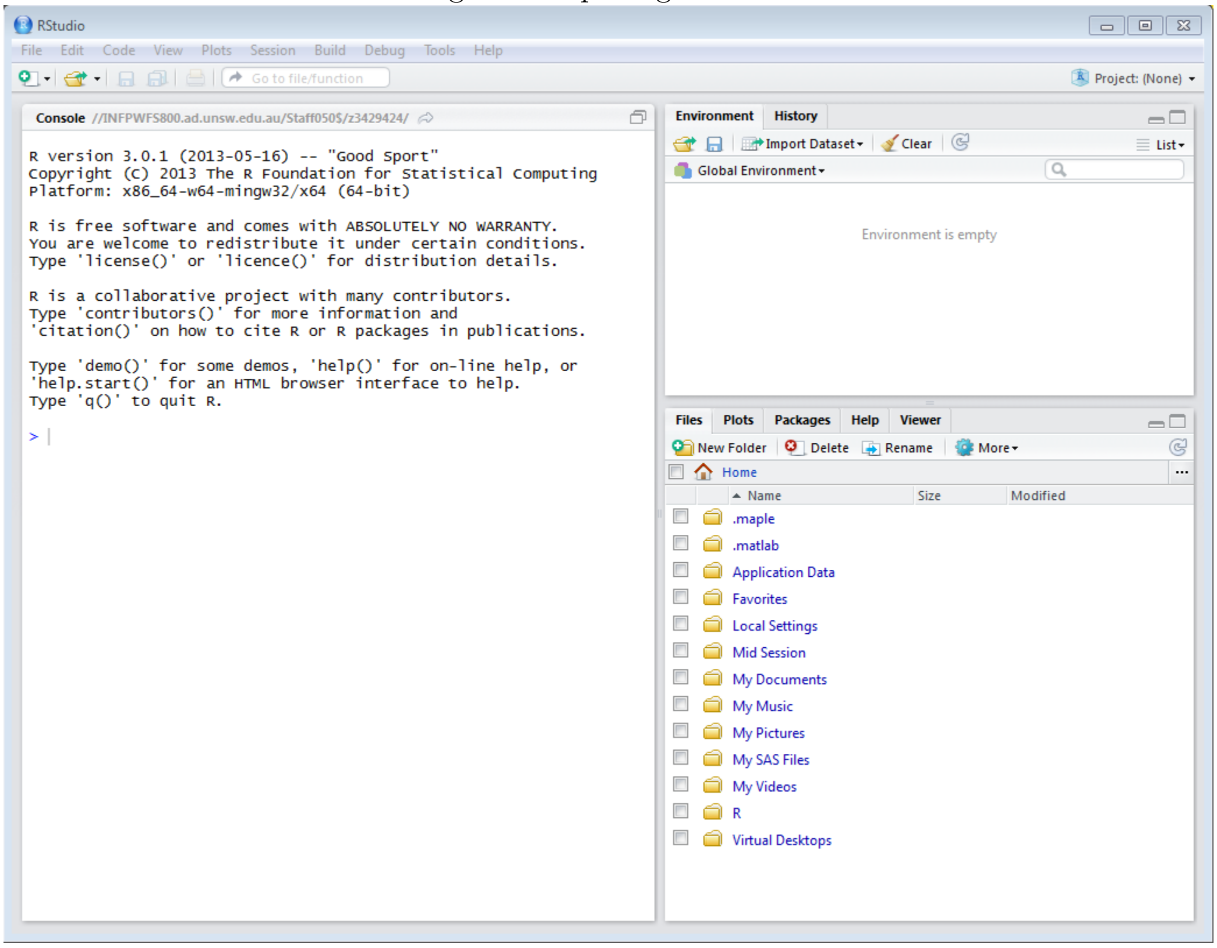
Handy tip: If you don't have your own computer and instead you often use one of several different computers, a useful trick is to install R and RStudio onto a USB stick (instead of installing it onto the computer's hard drive). A standard Rplus RStudio installation takes up less than 100M of space, so it will fit easily onto most USB sticks. Then you would be able to use RStudio on pretty much any computer that reads your USB stick! For more information, check out https://support.rstudio.com/hc/en-us/articles/200534467-Creating-a-Portable-Version-of-RStudio-for-a-USB-Drive.

R1.2 How to open RStudio

Navigate to the RStudio program in the start menu and launch the program. You can either look for the RStudio folder itself which contains the shortcut to the program, or you can type 'RStudio' in the run/search bar to look for it directly.

Once RStudio opens up (it make take a little while), you will end up up something like in Figure 1.

Figure 1: Opening RStudio



Console //INFPWFS800.ad.unsw.edu.au/Staff050$/z3429424/

```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

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

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

> |
```

Environment is empty

```
Name
.maple
.matlab
Application Data
Favorites
Local Settings
Mid Session
My Documents
My Music
My Pictures
My SAS Files
My Videos
R
Virtual Desktops
```

Like most Windows programs, there are several commands at the top such as *File*, *Edit* and so on. Below this, you should see three panels in RStudio:

- The big left panel is called the *console*. Whenever RStudio is opened, there are always some introductory message printed, informing you of the version of R you are using, how to cite R if you use in as part of a journal article and so on. At the bottom of all of this, you can see a read prompt symbol > (it may be a different color to red), followed by a blinking 'I'. Whenever it is blinking, that means the console is active, and so if you type in anything now it will be read into the console. However, if you click on either of two right panels, then the 'I' will stop blinking and become a slightly faded color. This means your console is now inactive, and anything you type won't be read in.

- The top right panel will contains several tabs. In the case of Figure 1, you can see three tabs in *Environment*, *History*, *Help*. We'll explain what *Environment* is shortly. *History* is generally not too useful, because you will learn a better way to save your work and history. *Help* is an a very helpful tab: Any time you want to search what a particular command in RStudio does, click on *Help*, type what you want in the search bar at the top right hand corner (with the magnifying glass) and press 'Enter'.

- The bottom right panel also contains several tabs. In the case of Figure 1, you can see three tabs in *Files*, *Plots*, *Packages*. *Files* is a bit like Windows Explorer: it shows everything in your "working directory" (we will discuss what this term means later). *Plots* is another useful tab, which is automatically 'clicked on' whenever you construct a plot (we shall see this later). The *Packages* tab will not be used in this course, but comes in handy when you want to download add-ons or packages to do more advanced statistical analysis.

Note: Depending the computer different tabs maybe located in different places – *e.g.* the *Help* tab may be on the bottom right panel instead of the top right. In ALL of the notes below, we shall go by the tab positions seen in Figure 1, but it should not be too hard to located these tabs in your own computer (hopefully).

R1.3 How to type data into `RStudio`

If your dataset is small, you can enter it in manually. For example, consider the dataset:

2.3 2.5 6.5 7.4 1.1 3.9

To enter it into `RStudio` and to store it as the object `dat`, type at the prompt

```
dat <- c(2.3, 2.5, 6.5, 7.4, 1.1, 3.9)
```

then press 'Enter'. You can think of the 'c' as meaning to "combine" these bunch of numbers together.

You can now type `dat` and press 'Enter' to check that `RStudio` has recorded your data correctly – it should return to you the values you entered. In addition, if you look at the *Environment* tab in the top right panel, you'll see that there is `dat` object has been created. You can now guess that the *Environment* tab basically keeps a record of every object that you have created since opening `RStudio`.

Handy tip: Note that if you created a new object which has the same name as an object that already exists in the *Environment*, then it overwrite the old object. For instance, if now type `dat <- c(1, 2, 6, 4)` and press 'Enter', and then press check out what `dat` is, you'll see the old dataset has been replaced by your new one.

Now that `RStudio` has your data stored in it, you can do some calculations to summarise the variable `dat`, such as finding the mean, median, and so on. We will do this a little later. The object `dat` will remain in the `RStudio` memory i.e., in *Environment* until you either overwrite it using another `dat<-...` command, or until you close `RStudio`. This is true of all objects that are created and stored in `RStudio`.

Handy tip: If at any time you are typing into console and the prompt changes into a +, then it means `RStudio` believes you have not finished typing. In this case you have two options: I) press the escape key `Esc` or press `Ctrl+C` you will be back in business; II) finish your typing! For instance, if you type `2+`, then you'll see `+` come up on the next line. That is because `RStudio` reckons you haven't finished, and rightly so. . . what do you want to add to 2? If you now type `3`, then you'll see the answer `5` pop up, followed by `>`, and you are back to business.

R1.4 How to import a text file into RStudio

You can import data into `RStudio` if it is available as a tab-delimited text file. All datasets we will use are available in **Computing Information** on Moodle, and most are already in this format. Importing a dataset into `R` involves three steps:

1. Save the file (as tab-delimited text) in a directory you plan to do your work in. If using a university computer, you should use the `H` drive – maybe start a `MATH1041` directory on the `H` drive and work there (`H:\MATH1041`).

   Go to Moodle and save `survey.txt` to the directory you want to work in today.

2. Tell `RStudio` which directory you will work in (`H:\MATH1041` or otherwise):
   Session. . . Set Working Directory. . . Choose Directory. . .

3. Use the `read.table` function to load the file into `RStudio`. For example, to load `survey.txt` and store it under the name `survey`, type at the prompt:

```
survey <- read.table("survey.txt", header = T)
```

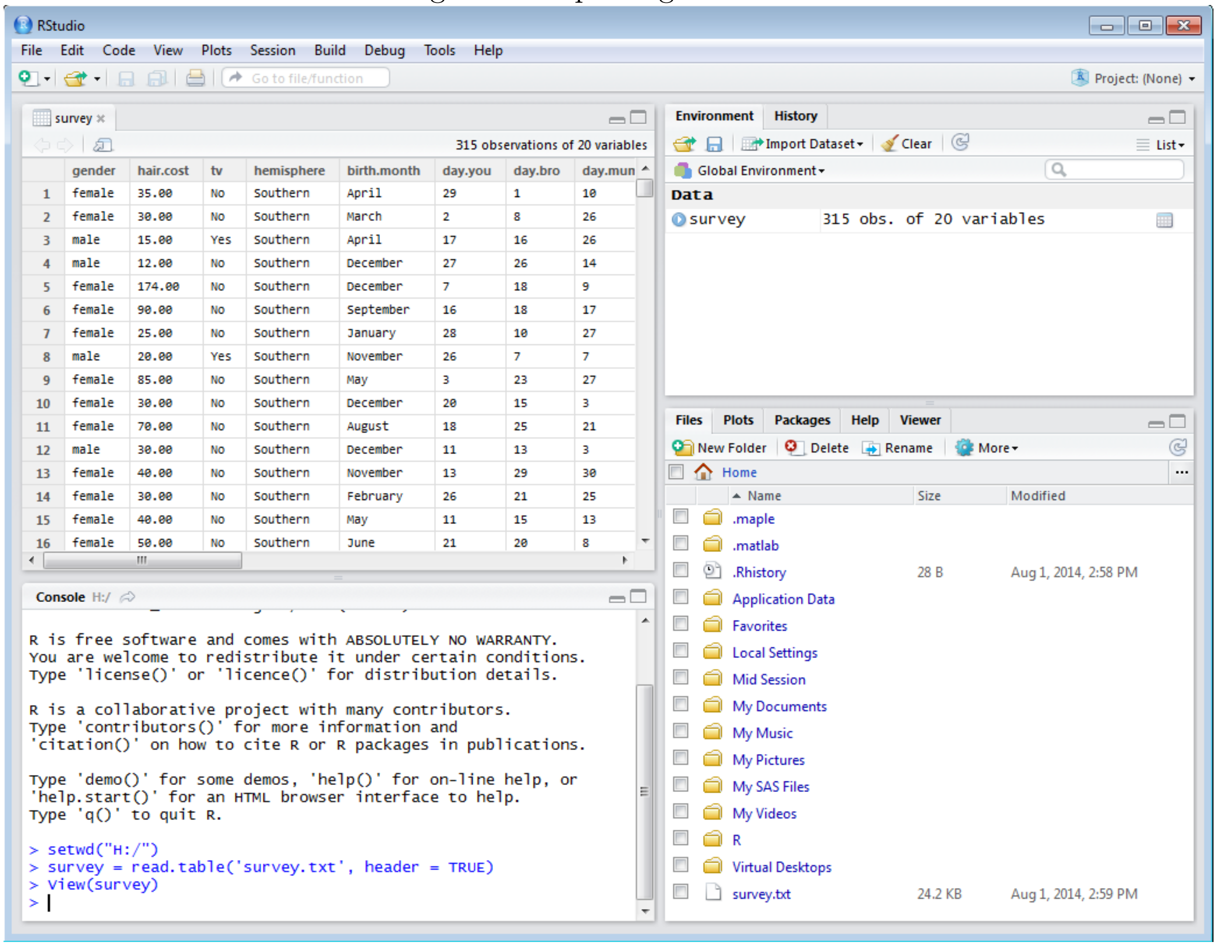
You'll notice in the *Environment* tab that there is now a new object called `survey`. This is seen in Figure 2 in the top right panel, where `RStudio` also informs us how many rows and columns there are in `survey`. What's more, if you actually click on this `survey` in the *Environment* tab, you'll notice that a preview of the survey dataset as read in by `RStudio` pops up in the top left panel (in turn shrinking the console to the bottom right panel). Again, we this is Figure 2. If you want to close the preview of the dataset, just click the '×' next to the newly opened `survey` view.

Tab-delimited files have columns of the data separated using the Tab key. `RStudio` can import a few other data formats as well as tab-delimited text, but this is the format we will use in the course. If your data are available in a different format, *e.g.* as an Excel spreadsheet, you will need to save it as a text file before importing. Note the text file needs to contain nothing except the variable names (in a "header row") and the data, with no missing values, so sometimes there is a bit of work to do in order to get your dataset into this format so that `RStudio` can read it in without an error.

Handy tip: We always recommend that THE FIRST thing you do once opening `RStudio` is to set your working directory to the place where you plan to store all your work. (On university computers, this should be somewhere on the `H drive`.) Start in a place where everything you will need is located, and life's a breeze.

Handy tip: It is good practice to give objects in `RStudio` informative names. For example in the above, we called the dataset we read in `survey` because it was a survey of

Figure 2: Importing a text file



315 observations of 20 variables

	gender	hair.cost	tv	hemisphere	birth.month	day.you	day.bro	day.mun
1	female	35.00	No	Southern	April	29	1	10
2	female	30.00	No	Southern	March	2	8	26
3	male	15.00	Yes	Southern	April	17	16	26
4	male	12.00	No	Southern	December	27	26	14
5	female	174.00	No	Southern	December	7	18	9
6	female	90.00	No	Southern	September	16	18	17
7	female	25.00	No	Southern	January	28	10	27
8	male	20.00	Yes	Southern	November	26	7	7
9	female	85.00	No	Southern	May	3	23	27
10	female	30.00	No	Southern	December	20	15	3
11	female	70.00	No	Southern	August	18	25	21
12	male	30.00	No	Southern	December	11	13	3
13	female	40.00	No	Southern	November	13	29	30
14	female	30.00	No	Southern	February	26	21	25
15	female	40.00	No	Southern	May	11	15	13
16	female	50.00	No	Southern	June	21	20	8

Data

survey 315 obs. of 20 variables

	Name	Size	Modified
	.maple		
	.matlab		
	.Rhistory	28 B	Aug 1, 2014, 2:58 PM
	Application Data		
	Favorites		
	Local Settings		
	Mid Session		
	My Documents		
	My Music		
	My Pictures		
	My SAS Files		
	My Videos		
	R		
	Virtual Desktops		
	survey.txt	24.2 KB	Aug 1, 2014, 2:59 PM

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

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

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

> setwd("H:/")
> survey = read.table('survey.txt', header = TRUE)
> View(survey)
> |
```

MATH1041 responses. Using generic names like `dat` is uninformative and will overwrite your data next time you read something in and call it `dat`!

To check out the `survey` data (as well as to just check that `RStudio` read things in correctly), there are some useful commands you can use. First, if you just want to view the entire file, then you can type:

```
survey
```

Although clicking on the object in the *Environment* tab is probably more useful. A good way to get a sense for what is in a dataset is to use the `str` function:

```
str(survey)
```

or maybe `summary`:

```
summary(survey)
```

If you just want the first 10 rows of the dataset, then type:

```
survey[1:10,]
```

The

```
1:10
```

is a shorthand in `RStudio` for writing `c(1,2,3,4,5,6,7,8,9,10)`. Alternatively, if you want the first second and fourth columns:

```
survey[,c(2,4)]
```

If you want a more general glimpse of what the beginning or 'head' of the dataset looks like:

```
head(survey)
```

R1.5 How to import data from Excel into `RStudio`

We will import the `smokePregnant.xls` dataset (which an Excel spreadsheet) into `RStudio`, by first saving it as a tab-delimited text file, then importing this file into `RStudio`. First, download `smokePregnant.xls` from `Moodle` and save it to your `H` drive.

To save `smokePregnant.xls` as a tab-delimited text file:

- Open `smokePregnant.xls` in `Excel`. Then highlight, right click and select delete to remove the top five 'useless' rows of the dataset, so that row 1 contains the variable names, and the rows immediately below contain the data.
- Use **File. . . Save as. . .** to save the data as a Text (Tab-delimited) (*.txt) file titled `smokePregnant.txt` in your `H` drive. This document is in a form that `RStudio` can read.
- Now open `RStudio` set your working directory to the `H` drive, and then type:
  ```
  smoke.preg <- read.table("smokePregnant.txt", header=T)
  ```
  This command stores the dataset from `smokePregnant.txt` as the object `smoke.preg`, and uses the first row of the dataset to label each variable.

To check that `RStudio` has stored the dataset correctly, type

```
smoke.preg
```

or click on the `smoke.preg` object in the *Environment tab.*

R1.6 How to save your work in `RStudio`

Because `RStudio` is a command-line language, you don't usually need to save every object that has been created. Instead, it is usually sufficient to just save the commands you typed into `RStudio`. We'll show you two ways to do this. The first is simple but not very practical. The second requires a little bit more fiddling but is AWESOME.

Saving your commands

To keep a history of every command you have typed during since opening `RStudio`, click on the *History* tab in the top left panel. You will now see a record of all the commands that you types in. Now click on the disk symbol, and save this record in your `H` drive, giving it any name you want followed by the '.txt' format.

A text file will be created, which will list every single command you have typed in your current `RStudio` session. In principle this is really cool, because you can then rerun your analyses at a later date by simply opening this text file, and copying and pasting the commands into `RStudio`. In other words, if you ever need to go "back to the drawing board" to check out some commands or do some revision (for a lab test maybe!), then it is easy to repeat your analysis on `RStudio` – just copy and paste all commands from this history file.

However, constantly flicking back and forth copying and pasting is very tedious. Also, you'll see that the text file has all the text in black. Can you imagine how hard it would be if you had hundreds of commands in your history file and you are just looking for one of them? We now look a much better way of doing things, which is virtually what all practicing statisticians using `RStudio` do.

R1.7 Using and saving `R` script files

Rather than saving typing all your commands in the console (left pane) and then saving a record before you exit, a much better way of doing things is to write all your commands from the beginning in a `R` script file. This approach offers a number of advantages, which you shall witness below.

First, once you have opened `RStudio` (and set your working directory to `H` drive, of course), click on the **File. . . New. . . R Script**. At this point, what you'll see happen is that the console on the left pane will shrink, and a `R` script file will open up as a new, top left panel names with the name `Untitled 1`; see Figure 3. Note that previously the top left panel was also where datasets were previewed; see Figure 2.

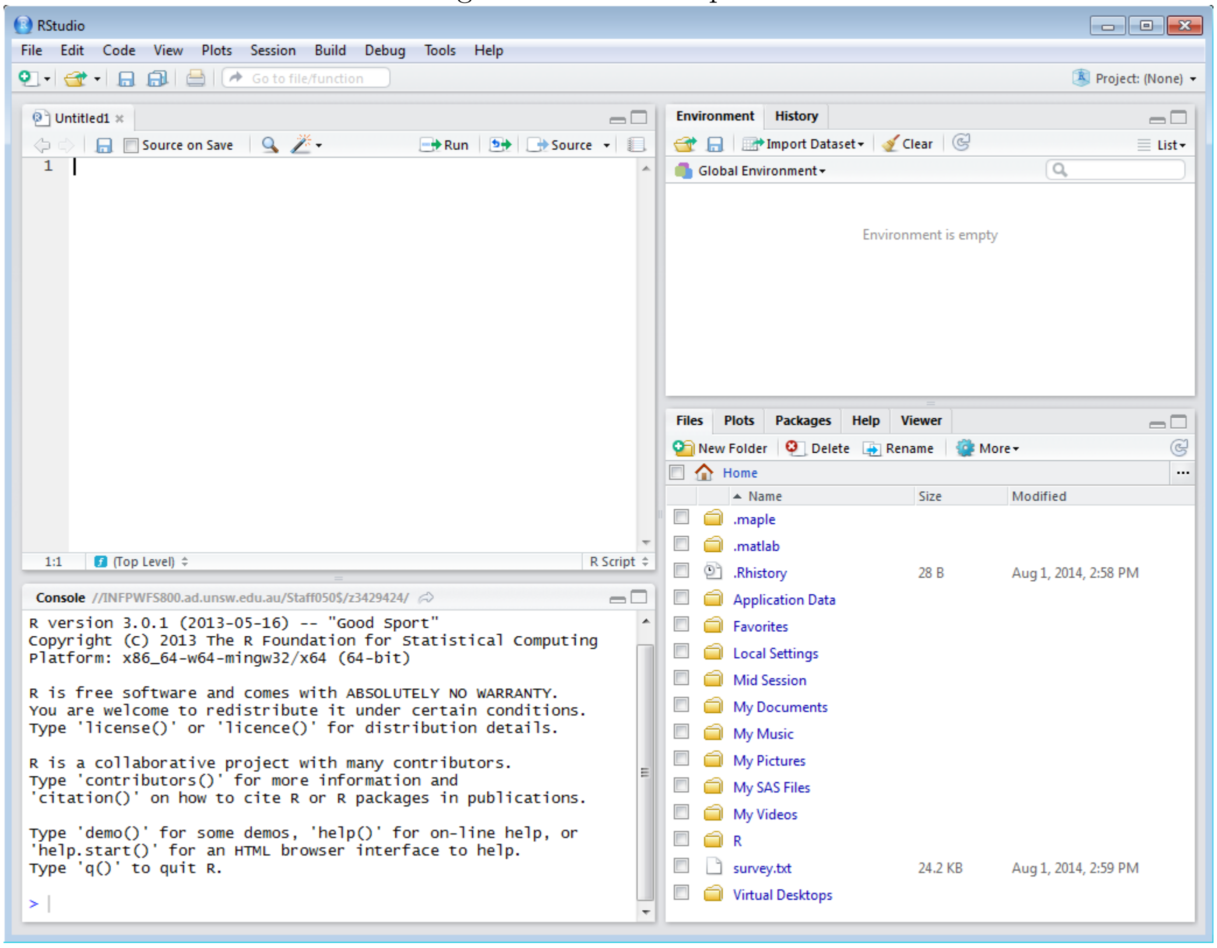
This is now your `R` script file, where you can write whatever you need. The key thing however is with this approach, rather than copying and pasting commands into the console on the panel below, you can type the commands in the `R` script file and directly execute from there! For example, in Figure 4, you can see that a line of data called marks has been typed into the `R` script file. Note this has NOT been run yet – it's simply sitting in the script file.

To run this command, ensure that the blinking 'I' is somewhere on that line (anywhere on that line will do). Then move your cursor over to the 'Run' button on the right and press it. Once you've pressed you, you'll notice that the command has been run in the console on the bottom left panel. That's the beauty of an `R` script file!

If there's a whole bunch of commands you want to run at once in your `R` script file, then what you can do in highlight them all (you'll notice the highlighted lines will be in highlighted blue), and then press the 'Run' button.

Handy tip: If you prefer to use the keyboard, then instead of pushing the 'Run' button, you can press 'Ctrl + Enter' while the blinking 'I' is somewhere on that line.

Figure 3: New R script file



```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

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

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

>
```

Name	Size	Modified
.maple		
.matlab		
.Rhistory	28 B	Aug 1, 2014, 2:58 PM
Application Data		
Favorites		
Local Settings		
Mid Session		
My Documents		
My Music		
My Pictures		
My SAS Files		
My Videos		
R		
survey.txt	24.2 KB	Aug 1, 2014, 2:59 PM
Virtual Desktops		

Aside from running commands directly off it, another major advantage of an R script file is syntax assistance:

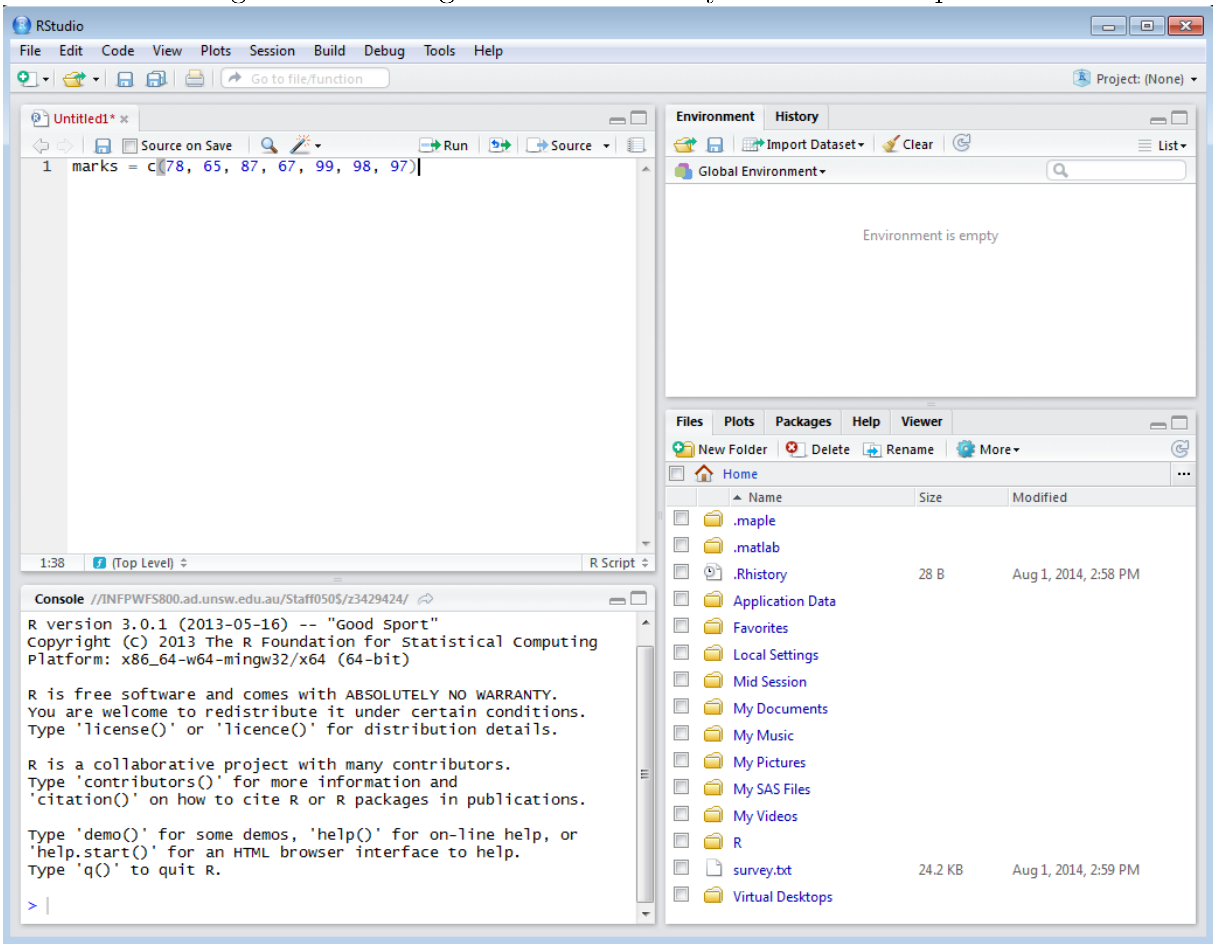
- RStudio will automatically highlight corresponding brackets. This is seen in Figure 5, where there command is designed to calculate $1/(\sqrt{(2)}/(5 \times 6))$. You'll see that my 'I' is at to the right of the final ')' bracket, and what RStudio has done is highlight the corresponding '(' bracket.

- Bracket completion. This is kind of related to the above. If you type an (is an R script file, then RStudio will automatically put a ')' to complete bracket AND move the 'I' blinker to in between those brackets.

- Basic syntax coloring. R scripts files come with some basic coloring to help differentiate things. This is illustrated in Figure 6 (don't worry about the commands for now...in time you'll learn what they are doing!). Most text is colored in black, all symbols are colored in purple, and anything inside quotation marks are colored in blue. There's also some green text, which we'll talk about now.

- You may also want to write titles for different sections and comments in the R script

Figure 4: Running commands directly from the R script file



```
marks = c(78, 65, 87, 67, 99, 98, 97)
```

```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

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

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

> |
```

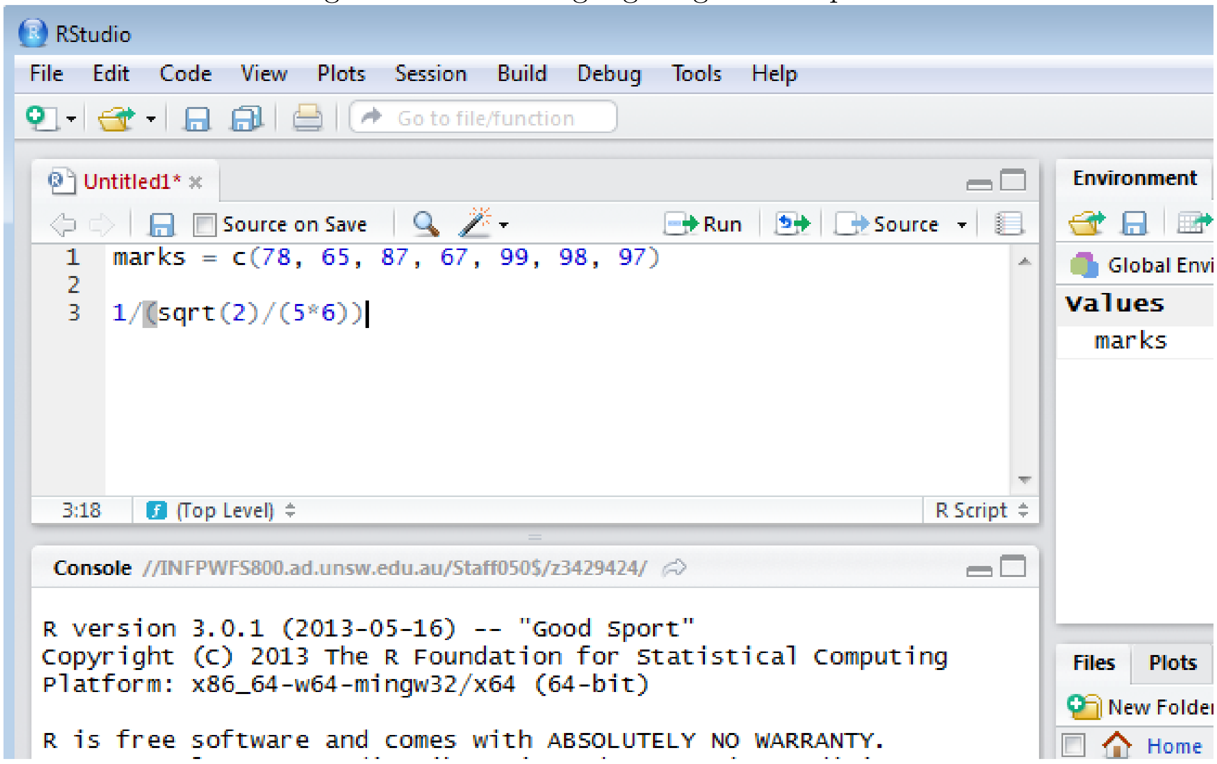
Name	Size	Modified
.maple		
.matlab		
.Rhistory	28 B	Aug 1, 2014, 2:58 PM
Application Data		
Favorites		
Local Settings		
Mid Session		
My Documents		
My Music		
My Pictures		
My SAS Files		
My Videos		
R		
survey.txt	24.2 KB	Aug 1, 2014, 2:59 PM
Virtual Desktops		

file as you go along, to remind yourself what you are doing and what the results mean. This is really useful if you plan to return to these script files later on (maybe for revision!). To put comments in, use the special comment hash character #. Anything you write after # and which is before a new line in the R script file, will appear in green and be a comments. This is seen in Figure 6, where a title 'Script for Week 4' has been given to the file, and some comments and sub-headings here.

Note that a double hash ## has been used in Figure 6 – there's reason for this, just force of habit!

Once you have types all your commands, comments, and run some stuff off your R script file, it's time to save it (if you want to, of course). To do this, make sure your there is blinking 'I' somewhere in the R script file – this ensures that it is the script file which is active and not the console or the other panels. Then you can either go to **File...Save** or click the disk symbol below `Untitled 1`. Then navigate to your H drive, give a name to your R script file and click 'Save'. The only thing you have to remember is that instead of a '.txt' format, please ensure you use a '.R' ending. This ensure the saved file is an official R script file. Once your file is saved, you'll notice that `Untitled 1` is replaced by your filename – *e.g.* `week4solution.R` in Figure 6.

Figure 5: Bracket highlighting in R script files



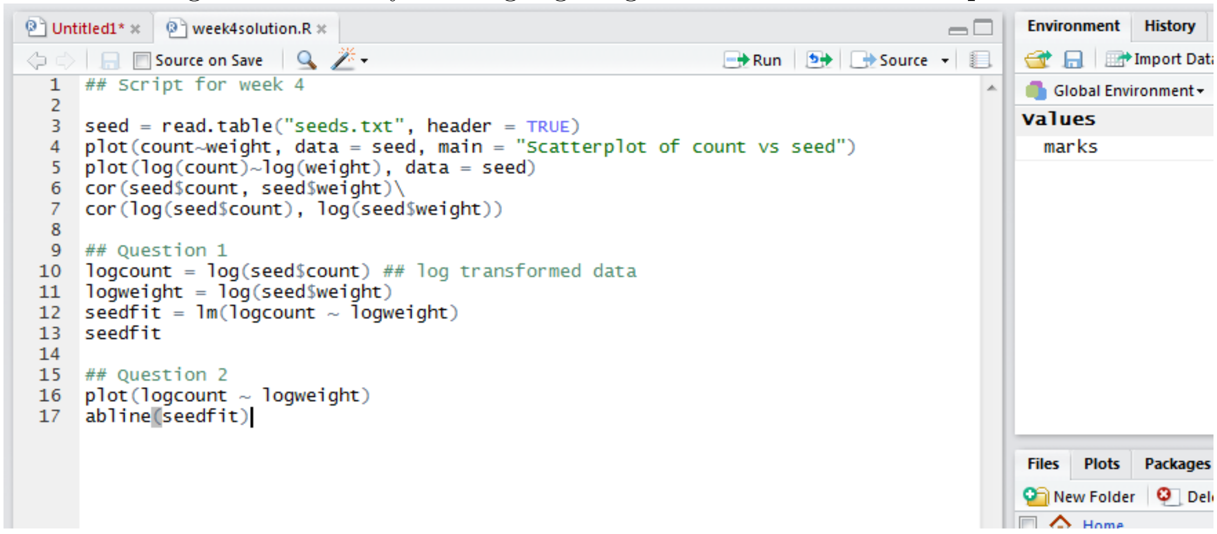
```
marks = c(78, 65, 87, 67, 99, 98, 97)

1/(sqrt(2)/(5*6))
```

```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
```

Figure 6: Basic syntax highlighting and comments in R script files



```
## Script for week 4

seed = read.table("seeds.txt", header = TRUE)
plot(count~weight, data = seed, main = "Scatterplot of count vs seed")
plot(log(count)~log(weight), data = seed)
cor(seed$count, seed$weight)\
cor(log(seed$count), log(seed$weight))

## Question 1
logcount = log(seed$count) ## log transformed data
logweight = log(seed$weight)
seedfit = lm(logcount ~ logweight)
seedfit

## Question 2
plot(logcount ~ logweight)
abline(seedfit)
```

Finally, note that you can have multiple R script files opened at once. In Figure 6 for instance, there is an `Untitled 1` as well as an `week4solution.R` script file opened. This is analogous to having multiple tabs in Firefox, and comes in handy if you are working on several R script files at once (which we won't be doing in this course).

Handy tip: Another awesome benefit of using R script files is that `RStudio` is automatically linked to it. That is, you can go to Windows Explorer, navigate to your H drive, right click on an R script file which you have saved previously, then choose to open with RStudio directly! This saves you from having to open RStudio and then look for the script file. The other major advantage of opening R script files this way is that RStudio will AUTOMATICALLY set the working directory to the directory where you opened the script file from i.e., your H drive.

Copying numerical output

Often, when preparing a document summarizing the results of an analysis, it is most convenient to present results using a Word processor such as Microsoft Word. There are two types of RStudio output that you might want to copy from RStudio across to Word when writing up a report or assignment, as below.

To do this, we will use standard copy-and-paste techniques. That is, we will highlight what we want to copy, then go to **Edit...Copy**, then we will move where we want to paste the output and go **Edit...Paste**.

- First, you will need some RStudio output that you want to copy across to Word. For example, calculate a five-number summary for the travel time variable form the survey dataset as described in Section R3.2: Numerical summaries of a quantitative variable.
- To tell the computer what you want to copy from RStudio, highlight the output by starting at the top left corner of the output, then holding the mouse button down while dragging the mouse across the output, such that the whole of the output to be copied is highlighted. Then go to **Edit...Copy**.
- Open a (new) word document. To save the five-number summary from section R3.2 into an empty document, simply go to **Edit...Paste**.

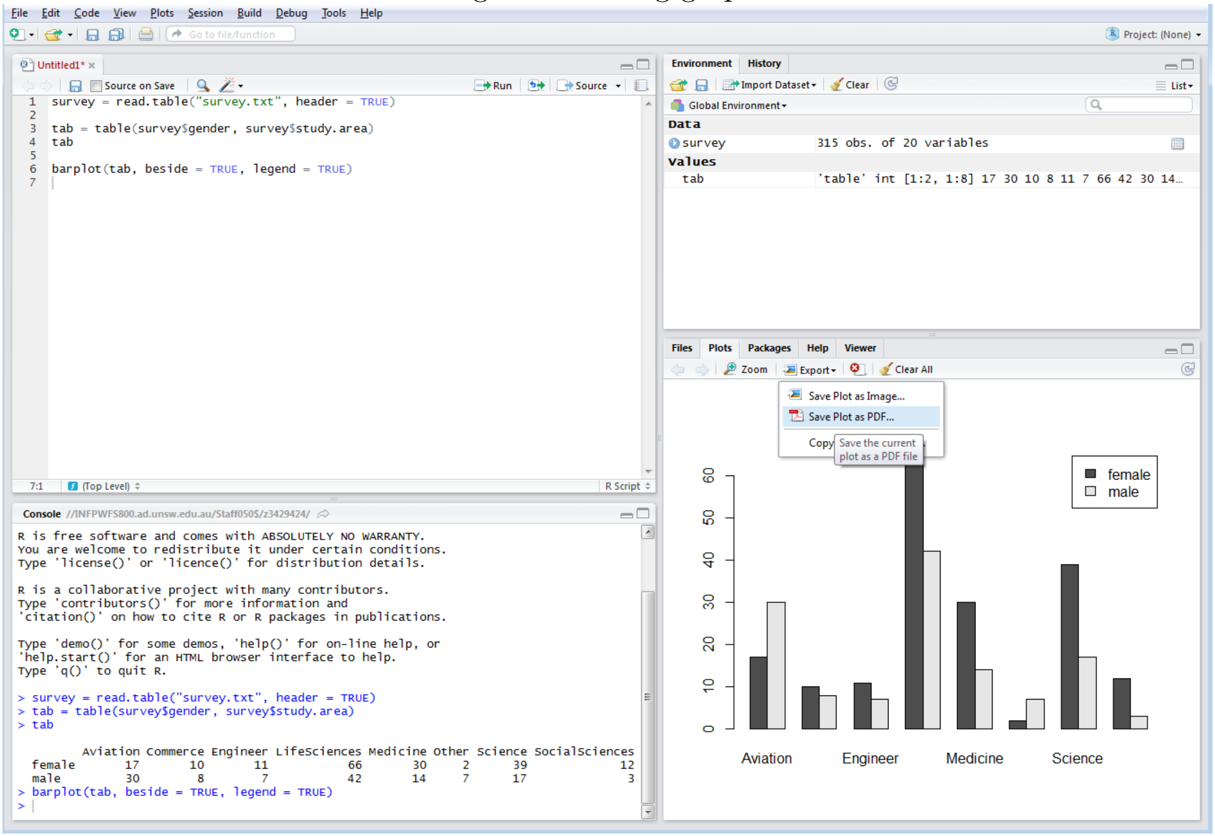
Constructing and saving Graphs

RStudio makes it very easy to save any plots you have constructed. In the below, you will see how to save plots in PDFs format. Once you have done this, then you can open the plot in Acrobat Reader (for instance) and copy and paste sections it into Microsoft Word, or print them, or whatever you want.

- First, create a graph. For example, work through section R2.2: How to graph one quantitative variable using RStudio.
- When relevant command to create a graph is run (from your R script file), you notice the bottom right panel jumps to the *Plots* tab, and your plot is there is all its glory!
- To save the graph as a PDF, click on **Export...Save as PDF**. A new window will not open up. You can change the size and orientation plot, although usually the default is good enough. See Figure 7.

- Importantly though, click on **Directory** and navigate to your `H` drive if you're already there (the default directory to save graphs is your working directory).
- Change the filename to something useful or just leave as it is as `Rplot`. Note however you do NOT need to put a .pdf format at the end of the filename – `RStudio` already knows you are exporting as pdf!
- Click 'Save' and your plot is saved!

Figure 7: Saving graphs



```
survey = read.table("survey.txt", header = TRUE)

tab = table(survey$gender, survey$study.area)
tab

barplot(tab, beside = TRUE, legend = TRUE)
```

```
Data
survey          315 obs. of 20 variables
Values
tab             'table' int [1:2, 1:8] 17 30 10 8 11 7 66 42 30 14…
```

```
Console //INFPWFS800.ad.unsw.edu.au/Staff050$/z3429424/
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

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

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

> survey = read.table("survey.txt", header = TRUE)
> tab = table(survey$gender, survey$study.area)
> tab

         Aviation Commerce Engineer LifeSciences Medicine Other Science SocialSciences
  female       17       10       11           66       30     2      39             12
  male         30        8        7           42       14     7      17              3
> barplot(tab, beside = TRUE, legend = TRUE)
> 
```

Handy tip: Once you've constructed your plot, if you want to look at it in more detail, click the **Zoom** button and a bigger image will pop up. Close it when finished.

Handy tip: `RStudio` is smart enough to keep a records of all the plots you've made in a session. You can check previous plots by clicking on the left and right arrows.

R1.8 Getting help on `RStudio`

There are a few different ways to find help on `RStudio`, if you want more information on anything:

- On the *Help* tab in the top right panel, search for the command of interest in the search bar with the magnifying glass.

- To search for help on a particular technique, use the `help.search` command, *e.g.* to find out what functions are available for constructing histograms, type
  `help.search("histogram")`

  in your `R` script file and run it.

- If you know about a particular function enter a question mark '?' followed by the command *e.g.* to find out more about the `hist` command, type
  `?hist`

- Google is your friend! There are also awesome brief introduction manuals out there – *e.g.* 'The `R` Guide', which you can download from `http://cran.r-project.org/doc/contrib/Owen-TheRGuide.pdf`

R1.9 Exiting `RStudio`

At the end of any `RStudio` session, after you have saved your `R` script file, quit `RStudio` by going to **File. . . Exit**, or pressing the '×' at the top right hand corner. Note that you'll then be asked whether you want to save your workspace i.e., all the objects located in the *Environment* tab; see Figure 8. You can safely press DON'T SAVE, because you have already got a history of your commands in your `R` script file – any objects can always be recreated when by running commands off the script file.

R1.10 Quick fire steps

So here is a list of quick fire steps you are recommended to run:

1. Open `RStudio`, or go to an `R` script file and open with `RStudio` on this.

2. Set working directory via **Session. . . Set Working Directory. . . Choose Directory. . .** . If you opened an `R` script file in Step 1 then you can skip this step.

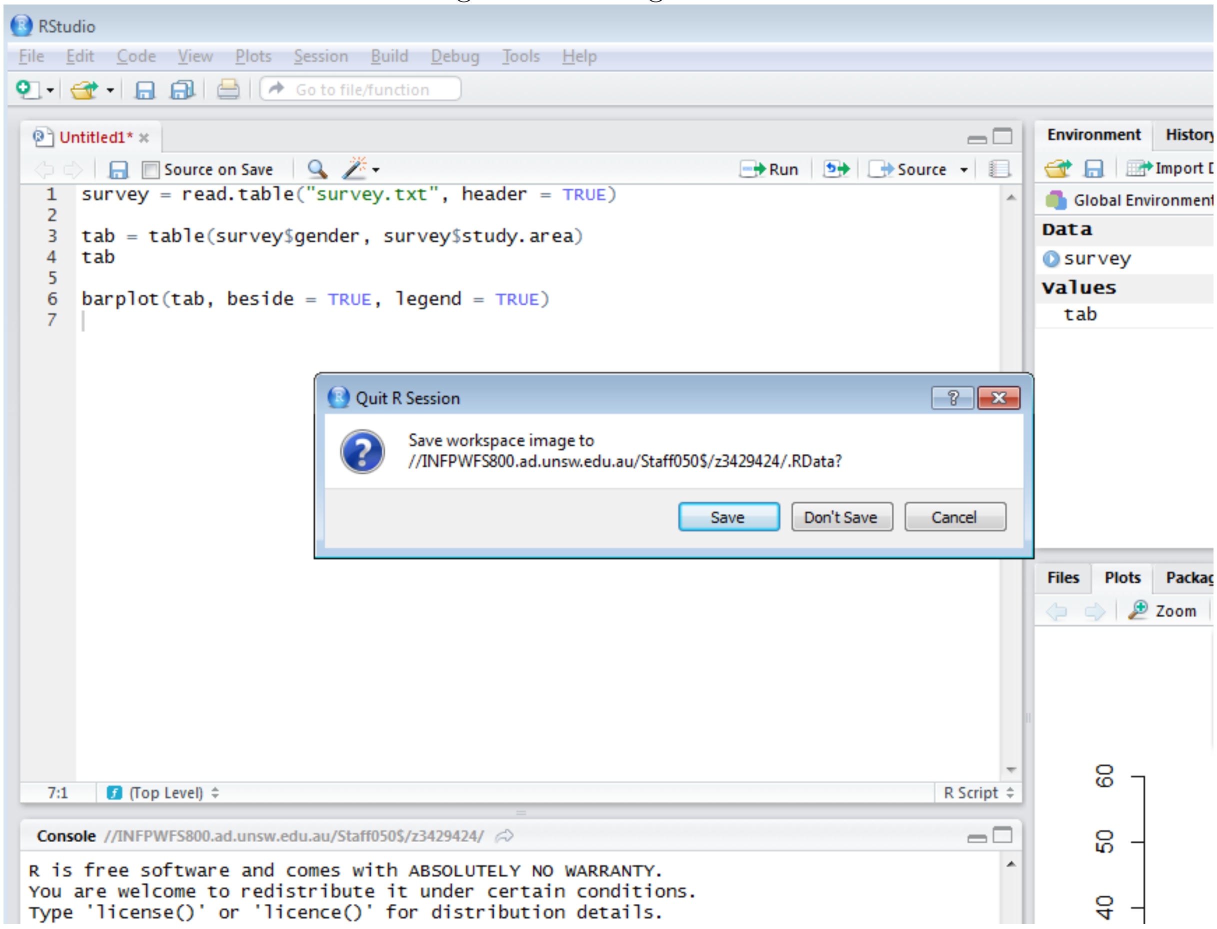
3. Open a new `R` script file via **File. . . New. . . R Script**. If you opened an `R` script file in Step 1 and you want to work on this now, then you can skip this step. Remember you can have multiple `R` script files open at once, much like multiple tabs in Firefox.

4. Type whatever commands you want in the `R` script files, and run them by pressing 'Run' or pressing 'Ctrl + Enter'.

5. Save your `R` script files by going to **File. . . Save**.

6. When you're finished, go to **File. . . Exit**, and click 'No' to saving the workspace.

Figure 8: Exiting RStudio


```
survey = read.table("survey.txt", header = TRUE)

tab = table(survey$gender, survey$study.area)
tab

barplot(tab, beside = TRUE, legend = TRUE)
```

Quit R Session

Save workspace image to //INFPWFS800.ad.unsw.edu.au/Staff050$/z3429424/.RData?

Save | Don't Save | Cancel

Console //INFPWFS800.ad.unsw.edu.au/Staff050$/z3429424/

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

R2 Summarising data graphically using RStudio

R and RStudio is a powerful graphical tool – you can create all sorts of different graphs, usually pretty easily too! In particular, it is easy to construct boxplots and normal quantile plots using RStudio, whereas this is very difficult when using Excel. The one drawback compared to Excel is that to make graphs "picture perfect" with titles and axis labels in the right place etc. . . is rather tricky (more like an 'art form'!). In this course though, we want to have appropriately labelled graphs, but not a 'perfect work of art'.

Note also that for all of the below plots, you can change the labels of the x-axis, the y-axis or the main title. To do this, use the `xlab`, `ylab` and `main` options respectively. This is illustrated below for bar charts.

R2.1 How to graph one categorical variable using RStudio

To construct a bar chart of the `gender` variable from the `survey` dataset, type either
`plot(survey$gender)`

or
`barplot(table(survey$gender))`

where the `table()` function is used to create a table of frequencies, i.e. it counts how many times each value occurs and puts the results in a table.

The syntax `A$B` sign means “in dataset A, find a column called B”. If column B can be found, then the command will run without problems. However if there is no column B in dataset A, then you’ll errors popping up in the console on the bottom right panel.

You can then improve your bar graph: change the names of the bars to “Female” and “Male”, add the main title “A bar chart of the gender of MATH1041 students”, label the x-axis “Gender”, label the y-axis “Counts” and change the color of the bars to dark red:

```
barplot(table(gender), names.arg=c("Female","Male"), main="A bar chart of the
gender of MATH1041 students", xlab="Gender", ylab="Counts", col="darkred")
```

Could you modify the above so that the y-axis has the label “Frequency”?

R2.2 How to graph one quantitative variable using RStudio

To graph the `travel.time` variable from the `survey` dataset as a boxplot, type
`boxplot(survey$travel.time)`

For a histogram:
`hist(survey$travel.time)`

R2.3 How to graph one categorical variable against another using RStudio

A clustered bar chart is an effective tool for displaying the relationship between two categorical variables. RStudio can construct a clustered bar chart from a table of frequencies. To construct a table of frequencies of the relationship between gender and study area for the MATH1041 `survey` data, and store it as the object `tab`:
`tab <- table(survey$gender, survey$study.area)`

Have a look what the created object, `tab`, looks like, by typing `tab` and running it.

Now to construct a clustered bar chart based on this table, including a legend:
`barplot(tab,beside=T,legend=T)`

The `T` is shorthand of `TRUE`. In this case, this command is telling RStudio to put the bars corresponding to ‘male’ and ‘female’ next to each other, and to make a legend.

Alternatively, if you have already summarized your data in a table and you want to enter your table of frequencies directly into RStudio and store them in a `table`, you can do this as described in Section R3.3.

R2.4 How to graph a quantitative versus a categorical variable using RStudio

To construct comparative boxplots for the hair cut costs of different genders, type any **one** of the following:

```
plot(survey$hair.cost~survey$gender)
boxplot(survey$hair.cost~survey$gender)
plot(hair.cost~gender, data=survey)
```

Note that most functions have an optional `data=` argument so you don't have to use the "`survey$`" form every time you want to use a variable from within the dataset `survey`.

The "~" stands for "against", that is, `plot(hair.cost~gender, data=survey)` means "plot `hair.cost` against `gender`, for the dataset `survey`." In other words, for each category of gender, make a boxplot of hair.cost.

R2.5 How to graph one quantitative variable versus another quantitative variable using RStudio

To construct a scatterplot of `hair.cost` against `travel.time` from the `survey` dataset, use the command:

```
plot(survey$hair.cost~survey$travel.time)
```

or equivalently, use:

```
plot(hair.cost~travel.time, data=survey)
```

Note the use of the "~" again. This time is means "plot `hair.cost` against `travel.time`" In other words, hair cut cost goes on the y-axis and travel time on the x-axis.

Adding a trend line to a scatterplot

Continuing from the previous section:

```
reg <- lm(hair.cost~travel.time, data=survey)
abline(reg)
```

The `lm` command fits a least squares regression line. The `abline` command is a general function for adding straight lines to a scatterplot.

R2.6 How to produce a normal quantile plot using RStudio

If you have the `survey` dataset loaded, you can produce a normal quantile plot of hair cost using:

```
qqnorm(survey$hair.cost)
```

If your data were perfectly normally distributed, the points would lie on a straight line. You can add this straight line to your normal quantile plot using:

`qqline(survey$hair.cost)`

Have a look at the normal quantile plot. How well do you think the hair cost data approximate normality? If you think the data are skewed, are they left-skewed or right-skewed?

R3 Summarising data numerically using `RStudio`

The below methods will be illustrated using the `dat` dataset that we entered manually in Section R1.3: How to type data into `RStudio`, and the `survey` dataset, which we imported from the `survey.txt` file in Section R1.4: How to import a text file into `RStudio`.

R3.1 Numerical summaries of a categorical variable using `RStudio`

The key function for numerically summarizing a categorical variable is the `table` function. We will illustrate the use of this function on the `gender` variable from the `survey` dataset.

After importing the `survey.txt` dataset and storing it as `survey`, type:
`survey$gender`

This command displays the raw data stored in the `gender` column of the `survey` dataset. Note that entries are either `Male` or `Female`, hence this variable is categorical. Note also that there are hundreds of entries, so we would like to create a numerical summary to make sense of them all!

Use the `table` function to numerically summarise the `gender` variable as follows:
`table(survey$gender)`

This returns a frequency table of the counts of the number of responses in each category.

It is also possible to use the `summary` function to obtain similar output.

R3.2 Numerical summaries of a quantitative variable using `RStudio`

Calculating numerical summaries of quantitative variables is pretty easy! For the `travel.time` variable from the `survey` dataset, for instance,

- To calculate the mean, type: `mean(survey$travel.time)`
- To calculate the standard deviation, type: `sd(survey$travel.time)`
- To calculate the median, type: `median(survey$travel.time)`
- To calculate the 25th percentile (otherwise known as the first quartile), type: `quantile(survey$travel.time,0.25)`

- To calculate a five number summary, type `summary(survey$travel.time)`. Note that this actually returns a six number summary – it throws in the mean as well for good measure. It also calculates the quartiles slightly differently to how we did it in class, so if you do the calculations by hand you won't get exactly the same answers as `RStudio`.

All of these commands will work for any quantitative variable. For example, if you typed in `dat` at Section R1.3, you could use `summary(dat)` for a five-number summary.

R3.3 How to numerically summarise the relationship between two categorical variables using RStudio

To numerically summarise the relationship between categorical variables, use the `table` function. Remember we also used this command for summarizing one categorical variable.

For example, we might wish to construct a table of frequencies to summarise the relationship between the `gender` and `study.area` variables, to look at how study area varies between males and females. To do this, type:

```
table(survey$gender, survey$study.area)
```

Handy tip: Remember we actually ran this command earlier in Section R2.3: How to Graph one Categorical Variable Against Another. The difference there was that we labeled the resulting table `tab`, because we wanted to construct a barplot from this table.

How to type in a table of frequencies

Sometimes, you may have a table of frequencies that has been calculated in another software package or given to you on paper, which you want to use for analyses in `RStudio`. For instance, you may want to construct a clustered bar chart as shown in Section R2.3: How to Graph one Categorical Variable Against Another.

Consider the following example:

	Virgin Blue	QANTAS
0-3 min late	22	11
3-15 min late	6	21
$>$ 15 min late	0	2

To enter the data into `RStudio` and store in a `table` called `tab`. Try running the set of commands:

```
Virgin <- c(22, 6, 0)
QANTAS <- c(11, 21, 2)
tab <- as.table( cbind(Virgin, QANTAS) )
row.names(tab) <- c("0-3", "3-15", ">15")
```

In brief, the strategy is to 1) type the data in on a per column basis, so here we have two columns of data, 2) use `as.table(cbind(...))`. The command `cbind()` tells `RStudio` to combine my columns together, while putting `as.table()` outside of this informs `RStudio` to treat this 'thing' as a table, 3) `row.names(tab) <-...` is not actually required, but it almost always helps to label for each row in your table is talking about, in this case the # of minutes late.

Handy tip: Remember with `R` script files, you can run a whole bunch of commands that are different lines simultaneously, by highlighting them all and then clicking 'Run' or pressing 'Ctrl + Enter'.

After running all of the above commands, you can type `tab` and then run this to have a look at your pretty table! Afterwards, you could then use the `barplot` and `summary` commands on `tab` to construct a clustered bar chart and so on.

R3.4 How to numerically summarise the relationship between a quantitative and a categorical variable using `RStudio`

Consider the case where we want to create a numerical summary of the relationship between a quantitative variable and a categorical variable. For example, for the `survey` dataset, we might want to look at the relationship between the variables `hair.cost` and `gender`, to see how male and female MATH1041 students differ in how much they spend at the hair-dressers.

What we want to do here is to repeat the function for numerically summarizing a quantitative variable (as in Section R3.2: Numerical summaries of a quantitative variable) at each level of our categorical variable `gender`. This can be done using the function `by`. For example, to create a five-number summary of `hair.cost` for each level of `gender`:

```
by(survey$travel.time, survey$gender, summary)
```

To calculate the standard deviation of `hair.cost` for each `gender`:

```
by(survey$travel.time, survey$gender, sd)
```

R3.5 How to numerically summarise the relationship between two quantitative variables using `RStudio`

We will illustrate how to numerically summarise the relationship between two quantitative variables using the `survey` dataset. So before attempting the below, import the survey data into `RStudio` as described in Section R1.4: How to import a text file into `RStudio`.

How to calculate a correlation coefficient

To find the correlation between travel time and hair cut cost:

```
cor(survey$travel.time, survey$hair.cost)
```

Remember you can only calculate correlations between two quantitative variables. If you accidentally ran `cor(survey$travel.time, survey$gender)` then `RStudio` will print out an error!

How to estimate a least squares regression line

To find the least squares regression line between travel time and hair cut cost:

```
mymodel<-lm(hair.cost~travel.time, dat=survey)
mymodel
```

`lm` stands for "linear model". We will learn more about this powerful command later on in the course in Section R6.7:How to make inferences about the relationship between two quantitative variables.

The output from `mymodel` contains two numbers in a section titled `Coefficients:` One number is labeled `(Intercept)`, and this is the Y-intercept (which we labeled a in lecture notes, for the line $y = a + bx$). The other number is labeled `survey$travel.time` and this is the slope (which we labeled b in the lecture notes, where $y = a + bx$).

R4 Manipulating numbers using RStudio

R4.1 How to do simple arithmetic using RStudio

Most standard arithmetic can be done from the `RStudio` command line as you would expect.

For example, to find $2 \times 3 + 1$: `2*3+1`

To find $\sqrt{20}$: `sqrt(20)`

To find the square root of every integer from 1 to 20: `sqrt(1:20)`. Recall that `1:20` in a shortcut for listing all integers from 1 to 20.

You can also do arithmetic using objects you have previously calculated and stored on `RStudio`. For example, to find the value that is 2 standard deviations above the mean travel time for the `survey` data:

```
mnTrav <- mean(survey$travel.time)
sdTrav <- sd(survey$travel.time)
mnTrav + 2*sdTrav
```

Note : An `RStudio` ouput of `2.7e-03` means 2.7×10^{-3}, where $10^{-3} = \frac{1}{10^3} = \frac{1}{1000} = 0.001$. As a result, multplying by 10^{-3} moves the decimal point 3 places to the left. Therefore, 2.7e-03 is just another way of writing 0.0027.
Beware, the "e" in 2.7e-03 is NOT the exponential function. It is the "e" (for exponent) used in scientific notation to mean "times ten raised to the power of".

R4.2 Combining variables using RStudio

Sometimes we want to combine variables together, *e.g.* calculating the average of two variables. This is actually a special case of data transformation (where the transformation is a function of two variables) and it can be done in `RStudio` using the same method used for transforming data above.

To use the `survey` data to find the average of the days of the month on which students' parents were born:

```
day.avg <- (survey$day.mum + survey$day.dad) / 2
```

To plot a histogram to see what it looks like, type:

```
hist(day.avg)
```

How does this compare to the distribution of an individual person's birthday?

R4.3 Counting the number of values satisfying a condition using `RStudio`

To find out how many MATH1041 students take longer than an hour to get to uni, then type the command:

```
sum(survey$travel.time>60)
```

To understand what this command is doing, type:

```
survey$travel.time>60
```

Note that `RStudio` returns a variable that contains a series of `TRUE` and `FALSE` values, which has as many entries as the original `survey$travel.time` variable has. A value is `TRUE` when the corresponding entry in `survey$travel.time` is greater than 60, otherwise it is set to `FALSE`. To find the number of values which are TRUE, we simply sum over this condition command.

Another way to think about it is that `RStudio` treats `TRUE` as equal to 1 and `FALSE` equal to 0. Thus summing over this means you are only summing over students exceeding an hour travel time.

Sometimes you might want to sum the values satisfying some other type of logical condition. Here are a few other types of conditions worth knowing:

- `survey$travel.time == 60` means "travel time is *equal to* 60".
- `survey$travel.time != 60` means "travel time is *not equal to* 60".
- `survey$travel.time < 60` means "travel time is *less than* 60".
- `survey$travel.time >= 60` means "travel time is *greater than or equal to* 60".

R5 Probability and experimental design using RStudio

R5.1 How to generate random success/failure results, where the probability of success is p using RStudio

Often, when experimenting with the notion of probability, we want to generate random numbers to do a **computer simulation**. This section explains one method of simulating data so that there are two possible responses, "success" and "failure", with the probability of success is equal to p.

To generate k random success or failure responses, each with probability of success p, use the command `runif(k)>p`.

For example, to generate 10 random numbers which have probability 0.7 of success, and to assign them to the variable `x`, type:

```
x <- runif(10)>0.7
```

Then you can check out the result by typing `x`. Notice that each element in `x` is either `TRUE` (which means "success" or 1) or `FALSE` (which means "failure" or 0). Briefly, what the `runif(10)` does is tell `RStudio` to random generate 10 numbers between 0 and 1. The `>0.7` is then exactly the same as what we saw in Section R4.3: Counting the number of values satisfying a condition using `RStudio`.

Let's say our trial involves the number of successful shots from the free throw line at basketball. In this case, it would be nice to be able to label the successes "score" rather than "TRUE" and the failures "miss" rather than "FALSE". To do this, check out:

```
y <- rep("miss",10)
y[x==T] <- "score"
y
```

What does the above do? The `rep("miss",10)` means to repeat the word "miss" ten types. The next line, `y[x==T] <- "score"` is where the awesomeness lies – basically, it informs `RStudio` to search out which of the ten elements of `x` are equal to `TRUE`, and then change the corresponding elements in `y` from "miss" to "score". Notice how powerful this command actually is – it is changing the elements of one object (`y`) based on what's happening in another object (`x`)!

Often it is of interest to count the number of successful trials. Along the lines of Section R4.3: Counting the number of values satisfying a condition using `RStudio`, we could type `sum(y=="score")`

R5.2 Generating random numbers using RStudio

`RStudio` has a number of functions designed for random number generation. All start with the letter `r` for 'random', followed by an abbreviation for the distribution you want to generate random numbers from.

Distribution	`RStudio` function for `k` random numbers
Uniform (0 to 1)	`runif(k)`
N(`mu`, `sigma`)	`rnorm(k,mu,sigma)`
B(`n`, `p`)	`rbinom(k,n,p)`

So for example, to generate 10 uniform random numbers between 0 and 1:

```
runif(10)
```

Or to generate 100 random numbers from $N(3, 2)$:

```
rnorm(100,3,2)
```

Success and failure data can be thought of as binomial with `n` = 1. So the `rbinom` function can be used to generate success/failure data, as an alternative to the `runif(k)>p` approach suggested in the previous section.

Can you generate 10 random success/failure numbers which have probability 0.7 of success, and assign them to the variable `x`?

R5.3 How to randomise the order of a set of subjects using `RStudio`

The standard computer-based technique of randomizing a set of objects is to first assign each object a random number, then to sort these random numbers. This gives us a random ordering of the experimental subjects, so that we can partition them to treatment groups. There are two ways that we can do this in `RStudio`. The first (and perhaps easiest) way makes use of the `sample()` function. The second way is a more manual approach that uses the `runif()` and `sort()` functions. We will go through both ways, starting with the `sample()` function.

The `sample()` function

Suppose we want to randomise the assignment of 6 subjects into three groups, each of size 2. Our subjects have the following names: Andrew, David, Frank, Ian, Jono and Victor.

We will use the `sample()` function to find a random rearrangement of their names. To do this for six subjects, first we will add the names by creating a variables called `names`. Type:

```
names <- c("Andrew", "David", "Frank", "Ian", "Jono", "Victor")
```

Next, we use the `sample()` function on the `names` object:

```
sample(names)
```

Notice that the names have been randomly sorted. So we take the first two subjects from this randomization and assign them to the first treatment group, the second two subjects from this randomization are assigned to the second treatment group, and the last two are assigned to the last treatment group.

Of course, if you want to know more about the guts of the `sample()` function, you can always type 'sample' in the *Help* tab on the top right panel or run the command `?sample`.

The `sort()` function

To use the `sort()` function, we will use the same example as above (that is, we want to randomise the assignment of 6 subjects into three groups, each of size 2). To do this we first:

- Give each experimental subject a number, from 1 to 6.
- `ran <- runif(6)` will create a vector `ran` that contains 6 random numbers.
- `sort(ran, index.return=T)` will sort the vector of random numbers. The first object returned `$x` is the sorted random numbers, which we aren't interested in. The second object returned `$ix` is a random reordering of the numbers 1:6 (which corresponds to the reordering of `ran` required to sort its values).
- The first two numbers in `$ix` tell us which subjects to assign to the first treatment group, the second two numbers tell us which subjects are assigned to the second treatment group, and the last two are assigned to the last treatment group.

Using the same names of the six subjects above (named Andrew, David, Frank, Ian, Jono and Victor), type:

```
names <- c("Andrew", "David", "Frank", "Ian", "Jono", "Victor")
ran <- runif(6)
sort.ran <- sort(ran, index.return=T)
names[sort.ran$ix]
```

If we wish to assign these six people to three treatment groups (two to each group), then the first two names in the list tell us who has been assigned to the first treatment group, the second two names tell us who has been assigned to the second treatment group, and the last two names tell us who has been assigned to the last treatment group.

R5.4 How to calculate probabilities, quantiles and density curves for a normal distribution using RStudio

Instead of using standard normal tables to calculate probabilities and quantiles for the normal distribution, you can use RStudio to find exact values, as described below.

Note in the following that there are three key functions: `pnorm`, `qnorm` and `dnorm`. These function names all take their first letter from what the function is doing (they calculate a probability, quantile or density value, respectively) and the remainder of the function name is a description of the type of random variable the function works for (the normal distribution). Remember we saw a similar syntax earlier with `rnorm` in Section R5.2:Generating random numbers.

There are many other functions on RStudio for calculating similar quantities for other special types of random variables, some of which are described in later sections.

Computing probabilities for a standard normal random variable

The function `pnorm` returns cumulative probabilities for the standard normal distribution, that is, for $Z \sim N(0,1)$, `pnorm(z)` returns the value of $P(Z \leq z)$ for any value z i.e., find the area to left of z.

For example, to find $P(Z \leq 1.96)$, type:
`pnorm(1.96)`

Note this returns the area to the left – so a bit of extra work is needed if you want a different probability. *e.g.* To find $P(Z > 1.96)$:
`1-pnorm(1.96)`

since $P(Z > 1.96) = 1 - P(Z \leq 1.96)$. To find $P(-1.96 \leq Z \leq 1.96)$:
`pnorm(1.96)-pnorm(-1.96)`

since $P(-1.96 \leq Z \leq 1.96) = P(Z \leq 1.96) - P(Z < -1.96)$

Computing Probabilities for any Normal Random Variable

The function `pnorm` can be used for any other normal distribution as well, provided you include the mean and variance when you type your command into `RStudio`. That is, for $X \sim N(\mu, \sigma)$, `pnorm(z,`μ`,`σ`)` returns the value of $P(X \leq z)$ for any value z.

For example, to find $P(X \leq 4)$ where $X \sim N(3,2)$, type:
`pnorm(4,3,2)`

How do we calculate $P(X > 4)$?

Calculating quantiles for the standard normal distribution

The function `qnorm` returns quantiles for the standard normal distribution, that is, for $Z \sim N(0,1)$, `qnorm(p)` returns the value c so that $P(Z \leq c) = p$ for any value p between 0 and 1 i.e., find that magical number c such that the area to the left of c is equal to p. You can think of quantiles are doing the reverse operation of finding cumulative probabilities.

For example, to find c such that $P(Z \leq c) = 0.975$, type:
`qnorm(0.975)`

This returns about 1.96, which means that the value c that satisfies $P(Z \leq c) = 0.975$ is $c = 1.96$. In other words, 1.96 is the value from the standard normal distribution that has 97.5% of observations falling below it.

Question: How do we calculate $P(Z > c) = 0.975$?

Calculating quantiles for any normal distribution

As you may have guessed, the function `qnorm` can be used for any other normal distribution as well, provided that you include the mean and variance in your input. That is, for

$X \sim N(\mu,\sigma)$, `qnorm(p,μ,σ)` returns the value c so that $P(X \leq c) = p$ for any value p between 0 and 1.

For example, to find c such that $P(Z \leq c) = 0.7$ where $X \sim N(3,2)$, type:
`qnorm(0.7,3,2)`

How to plot the density curve of a normal distribution

The function `dnorm` can be used to find the value of the density curve of any variable $X \sim N(\mu,\sigma)$ at a specified value x. Combining this function with the `plot` function, we can easily construct the density curve of any normal distribution.

For example, plot the density curve of $X \sim N(12,3)$ for the range of values from 0 to 24:

- To first specify a set of values for which we will find the value of the density curve, between 0 and 24 and increasing by 0.1 each step:
  `x <- seq(0,24,0.1)`

  Type `x` to see what the result looks like. Why 0 to 24? Well that's because it is the range where we the majority of the density curve to lie. Recall that 99.7% of the area under a normal density curve is located ±3 standard deviations from the mean.

- To find the value of the density curve for $X \sim N(12,3)$ at each value of `x`:
  `f <- dnorm(x,12,3)`

  Note the cleverness of `RStudio` – you can actually calculate the density for all elements of `x` simultaneously!

- To plot the density curve by joining the dots between each point from `x`:
  `plot(x, f, type="l")`

  Note: The type above is the letter, "l" for lines , and NOT the number 1

Do you know how to change this plot so that it has a suitable title? Hint: return to Section R2: Summarising data graphically using `RStudio`, to find out how to control the title and axis labels added to graphs.)

R5.5 How to calculate probabilities for the binomial distribution using RStudio

We can use `RStudio` to understand the binomial distribution, an important discrete random variable.

Evaluating a binomial probability

To find $P(X = k)$ where $X \sim B(n,p)$, use `dbinom(k,n,p)`.

For example, to find $P(X = 7)$ where $X \sim B(15, 0.3)$, type:
`dbinom(7,15,0.3)`

Evaluating a cumulative binomial probability

To find $P(X \leq k)$ where $X \sim B(n, p)$, use `pbinom(k,n,p)`.

For example, to find $P(X \leq 2)$ where $X \sim B(15, 0.3)$, type:
`pbinom(2,15,0.3)`

Question: How do we find $P(X < 2)$? $P(X > 2)$?

How to plot the probability histogram of a binomial distribution using `RStudio`

To plot a probability histogram in `RStudio` for $X \sim B(48, 0.25)$, for the range of values of X between 0 and 24:

- First specify the set of values for which we will calculate probabilities i.e., from 0 up to 24, increasing by 1 each step:
  `xbin <- 0:24`

  Remember that this is short hand of for all numbers `c(1,2,3,...,23,24)`. Type `xbin` to see what the result looks like.

- To find the value of the probability function for $X \sim B(48, 0.25)$ at each value of `xbin`:
  `fbin <- dbinom(xbin,48,0.25)`

- To plot the probability function, use a barplot:
  `barplot(fbin, names.arg=xbin, space=0)`

  The `space=0` argument ensures no spaces between bars, like a histogram. The `names.arg = xbin` tells `RStudio` what is each is called. In this case, because it's a probability histogram, the 'name' of each bar is then the sequence of integers from 0 to 24. If you further want to **overlay a normal curve** on top of this plot, then try running:
  `lines(xbin+0.5, dnorm(xbin,12,3))`

  What was that `+0.5` for?

There are a few other ways to construct this plot on `RStudio`. You could plot the points alone using: `plot(xbin,fbin)`, or you could plot vertical lines dropping down to the X-axis from each point using `plot(xbin,fbin,type="h")`... feel free to experiment

R5.6 How to calculate probabilities and quantiles for a t-distribution using `RStudio`

Instead of using t tables to estimate probabilities and quantiles for the t-distribution, you can use `RStudio` to find exact values, as described below. The functions operate in an analogous manner to `pnorm` and `qnorm`.

Calculating probabilities for the t-distribution

The function `pt` returns cumulative probabilities for the t distribution. If $T \sim t(k)$ where k is the degrees of freedom, then we can find $P(T \leq x)$ using `pt(x,k)`.

For example, to find $P(T \leq 2)$ where $T \sim t(4)$, type:

```
pt(2,4)
```

Note that this returns the probability **to the left** which is consistent with what `RStudio` does for other distributions.

To find $P(T > 2)$ where $T \sim t(4)$, recall that $P(T > 2) = 1 - P(T \leq 2)$ and type:

```
1-pt(2,4)
```

Calculating quantiles for the t-distribution

The function `qt` returns quantiles for the t distribution. If $T \sim t(k)$ where k is the degrees of freedom, then we can find the value c that satisfies $P(T \leq c) = p$ for any p between 0 and 1, using `qt(p,k)`.

Let's say we want to find the lower 12% point from the t-distribution with 15 degrees of freedom. That is, we want to find c such that $P(T \leq c) = 0.12$ where $T \sim t(15)$. To do this, type:

```
qt(0.12,15)
```

Now let's say we want to find the upper 0.5% point of the t-distribution with 15 degrees of freedom. In this case we need to do a little manipulation to get the answer, in order to turn this expression into a left-tailed probability. If $P(T \geq c) = 0.005$ then $P(T < c) = 1 - 0.005 = 0.995$. So we can find the upper 0.5% point of $t(15)$ using:

```
qt(0.995,15)
```

A more direct way of doing this is to type:

```
qt(1-0.005,15)
```

R5.7 How to calculate probabilities and quantiles for the χ^2 distribution using `RStudio`

Instead of using χ^2 tables to estimate probabilities and quantiles for the χ^2 distribution, you can use `RStudio` to find exact values, as described below. Again, the functions operate in an analogous manner to `pnorm` and `qnorm`.

Calculating probabilities for the χ^2 distribution

The function `pchisq` returns cumulative probabilities for the χ^2 distribution. If $X \sim \chi^2(k)$ where k is the degrees of freedom, then we can find $P(X \leq x)$ using `pchisq(x,k)`.

For example, to find $P(X \leq 5.2)$ where $X \sim \chi^2(5)$, type:
```
pchisq(5.2,5)
```

Note that this returns the probability **to the left**, which is consistent with what `RStudio` does for other distributions – *e.g.* `pnorm` and `pt`.

To find $P(X > 5.2)$ where $X \sim \chi^2(5)$, we need to note that $P(X > 5.2) = 1 - P(X \leq 5.2)$ and type:
```
1-pchisq(5.2,5)
```

Calculating quantiles for the χ^2 distribution

The function `qchisq` returns quantiles for the χ^2 distribution. If $X \sim \chi^2(k)$ where k is the degrees of freedom, then we can find the value c that satisfies $P(X \leq c) = a$ for any a between 0 and 1, using `qchisq(a,k)`.

For example, let's say we want to find the lower 1% point from the χ^2 distribution with 10 degrees of freedom. That is, we want to find c such that $P(X \leq c) = 0.01$ where $X \sim \chi^2(10)$. To do this, we just type:
```
qchisq(0.01,10)
```

Now let's say we want to find the upper 5% point of the χ^2 distribution with 10 degrees of freedom. In this case we need to do a little manipulation to get the answer, in order to turn this expression into a left-tailed probability. If $P(X \geq c) = 0.05$ then $P(X < c) = 1 - 0.05 = 0.95$. So we can find the upper 5% point of $\chi^2(10)$ using:
```
qchisq(0.95,10)
```

A more direct way of doing this is to type
```
qchisq(1-0.05,15)
```

R6 Inference using `RStudio`

R6.1 One-sample inference about μ when σ is known using `RStudio`

Consider the following problem:

> (From Moore *et al*) Bottles of cola drink are supposed to contain 300 ml of cola. The distribution of contents is normal with standard deviation 3 ml. The contents of six bottles are

299.4 297.7 301.0 298.9 300.2 297.0

Is there evidence that the true mean amount of cola in bottles is less than 300ml?

First we will type in the data from the `Cola` dataset (given it only contains 6 observations):

```
Cola <- c(299.4, 297.7, 301.0, 298.9, 300.2, 297.0)
```

One-sample z-test for μ

The method for a one-sample z-test on `RStudio` can be done "by hand", with the calculations being done one step at a time. For the `Cola` data, where $H_0 : \mu = 300, \quad H_a : \mu < 300$:

```
m<-mean(Cola)
mu.0 <- 300
sigma <- 3
n <- 6
z <- (m - mu.0) / ( sigma / sqrt(n) )
```

Typing `z` then returns the observed value of the test statistic, and you should get -0.78928.

To find the P-value for the one-sided test, we want to calculate $P(Z < z)$:

```
pnorm(z)
```

which returns the P-value:

```
[1] 0.2149742
```

How would you modify the above method in order to calculate a P-value if $H_a : \mu \neq 300$? A two-sided test involves calculating $2P(Z > |z|) = 2P(Z \leq -|z|)$, so we would now compute the P-value using:

```
2*pnorm(z)
```

(And if the test statistic were positive, we would have used `2*pnorm(-z)`.)

Confidence interval for μ when σ is known using `RStudio`

Following on from the above working, in order to construct a 95% confidence interval for μ when σ is known:

```
z.star <- qnorm(0.975)
margin <- z.star * sigma / sqrt(n)
lower.ci <- m - margin
upper.ci <- m + margin
c(lower.ci, upper.ci)
```

which should return:

```
[1] 296.6329 301.4338
```

Why 0.975? That's because a 95% confidence interval means we want the central 95% of the area, which implies a total of 5% to either side of this. This implies we have 2.5% on the right, and so we want an upper 2.5% point or quantile.

How would you modify the above method in order to construct a 90% confidence interval for μ?

R6.2 One-sample inference about μ when σ is not known using `RStudio`

`RStudio` has an in-built function specially designed for making inferences about μ when σ is unknown (which is pretty much always!). We will illustrate this function using the following problem:

> An archaeologist discovers seven fossil skeletons from a previously unknown species of miniature horse. It is reasonable to assume that shoulder heights are normally distributed. Below are the shoulder heights, in centimeters.
>
> 45.3 47.1 44.2 46.8 46.5 45.5 47.6
>
> A present-day species that might be closely related to these fossil skeletons has average shoulder height of 44.1 centimeters.
>
> *Is there evidence that the fossil skeletons are of a different-sized animal?*

One-sample t-test for μ

A one-sample t-test for μ is straightforward on `RStudio`, using the function `t.test`. To do a one-sample t-test for the above problem:

```
shoulder <- c(45.3, 47.1, 44.2, 46.8, 46.5, 45.5, 47.6)
t.test(shoulder, mu=44.1)
```

The expression `mu=44.1` in the above command tells `RStudio` that the null hypothesis is $H_0 : \mu = 44.1$. If no such value is entered, `RStudio` will assume that a default null hypothesis is $H_0 : \mu = 0$.

The default for `t.test()` is a two-sided tests. However you can ask `RStudio` to do one-sided t-tests. For a one-sided t-test of H_0 against $H_a : \mu > 44.1$ for the above shoulder height data, you would type:

```
t.test(shoulder, mu=44.1, alternative="greater")
```

Using `alternative="less"` would test H_0 against $H_a : \mu < 44.1$.

Confidence interval for μ when σ is unknown

You can use the `t.test` output to construct a level C confidence interval for μ. You might have noticed in the output from

```
t.test(shoulder, mu=44.1)
```

that there are some additional lines

```
95 percent confidence interval:
45.04226 47.24346
```

which give the lower and upper limits of the 95% confidence interval for μ.

To construct a 98% confidence interval, type:

```
t.test(shoulder, conf.level=0.98)
```

Note: You could include `mu=44.1` in this command if you wanted, but this would ONLY affect the P-value in the output and not the confidence interval.

Double Note: Setting the `alternative` argument will produce a one-sided confidence interval, rather than a two-sided one.

For more details on any of the above, see the help for the `t.test` function by typing `?t.test` or searching `t.test` in the *Help* tab.

R6.3 Making inferences about p using `RStudio`

Now we will consider the case where we have a simple random sample from a categorical variable that has two possible outcomes, "success" and "failure", and we want to make inferences about p, the probability of "success". We will focus on the following problem:

> Many mammals have a higher likelihood of giving birth to offspring in Spring than at other times of the year. Is this true of humans too?
>
> Out of 381 MATH1041 students, 96 of them were born in the Spring months (September-October).
>
> *Does this provide evidence that people are more likely to be born in Spring?*

The function for one-sample inference about proportions is called `prop.test`. To do a one-sample test of p, testing $H_0:\ p = 0.25$ against $H_a : p > 0.25$ for the above problem:

```
prop.test(96,381,0.25,alternative="greater",correct=FALSE)
```

which should return a P-value of `0.4646`. (The argument `correct=FALSE` tells `RStudio` not to use a continuity correction, see the textbook for details.)

The test statistic in the `prop.test` output is a chi-squared statistic – we will meet this type of statistic in a few weeks, but in this special case, it is the square of the usual Z-statistic, and returns exactly the same P-value.

To get a 95% confidence interval for p, the true chance of a MATH1041 student being born in Spring:

```
prop.test(96,381,0.25,correct=FALSE)
```

which returns in the output

```
95 percent confidence interval:
0.2109987 0.2978900
```

We needed to rerun `prop.test` as a two-tailed test, without `alternative="greater"`, in order to get the usual "two-sided" confidence interval for p.

You can change the confidence level using the `conf.level` argument. For example, for a 90% confidence interval for p:
`prop.test(96,381,0.25,conf.level=0.9,correct=FALSE)`

*How would you calculate a **99%** confidence interval?*

WARNING! When using the `prop.test` function, the P-value and confidence intervals WILL NOT ALWAYS be exact compared to the usual by-hand calculations shown in lectures. This is due to a small sample size correction that has been programmed inside the function. For assignments and online labs, please use the `prop.test` function rather than the by-hand calculations shown in lectures.

R6.4 Inference about the mean difference from two independent samples using `RStudio`

Two-sample inference about means is easy to do on `RStudio`, as we can again use the `t.test` function. Note that this is the same function we used for one-sample inference about means, but it is used in a slightly different way here. There are two different types of method, depending on how your data are stored.

Two-sample t test for a quantitative and a categorical variable

We will use the `survey` dataset to do a two-sample t-test to see if there is evidence that the mean amount students spend on a hair-cut differs for males and females. Assuming you've imported the `survey` dataset into `RStudio` (see Section R1.4:How to import a text file into `RStudio`) then type:
`t.test(survey$hair.cost~survey$gender, var.equal=T)`

Note that this uses the same function as a one-sample t-test, but `RStudio` knows to do a two-sample t-test because two variables were specified in the above command – a quantitative variable (`hair.cost`) which contains the measurements, and a categorical variable (`survey`) that says which category each measurement belongs to. Note the use of the '~' here, which means "against". Recall we used this symbol also when constructing comparative boxplots and scatterplots!

If you don't specify the option `var.equal=T` then `RStudio` will do a t-test without assuming equal variances, as described in Moore *et al.*

If we wanted to do a one-sided test rather than a two-sided test, we could specify this using `alternative="greater"` or `alternative="less"` as previously.

Two-sample t test for two sets of quantitative measurements

Sometimes, rather than being given a dataset containing the quantitative variable and the categorical variable of interest, we are simply given two sets of measurements, corresponding to the two samples of data. `RStudio` is able to perform a two-sample t-test in this situation, as described for the following problem from week 9 lectures:

> What is the effect of smoking during pregnancy on children?
>
> A randomized controlled experiment investigated this by injecting pregnant guinea pigs with 0.5 mg/kg nicotine hydrogen tartrate in saline solution. The control group was injected with the saline solution.
>
> The learning capabilities of the **offspring** of these guinea pigs was then measured using the number of errors that were made trying to find food in a maze.
>
> *Is there a detrimental effect of nicotine on the development of the guinea pig offspring?*

First of all, you will need to import the dataset from the Excel file `smokePregnant.xls`, and save it in `RStudio` as the object `smoke.preg`. Please see Section R1.5: How to import data from Excel into `RStudio`, for details on how to do this.

If you now type:

```
smoke.preg
```

then you will notice that the dataset has two columns: `smoke.preg$Treatment` contains the data for treatment guinea pigs, and `smoke.preg$Control` contains the data for the control group. To do a two-sample t-test of H_0 : $\mu_{\text{Treatment}} = \mu_{\text{Control}}$ versus H_a : $\mu_{\text{Treatment}} > \mu_{\text{Control}}$:

```
t.test(smoke.preg$Treatment, smoke.preg$Control, alternative="greater", var.equal=T)
```

Confidence interval for $\mu_1 - \mu_2$

You can use the `t.test` output to construct a confidence interval for the mean difference ($\mu_1 - \mu_2$). For example, for the guinea pig data stored in `smoke.preg` from above:

```
t.test(smoke.preg$Treatment, smoke.preg$Control, var.equal=T)
```

which will return in the output

```
95 percent confidence interval:
4.460667 37.339333
```

As usual, we can change the confidence level using the `conf.level` option, for example:

```
t.test(smoke.preg$Treatment, smoke.preg$Control, var.equal=T, conf.level=0.90)
```

Note that μ_1 is equivalent to $\mu_{\text{Treatment}}$ and μ_2 is equivalent to μ_{Control}. This is because in the `t.test()` command above, we put `smoke.preg$Treatment` first and `smoke.preg$Control` second.

As previously, we want to make sure we don't use the `alternative` argument of the `t.test` function to specify a one-sided test when we are specifically interested in calculating a confidence interval.

R6.5 Inference about the mean difference from a paired sample using RStudio

The `t.test` function on `RStudio` can also be used to analyze paired data. Consider the following problem from week 10 lectures:

> Do ravens intentionally fly towards gunshot sounds (to scavenge on the carcass they expect to find)? Crow White (what an appropriate name!) addressed this question by counting raven numbers at a location, firing a gun, then counting raven numbers 10 minutes later. He did this in 12 locations. Results:

location	1	2	3	4	5	6	7	8	9	10	11	12
before	1	0	1	0	0	0	0	5	1	1	2	0
after	2	3	2	2	1	1	2	2	4	2	0	3

> *Is there evidence that ravens fly towards the location of gunshots?*

First import the data into `RStudio` from the `ravens.xls` Excel file, available on the Computing page on Moodle. Please see Section R1.5: How to import data from Excel into `RStudio`, for details on how to do this. Note the top four rows of the `ravens.xls` need to be deleted before saving as a text file, so that row 1 contains the variable names, and so that the rows immediately below contain the data.

Suppose you imported the dataset using a command like

```
ravens <- read.table("ravens.txt", header=T)
```

Then, we can view the data by typing

```
ravens
```

Note that there are three columns, and in particular, `ravens$before` contains the "before" counts, while `ravens$after` contains the "after" counts.

Paired *t*-test

To do a paired t-test for the raven data, to test $H_0 : \mu_{\text{after}} = \mu_{\text{before}}$ against $H_a : \mu_{\text{after}} > \mu_{\text{before}}$:

```
t.test(ravens$after, ravens$before, paired=T, alt="greater")
```

We have used the function `t.test` to do this test, just as we did for inference about means from one-sample or from two independent samples. Moreover, because two variables were entered rather than one, then `RStudio` knew to do some form of t-test involving two samples.

However, we have now also specified `paired=T`, and so `RStudio` knows to do a paired t-test rather than a two-sample t-test. It is important to remember to include `paired=T`, otherwise `RStudio` will do the wrong type of two-sample test!

How would you modify the above command in order to do a ***two-sided*** *paired t-test using the raven data?*

Confidence interval for the mean difference from a paired sample

To construct a confidence interval for the mean difference from a paired sample, using the `raven` dataset:

```
t.test(ravens$after, ravens$before, paired=T)
```

This will return in the output:

```
95 percent confidence interval:
-0.1117494 2.2784161
```

As previously, we can change the confidence level using the `conf.level` option.

R6.6 How to make inferences about the relationship between two categorical variables using `RStudio`

To do a χ^2 test for independence of two categorical variables using `RStudio`, you first need to construct a table of frequencies to summarise the data (see *Section R3.3: How to numerically summarise the relationship between two categorical variables using* ***`RStudio`***).

For example, to do a χ^2 test for independence of the `gender` and `study.area` variables from the `survey` dataset, first do

```
tab <- table(survey$gender, survey$study.area)
```

From there, if you want to do a χ^2 test for independence and you do NOT need the expected counts, you can simply run the `summary` function on the table of frequencies:

```
summary(tab)
```

On the other hand, if you want to do a χ^2 test for independence and you need the expected counts (you will need them to check that the assumptions are satisfied), run the `chisq.test` function on the table of frequencies:

```
Xsq <- chisq.test(tab)
Xsq
```

The instruction on the first line applies the `chisq.test` function to the table called `tab` and stores the result in a variable called `Xsq`. The instruction `Xsq` on the second line asks `RStudio` to display the content of `Xsq` in the console.

The value labeled `X-squared =` gives the value of the X^2 test statistic, `df =` tells us the degrees of freedom, and `p-value =` tells us the P-value.

To get the observed counts and the expected counts, use respectively:

```
Xsq$observed
Xsq$expected
```

If you want to get the individuals values "$(observed - expected)^2/(expected)$" use:

```
(Xsq$residuals)^2
```

Is there evidence that MATH1041 males and females have different subject preferences?

Alternatively, if you have already summarized your data in a table and you want to enter your table of frequencies directly into `RStudio` and store them in a `table`, you can do this as described in *Section R3.3:How to type in a table of frequencies*. After creating the table, then do `chisq.test` to conduct the χ^2 test for independence.

R6.7 How to make inferences about the relationship between two quantitative variables using `RStudio`

We will illustrate how to make inferences about the relationship between two quantitative variables using the `concept` dataset. Please make sure you have imported the `concept` dataset into `RStudio` (see Section R1.4 for how to import a text file).

The first step is to fit a least squares regression line between GPA and IQ of the students, and store it as an object. We will call it `mymodel`:

```
mymodel <- lm(GPA~IQ, data=concept)
```

`lm` stands for "linear model", so we just told `R` to fit a linear regression line to predict GPA using IQ, where these columns are found in `concept`. Note that alternatively, you can have typed

```
mymodel <- lm(concept$GPA~concept$IQ)
```

Everything you need to know about your linear regression fit is now stored in the object `mymodel`. To find out what sort of information is available, try typing

```
names(mymodel)
```

Information is stored within `mymodel` under each of the labels that are printed out when you type the above line. For example,

```
mymodel$coefficients
```

will return the estimated slope and intercept of your least squares regression line. By exploring the different objects that are stored in `mymodel`, you could find out all you need to know about a linear regression fit. Below are some tricks to cut straight to the most important bits.

How to obtain linear regression summary output

For summary output from your linear regression, simply type:

```
summary(mymodel)
```

See if you can answer the following questions, using the summary output:

- Is there evidence of a relationship between GPA and IQ for the students?
- What is the equation of the least squares regression line for predicting a students GPA from their IQ?
- What proportion of variation in GPA is explained by IQ?

Constructing residual plots

The residuals from your fit can be accessed by typing either `mymodel$residuals` or `residuals(mymodel)`. The predicted values from a linear regression fit can be accessed using either `mymodel$fitted` or `fitted(mymodel)`.

And to construct a residuals versus fitted values plot, you could type:

```
plot(fitted(mymodel), residuals(mymodel))
```

So, for example, to construct a normal quantile plot of residuals, you could type:

```
qqnorm(mymodel$residuals)
```

A shortcut to do these two plots, plus a couple more, is:

```
plot(mymodel)
```

To create a scatterplot and add the fitted line, type (See also Section R2.5: How to graph one quantitative variable versus another quantitative variable using `RStudio`):

```
plot(GPA~IQ, data=concept)
abline(mymodel)
```