1 Workshop 1

1.1 Learning objectives

- To be able to start and use R-studio, becoming familiar with the different windows and the R environment
- To read in data from tablular text files.
- To calculate descriptive statistics for data and subsets of data
- To produce appropriately labelled plots from one set of x-y data

1.2 Starting R

Launch R Studio. A window appears with a console window (and others) inside. If you launch R in a terminal (on Mac or Linux) then the prompt changes after some version boilerplate.

The blue `>` in the Console window is where we will be typing. R is a command line driven program so all commands are typed (or read from a script you prepared earlier. The `>` in this exercise is where you type - lines starting with a number in square brackets like this `[1]` are the computer's response. Don't type the `>` at the start of the line. R is a very fussy language - all the commands must be typed in exactly with the right number of brackets of the right type. If you get an error, try to spot where the mistake is and attempt it again. In particular check that you haven't mixed up a `1` (one) and `l` (ell), or `|` (pipe, a vertical line) and `/` (forward slash), especially when displayed in a slanting or italic font.

At the back of this document is a table of commands and options that you find useful. At the moment it is blank. As you discover seful commands, make a note in this table as a reminder for when you perform experimental analyses.

1.3 The command prompt

Try typing in a simple arithmetic expression:

```
> 1+2
```

```
[1] 3
```

The computer has performed the calculation and returned a single value. R typically works on sets (vectors or lists) of values. We can describe a set by placing it inside `c( ...)` with each value separated by a comma. When we perform a calculation, R works on all members of the set.

```
> c(1,2) + 1
```

```
[1] 2 3
```

As our set contains two values, R gives us two values in return. We can perform calculations using sets as well. R will perform the operation between the first elements of each set, then the second, then the third and will recycle the shorter set as many times as necessary.

```
> c(1,2) * c(3,4)
```

```
[1] 3 8
```

Try different sets and different operators (`+ - * / **` are all valid) A list of operators and a crib sheet for commands is given at the end of this document

1.4 Variables

We can save values for reuse by giving them names

```
> myvalues <- c(1,2,3,4,5)
> myvalues
```

```
[1] 1 2 3 4 5
```

These names can be anything that isn't an existing command or keyword. Note that we assign them right to left `variable <- data` and that the symbol `<-` is typically used instead of `=`.

We can then reuse these names wherever we want.

```
> myvalues + 1
```

```
[1] 2 3 4 5 6
```

```
> c(myvalues, myvalues)
```

```
 [1] 1 2 3 4 5 1 2 3 4 5
```

Note that `c(...)` has joined (concatenated) the two sets into one longer set.

```
> morevalues <- c(myvalues, myvalues + 5)
> morevalues
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
> length(morevalues)
```

```
[1] 10
```

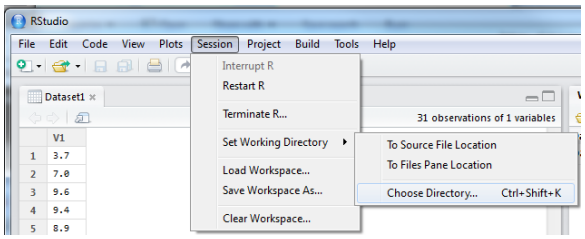
We can get basic statistics on the set with the functions `mean()`, `median()`, `sd()` (for standard deviation) and `var()` (for variance). Standard Error can be calculated by taking the square root (`sqrt()`) of the variance divided by the number of datapoints. You can find more information about any function by caling `help(` *`function name`* `)`.

```
> mean(morevalues)

[1] 5.5

> median(morevalues)

[1] 5.5

> sd(morevalues)

[1] 3.02765

> sqrt( var(morevalues) / length(morevalues) )

[1] 0.9574271
```

Help in R is easily obtained by typing `help(function)` to get help on how to use `function()`. Try this to see the options that can be used with `sd` (ie. `help(sd)`)

1.4.1 Setting your Working Directory

Typically you will be keeping your data well organised and have a specific directory (folder) in which you wish to be working. We will refer to this as the **working directory** and can set this in R Studio. (You will probably choose to have a different working directory for each project). From the menu select *Session* > *Set Working Directory* > *Choose Directory ...* and navigate to the directory in which you have saved your data. You will not see any files in the directory - just select it anyway by clicking Choose.



1.5 Reading from a file

We can use the 'scan()' function to read data from a text file into R variables. If we have data stored in the file *Dataset1.txt* then we can read this in to a variable in R. Download the file from the VLE to the folder you created for the project and take a look at the file using a program such as Wordpad.[1]

[1] If this workshop is being run outside a module then the workshop leader will point you tot he correct URL for the files.

```
   3.7
   7.0
   9.6
   9.4
   8.9
 and so on ...
```

Note that this is a plain text file. R cannot read data from Word documents, just from plain text files. Read in the data ta the variable `ds` using the `scan()` function.

```
> ds <- scan("Dataset1.txt")
> ds
```

```
 [1]  3.7  7.0  9.6  9.4  8.9  6.7  8.4  8.1  4.4  3.3  3.7  2.9  1.7  1.5  0.7
[16] -0.8  1.1  1.1  1.8  1.3  1.6  1.7  0.9  1.1  0.8  2.6  4.4  4.8  6.5  5.8
[31]  4.1
```

Challenge: *Count how many elements are in the list using* `length()` *and calculate the mean, median, standard deviation and variance of these data.*

We can select an individual datapoint by placing it's row number in square brackets after the variable name. The `[number]` is known as a subscript.

```
> ds[4]
```

```
[1] 9.4
```

and a range of data points by separating the first and last points in the range with a :

```
> ds[4:8]
```

```
[1] 9.4 8.9 6.7 8.4 8.1
```

We can also select data by giving a list of elements to retrieve (remember that a list is specified using the `c( ...)` function) Retrieve the first three odd numbered values

```
> ds[c(1,3,5)]
```

```
[1] 3.7 9.6 8.9
```

Challenge: *Calculate the mean and standard deviation for the first 15 and for the last 15 datapoints in the set*

1.6 Reading tabular data

The data in *Dataset1* is a single column of data. Often the data we want to read is in tabular form and we want to keep all the columns together to preserve the rows. We will read in a set of data describing athletes from the Australian Institute of Sport. Ensure you have downloaded the file `sport.txt` to your working directory.

We will use the command `read.table()` to import the data into the variable `ais`. You could also use the *Tools* > *Import Dataset* > *From Text File* menu option but ensure you set the variable name (top left corner in teh import wizard box) to `ais`.

```
> ais <- read.table("sport.txt", sep="\t", header=T)
```

This has read the table into the variable `ais`.

Examine the contents of `ais` by clicking on it in the Workspace (upper right window). It will then open in the upper left. There are five columns, Gender, Sport, Bodyfat, Height, and Weight. Scroll down and look at the values in each column.

Question: *Which of these columns contain categorical (factorial) data and which contain continuous data?*

`ais` is a data frame. Each column is named and each row is numbered. We can access a column as a list using the convention `variable$columname`. Try these and see what results you get:

```
> ais
> ais$Height
> ais[2]
> ais$Weight[4:10]
```

Note that the names are *case sensitive*. Upper and lower case are different. We can list the column names with `names(variable)` and use the columns like we would any other list.

```
> names(ais)
```

```
[1] "Gender"  "Sport"   "Bodyfat" "Height"  "Weight"
```

```
> mean(ais$Bodyfat)
```

```
[1] 13.50743
```

It is often inconvenient to have to type `ais$` every time we want to use the data. We can *import* the data frame into our current workspace so all the columns areindependent variables using the function `attach`.

```
> attach(ais)
```

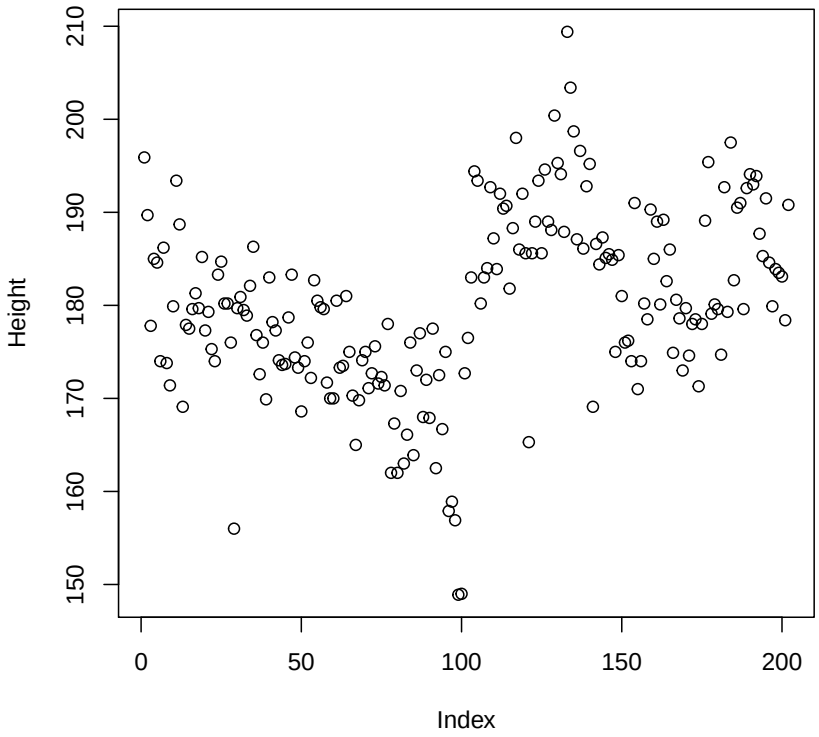
Now we can use the columns directly. **Warning: Importing data like this will hide any data with the same name as any of the columns.** You can reverse `attach()` with `detach()`.

Challenge: *Calculate the mean, median, min, max and standard deviation of the Height, Weight and Bodyfat data*

1.6.1 Simple Plots

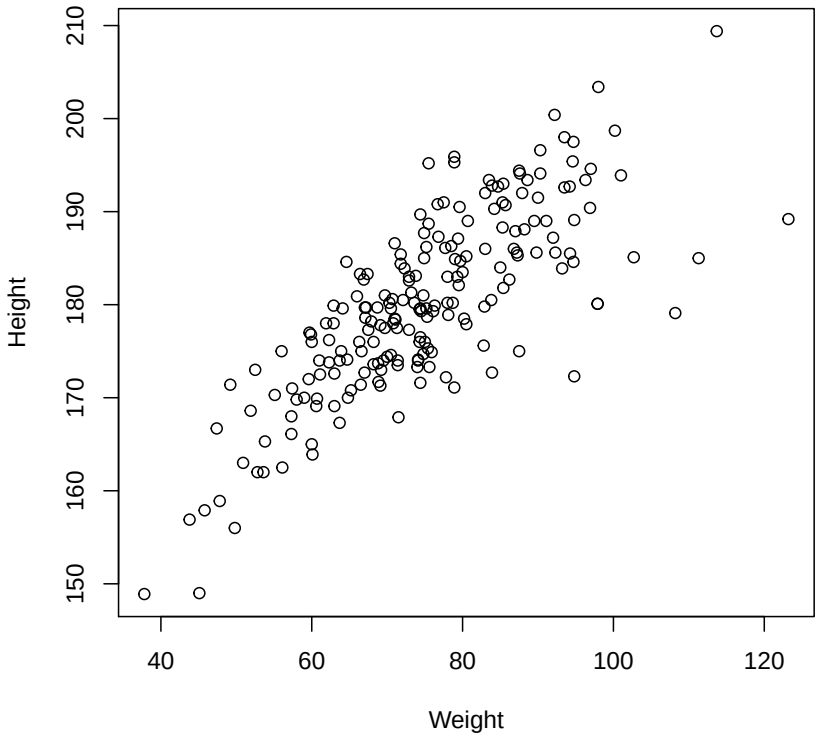
Viewing data graphically is often a good way to begin to understand the relationships. R provides simple yet very powerful plotting functions. The simplest plot is to just plot one variable. This will be displayed on the y-axis and R will automatically order them by row number on the x-axis.

```
> plot(Height)
```



A more advanced plot would try to make some sense of the data. If we want to plot the height versus the weight then we can easily do this.

```
> plot(Weight, Height)
```



The first set of data is plotted as the x-axis, and the second set as the y-axis.

This plot would not get you very good marks in a report. *Can you spot the key things that are missing?*

Fortunately the `plot()` function has many additional options that can help display the data in a more apropriate way. These are called by adding them to the list of arguments for plot; e.g. `plot( xdata, ydata, ...)` where `...` is the list of additional options. You can find a detailed description with `help(plot)` and `help(par)`. This includes setting the axis labels and the main title with `xlab="label"`, `ylab="label"` and `main="title"` where *label* and *title* are the text you wish to appear placed between speech marks ("). These parameters are included in the plot command like *`plot(x,y, param1=nnn, param2="nnn", ...)`* and so on.

Repeat the plot of Height versus Weight including the additional parameters to produce a properly labelled plot and show it to a demonstrator before finishing the workshop.

2 Reference

2.1 Operators

2.1.1 Arithmetic Operators

Operator	Description
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`^` or `**`	exponentiation
`x %% y`	modulus (x mod y) `5%%2` is 1
`x %/% y`	integer division `5%/%2` is 2

2.1.2 Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

2.2 Useful commands

Command	**option**	**notes**
help(*command*)	–	Provides the help for *command*