Introduction to R

The Basics

R is a piece of software that you control via its statistical programming language. It is extremely powerful and as such is now widely used for academic research as well as in the commercial sector not least by companies such as Google and Facebook. Unlike software such as Excel or SPSS the user has to type commands to get it to execute tasks such as loading in a dataset or performing a calculation. The biggest advantage of this approach is that you can build up a document, or script, that provides a record of what you have done and enables to easy repetition of tasks. Once you get the hang of it, this approach makes for much more efficient data analysis (we promise!), but it can be hugely frustrating to start with as you will generate lots of error messages. Be patient and things will become easier with practice.

Need to add a bit about the R interface and command line etc.

At its absolute simplest R is a calculator. If you type

```
## [1] 15
```

in the command line window, it will give you an answer (after every line you need to hit enter to *execute* the code. Where R differs is that you can assign numbers names. These become *objects* in R and they are a really important concept. For example:

```
a<-5
b<-10
```

The `<-` symbol is used to *assign* the value to the name, for example we assigned the integer `5` to the object `a`. To see what each object contains you can just type

```
print(a)
```

```
## [1] 5
```

Where the bit in the brackets is the object name. Objects can then be treated in the same way as the numbers they contain. For example:

```
a*b
```

```
## [1] 50
```

Or even used to create new objects:

```
c<- a*b
print(c)
```

```
## [1] 50
```

The real power of R comes when we can begin to execute *functions* on objects. Until now our objects have been extremely simple integers. It is possible to build up more complex objects. In the first instance we will use the `c()` function for this. ‘c’ means concatenate and essentially groups things together.

```
DOB<- c(1993,1993,1994,1991)
```

Type `print(dob)` to see the result. We can now execute some common statistical functions on this object

```
mean(DOB)
```

```
## [1] 1993
```

```
median(DOB)
```

```
## [1] 1993
```

```
range(DOB)
```

```
## [1] 1991 1994
```

R can also calculate a whole load of descriptive statistics using the `summary()` command:

```
summary(DOB)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1990    1990    1990    1990    1990    1990
```

The structure of the `DOB` object - essentially a group of numbers - is known as a *vector* object in R. To build but more complex objects that, for example, resemble a spreadsheet with multiple columns of data it is possible to create a class of object know as a data frame. This is probably the most commonly used class of object in R. We can create one here by combining two vectors.

```
Singers<- c("Zain", "Liam", "Harry", "Louis")
One.Direction<- data.frame(Singers, DOB)
```

If you type `print(One.Direction)` you will see our data frame.

```
print(One.Direction)
```

```
##   Singers  DOB
## 1    Zain 1993
## 2    Liam 1993
## 3   Harry 1994
## 4   Louis 1991
```

Recap

You have just:

1. Entered your first commands into the R *command line interface*
2. Created *objects* in R
3. Calculated a number of descriptive statistics using R *functions*
4. Created a *vector* of values (the DOB object)
5. Created a data frame (called One.Direction)

Tips

1. R is *case sensitive* so you need to make sure that you capitlise everything correctly.
2. It is important to come up with good names for your objects. In the case of the One.Direction object I used a full-stop to separate the words and capitilisation. It is good practice to keep the object names as short as posssible so I could have gone for OneDirection or one.dir. You cannot start an object name with a number so 1D won't work.
3. If you press the up arrow in the command line you will be able to edit the previous lines of code you inputted.

Next Steps

In the previous section R may have seemed fairly labour-intensive. We had to enter all our data manually and each line of code had to be written into the command line. Fortunately this isn't really the case. In R Studio look to the top left corner and you will see a plus symbol, click on it and select "R Script".

This should give you a blank document that looks a bit like the command line. The difference is that anything you type here can be saved as a *script* and re-run at a later date. When writing a script it is important to keep notes about what each step is doing. To do this the `#` symbol is put before any code. This *comments out* that particular line so that R ignores it when the script is run. Type the following into the script:

```r
#This is my first R script

My.Data<- data.frame(0:10, 20:30)
```

```r
print(My.Data)
```

In the scripting window if you highlight all the code you have written and press the run button on the top on the scritping window you will see that the code is sent to the command line and the comment after the `#` is ignored.

From now on you should type your code in the scripting window and then use the Run button to execute it. If you have an error then edit the line in the script and hit run again.

The My.Data object is a data frame in need of some sensible column headings. You can add these by typing:

```r
#Add column names
names(My.Data)<- c("X", "Y")

#print My.Data object to check names were added successfully.
print(My.Data)
```

```
##     X  Y
## 1   0 20
## 2   1 21
## 3   2 22
## 4   3 23
## 5   4 24
## 6   5 25
## 7   6 26
## 8   7 27
## 9   8 28
## 10  9 29
## 11 10 30
```

Until we have generated the data used in the examples above. One of R's great strenghts is its ability to load in data from almost any file format. Comma Separated Value (CSV) files are our preferred choice. These can be thought of as stripped down Excel spreadsheets. They are an extremely simple format so they are easily *machine readable* and can be read in and written out of R. Since we are now reading and writing files it is good practice to tell R what your *working directory* is. This is the folder on your computer where you wish to store the data files you are working with. On the lower left of the RStudio screen is a window with a "Files" tab. I you click on this tab you can then navigate to the folder you wish to use. You can then click on the "More"" button and then "Set as Working Directory". You should then see some code similar to the below appear in the command line. It is also posible to type the code in manually.

```
#Set the working directory. The bit between the "" needs to specify the path to the folder you wish to

setwd("~/Dropbox/Q-Step/Qstep-R-Intro")
```

Once the working directory is setup it is then possible to load in a csv file. In this case it shows London's historic population for each of its Boroughs.

```
#load in csv file
pop<- read.csv("census-historic-population-borough.csv")
```

To view the object type:

```
print(pop)
```

Or if you only want to see the top 10 or bottom 10 rows you can use the `head()` and `tail()` commands. These are particularly useful if you have large data frames.

```
head(pop)

tail(pop)
```

To get to know a bit more about the file you have loaded R has a number of useful functions

```
#Get the number of columns

ncol(pop)
```

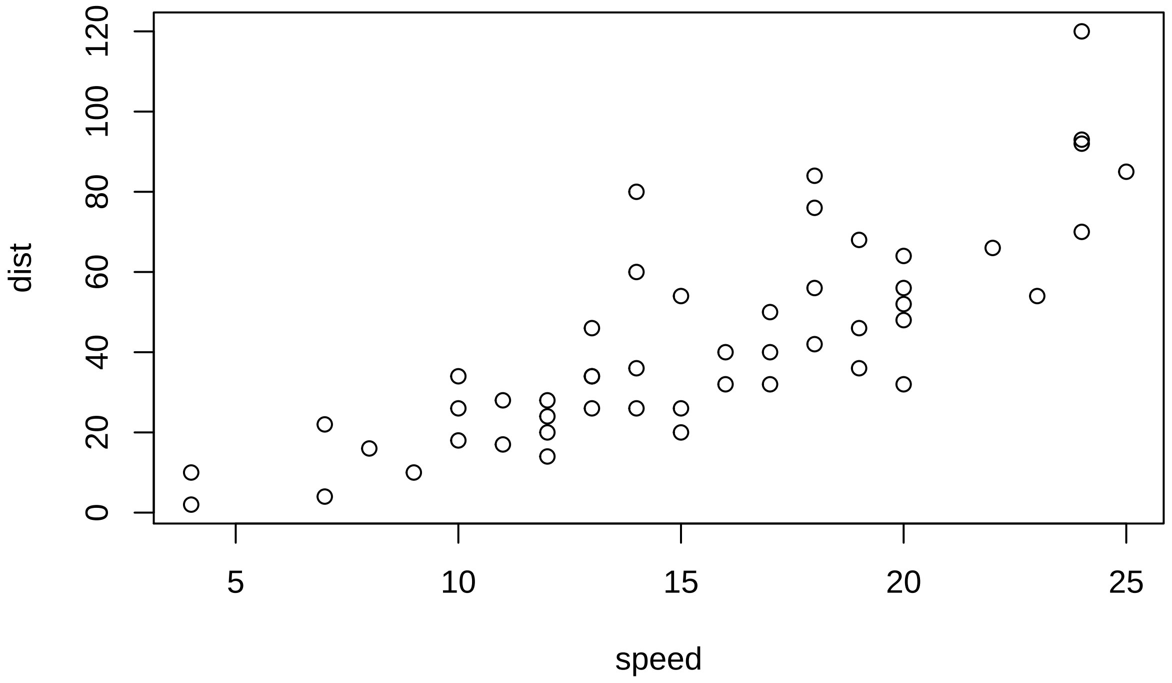
```
## [1] 24
```

```
#Get the number of rows

nrow(pop)
```

```
## [1] 36
```

```
#List the column headings

names(pop)
```

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.