Basics of R

You should all be familiar we the dataset called 'students' from the pre-practical worksheet. If you have not saved this, or you deleted it, please now download this excel file from Moodle.

You need to save this file in a convenient folder, this folder is where you will access and save data, scripts and plots throughout the course, for ease lets all name this folder 'SaRD' and in this create a folder called 'Week_1'. Remember the 'students' excel file contains two sheets, the first (labelled 'plot') contains the data plus a basic scatter plot, the second sheet (labelled 'data') contains only the data we are going to use in R. Our first step is to create a csv file from the students dataset and save it in your new 'Week_1' folder.

1. Setting up your script & importing data

All script should be written and saved using R-Studio's script editor, never be tempted to use word possessing programs such as MS Word, autocorrect features will destroy your carefully written code!

- Open R-Studio
- Click File>New File>R Script (this open a new Script Editor window)
- Its good R practice when starting a new script to add some annotation (meta data) at the beginning of the script, such as the date the script was edited, who edited it, what data was use, what analysis was done.

```
# 15/01/2019 Your Name
# Data: student weight and height from tiny made up survey
# Analysis: data exploration and plotting
```

A couple of other house keeping rules for new script are:

- Always make your next line of script **rm(list=ls())** (this removes everything from R and gives you a clean slate to work from).
- The second line of script will typically be used to set your working directory (i.e. the location of your files and where you want to save your work) and will look like this….

  setwd("H:/Documents/SaRD/Week_1")

- Other useful functions at this stage are **getwd()** which tells you what your current working directory is, and **list.files()** which tells you what files are in your directory.

Now you will likely want to import some data, this can be done using the file path to the folder you saved your csv file above…

mydata<-read.csv("H:/Documents/SaRD/Week_1/students.csv")

- **OR**, as you have already specified the path when you set your working directory you can more simply use…

  mydata<-read.csv("students.csv")

So the first block of text in your script EVERY SESSION should look something like this….

```
# 16/01/2018 Your Name
# Data: student weight and height from tiny made up survey
# Analysis: data exploration and plotting

# clear r console
rm(list=ls())
# set working directory
setwd("H:/Documents/SaRD/Week_1")
# check working directory
getwd()
# list files in working directory
list.files()

# import data
mydata<-read.csv("students.csv")
```

Now lets leave this dataset for a minute and focus on some basic R features

2. Help in R

Let's assume you want to do a linear regression.

Scenario 1: you do not know the exact name of the function you want to use, but you know it is to perform a linear model. **Help.search()** performs a broad search. It will come with all the functions that contain the word linear model.

```
help.search("linear model")
```

The output gives you first the name of the package where to find the function and then the name of the function (for instance, glm.nb is a function in the library MASS that performs

Negative Binomial Generalized Linear Models [don't worry about what this is, just the searching bit])

Scenario 2: you know the name of the function but you need help about how to use it and what arguments it takes.

```
?lm
```

A description of the function appears. Have a good look at that page and see what information it gives you.
Note: one of the most important sections in the help page is Examples. You can copy and paste the Examples script into the script editor panel and see how a function works.

3. Objects in R

Creating, listing and deleting objects

You create objects using the assign operator <-

```
n<-27
n
```

n is an object and contains the value 27. You can use it in functions, operations, plots

```
n*2 # multiply n by 2
n^2 # n is squared

x<-53 ; x
```

x is another object and contains the value 53. Note: the use of the semicolon **;** to separate two commands.

```
n+x ; n-x ; n

n<-34
n
```

Warning: if you assign a different value to an object that already exists (i.e. has the same name), its previous value is erased. n is now 34 and not 27.

R is **case sensitive**: below n and N are two different objects. This is also true for functions. mean (x) will give you the average of the object x but Mean(x) won't.

```
n<-27
N<-34

x<-c(12,7,23,90,45) # c() is used to concatenate (join) values into a vector
mean(x)
Mean(x)

# Let's check manually:
(12+7+23+90+45)/5 # remember that R can be used for simple operations as well.
```

An object can be anything: a single value, a vector, a list, a data frame, a matrix, the output of a function.

```
y<-c(23,18,90)
y # object y is a vector of 3 values
M<-c("R Rulz!") # when specifying character or strings you need quotes (" ")
M # M is a string vector
Mean<-mean(y)
Mean # Mean is the output of the function mean applied to the object y
```

Use the function ls() to list all the objects stored in the R environment memory.
Use function rm() to erase objects from the R environment memory

```
ls() # to list all objects in the memory

rm(y) ; ls() # removes y only

rm(list=ls()) # remove all objects
ls() # what is left?
```

4. Attributes

Objects in R are characterised by their names, their contents and their attributes. The mode or length of an object for instance are attributes. Attributes are important because the action of a function on an object depends on the attribute of the object.

The mode of an object can be either

- Numeric
- Character
- Logical (False or True)
- Integer

```
x<-c(1:10)
mode(x) # what is x?

treatment<-c("blue","red")
mode(treatment) # what is treatment?

vect<-c(T,F,T,T) # or vect<-c(TRUE,FALSE,TRUE,TRUE)
mode(vect) # what is vect?
```

Objects can be converted into another mode if required by functions as.numeric(), as.integer(), as.character(), as.factor(), etc...

5. Operators

Operators are symbols that tell the software to perform a specific task (e.g. mathematical and logical tasks). The R language has many built-in operators; the following table gives several of the most common along with a short description. We will see some of these in action over the next few weeks.

Arithmetic	Comparison	Logical
+ addition	< lesser than	! x NOT
- subtraction	> greater than	x & y AND
* multiplication	<= lesser than or equal to	x | y OR
/ division	>= greater than or equal to	%in% IN
^ power	== equal	
	!= different	

Manipulating data

1. Indexing of vectors

Let vec be the vector containing values (2,4,5,9,10). It is a vector of length 5 (because it contains 5 elements). To access the i^{th} value of vector vec, you type vec[i].

```
vec<-c(2,4,5,9,10)
vec[3] # gives 5, the 3rd value of the vector
vec[c(1,3)] # gives 2 and 5, the 1st and 3rd values of the vector
vec[6] # gives NA (Non Attributed) because the length of the vector is only 5, it
does not have a 6th value.
```

Values from a vector can be replaced or removed

```
vec[3]<-6 # replace third value by 6. Check it.
vec[c(1,5)]<-c(0,0) # replace 1st and 5th values by 0. Check it
vec[-3] # remove 3rd value
```

Logical operators can be used to select data within the vector

```
vec<-c(10,4,8,2,9,13,7,10)
vec[vec>=10] # gives all values of vec >= 10
vec[vec>9 & vec <20] # gives values of vec >9 AND <20
vec[vec>9 | vec <20] # gives values of vec >9 OR <20
vec[vec != 10] # gives values of vec that ARE NOT equal to 10
```

2. Create a data frame

Let's create our own dataframe, and add it to our existing 'students' dataset. Remember we deleted the 'students' dataset earlier so the first thing to do is import the student.csv file as you did above. Then use the following code,

```
pupil<-seq(1,51,by=1) # pupil is a sequence of numbers from 1 to 51, by increment of 1
scores<-c(20,25,19,34,25,18,36,37,43,29,26,33,31,49,50,29,48,
      45,59,67,70,54,52,55,59,68,63,68,61,55,59,69,71,72,
      69,68,65,75,73,71,79,89,83,90,83,82,85,81,85,71,69)
# scores is a vector formed by concatenating c() values
breakfast<-gl(3,17,51, labels = c("none", "muffin", "fry.up")) # breakfast is a factor with 3
# levels, each replicated 17 times (51 in total)

# see help of functions to get a better understanding of how they work
# copy and paste some of the examples provided in the help file
?seq
?gl # stands for generating levels
```

Now combine all your variables into a data frame: you want each of your vectors to become a column in your data frame, therefore you want to column-bind you vectors (**cbind**). You want the combined columns to be a data frame object (**data.frame**). You can now also add your students data (mydata) to this dataframe.

```
df<-data.frame(cbind(pupil,scores,breakfast,mydata)) # combine all data
```

Now look at your new dataset, either in the R Studio data window or by running *df*

3. Access portions of a data frame

Contrary to vectors which are 1-dimensional, data frames have 2 dimensions. They have a number of rows *i* and a number of columns *j*.
Lets try this in our df data frame, to access the data point in the i^{th} row and j^{th} column, type df[i,j].

```
dim(df) # gives the dimension of data frame df. The first number is the number of rows
df[1,] # selects the first row
df[,1] # selects the first column
df[1,1] # selects the value in the first row and first column

#Do not forget to use commas when you subset elements of a data frame. A data frame
# has 2 dimensions [rows,columns].
```

Columns of a data frame can also be accessed by their names by typing df$name.of.column

```
names(df) # gives the column names of the data frame
df$scores
df[,"breakfast"] # when selecting names of columns do not forget the quotes
df[df$scores<50,] # selects rows of the data frame where x < 50
df[df$breakfast=="muffin" & df$scores<50, ] # selects rows of the data frame where
#breakfast = 1 and x <50
df[df$breakfast=="fry.up", "scores"] # select exam scores for fry up breakfast eaters
df[df$breakfast=="none", "pupil"] # select pupil numbers for people that skip breakfast
```

4. Useful functions for vectors and data frame

Below are some useful functions for use with vectors and data frames. In your spare time you can see the help Examples related to each, copy and paste the examples of the help page to see how they work. Try changing some of elements in the examples to see what happens.

4.1. Vectors

Functions	
c(x,y,...)	Combine numbers,vectors into a single vector
seq(from= ,to = ,by= ,...)	Generate regular sequences by a certain increment
seq (from=, to= ,length= ,...)	Generate regular sequences of a certain length
rep(x, times= ,...)	Replicates the values in x a certain number of times
rnorm(n, mean= ,sd=)	random generation for the normal distribution with mean equal to mean and standard deviation equal to sd
gl(n,k,...)	Generate factors by specifying the pattern of their levels
factor(x,...)	Encode a vector as a factor
length(x)	Get the length of vectors or factors x

4.2. Data frames

Functions	
cbind(x,y) **rbind(x,y)**	Combine vectors, factors or data frames of the same length by column or by row
rownames(x) **colnames(x)**	Retrieve row names or column names of a data frame
dim(x)	Get the dimension of data frame x
merge(x,y,by=)	Merge data frames x and y by column names

Exploratory data analysis

1. Check the data

All the functions below have some overlap in their use, but it is important to use them to explore data.

summary()	Gives summary statistics of each column. check NAs, outliers, levels of factors, etc...
str()	Gives the mode of each variable. Check before an analysis that the variable is in the correct mode. For instance if you want to do an anova with year rather than a regression, convert it into a factor
dim()	Gives you the dimension of the data frame.
names()	Quick recap of the names of your columns.
head()	Gives the first lines of the data frame
table()	To summarize numbers of individuals per category

```
dim(df) # df is a data frame with 51 rows and 5 columns
names(df) # the columns are called pupil, scores, breakfast, height, weight
summary(df)

# The summary gives you the possibility to check numerically for outliers, number of replicates
# in various treatments, presence of missing values...

str(df)

# str reveals that you have a data frame of 51 observations and 5 variables.
# It states the mode of each of the variable; scores is considered as an numeric,  breakfast is
considered as a factor.
```

2. Summary statistics

Here are some basic functions to get a first numerical summary of your data

mean(x)	mean of elements of x
var(x)	variance
sd()	standard deviation
min(x)	minimum
max(x)	maximum
median(x)	median
range(x)	gives min(x) and max(x)
cor(x,y)	correlation between x and y
rowMeans/rowSum colMeans/ColSum	row and column sums and means for numeric arrays
tapply(x,index,fun)	Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

```
mean(df$scores)
var(df$scores)
min(df$ scores)
range(df$scores)

#tapply is an interesting function to obtain values from a specific function by groups.
# for instance, if we want to know the mean of scores for different breakfast types
tapply(df$scores, df$breakfast, mean)
```

Graphics

1. The function plot()

The function plot() is the general plotting command. A plot is not stored into an object like data frames and vectors are. A plot is sent to a graphical device that can either be a new window or a file (pdf for example).

```
# the command plot automatically open a graphical window and displays the plot
plot(df$height~df$weight)

## If you want several plots in the same window, you can split the screen
par(mfrow=c(1,2)) # the 1st number states the number of rows, the 2nd the number of columns
plot(df$height~df$weight)
plot(height~weight,data=df)
```

Here is a table with some of the most basic plotting functions in R

plot(x,y) **plot(y~x)**	plot values of y(y-axis) as a function of x(x-axis)
boxplot(x) **boxplot(x~factor)**	box and whisker plot
coplot(y~xlz)	scatterplot of y as a function of x for each value (or interval of values) of z
hist(x)	histogram of the frequencies of x
barplot(x)	histogram of the values of x

Boxplots are informative graphics to use with categorical variables. They inform on the overall distribution of the data (its location, its spread, whether it is skewed). The horizontal line within the box represents the median of the response variable for each category of the factor. The hinges of the box are roughly the 1st and 3rd quartiles of the distribution of the response variable. The “whiskers” are extreme data points within a factor of 1.5 of the hinge. Outliers (data beyond the whiskers) are shown by themselves.

```
boxplot(scores~breakfast,data=df)
# there are no outliers in this case. We can clearly see the difference between
# different breakfasts and students exam scores
# boxplots are also a good way to spot errors in data entry.
```

2. Graphical parameters

?par will give you ALL the possible graphical parameters to customise your plot.

```
# Let's customise our plot of weight vs height
plot(weight~height, data=df)
# add axis labels
plot(weight~height, xlab="Student Height (m)", ylab="Student Weight (kg)", main=("Student
weight as a function of height"), data=df)
# give a different colour to student's breakfast choice
plot(weight~height,xlab="Student Height (m)", ylab="Student Weight (kg)", main=("Student
weight as a function of height"), col=(breakfast), data=df)

# We can adjust and x and y dimension using xlim and ylim.
plot(weight~height,xlab="Student Height (m)", ylab="Student Weight (kg)", main=("Student
weight as a function of height"), ylim=c(0,100), col=(breakfast), data=df)
```

3. Graphic commands

Once you have created a plot, you may want to add some more data or some text/legend to the plot. Here is a table with some of the most useful commands.

points(x,y)	adds points of coordinates
lines(x,y)	add lines of coordinates
abline(v=)	add a vertical line
abline(h=)	add a horizontal line
abline(lm(y~x))	add a regression line of y~x
arrows(x1,y1,x2,y2, code=3, angle=90)	add arrows between point 1 and 2. Useful for confidence intervals
legend(x,y,text)	add legend at coordinates x,y
title()	add a title
axis()	add axis to the plot

```
# Let's first plot the data points of muffin eaters in blue filled triangles
plot(weight~height, xlab="Student Height (m)", ylab="Student Weight (kg)", main=("Student
weight as a function of height"),
    col="blue", pch=17,  ylim=c(60,100),  xlim=c(1.5,2.2), data=df[df$breakfast=="muffin",])

# We have subset the muffin eater from the data frame by this command:
#data=df[df$breakfast=="muffin",]
#now we can add the data points for another breakfast type in red filled circles
points(weight~height, col="red", pch=16, data=df[df$breakfast=="fry.up",])
```

4. Add a linear regression lines to a graphic

This is a straightforward process involving your first bit of statistics in R!

```
# Let's focus only on muffins eaters first by making a subset of the data with just muffin eaters:
df.muffins<-df[df$breakfast=="muffin",]
# Create mod1, an object that is going to contain the linear model of weight as function of height
for the students that have muffins for breakfast
mod1<-lm(weight~height, data=df.muffins)
# Plot the muffin points
plot(weight~height, xlab="Student Height (m)", ylab="Student Weight (kg)",
main=("Student weight as a function of height"), col="blue", pch=17,  ylim=c(60,100),
xlim=c(1.5,2.2), data=df[df$breakfast=="muffin",])

# plot the line of the regression on the graph using function abline()
abline(coef(mod1))
# coef(mod1) gives the intercept and slope of the model calculated from df.muffins data frame.
# you may want to have a line that is blue and wider
abline(coef(mod1), lwd=2, col="blue")

# now lets do the same for the fry.up eating students
df.fry.up<-df[df$breakfast=="fry.up",]
mod2<-lm(weight~height, data=df.fry.up)
# add these objects to your existing plot by running the following code
points(weight~height, col="red", pch=16, data=df[df$breakfast=="fry.up",])
abline(coef(mod2), lwd=2, col="red")

# You might also want to add a legend to your plot
legend('topleft', c("Muffin", "Fry Up"), lty=1, col=c('blue', 'red'), bty='n', cex=1)
```

5. Save a graphic

When you are on the R-Studio Graphics window, you can go to the Export tap and save the image in various formats (e.g. png, pdf). There is a whole range of options available.

If you are planning to create numerous graphs you can save them directly into your folder. For help look at **?png**

```
# Save using pdf - first give a name to your plot
pdf("FirstRplot.pdf", width = 8, height = 6)
# plot your data
plot(weight~height, xlab="Student Height (m)", ylab="Student Weight (kg)",
main=("Student weight as a function of height"), col="blue", pch=17,  ylim=c(60,100),
xlim=c(1.5,2.2), data=df[df$breakfast=="muffin",])
abline(mod1, lwd=2, col="blue")
points(weight~height, col="red", pch=16, data=df[df$breakfast=="fry.up",])
abline(mod2, lwd=2, col="red")
legend('topleft', c("Muffin", "Fry Up"), lty=1, col=c('blue', 'red'), bty='n', cex=1)

# close the graphic window
dev.off()
# You should now be able to look at the picture in your working directory, if you forgot where
this is use getwd()
```

You may also want to save your data frame by exporting it as a csv file

```
# save data frame as csv to open in excel
write.csv(students, file = "students.2.csv", row.names = FALSE)
```

Packages

Installing Packages

R already has an implemented set of commands and functions. However, you may want to use some functions that are not implemented by default and are located in other packages. The full list of packages is available from the R cran webpage http://cran.r-project.org/. Over the next few weeks we will use various packages from this list.

Below I have included several sets of instructions for loading packages in R. Which set of instructions you need will depend on the system you are working on (uni PC, personal computer with admin rights, computer without admin rights).

1. For the most straightforward situation (personal computer with admin rights) the following set of instructions should work.

- Click on Tools in the main R Studio toolbar
- Select Install packages…
- Choose package **ggplot2** and select install (this will automatically download the files needed for the ggplot2 package)
- To use the package you will also need to load ggplot2 for each R session (see the 'Plotting with package ggplot2' section below)

2. For University PC's you need the following set of instructions.

- Open the C Drive and add a new folder called **Rpackages**
- Open R studio
- Add and run the following code at the start of your script **.libPaths("C:/Rpackages")**
- Now use the instructions for section 1 above to install ggplot2 and load the package.

3. If you do not have administrative rights, install packages into a working folder and then attach them using the following code.

```
folder<- getwd()
options(repos="http://cran.ma.imperial.ac.uk")
install.packages("ggplot2", destdir=folder, lib=folder) # do not forget quotes
library(ggplot2,lib=folder) # once the package is installed you still need to load
# it using the command library()
```

Plotting with package ggplot2

R is very flexible and by using additional packages you can expand the number of functions that can be implemented. Below are some plotting tools for the plotting package ggplot2. This package simplifies many of the more tricky plotting applications such as error bars and modelled lines.

```
library(ggplot2) # load package

ggplot(df, aes(x=weight, y=height, color=breakfast)) +
  geom_point(shape=1) +
  geom_smooth(method=lm, se=TRUE) +
  facet_grid(breakfast ~ .)

# use http://www.cookbook-r.com/Graphs/ to explore further plotting tools in ggplot2
```

The end: Getting these basic skills down is very important before we move into the statistical analysis. Make sure you have understood the practical material today and have a clean **working** and **annotated** script before moving on.

Bonus work

If you would like more practice on these basic skills then please access the swirl package in R (see http://swirlstats.com for instructions) and work through the first lesson. The swirl package runs in R and instructs you as you work through the activities.