Data Science in the Classroom

Introduction

For this activity we will be exploring the R statistical language using the R Studio Integrated Development Environment (IDE). An IDE is an application used to consolidate various features of a programming environment and can make the process of coding faster and easier.

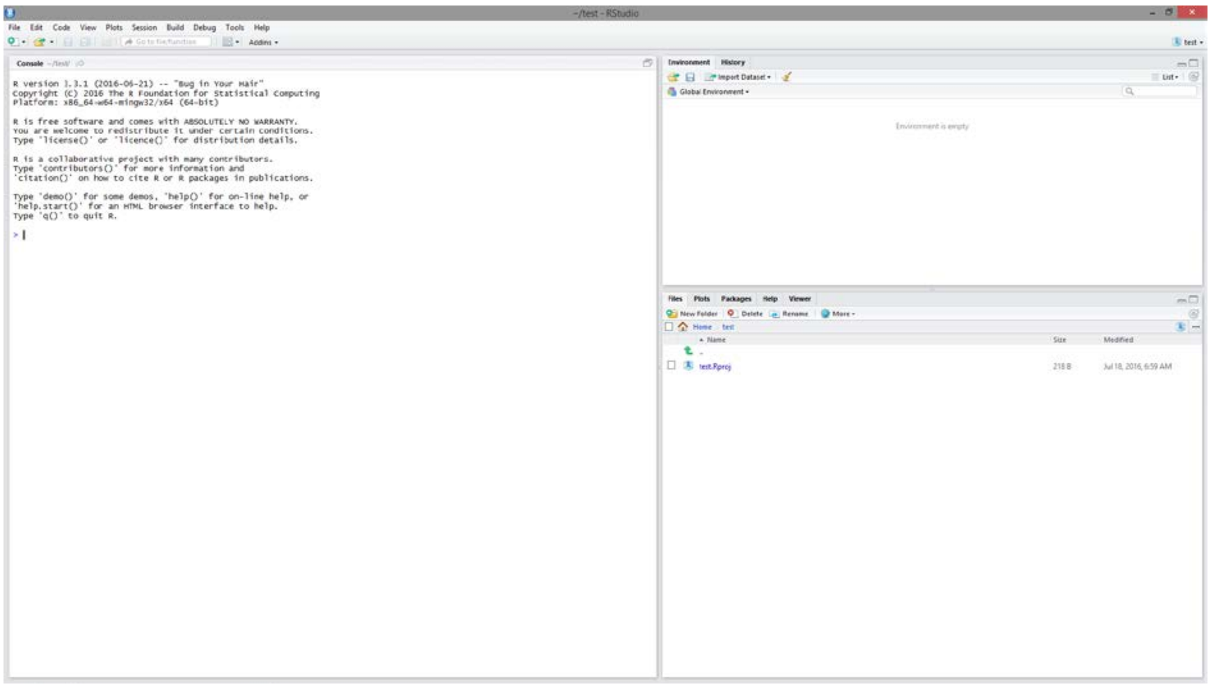
To complete this activity at school or home you will need to have R version 2.11 or higher installed on your computer in order to install the R Studio IDE.

Loading R Studio

If you are using Windows, go to your Start Button and type in “RStudio”.

Click on the R Studio Icon to open the application.

If everything is successful, R Studio will open and you will see the following screen:



```
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 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.

>
```

The left window shows the Console view. This gives you a way to interact with the R environment by typing instructions and reading system messages.

The greater-than symbol, > is the prompt that tells us that R Studio is ready for interaction.

Hello World

To output a message to the console you need to type the following command:

```
print("Hello R!")
```

Try changing the message between the speech marks to make R output your own custom greeting to the console.

Notice that this instruction is made from two parts; the print command and the content to be printed. The use of parenthesis also tells us that we are dealing with a function. A function is a set of procedure and routines. A function might accomplish a task like calculating the area of a circle or finding the square root of a number. The R language is based the idea of nesting functions within functions.

Hello Mathematics

You can also input mathematics instructions using the following operands:

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

Try typing the following to the console:

```
5^2
```

This is telling R to take 5 and square it, ie. 5^2. The resulting answer is output to the console.

What is Modulus?

What is the Modulus of 10 and 3?

Variables in R

In R, you can create and assign variables using the following code:

```
x <- 5
```

This instruction tells R to create a variable called ‘x’ and store the number 5 inside it.

The R language refers to its variable objects as vectors. A vector is a sequence of data elements of the same basic type.

To show the contents of ‘x’ you use the print function:

```
print (x)
```

You can also just type the object name and hit enter.

What would be the output of the following code?

```
hours_worked <- 10
rate_of_pay <- 20
print (hours_worked * rate_of_pay)
```

Variable naming follow the conventions used for most other languages:

Try to use descriptive names, separate each word with a capital letter or underscore.

Don’t start variable names with numbers or symbols. Don’t use words that are also used as instructions by R. Don’t use any other symbol other than an underscore.

Which of the following are legal variable names in R?

```
30daysOfSummer
print
$gogo_machine
Xy26576276
time_of_day
```

Programming in R

The easiest way to start programming in R is to enter commands sequentially to the console.

Enter the following code to print the 10 times tables. The paste command allows string concatenation using comma separated values. After the ‘for’ command is entered, a plus symbol + will appear to let you know to enter the rest of the commands on a new line.

```
for(i in 1:5)
{print(paste("10 x", i))}
```

RStudio will output the following:

```
[1] "10 x 1"
[1] "10 x 2"
[1] "10 x 3"
[1] "10 x 4"
[1] "10 x 5"
```

Modify the code above to increase to loop from 5 times to 10 times.

Modify the print/paste command using additional commas and speech marks to output the answers of the multiplication. The output should look like the following:

```
[1] "10 x 1 = 10"
[1] "10 x 2 = 20"
[1] "10 x 3 = 30"
[1] "10 x 4 = 40"
[1] "10 x 5 = 50"
[1] "10 x 6 = 60"
[1] "10 x 7 = 70"
[1] "10 x 8 = 80"
[1] "10 x 9 = 90"
[1] "10 x 10 = 100"
```

Your modified code would look like the following:

```
for(i in 1:10)

+ {print(paste("10 x", i, "=", 10 *i))}
```

`A while loop looks like the following:`

```
x <- 1
while(x < 5) {x <- x+1; print(x);}
```

Data Science in R

The main benefit to using R for number crunching over non-statistical programs like Excel, is the ability to handle enormous quantities of data, and with accuracy. Excel has been known to have mathematical issues that can render incorrect results with large floating point values.

As mentioned earlier, we store objects of the same data type inside vectors. We access the vector storage command using ‘c()’. This is demonstrated in the following way:

ages <- c(13,14,13,12,14,15,15,13,14,11,15)

This creates a vector with the above list of integers. Vectors are like arrays in other programming languages. It’s important to note that you can nest vectors inside other vectors like the following:

```
ax <- 'cat'
ay <- 'dog'
animals <- c(ax, ay)
```

Making Pictures

We will now attempt to plot data. Firstly, create an x object and store the integers from 1 to 10 inside it:

```
x <- c(1,2,3,4,5,6,7,8,9,10)
```

This will be our independent variable (eg. Time).

Next create a y object and store the following values:

```
y <- c(3,6,9,12,15,18,21,24,27,30)
```

This could be our dependent variable (eg. Temperature).

You can create a simple plot of these two objects using the ‘plot()’ function:

```
plot (x, y)
```

Give your graph a title using the main attribute:

```
plot (x,y, main = "time vs temperature")
```

X and Y axes are labelled using ‘xlab’ and ‘ylab’:

```
plot (x,y, main = "time versus temperature", xlab="time", ylab="temperatur
e")
```

Making Predictions with R

Using the time and temperature values above, pretend that we wanted find the value of the temperature when the x value (time) was 6.5 units. Given the linearity of the values, this would be easy for us to guess.

We use our guesses to approximate a ‘function’ for the values.

For example, what would be the function associated with the following data?

2, 4, 6, 8, 10, 12

3.98, 8.67, 15.76, 24.89, 35.789

As the data becomes more complex, we need to use the statistical idea of Linear Regression.

Linear Regression attempts to model a function based on input data using the formula of a straight line:

$y = mx + b$

To use Linear Regression in R we use the 'lm()' function. It takes two objects, the dependant variable followed by the independent variable, separated by the tilde symbol ~ :

```
lm(y ~ x)
```

If you typed this into R, it would return the following output:

```
Coefficients:
(Intercept)            x
 -4.494e-15    3.000e+00
```

We can use this information to make future predictions by performing the following actions:

```
linear_regression <- m(y ~ x)
```

This stores the information inside an object called linear_regression.

```
coefficient <- coefficients(linear_regression)
```

This stores the coefficient information for our linear regression inside an object called coefficient.

```
x_value = 12
```

This is the value we want to use to make a prediction.

```
prediction = coefficient[2] * x_value + coefficient[1]
```

This is our formula of a line, y = mx + b. The answer is stored inside the prediction object.

Type prediction to see the value of y when x is 12.

The completed code looks like the following:

```
x <- c(1,2,3,4,5,6,7,8,9,10)
y <- c(3,6,9,12,15,18,21,24,27,30)
linear_regression <- lm(y ~ x)
coefficient <- coefficients(linear_regression)
x_value = 12
prediction = coefficient[2] * x_value + coefficient[1]
prediction
```

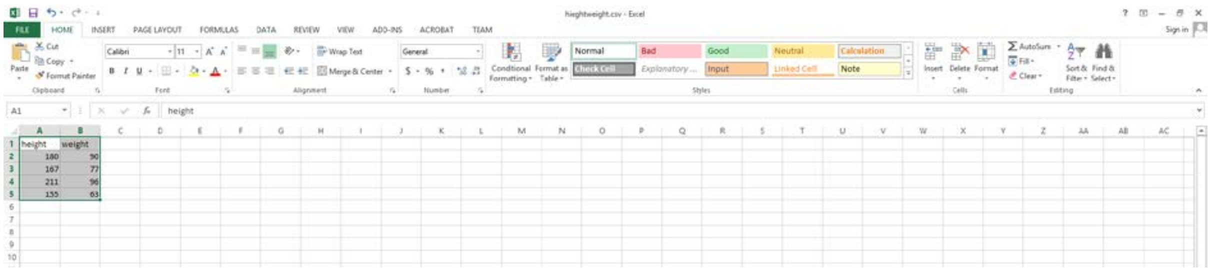
Height and Weight Prediction Activity

Get all the students in the class to enter values for their height and weight in place visible by all.

Using the newly acquired knowledge above, find the Linear Regression information for the height and weight samples in the classroom and use it to make a prediction about the teachers weight given their height.

Importing from Excel

Excel has easy and familiar to use interface for data entry.



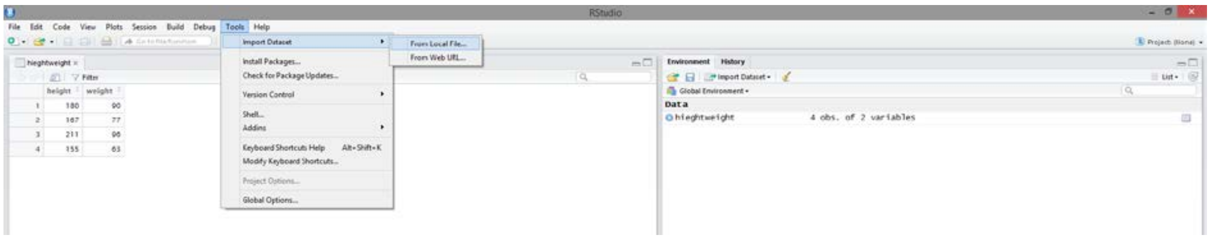
A1 height

	A	B
1	height	weight
2	180	90
3	167	77
4	211	96
5	155	63

Note that you should always start at the very top left and avoid unnecessary information entered into the tables. Keep it simple. One top row for headings will work for an import.

Export the spreadsheet as a .csv (comma separated) file.

Inside RStudio goto tools -> import dataset:

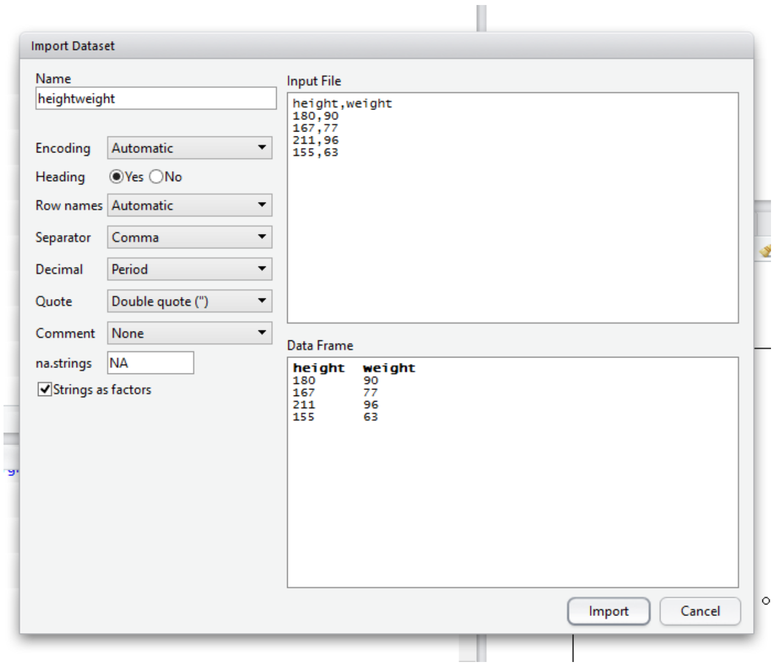


	height	weight
1	180	90
2	167	77
3	211	96
4	155	63

```
Data
hieghtweight          4 obs. of 2 variables
```

Give your data set object a name. I've called mine 'heightweight'.

We use this name to perform actions on that set of data:



Name: heightweight
Encoding: Automatic
Heading: [✓ Yes] No
Row names: Automatic
Separator: Comma
Decimal: Period
Quote: Double quote (")
Comment: None
na.strings: NA
[✓ Strings as factors]

Input File

```
height,weight
180,90
167,77
211,96
155,63
```

Data Frame

```
height   weight
180      90
167      77
211      96
155      63
```

Try typing in the name of your data set once it has loaded. It should return all the values:

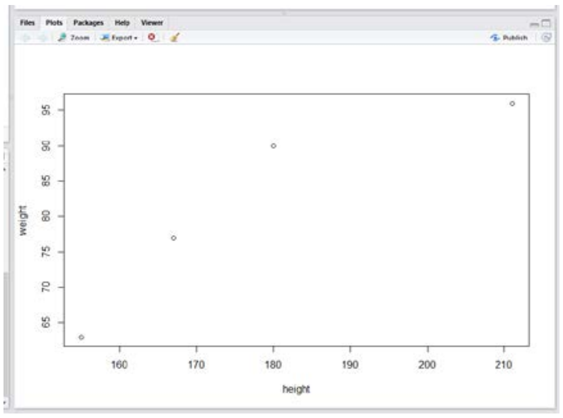
```
Heightweight
```

Next use the summary function to give you statistical information about your data:

```
summary(heightweight)
```

And then try the old plot function:

```
plot(heightweight, main=”height vs weight”, xlab=”height”, ylab=”weight”)
```


Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
weight
height