Basics of R Studio

The module covers concepts such as:

- Setting up R Studio
- Getting Data into R
- Tidying Data
- Basic Analysis
- Data Visualization with ggplot2

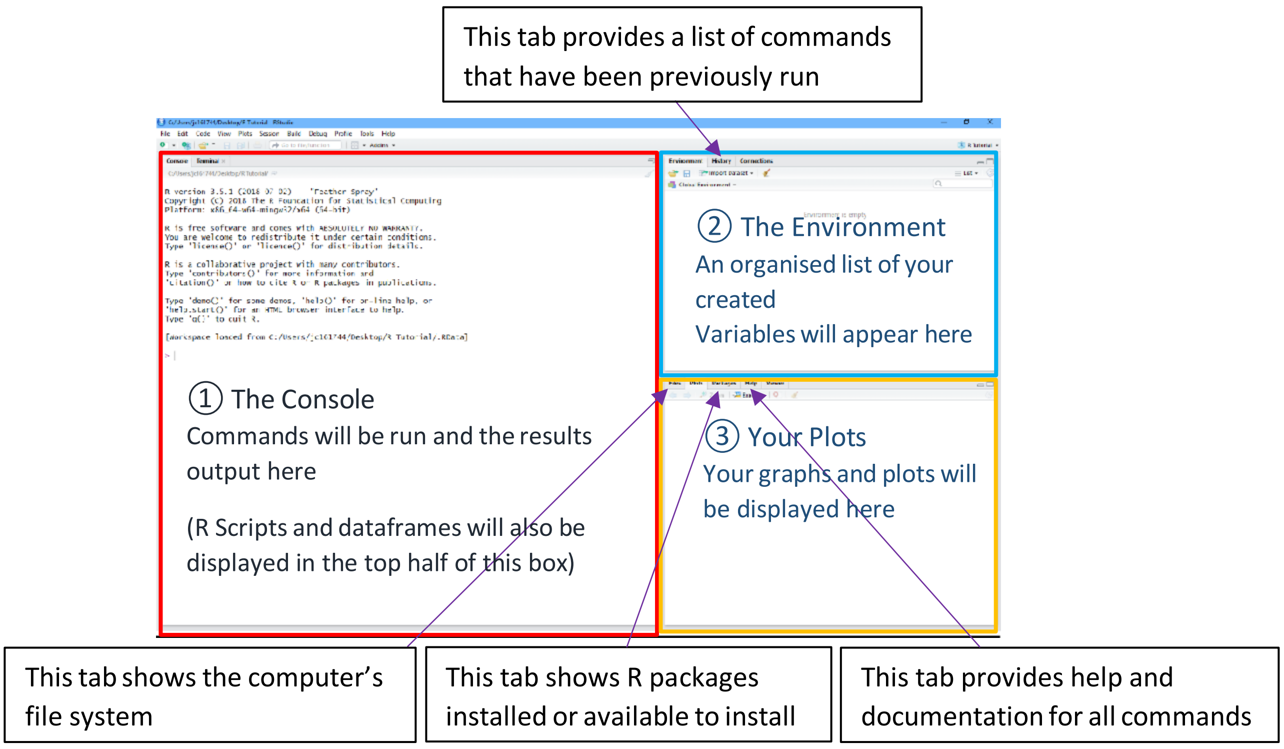
TheLearningCentre

UNLOCK YOUR POTENTIAL

1. Setting Up R-Studio

The R Studio Interface

When opening R Studio, you will be presented with the following screen. The colours and fonts may vary depending on your Operating System but the layout will be the same.



This tab provides a list of commands that have been previously run

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 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.

[Workspace loaded from C:/Users/jc161744/Desktop/R Tutorial/.RData]

> |
```

① The Console
Commands will be run and the results output here

(R Scripts and dataframes will also be displayed in the top half of this box)

② The Environment
An organised list of your created Variables will appear here

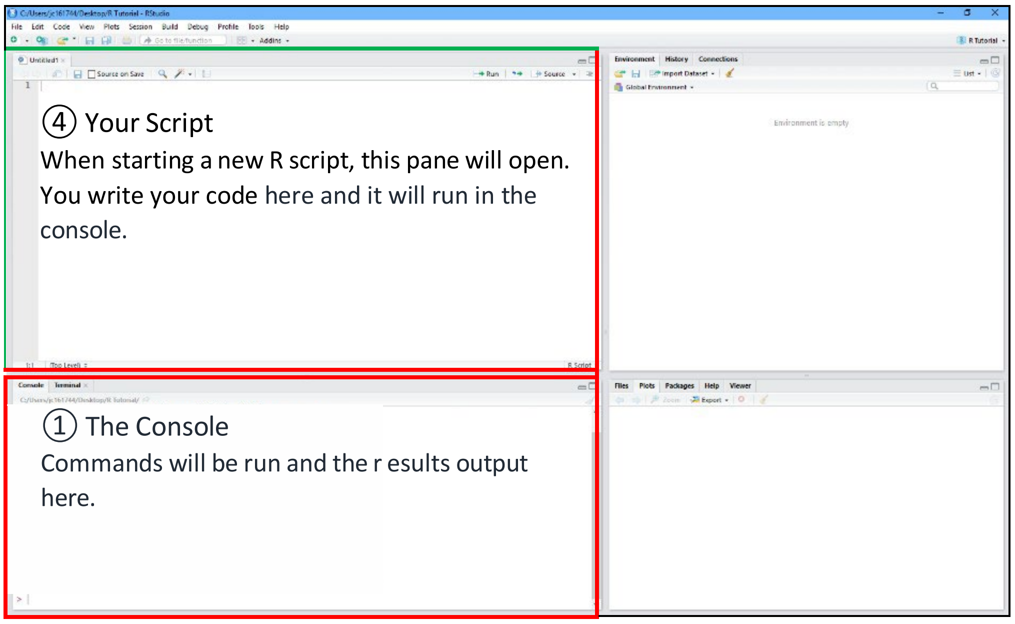
③ Your Plots
Your graphs and plots will be displayed here

This tab shows the computer's file system

This tab shows R packages installed or available to install

This tab provides help and documentation for all commands

On creating a new R script (see next section), the script panel will open. You will write your R code/script in here and it will be run in the console.

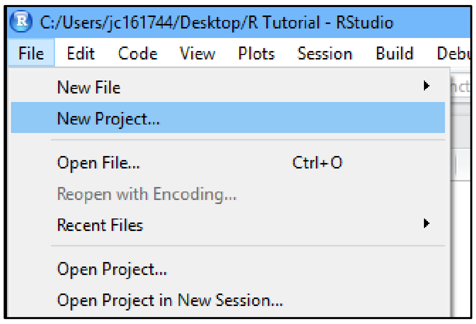


④ Your Script
When starting a new R script, this pane will open. You write your code here and it will run in the console.

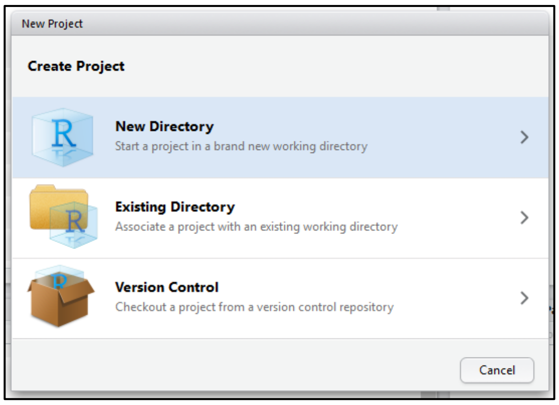
① The Console
Commands will be run and the r esults output here.

Start a New Project in R

When starting to work with a new dataset, a New Project should be created.



Creating a New Directory makes a default working directory and a logical place to store all associated files such as raw data spreadsheets.


New Project
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

Any associated excel documents or text files can be saved into this new folder and easily accessed from within R. You can then perform data analysis or produce visualisations with your imported data.

Directions:

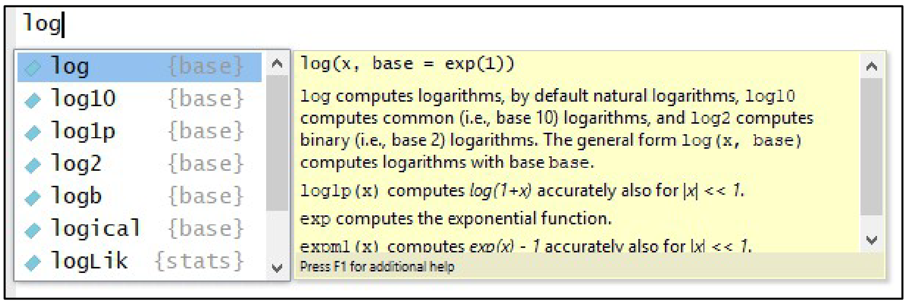
1. Open R Studio
2. Create a New Project using a New Directory in a location of your choosing

2. Basic Commands in R Studio

Run a Command

Base R Cheat Sheet: https://www.rstudio.com/resources/cheatsheets/

Commands in R can be entered directly into the console at the bottom left of the screen. Entering 3 or more characters of a command into the console or a script will open the suggested command menu. This menu suggests commands or the names of variables you have intended to type, alongside a description and suggested use. Entering ? followed by any R command will open a help page in the help tab found in the bottom right hand corner of the screen (eg. ?log10 opens the log10 help page). This help page will offer settings and formatting for each command, as well as an example.



```
log
```

log	{base}
log10	{base}
log1p	{base}
log2	{base}
logb	{base}
logical	{base}
logLik	{stats}

```
log(x, base = exp(1))
```

log computes logarithms, by default natural logarithms, log10 computes common (i.e., base 10) logarithms, and log2 computes binary (i.e., base 2) logarithms. The general form log(x, base) computes logarithms with base base.

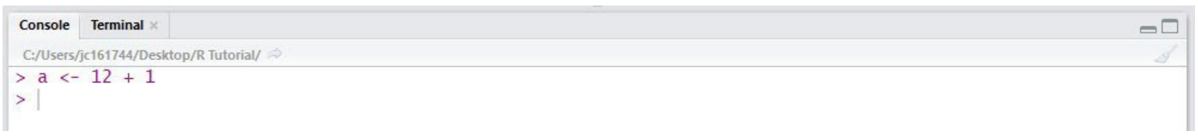
log1p(x) computes *log(1+x)* accurately also for |*x*| << *1*.

exp computes the exponential function.

expm1(x) computes *exp(x) - 1* accurately also for |*x*| << *1*.

Press F1 for additional help

On completing a command and pressing enter, R will immediate run the code, print the output and move to a new line. Using the ↑ key will repeat the last command entered into the console.



```
> a <- 12 + 1
> 
```

At its heart, R is a calculator and can accept any mathematical calculation directly into the console. The following table represent just a few of the available mathematical operators:

R Command	Mathematical Operator	Example
*	x (multiply)	10 * 2 = 20
/	÷ (divide)	50 / 2 = 25
^	(exponent/power)	10^2 = 10^2 = 100
log10(x)	$\log_{10}x$	log10(100) = 2
log(x, base)	$\log_{base}x$	log(25,5) = 2
sum(a,b,c)	a + b + c	sum(1,2,3,4) = 10

Directions:

Enter the following commands into the console, pressing enter after every line:

1. (5^2+5)/3

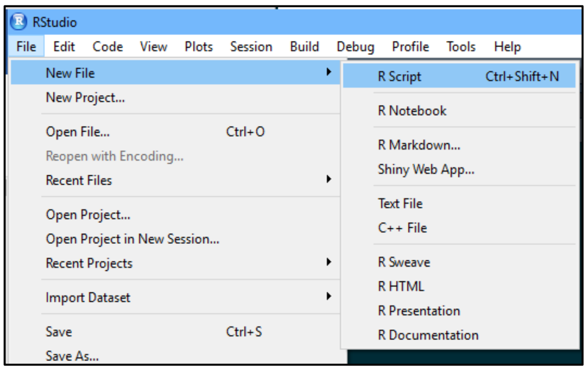
2. log10(90+5*2)

3. sum(50, 3, 5)

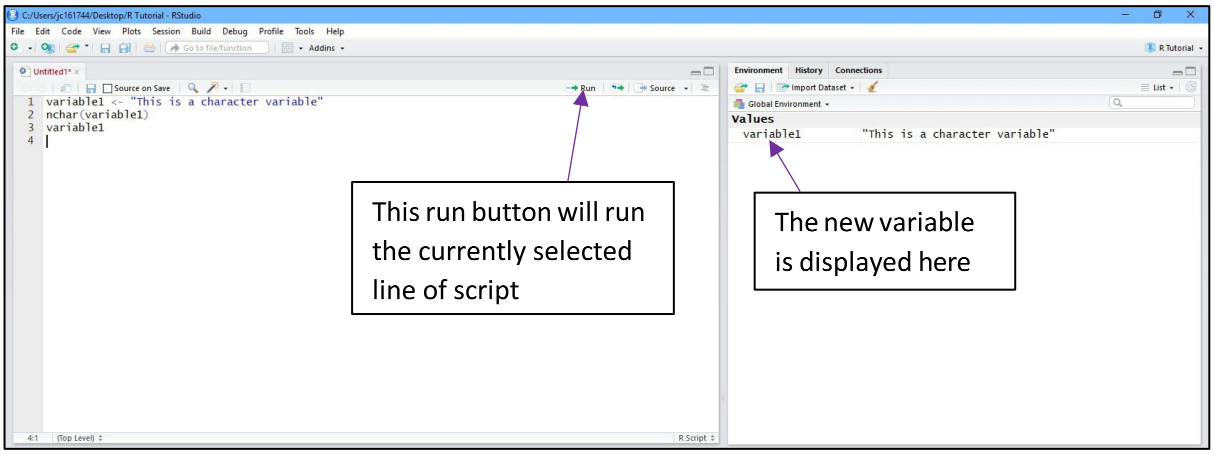
4. sum(log(32,2), 8^4, 4+5, 12/3)

Using R Scripts

Commands can be run directly from the console, but creating an R Script allows you to edit and reuse previous commands and to create more complicated lists of commands. A script also allows you to save your commands to be reopened later.



Multiple commands can be entered into a script, one after the other across multiple lines.



```
variable1 <- "This is a character variable"
nchar(variable1)
variable1
```

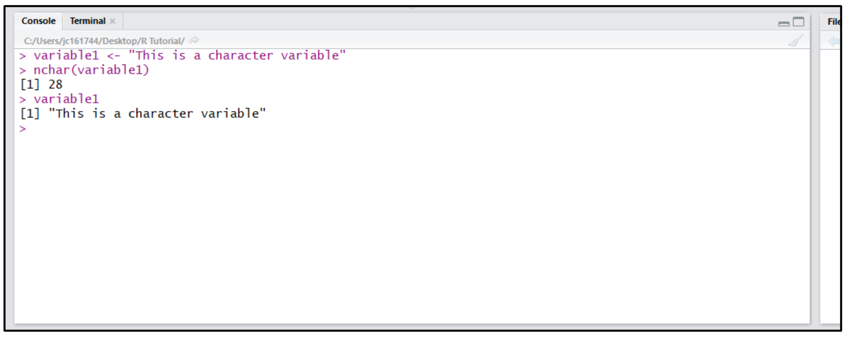
```
Values
  variable1           "This is a character variable"
```

This run button will run the currently selected line of script

The new variable is displayed here

The script above creates a new variable called variable1 with the value “This is a character variable”. The next command counts the number of characters in the variable. The final command returns the value of variable1.

To run the script one line at a time, navigate the cursor to the appropriate line and press CTRL + Enter. To run all commands from the start, press CTRL + Shift + Enter.



```
C:/Users/jc161744/Desktop/R Tutorial/
> variable1 <- "This is a character variable"
> nchar(variable1)
[1] 28
> variable1
[1] "This is a character variable"
>
```

Directions:

1. Create a new script

2. Enter the following code into your script:
   variable2 <- c(12,1,15,3,20,6,11)
   variable2
   mean(variable2)
   variable2 + 2

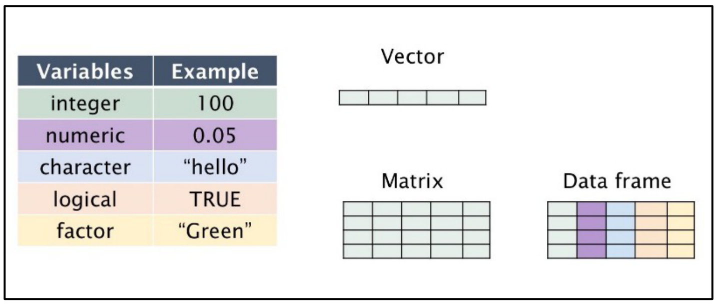
3. Run your code (Ctrl + Shift + Enter)

4. Save your script

3. Getting Data into R-Studio

Getting Data into R

Variables can be thought of as a labelled container used to store information. Variables allow us to recall saved information to later use in calculations. Variables can store many different things in R studio, from single values to tables of information, images and graphs.

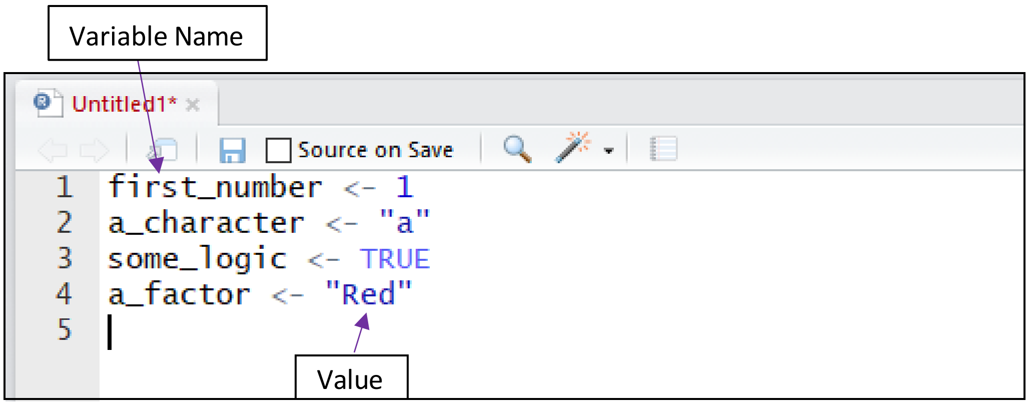


Variables	Example
integer	100
numeric	0.05
character	“hello”
logical	TRUE
factor	“Green”

(Source: https://sydney-informatics-hub.github.io/lessonbmc/02-BMC_R_Day1_B/index.html)

Assigning a Value to a Variable:

Storing a value or “assigning” it to a variable is completed using either <- or = function. The name given to a variable should describe the information or data being stored. This helps when revisiting old code or when sharing it with others.



```
first_number <- 1
a_character <- "a"
some_logic <- TRUE
a_factor <- "Red"
```

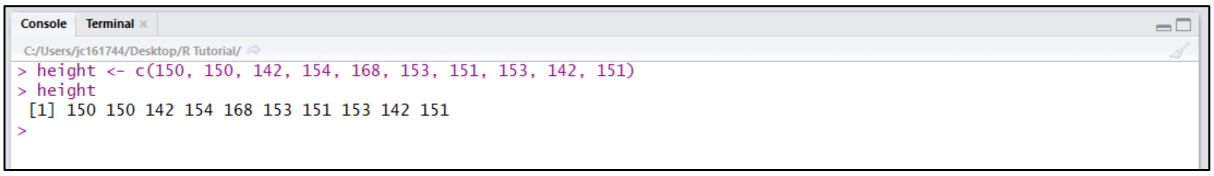
Creating a Vector

A variable can hold one value or many values. A vector is used to store more than one value of the same type. The *combine or c()* function (type *?c* in the console for the further information) allows us to combine them into a single list of values.

Example:

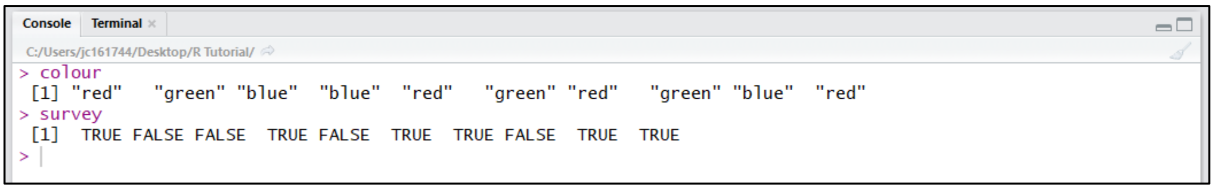
A class of 10 students have been surveyed and their heights recorded:
150cm, 150cm, 142cm, 154cm, 168cm, 153cm, 151cm, 153cm, 142cm and 151cm

To begin analysing this data in R, the values must be stored in a variable. In the following, the variable *height* has been created to contain our values:



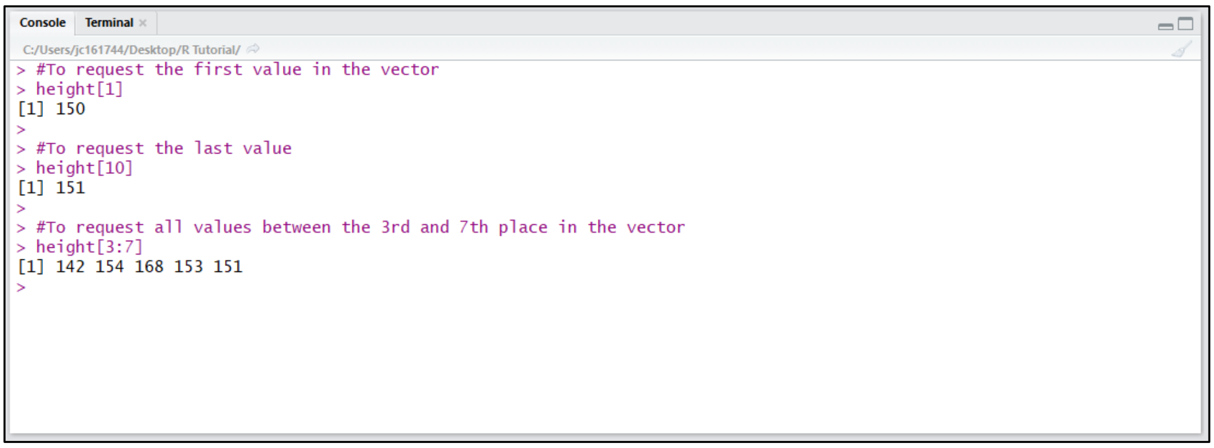
```
> height <- c(150, 150, 142, 154, 168, 153, 151, 153, 142, 151)
> height
 [1] 150 150 142 154 168 153 151 153 142 151
>
```

The vector can then be recalled using the variable name *height*. Vectors for colour (text strings) and survey (logical values) have also created.



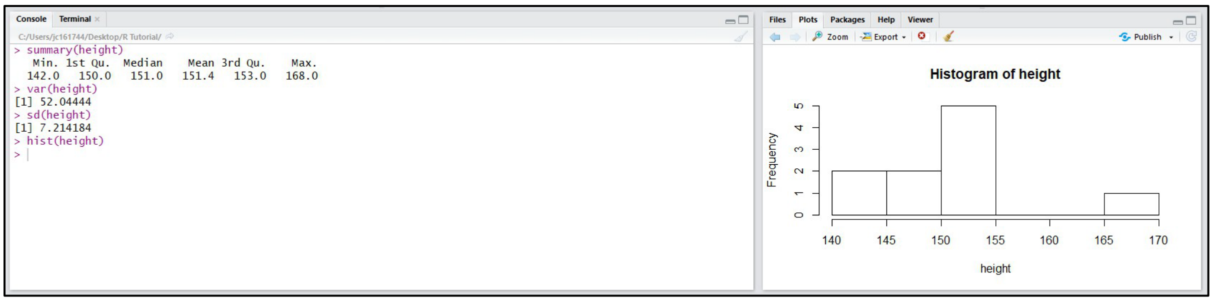
```
> colour
 [1] "red"   "green" "blue"  "blue"  "red"   "green" "red"   "green" "blue"  "red"
> survey
 [1]  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
>
```

To recall a single value from the vector, use the variable name followed by the position of the value in the list and surrounded by [ ] brackets.



```
> #To request the first value in the vector
> height[1]
[1] 150
>
> #To request the last value
> height[10]
[1] 151
>
> #To request all values between the 3rd and 7th place in the vector
> height[3:7]
[1] 142 154 168 153 151
>
```

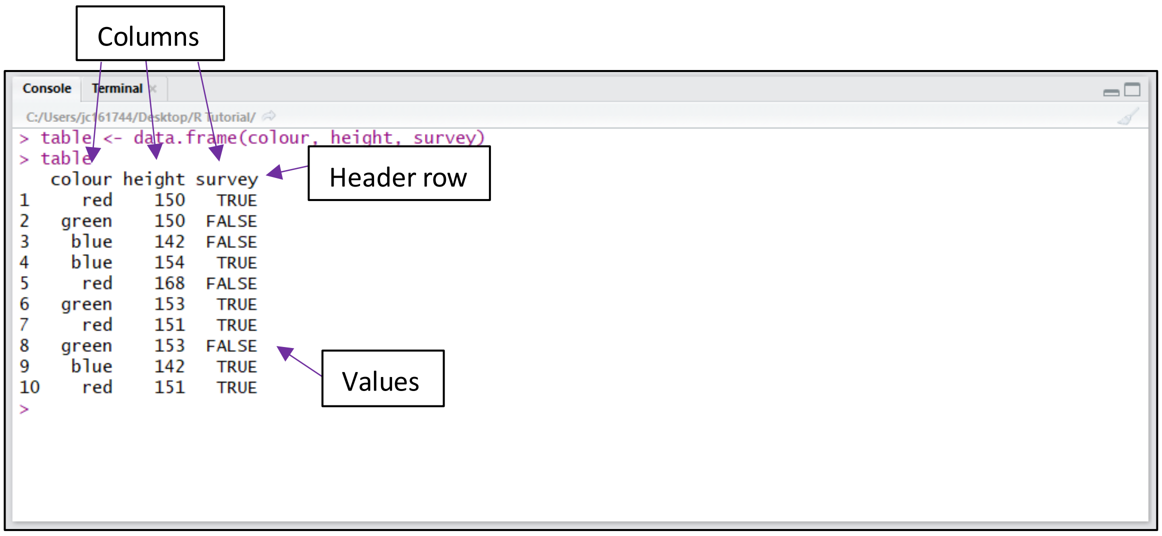
Useful statistical analysis can be performed on single set of values using vectors:



```
> summary(height)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  142.0   150.0   151.0   151.4   153.0   168.0 
> var(height)
[1] 52.04444
> sd(height)
[1] 7.214184
> hist(height)
> 
```

Combining Vectors into a Dataframe

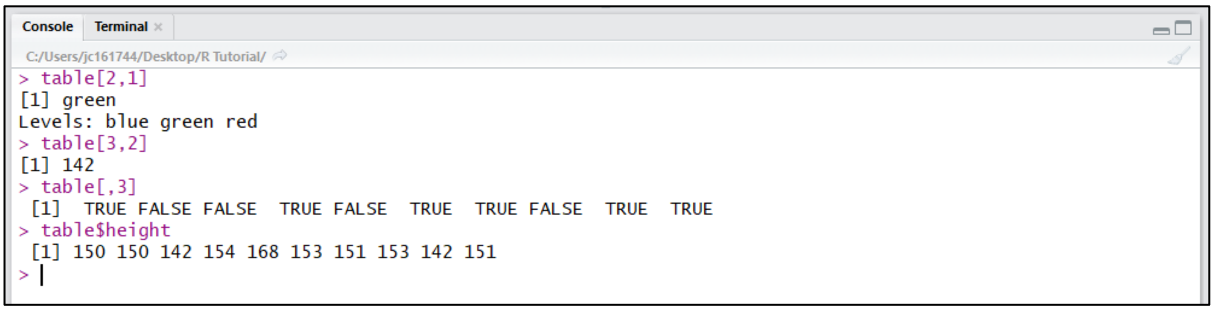
A dataset will often contain more than 1 variable of interest. To combine vectors of the same length together into a table or "dataframe" (similar to an excel table), we use the *data.frame()* command. Below we have combined 3 vectors (colour, height and survey), containing 10 values each, together into a dataframe. The table has been assigned to the variable *table*.



```
> table <- data.frame(colour, height, survey)
> table
  colour height survey
1    red    150   TRUE
2  green    150  FALSE
3   blue    142  FALSE
4   blue    154   TRUE
5    red    168  FALSE
6  green    153   TRUE
7    red    151   TRUE
8  green    153  FALSE
9   blue    142   TRUE
10   red    151   TRUE
>
```

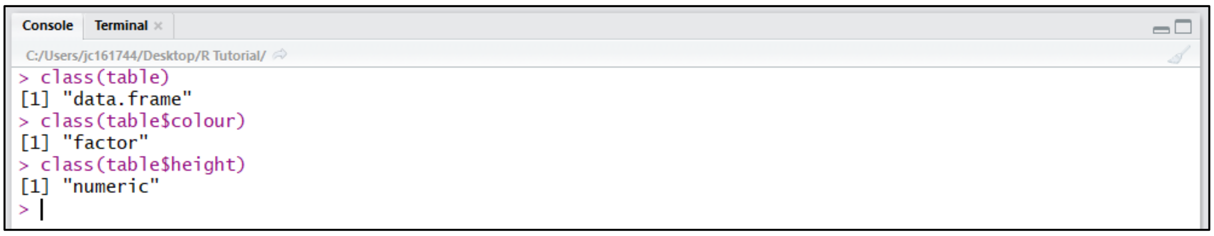
The top line of the dataframe is called the header row and contains a descriptive name for the values in each column.

Below the header row, values can be referred to individually or in whole rows and columns. To retrieve a single value from a dataframe assigned to a variable, the variable name is used then followed by the coordinates of the value within [row, column] square brackets (eg. table[2,1] returns green and table[3,2] returns 142). Whole rows can be returned individually using coordinates (eg. table[3,] returns all values for row 3). Single columns can be returned by either referring to them using a coordinate with the row blank (eg. table[,3] returns all values in 3rd column), or by using the variable name followed by a $ and the column header (eg. table$height returns all "height" values from the dataframe "table").



```
> table[2,1]
[1] green
Levels: blue green red
> table[3,2]
[1] 142
> table[,3]
 [1]  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
> table$height
 [1] 150 150 142 154 168 153 151 153 142 151
> |
```

Note: You can find out what type of data is stored within a variable or a column in a dataframe using the class( ) function.



```
> class(table)
[1] "data.frame"
> class(table$colour)
[1] "factor"
> class(table$height)
[1] "numeric"
> |
```

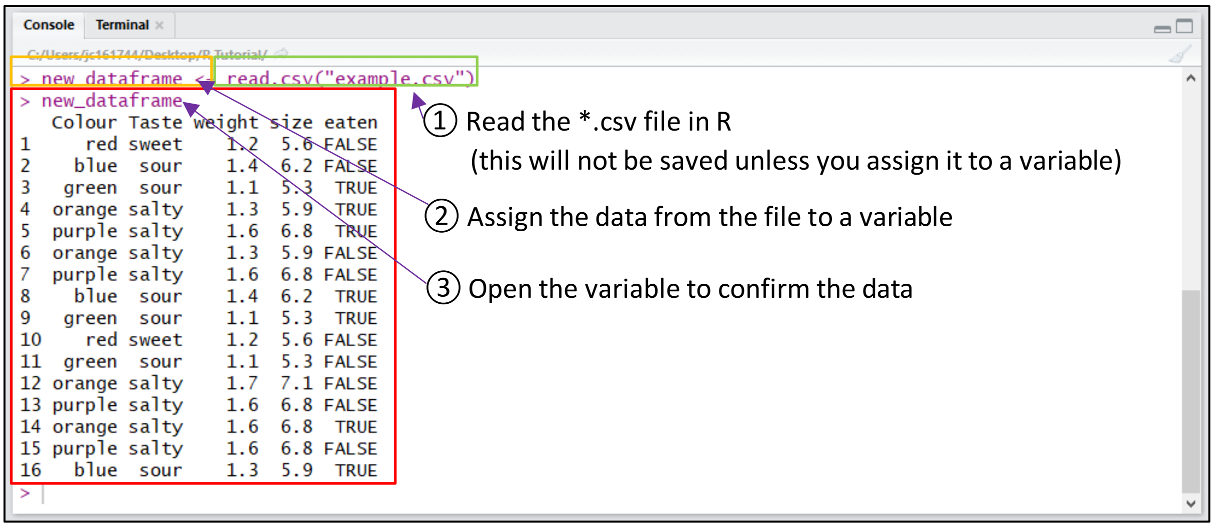
Some examples of classes are: “matrix”, “data.frame”, ”array”, “factor”, “numeric”, “logic”

Directions:

1. Create vectors
2. Create a dataframe
3. Use coordinates to find cells
4. Use coordinates to find rows/columns
5. Use variable names
6. Use coordinates to produce a table WITHOUT a row or column

Importing an Excel File into R

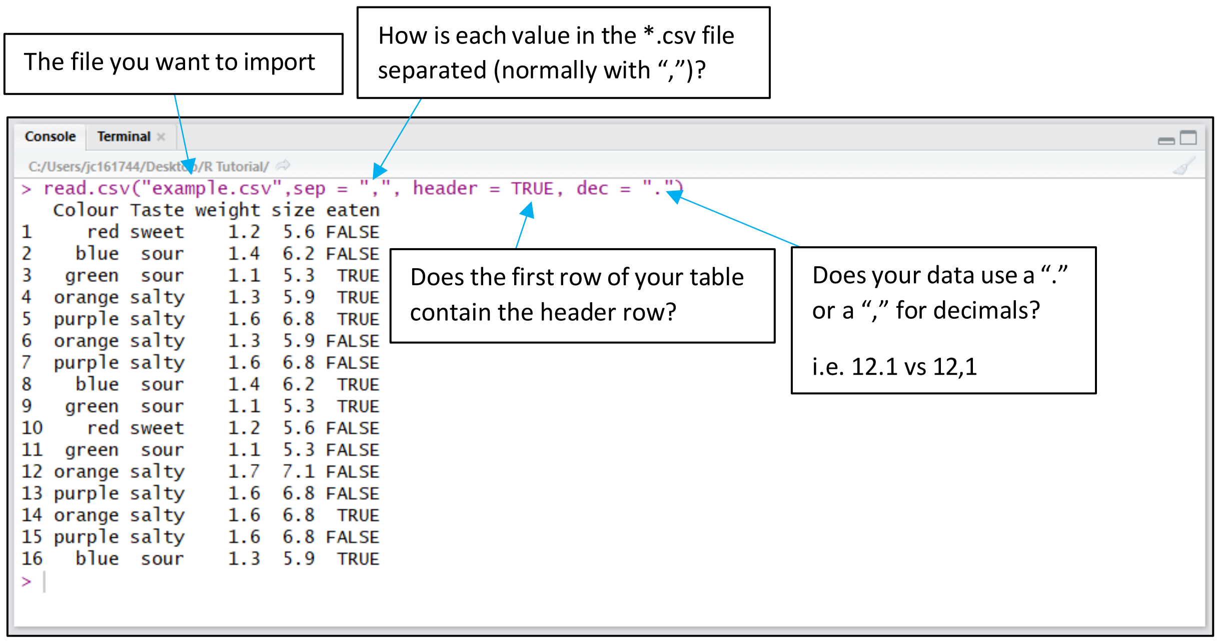
In Excel, simplify your table to basic column titles and no unnecessary data as R will attempt to manipulate the structure into columns. Save your excel file in *.csv (MS-DOS) into your R Project folder. In R Studio use the read.csv function with your file name in " " to read:



Console
Terminal
① Read the *.csv file in R
(this will not be saved unless you assign it to a variable)
② Assign the data from the file to a variable
③ Open the variable to confirm the data

```
C:/Users/jc161744/Desktop/R Tutorial/
> new_dataframe <- read.csv("example.csv")
> new_dataframe
   Colour Taste weight size eaten
1     red sweet    1.2  5.6 FALSE
2    blue  sour    1.4  6.2 FALSE
3   green  sour    1.1  5.3  TRUE
4  orange salty    1.3  5.9  TRUE
5  purple salty    1.6  6.8  TRUE
6  orange salty    1.3  5.9 FALSE
7  purple salty    1.6  6.8 FALSE
8    blue  sour    1.4  6.2  TRUE
9   green  sour    1.1  5.3  TRUE
10    red sweet    1.2  5.6 FALSE
11  green  sour    1.1  5.3 FALSE
12 orange salty    1.7  7.1 FALSE
13 purple salty    1.6  6.8 FALSE
14 orange salty    1.6  6.8  TRUE
15 purple salty    1.6  6.8 FALSE
16   blue  sour    1.3  5.9  TRUE
> 
```

The read.csv function can include settings or parameters that may need to be set for the file to be read correctly (see ?read.csv for more info). Parameters are entered after the file name and separated by a comma. Some of the more useful parameters are shown below:



The file you want to import
How is each value in the *.csv file separated (normally with ",")?
Does the first row of your table contain the header row?
Does your data use a "." or a "," for decimals?
i.e. 12.1 vs 12,1
Console
Terminal

```
C:/Users/jc161744/Desktop/R Tutorial/
> read.csv("example.csv",sep = ",", header = TRUE, dec = ".")
   Colour Taste weight size eaten
1     red sweet    1.2  5.6 FALSE
2    blue  sour    1.4  6.2 FALSE
3   green  sour    1.1  5.3  TRUE
4  orange salty    1.3  5.9  TRUE
5  purple salty    1.6  6.8  TRUE
6  orange salty    1.3  5.9 FALSE
7  purple salty    1.6  6.8 FALSE
8    blue  sour    1.4  6.2  TRUE
9   green  sour    1.1  5.3  TRUE
10    red sweet    1.2  5.6 FALSE
11  green  sour    1.1  5.3 FALSE
12 orange salty    1.7  7.1 FALSE
13 purple salty    1.6  6.8 FALSE
14 orange salty    1.6  6.8  TRUE
15 purple salty    1.6  6.8 FALSE
16   blue  sour    1.3  5.9  TRUE
> 
```

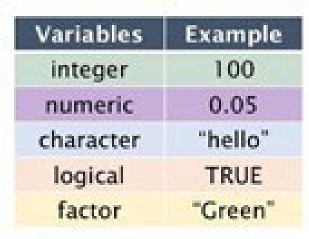
Note: Other read functions exist such as readxl to directly read xls files, but functionality and options vary. Help documentation can be found within R using the appropriate help command (eg. ?readxl command for readxl).

4. Basic Tidying of your Data

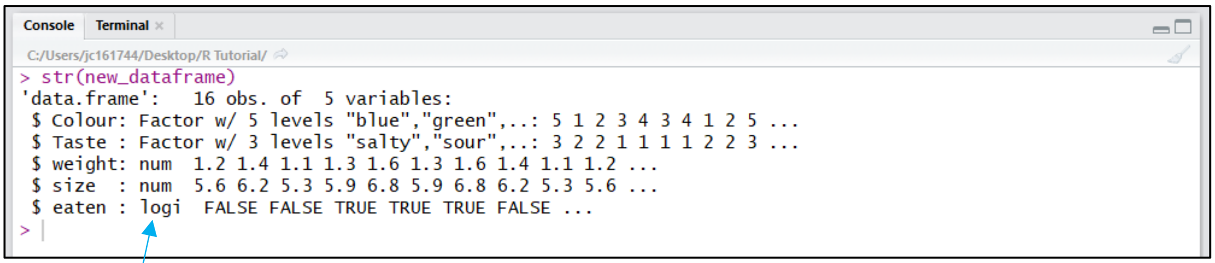
Clean Up Your Data

As with all statistical software, your data must be in the correct format for visualisation and analysis to function correctly.

To check the type of each variable in a dataframe, the class() function could be used separately. A quicker way to check the variables in a dataframe is the structure or str() function, which displays all variables at once.



Variables	Example
integer	100
numeric	0.05
character	"hello"
logical	TRUE
factor	"Green"



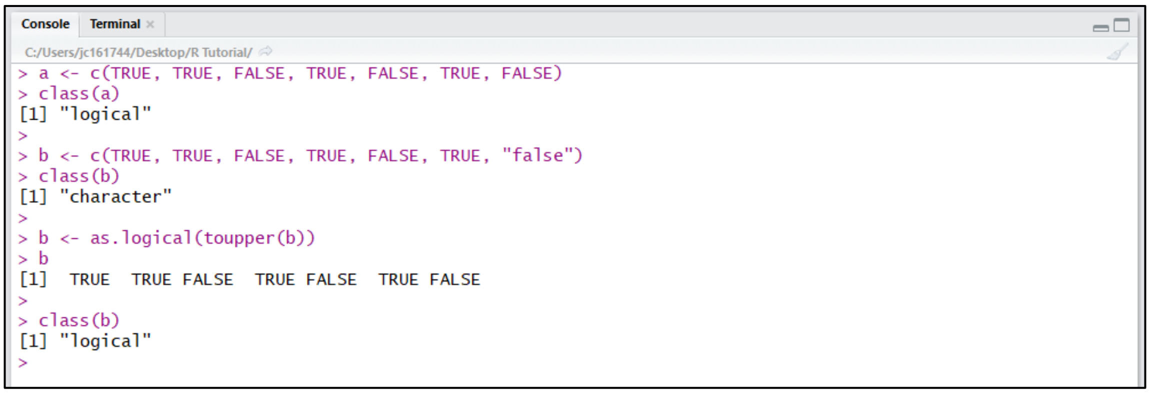
```
> str(new_dataframe)
'data.frame':	16 obs. of  5 variables:
 $ Colour: Factor w/ 5 levels "blue","green",..: 5 1 2 3 4 3 4 1 2 5 ...
 $ Taste : Factor w/ 3 levels "salty","sour",..: 3 2 2 1 1 1 1 2 2 3 ...
 $ weight: num  1.2 1.4 1.1 1.3 1.6 1.3 1.6 1.4 1.1 1.2 ...
 $ size  : num  5.6 6.2 5.3 5.9 6.8 5.9 6.8 6.2 5.3 5.6 ...
 $ eaten : logi  FALSE FALSE TRUE TRUE TRUE FALSE ...
> 
```

The first and second variables are categories or "factors"
The third variable is a number or "numeric"
The fourth is True/False or "logical"

While numeric variables generally cause few problems, logical and factor variables often do have issues due to R interpreting the variable type when importing from an Excel file.

- **Problem:** One of the logical values is in lower case (eg. TRUE, TRUE, false, FALSE), so all values are interpreted as a character type variable ("TRUE", "TRUE", "false", "FALSE"). R interprets logical variables only if all values are in CAPITAL letters. This prevents any analysis that relies on a comparison between the TRUE and FALSE values.

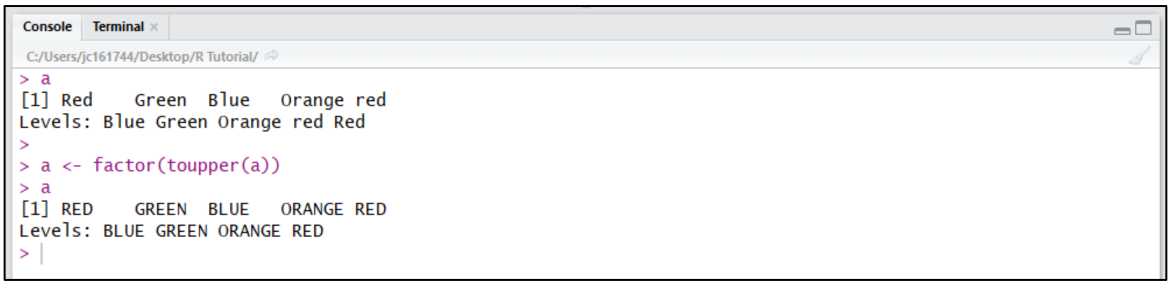
Solution: The problem is most easily fixed in Excel before the data is imported. Converting the variables in R can be achieved using the toupper( ) function.



```
> a <- c(TRUE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE)
> class(a)
[1] "logical"
>
> b <- c(TRUE, TRUE, FALSE, TRUE, FALSE, TRUE, "false")
> class(b)
[1] "character"
>
> b <- as.logical(toupper(b))
> b
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE
>
> class(b)
[1] "logical"
>
```

- **Problem:** R automatically converted categories into groups or "levels" but there were mixed upper and lower case letters (e.g "Green", "GREEN" and "green" will all become different levels).

  Solution: These are easily converted in Excel before importing to R-Studio, by converting all letters to either upper or lower case. Converting values between upper and lower case in R is achieved using the toupper( ) or tolower( ) functions which converts all values to the same case. This command also converts factors to character type variables. The factor( ) function is then used to convert the variable back into a factor, and determining new levels.

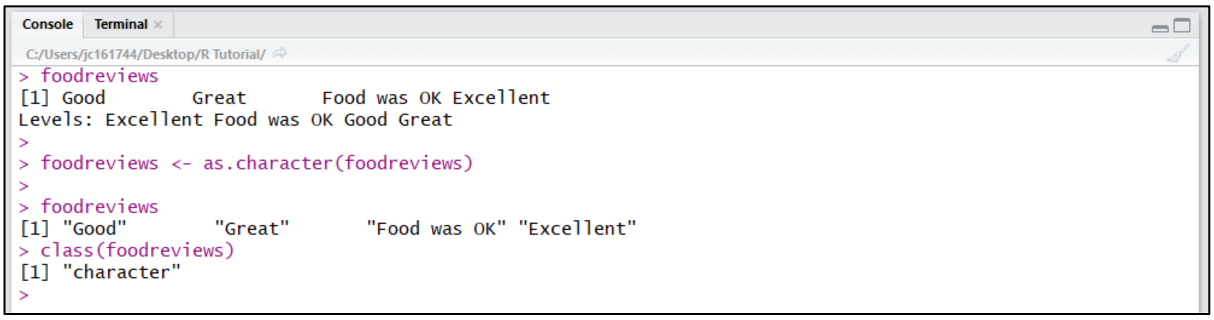


```
> a
[1] Red    Green  Blue   Orange red
Levels: Blue Green Orange red Red
>
> a <- factor(toupper(a))
> a
[1] RED    GREEN  BLUE   ORANGE RED
Levels: BLUE GREEN ORANGE RED
> |
```

Note: There are methods of manually declaring levels in R and this information can be found in the help documentation (ie. ?factor, ?gl, ?levels). If there is an order to the levels (high, medium and low these can also be declared – see help documentation)

- **Problem:** R has interpreted strings of text as factors

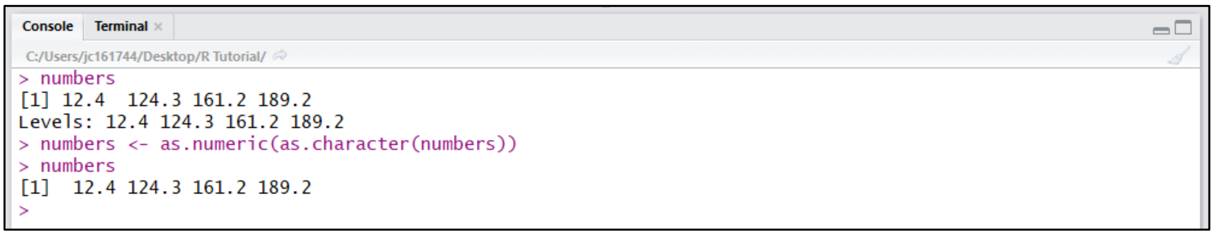
  Solution: To convert a factor into text or “character” type variable the as.character() function is used.



```
> foodreviews
[1] Good         Great        Food was OK Excellent
Levels: Excellent Food was OK Good Great
>
> foodreviews <- as.character(foodreviews)
>
> foodreviews
[1] "Good"          "Great"        "Food was OK" "Excellent"
> class(foodreviews)
[1] "character"
>
```

- **Problem:** R has interpreted numeric data as factors

  Solution: To convert a factor into a number is a 2 step process. The factor must be converted to a character first using as.character(), then converted to a number using as.numeric().



```
> numbers
[1] 12.4  124.3 161.2 189.2
Levels: 12.4 124.3 161.2 189.2
> numbers <- as.numeric(as.character(numbers))
> numbers
[1]  12.4 124.3 161.2 189.2
>
```

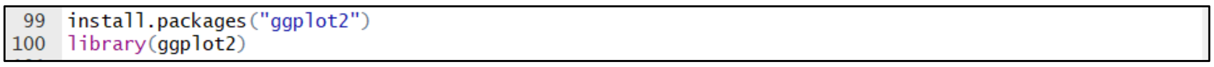
5. Graphing in ggplot2

Graphing with ggplot2

ggplot2 Cheatsheet: https://raw.githubusercontent.com/rstudio/cheatsheets/main/gganimate.pdf

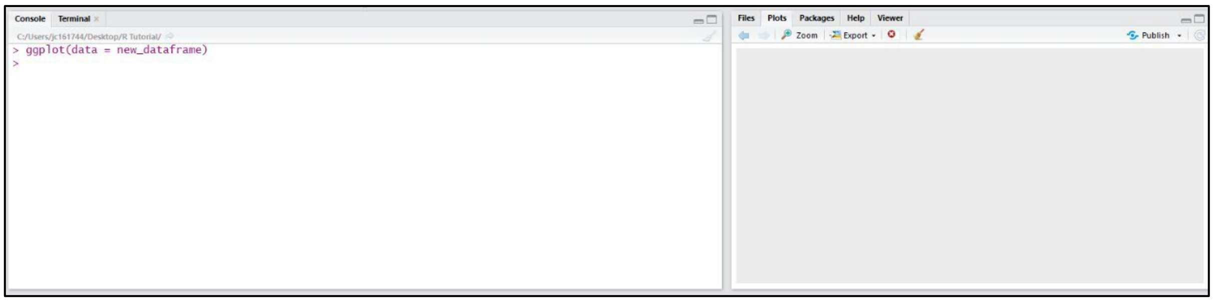
While R comes with many useful functions built in, other plugins or “packages” can be downloaded and added to bring extra functionality. One of the more useful packages is ggplot2 (based on The Grammar of Graphics by Leland Wilkinson), which introduces a variety of powerful visualisation and graphing options.

The ggplot2 package must first be downloaded and installed into R Studio using the command install.packages(“ggplot2”). This command will only need to be run once, as the package will then be available to load in the future. Once installed, the package must be loaded into your current R session using library(ggplot2). The package will need to be loaded for every new R session/project.



```
install.packages("ggplot2")
library(ggplot2)
```

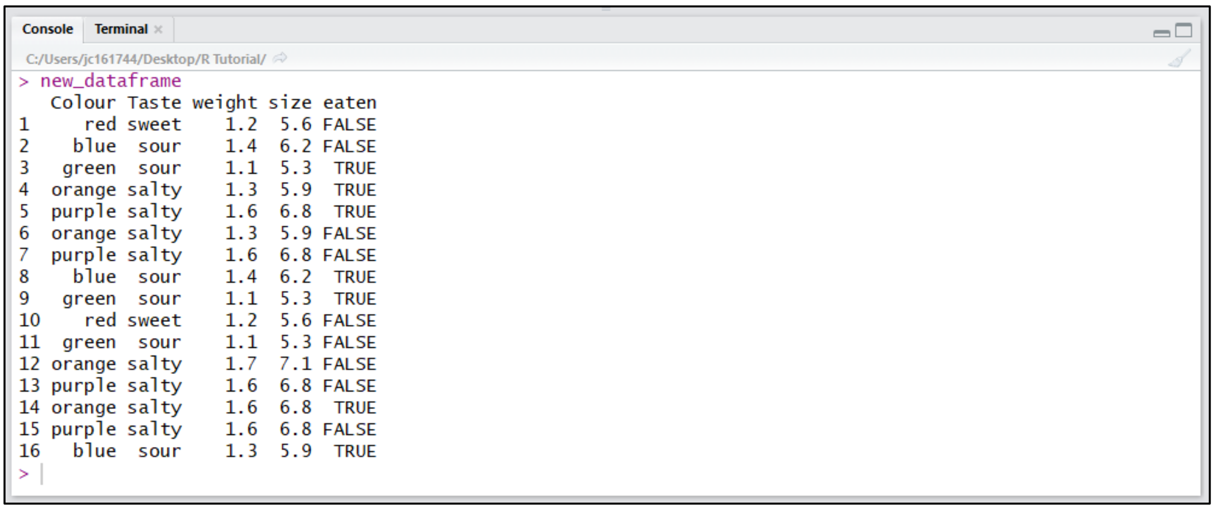
The package ggplot2 simplifies data visualisation. You provide the data, how the variables should be interpreted, and graphical options if required, while ggplot2 handles the details. To load the data that you would like to visualise, the ggplot(data = *yourdata*) function is used.



```
> ggplot(data = new_dataframe)
>
```

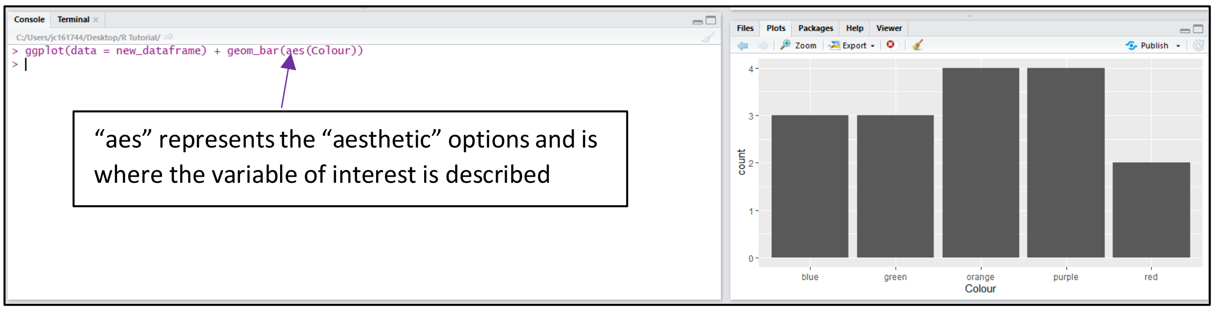
The next step is choosing a visualisation type or "geom". This option comes after the ggplot function and they are separated by a +. There are many different types of graph and the choice of which to use depends on your data. The most common types are geom_histogram for a histogram, geom_bar for a bar chart, geom_boxplot for a box and whisker plot, geom_point for a scatterplot, and geom_smooth for a line of best fit or fitted curve. Many of these different graphs can be overlayed with other visualisations, with each requiring you to define the variable/s to display. Many also require extra parameters, all of which can be found in help documentation.

The following visualisation examples will use the following dataframe:



```
> new_dataframe
   Colour Taste weight size eaten
1     red sweet    1.2  5.6 FALSE
2    blue  sour    1.4  6.2 FALSE
3   green  sour    1.1  5.3  TRUE
4  orange salty    1.3  5.9  TRUE
5  purple salty    1.6  6.8  TRUE
6  orange salty    1.3  5.9 FALSE
7  purple salty    1.6  6.8 FALSE
8    blue  sour    1.4  6.2  TRUE
9   green  sour    1.1  5.3  TRUE
10    red sweet    1.2  5.6 FALSE
11  green  sour    1.1  5.3 FALSE
12 orange salty    1.7  7.1 FALSE
13 purple salty    1.6  6.8 FALSE
14 orange salty    1.6  6.8  TRUE
15 purple salty    1.6  6.8 FALSE
16   blue  sour    1.3  5.9  TRUE
>
```

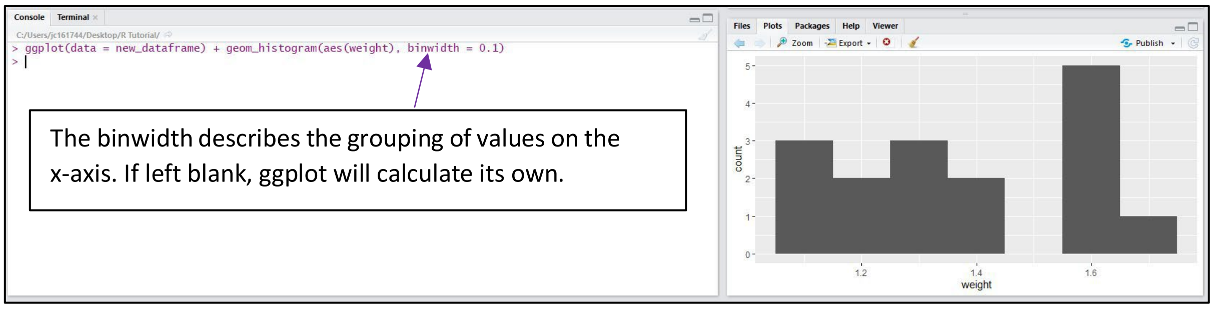
A bar chart representing the frequency for each "colour":



```
> ggplot(data = new_dataframe) + geom_bar(aes(Colour))
>
```

"aes" represents the "aesthetic" options and is where the variable of interest is described

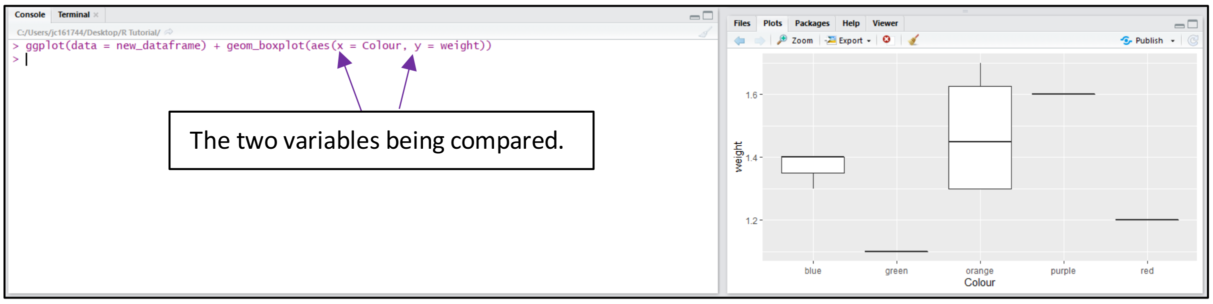
A histogram representing the range of "weight":



```
C:/Users/jc161744/Desktop/R Tutorial/
> ggplot(data = new_dataframe) + geom_histogram(aes(weight), binwidth = 0.1)
> |
```

The binwidth describes the grouping of values on the x-axis. If left blank, ggplot will calculate its own.

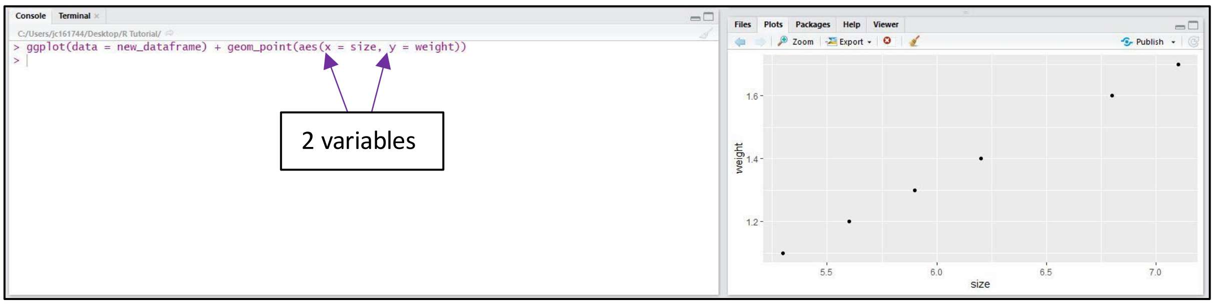
A box and whisker plot representing the “weight” for each “colour”:



```
C:/Users/jc161744/Desktop/R Tutorial/
> ggplot(data = new_dataframe) + geom_boxplot(aes(x = Colour, y = weight))
> |
```

The two variables being compared.

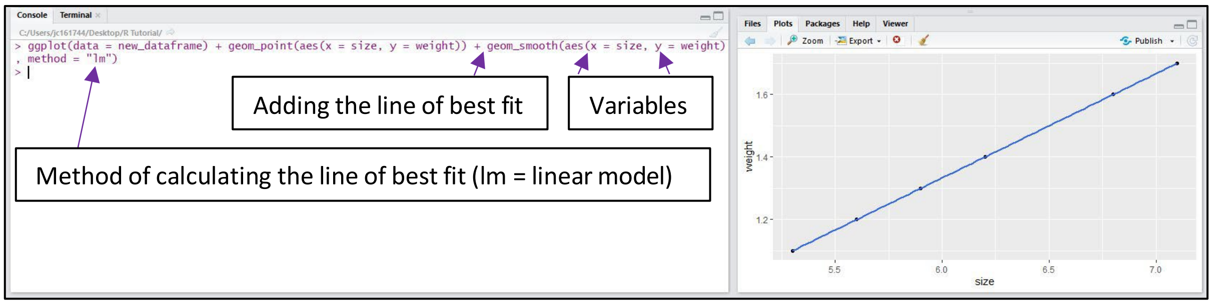
A scatterplot representing “size” compared to “weight”:



```
C:/Users/jc161744/Desktop/R Tutorial/
> ggplot(data = new_dataframe) + geom_point(aes(x = size, y = weight))
> |
```

2 variables

The above scatterplot but with a fitted line using a linear model:



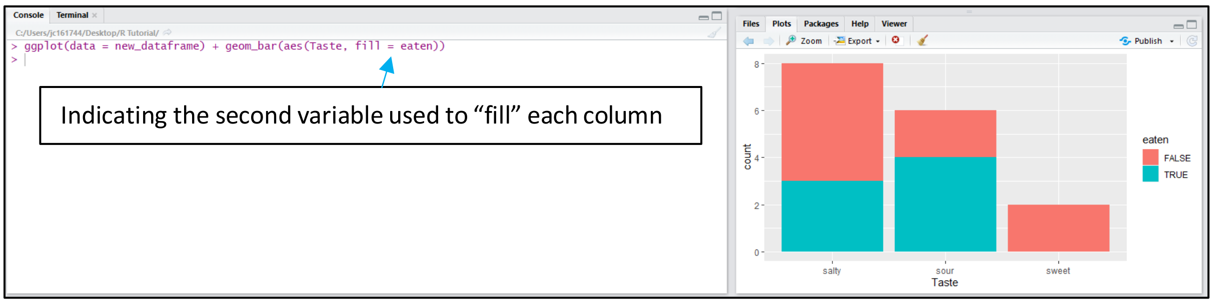
```
C:/Users/jc161744/Desktop/R Tutorial/
> ggplot(data = new_dataframe) + geom_point(aes(x = size, y = weight)) + geom_smooth(aes(x = size, y = weight)
, method = "lm")
> |
```

Adding the line of best fit

Variables

Method of calculating the line of best fit (lm = linear model)

A bar chart representing the frequency of “taste” and the ratio of “eaten” for each:



```
C:/Users/jc161744/Desktop/R Tutorial/
> ggplot(data = new_dataframe) + geom_bar(aes(Taste, fill = eaten))
> |
```

Indicating the second variable used to “fill” each column

Basic R Studio Tutorials

- https://data-flair.training/blogs/rstudio-tutorial/

More ggplot2 graphing ideas can be found at:

- https://ggplot2.tidyverse.org/
- https://www.r-graph-gallery.com/
- http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html
- https://datacarpentry.org/R-ecology-lesson/04-visualization-ggplot2.html

Official R Cheat Sheets can be found at:

- https://rstudio.com/resources/cheatsheets/

TheLearningCentre
UNLOCK YOUR POTENTIAL

www.jcu.edu.au/students/learning-centre