



## Introduction to R programming and using Rstudio.

Anders Jensen

### Contents

Introduction .....	2
Key advantages of using R programming.....	2
Installing R and RStudio.....	3
On campus.....	3
Off campus.....	4
Getting to know RStudio .....	5
Starting your first R script.....	6
Writing your first line of code .....	7
Saving information into the environment.....	8
Reading data into RStudio.....	9
Setting the working directory .....	10
Reading in a .csv file.....	11
Reading in an .xlsx file.....	11
Checking data.....	12
Working with your data.....	13
Subset data based on a condition.....	13
Selecting specific columns .....	13
Creating a new column.....	14
Changing the type of a specific column.....	14
Making a vector from a column of interest .....	15
Descriptive Statistics.....	15
Introduction to statistical testing.....	16
Normality testing .....	17
Performing statistical tests in R .....	17
Homework .....	20
Common problems and solutions.....	20
Conclusions.....	20
Worked solutions.....	21

## Introduction

For all different fields across the university, one of the most common struggles we will all face is working with and analysing data. There are many different types of software available for people to use, such as SPSS, Microsoft Excel, Origin, MATLAB and more. These tools will often provide users with a friendly interface designed with the sole purpose of simplifying tasks for users. This is great for beginners; however, the simplicity also leads to limitations.

R programming is a coding language specifically designed for data analysis and statistical computing. Along with RStudio, which is a user interface to make R function more effectively, R is capable of performing every task in the aforementioned software and much more.

You may think that learning an entire coding language is a bit overkill; however, the advantages you will gain in terms of both employability and efficiency cannot be understated.

The aim of this document is to give you a brief introduction to R programming so that the basics are covered. One document is not enough to go through every aspect of RStudio; however, the [KnowHow Statistics & Coding page](#) have many resources/videos on topics that are more specific.

## Key advantages of using R programming

- **Versatility:** Easy to work with various types of data from simple descriptive statistics to complex machine learning algorithms.
- **Efficiency:** Repetitive tasks can be streamlined to increase your productivity.
- **Collaboration:** You can implement code from other people into your own work and then share your code with others for reproducible analysis pipelines.
- **Customisation:** R can allow users to customise aspects of the analysis and visualisation, which may be more difficult with other software.

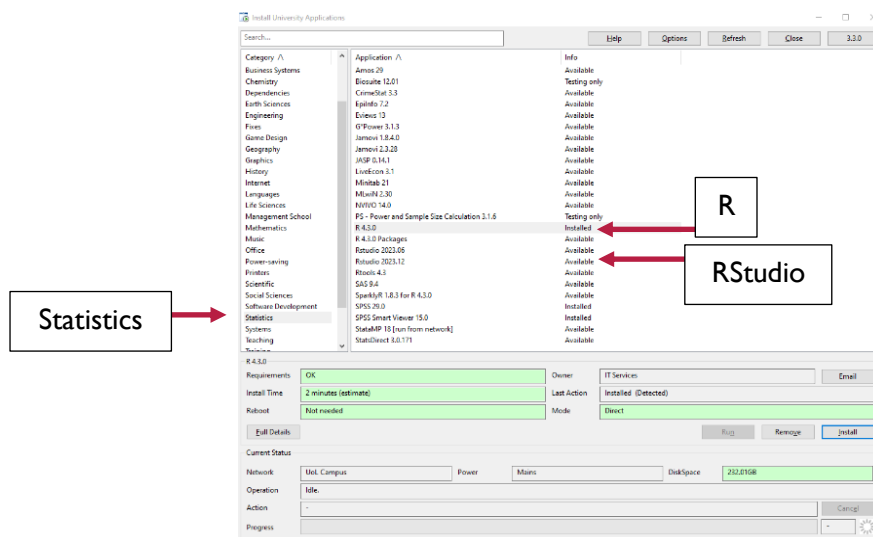
## Installing R and RStudio

The first step into using R is making sure you can access both R and RStudio. R is an open-source programming language so everything is freely available to download online or if you are working on campus you can use the Install University Applications app on the desktop. On the other hand RStudio is simply an interface, designed to assist with R programming.

### On campus



- 1) Click on the Install University Applications icon pictured here →
- 2) Then look for statistics in the category section and you should see both R and rstudio



- 3) Install R first and once that has been installed then install RStudio. It is also advisable to install RTools to ensure all the Libraries in this handbook work.
- 4) If you then search for RStudio in the bottom left of your desktop and click on the icon below to open it



## Off campus

If you want to use R and RStudio off-campus on your own computer the installation is slightly more complicated but still easy to follow. If you are having difficulties then double check for installation tutorials specific to the computer you are using.

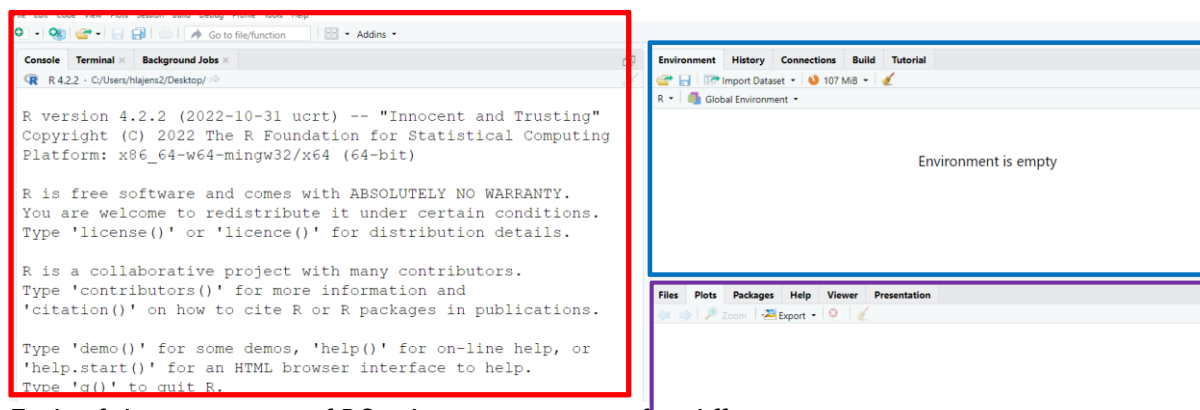
- 1) Visit (<https://cran.r-project.org/bin/windows/base/>) and click Download R-4.3.2 for Windows. The version number may be slightly different and it should be at the top left of the page.
- 2) This should then download R and then you need to install R on your computer by clicking on the file that was downloaded.
- 3) Follow the instructions from the pop up.
- 4) Visit (<https://posit.co/download/rstudio-desktop/>) and click Download RStudio Desktop for Windows (It should be written in a blue box)
- 5) Then click on the downloaded file to install RStudio
- 6) Follow the instructions from the pop up
- 7) You should now be able to open RStudio on your laptop by clicking the icon below



As mentioned above if you are using a computer that isn't a windows the download may vary slightly. Fortunately, there will be lots of help available online about downloading R on your device.

## Getting to know RStudio

When you first open RStudio on your computer you should see a single page split into three regions: the console (left), the environment (top right) and then the plots window (bottom right).



Each of these sections of RStudio are important for different reasons.

**Console:** This is where your code is run, you can write your code directly in here; however, as you will see later it is easier to make a new script so that your work can be saved for a later date. This is also where any error messages will be shown.

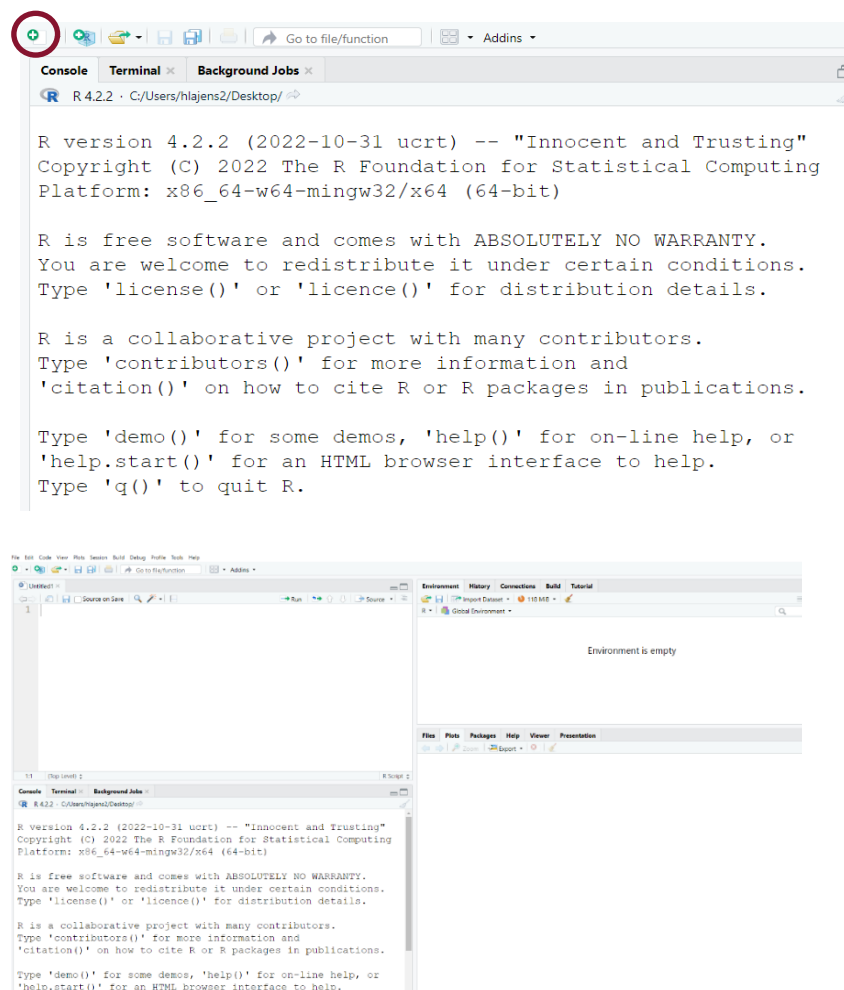
**Environment:** As you can see this region has multiple windows however the environment is the most important. This is where you can store different objects (data frames, variables, functions, etc.).

**Plots window:** Similarly to the environment there are other tabs in this window, however the plots window is the most important. This is where you can visualise any graphs or charts you make using RStudio.

## Starting your first R script

As mentioned above, it is not recommended to write code directly into the console as it is more difficult to save/annotate your work as you are going along. Instead it is better to load up a blank R script and write the code in there.

To do this, simply click on the small page icon on the top left of your screen (circled below). Then click “R script”. The shortcut for this is Ctrl+Shift+N.



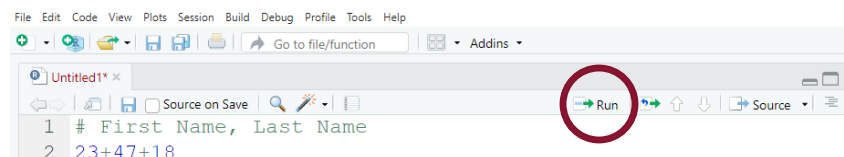
Your screen should now look like the image above. As you can see you now have an R script, which you can write in and then save as an .R file that can be opened in RStudio anytime you want to re-visit your code.

## Writing your first line of code

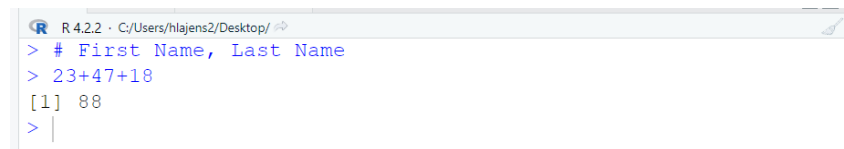
Now that you have an empty R script you can begin writing in some code. A useful tip before you start is to remember the importance of annotating your code. By using the # symbol you can write notes as you are going along, which makes your life much easier when you come back to your code in the future.

Use the # symbol in RStudio and write your name at the top of the script to begin with. After this we can start by doing some simple calculations using RStudio.

Write **23+47+18** into R and then press Ctrl+Enter, you can also click the button that says **Run** highlighted below. Make sure your cursor is on the correct line of code that you want to run.



If you have run it correctly then your console should look like this



**Try to calculate the following using R:**

8 x 19 (Hint: \*) =

96 % 6 (Hint: /) =

9<sup>3</sup> (Hint: ^) =

Square root of 100 (Hint: sqrt(100)) =

## Saving information into the environment

Now let's say we want to save this information into our environment in R we can do this very easily. To assign something to the environment in R you use the following syntax:

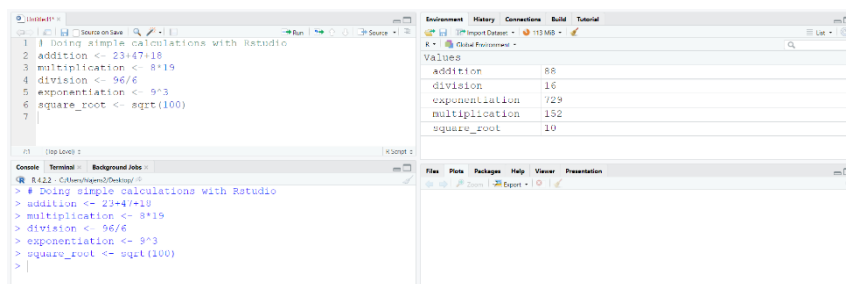
**Name** <- 8\*19

The **Name** can be whatever you want as long as it starts with a letter and has no spaces in it

The <- symbol can be written quickly with Alt+- and assigns something to the environment.

The **output** of anything to the right of the arrow is saved in the environment under the name you gave it

So using the previous example questions it should look like the image below. If you need to include a space in the name for some reason simply use a \_.



As you can see all the answers are now stored in my environment.

**Try to use this information to find a way to work out the sum of all the answers:**  
(Hint: name1+name2+name3....)



## Reading data into RStudio

Hopefully at this stage you are more comfortable with the RStudio interface and how to run lines of code and how to then save information into your working environment. The next and one of the most important steps is to be able to read in different types of data into RStudio.

For this workshop you will learn how to read in the two most common types of files, which are used to store data.

**CSV** – Comma separated values

**XLSX** – Excel spreadsheet (XML)

When you save a file containing data you should make sure about the type of file it is saved as. This is because the process of reading it into RStudio is slightly different.

For this workshop we will be working with a dataset called [mtcars.csv](#), the data are available to download via this workbook.

The mtcars data set contains 32 rows and 11 columns and contains data on cars from 1974. The variables include; miles per gallon, number of cylinders, displacement, horsepower, rear axle ratio, weight, 1/4 mile time, engine type, transmission, number of forward gears and number of carburettors.

If you would like to know more about each column then run the following code in RStudio:

```
?mtcars
```

## Setting the working directory

In order to make sure that RStudio can find your saved file you first need to make sure it is looking in the correct place. The best way to think of a directory is like an address:

Continent → Country → City → Street name → House number

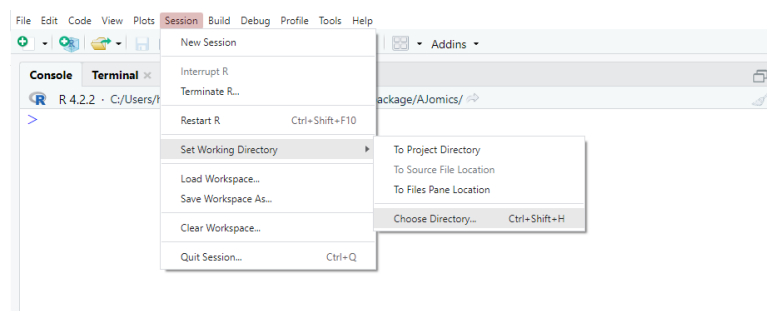
C drive → Users → hlajens2 → Desktop → Introduction to R

It is basically saying to RStudio that you need to look in this folder to find the correct document. In the same way as you wouldn't tell a delivery driver to deliver something to America, you would need to be more specific.

To ensure there are no problems, make a new folder called "Introduction to R" on your desktop and save "[mtcars.csv](#)" and "[mtcars.xlsx](#)" into that folder.

The easiest way to set your working directory is to click **Session** then click **Set Working Directory** then click **Choose directory** (Or Ctrl+Shift+H)

Then navigate to desktop and double click on the folder you made in the previous step.



Once you have found the folder then click **open** on the bottom right of the pop up.

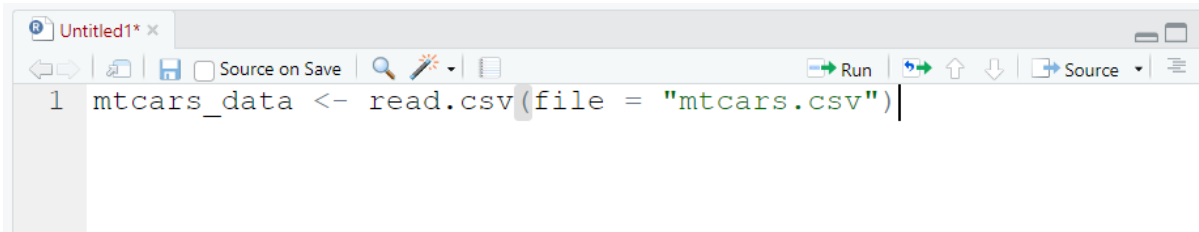
If successful you should get a message similar to this in your console:

```
> setwd("C:/Users/hlajens2/Desktop/Introduction to R")
```

## Reading in a .csv file

Csv files are commonly used when transferring data and are often the required format for many different kinds of software. Large datasets can be stored compactly, allowing a more efficient transfer of data.

To read a csv file into RStudio we are going to use the **read.csv** function, which does exactly what it says on the tin.

A screenshot of the RStudio code editor window. The title bar says 'Untitled1\*'. The toolbar includes icons for back, forward, save, source on save, search, and a palette. The code area contains a single line: `1 mtcars_data <- read.csv(file = "mtcars.csv")`. The cursor is at the end of the line.

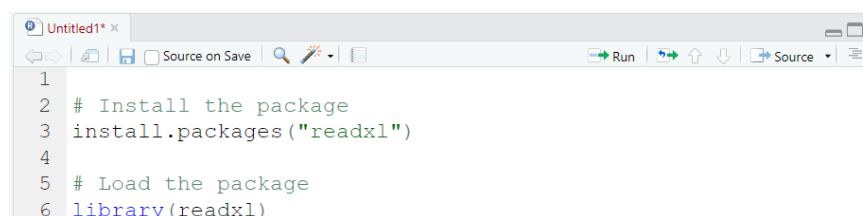
By running this line of code you should see your data in the environment section named `mtcars_data`. Remember that it doesn't matter what you call it, the text to the left of the `<-` is purely there to assign a name to the object. Try to make your names as descriptive as possible as this will be very useful when working on large projects.

## Reading in an .xlsx file

To read data into RStudio in the xlsx format you first need to install a package that is capable of opening these types of files. Packages are just a collection of custom functions that can be used for a variety of tasks.

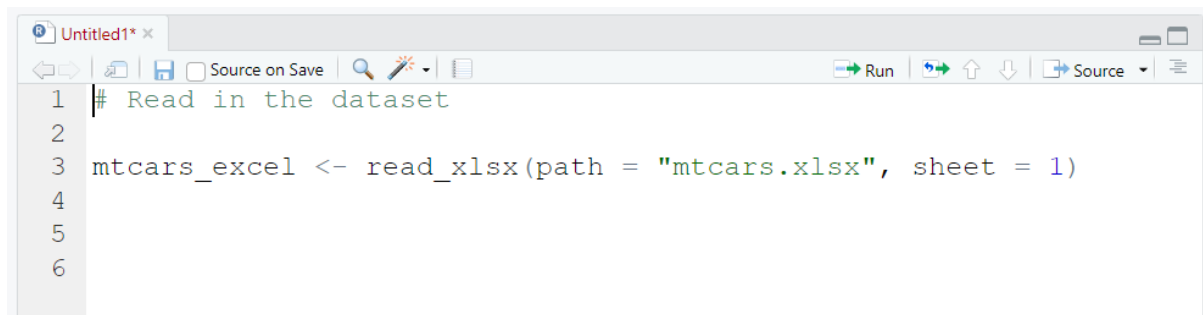
There are many available packages for reading in .xlsx files but the one we will be using is called "readxl"

To install a package we need to run the following code:

A screenshot of the RStudio code editor window. The title bar says 'Untitled1\*'. The toolbar includes icons for back, forward, save, source on save, search, and a palette. The code area contains six lines: `1`, `2 # Install the package`, `3 install.packages("readxl")`, `4`, `5 # Load the package`, and `6 library(readxl)`.

As you can see you need to install the package and then load it into RStudio using the **library** function. You do not need to install packages every single time, however if you close down RStudio and re-open it you will need to load the package again.

To then read the file into RStudio you need to use **read\_xlsx** function. As you can see below I have specified the file name (**path**) and the sheet number (**sheet**).

A screenshot of the RStudio code editor window. The title bar shows 'Untitled1\*'. The toolbar includes icons for running code, saving, and other standard RStudio functions. The code editor contains the following R code:

```
1 # Read in the dataset
2
3 mtcars_excel <- read_xlsx(path = "mtcars.xlsx", sheet = 1)
4
5
6
```

Both files are exactly the same and the data should look the same, the point of this was just to show that different functions are needed to effectively perform the same task and that different types of files can be read into R but in different ways.

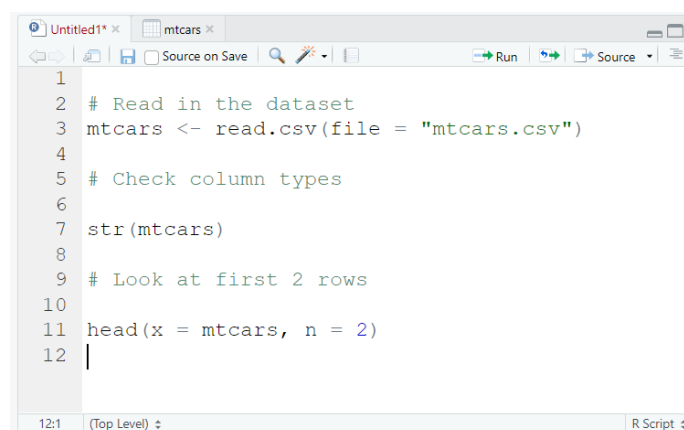
## Checking data

Before going on any further there are some very useful functions that can be used to summarise a data frame in RStudio.

The **str** function allows you to look at the types of data that exists in your columns (Numeric, Factor, Character, Integer, Logical, etc.)

The **head** function allows you to see a certain number of rows of the dataframe to see how they look and also to show the column names.

Try the following code and have a look at the output.

A screenshot of the RStudio code editor window. The title bar shows 'Untitled1\*' and 'mtcars'. The toolbar includes icons for running code, saving, and other standard RStudio functions. The code editor contains the following R code:

```
1
2 # Read in the dataset
3 mtcars <- read.csv(file = "mtcars.csv")
4
5 # Check column types
6
7 str(mtcars)
8
9 # Look at first 2 rows
10
11 head(x = mtcars, n = 2)
12 |
```

You should see that some columns are chr (Characters, written text), num (Numeric, continuous scale), int (Integer, whole numbers).

You should also see all the names of the columns when using the **head** function.

## Working with your data

Now that you have a dataset in RStudio and are happy with the columns we can begin a variety of different tasks.

There are too many things to all be covered in this document so we will work through five specific common examples.

### Subset data based on a condition

To do this we need to use the **subset** function.

Let's say we only want rows where the mpg (miles per gallon) is above 15.0 mpg.

```
1
2 # Keep only cars with mpg above 15
3 mtcars_highmpg <- subset(mtcars, mpg >= 15)
```

As you can see the `>=` selects everything larger than or equal to 15.0 mpg

You can use `==` if you want to select for an exact number

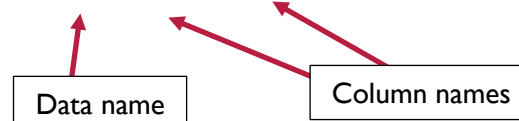
If you have multiple conditions

### Selecting specific columns

To do this we will use the **select** function from the dplyr package. Dplyr is an extremely useful package and is definitely worth spending some more time to learn.

Let's say we want a new dataframe that only contains mpg and qsec (quarter mile time)

```
1 # Install dplyr
2 install.packages("dplyr")
3
4 # load dplyr
5 library(dplyr)
6
7 # New data frame with mpg and qsec
8 mtcars_mpgqsec2 <- select(mtcars, mpg, qsec)
9
```



If you wanted to make a dataframe excluding mpg you would simply need to put a `!` before mpg

**select(mtcars, !mpg)**

## Creating a new column

To do this we will use the \$ operator, which is very important when working with data.

Let's say we want to make a column that is the horsepower per 1000 lbs (weight, wt). to do this we need to do horsepower/wt

```
13 # Making a new column in the data
14 mtcars$hpwt <- mtcars$hp/mtcars$wt
15
```



New column name

As you can see the \$ operator is used to look inside a dataframe and specify a column

## Changing the type of a specific column

In our mtcars example the am column is in binary where Transmission (0 = automatic, 1 = manual). This is fine but another way to show it is by using TRUE/FALSE statement.

Let's say we want to make the am column a TRUE if its manual and FALSE if its automatic.

```
16 # Making am column logical
17 mtcars$vs <- as.logical(mtcars$vs)
```

If you wanted the TRUE/FALSE statement to be the other way around so that automatic was true, simply use the ! operator before the mtcars inside the brackets.

as.logical, as.character, as.numeric can also all be used.

## Making a vector from a column of interest

Once again this involves using the \$ operator and allows us to save information from a certain column into our environment

Let's say we want to save all the names of the cars

```
19 # Save vector of carnames
20 carnames <- mtcars$name
```

This will then allow us use this vector in any further code.

These examples only provide a snippet of the tasks you may need to ever complete. The best way to solve these problems is knowing where to look. Datanovia, stackoverflow, Statology are all useful websites with helpful tutorials and quick fixes.

## Descriptive Statistics

The next thing to do is to learn how to use RStudio to perform very simple descriptive statistics.

The following functions are extremely useful for this:

**mean** – Calculate mean

**median** – Calculate median

**mode** – Calculate mode

**range** – Calculates the range

**sd** – Calculates the standard deviation

**sum** – Calculates the sum

**max** – Calculates the highest value

**min** – Calculates the lowest value

**Try to use this and what you have learnt so far to calculate descriptive statistics on the miles per gallon (mpg) of cars in the mtcars dataset**

**Do automatic cars have a higher average mpg than manual cars? (Hint: Subset)**

## Introduction to statistical testing

Statistical testing is the idea of using mathematical techniques to make inferences about the characteristics of a population with your sample data. We are assessing the likelihood of an observed difference being down to chance. We then use statistical testing to generate a p-value. The p-value can be used as a threshold for when we would consider something significant. A commonly used threshold of significance is 0.05 meaning there is a 5% chance the difference is due to chance.

There are too many statistical tests to cover in this document so we will go through three of the most common:

- 1) **Shapiro-Wilks normality testing**
- 2) **Two Sample t-test**
- 3) **Pearson's correlation test**

For this task we are attempting to answer two main questions:

- 1) **Do straight engines have a higher miles per gallon than v-shaped engines?**
- 2) **Is there a correlation between weight and miles per gallon?**

### **Q1) Do straight engines have a higher miles per gallon than v-shaped engines?**

To begin with we are comparing a difference with one continuous scale across two different categorical groups. This would suggest some variant of a t-test.

To understand which t-test we are doing we need to look at the assumptions of a t-test.

Are the data independent? Yes they are different cars

Is the data normally distributed? Not sure so let's find out....



## Normality testing

The shapiro-wilk test is a quantitative test to determine if data follows a normal distribution or not (i.e follows a bell-shaped curve on a histogram). Using our threshold of 0.05 we would say that a p-value larger than 0.05 is normally distributed and anything lower is non-normally distributed.

To do this on RStudio we first need to subset mtcars into v-shaped and straight engines.

```
# v shaped and straight engines subset
v_shaped <- subset(mtcars, vs == 0)
straight <- subset(mtcars, vs == 1)
```

Now we can check normality using the **shapiro.test** function to test for normality.

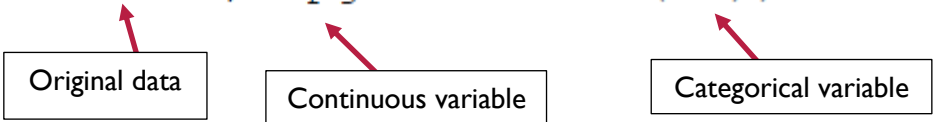
```
# Test mpg normality for both subsets
shapiro.test(v_shaped$mpg)
shapiro.test(straight$mpg)
```

## Performing statistical tests in R

If you look in the console you will see the output and notice that for both tests the p-value is greater than 0.05 so we can say the data is normally distributed. It may seem the other way around to other statistical tests; however the null hypothesis is that the data is normally distributed.

Now that we know the data is normally distributed, we can proceed with a parametric t-test. To do this we need to use the **t.test** function.

```
# Perform the t-test
t.test(data = mtcars, mpg~as.factor(vs))
```



Original data

Continuous variable

Categorical variable

Once again if you look in your console you will have a result from the t-test. You should be able to see if the difference is significant or not. Remember  $p < 0.05$  = significant result

**Use descriptive statistics and the t-test to answer if straight engines have a higher mpg than v-shaped engines.**

## **Q2) Is there a correlation between weight and miles per gallon?**

For this question we want to look for an association between two continuous scales. Therefore we need to do some type of correlation analysis.

First of all we need to check the distribution to see if we need a Pearson (parametric) or Spearman's rank (non-parametric)

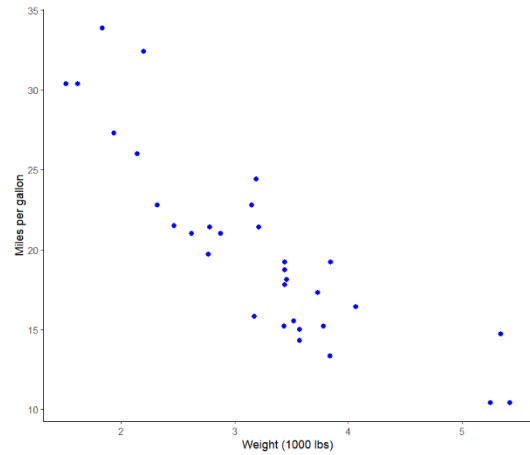
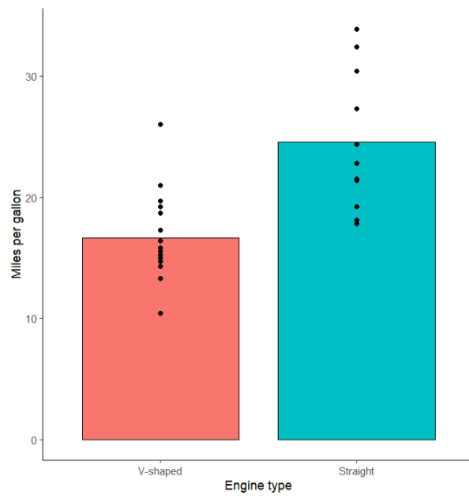
```
# Check normality of weight and mpg
shapiro.test(mtcars$wt)
shapiro.test(mtcars$mpg)
```

Depending on these results we then need to perform a correlation analysis

```
# Do correlation test
cor.test(mtcars$mpg, mtcars$wt, method = "pearson")
cor.test(mtcars$mpg, mtcars$wt, method = "spearman")
```

**Now you can interpret these results to determine if there is a correlation between the weight of cars and the miles per gallon.**

**We can then plot these results using the ggplot2 package. This will be covered in a separate document, but the graphs and code are available below.**



```
# Install ggplot
install.packages("ggplot2")

# Load ggplot
library(ggplot2)

# Make barplot
ggplot(mtcars, aes(x = as.factor(vs), y = mpg)) +
  geom_bar(aes(fill = as.factor(vs)), col = "black",
           stat = "summary",
           fun = mean,
           width = 0.8) +
  theme_classic() +
  labs(fill = "Engine shape", x = "Engine type",
       y = "Miles per gallon") +
  geom_jitter(position = position_dodge(width = 1)) +
  scale_x_discrete(labels = c("V-shaped", "Straight")) +
  scale_fill_discrete(breaks = c("0", "1"),
                     labels = c("V-shaped", "Straight"))

# Make scatter plot
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(size = 2, col = "blue") +
  theme_classic() +
  labs(x = "Weight (1000 lbs)", y = "Miles per gallon")
```

## Homework

If you have completed everything in this document and are comfortable with all the concepts then why not try analysing one of the built in datasets in RStudio.

To access these datasets simply run `data()` and a list of all the available datasets you can use.

Then run `data("name")` with the name of the dataset and it should appear in your environment. There are some worked examples available at the end of the document to guide you.

### **Have a go with any dataset and try to perform the following:**

- 1) Subset based on a condition
- 2) Descriptive statistics
- 3) A statistical test of any kind

## Common problems and solutions

As you progress with R programming your problems will become more and more complicated, however there is no need to fear because chances are there is someone who has had the same problem and there will always be help available online.

The major reason for errors when beginning R programming is spelling mistakes. Often you may have a capital letter where there shouldn't be one or vice versa. The best way to overcome this is to avoid writing too many things into R, instead use the tab button and then enter so that it is done automatically.

Another important thing is to begin understanding error messages. If a specific object hasn't been found then this is probably a spelling mistake. The more error messages you get the more experience you will have solving the problems.

If you get a specific error message then the best advice would be to simply copy and paste that error message into google and hope that someone has had the same problem. Stack overflow is the perfect website for this and if you are really stuck you can even post your error message into that and wait for a response.

## Conclusions

Hopefully this document has provided you with a brief overview of using R programming for data analysis and you are now confident to have a go with your own data. The most important thing to remember when coding is that it is okay not to know how to do something, the real trick comes from knowing where to look for help.

In the future we are hoping to bring out a data visualisation guide, which will explain how to use the `ggplot2` package in more detail.

For any additional support, [LinkedIn Learning](#) has courses on R and you can email [knowhow@liverpool.ac.uk](mailto:knowhow@liverpool.ac.uk) for any queries about this handbook.

## Worked solutions

### Calculations using RStudio

```
Multi <- 8*19
```

```
Div <- 96/6
```

```
Expo <- 9^3
```

```
Root <- sqrt(100)
```

```
Final_ans <- Multi+Div+Expo+Root
```

```
Final_ans
```

### Descriptive statistics of mtcars

```
Automatic <- subset(mtcars, am == 0)
```

```
Manual <- subset(mtcars, am == 1)
```

```
mean(Automatic$mpg)
```

```
mean(Manual$mpg)
```

```
# A second way using the dplyr package
```

```
library(dplyr)
```

```
mtcars %>%
```

```
  group_by(am) %>%
```

```
  summarise(mean = mean(mpg))
```

### Do straight engines have a higher mpg than v-shaped engines?

```
# subset data
```

```
straight <- subset(mtcars, vs == 1)
```

```
v_shaped <- subset(mtcars, vs == 0)
```

```
# Normality testing
```

```
shapiro.test(straight$mpg)
```

```
shapiro.test(v_shaped$mpg)
```

```
# Both are normally distributed so we can use a t-test to compare
```

```
t.test(straight$mpg, v_shaped$mpg)
```

```
# p < 0.05, which meets our significance threshold so we can say there is a statistically
```

```
# significant difference between the miles per gallon of straight and v_shaped engines.
```

## **Is there a correlation between weight and mpg?**

```
# normality testing
```

```
shapiro.test(mtcars$mpg)
```

```
shapiro.test(mtcars$wt)
```

```
# Both are normally distributed so we use Pearson correlation
```

```
cor.test(mtcars$mpg, mtcars$wt, method = "pearson")
```

```
# p < 0.05 meaning there is a significant negative correlation between weight
```

```
# and miles per gallon.
```

## **Homework example 1**

```
# Check datasets
```

```
data()
```

```
# Find out information about dataset
```

```
?Orange
```

```
# Read in dataset of interest
```

```
data("Orange")
```

```
# Find out mean and sd of circumference of trees
```

```
mean(Orange$circumference)
```

```
sd(Orange$circumference)
```

```
# Perform a correlation analysis to see if circumference and age are correlated
```

```
cor.test(Orange$age, Orange$circumference, method = "spearman")
```

## **Homework example 2**

```
# Check datasets
```

```
data()
```

```
# Find out information about dataset
```

```
?sleep
```

```
# Read in dataset of interest
```

```
data("sleep")
```

```
# Find out mean sleep increase with both drugs
```

```
drug_1 <- subset(x = sleep, group == 1)
drug_2 <- subset(x = sleep, group == 2)
mean(drug_1$extra)
mean(drug_2$extra)
# Is there a statistical difference between drug 1 and drug 2
t.test(drug_1$extra, drug_2$extra, paired = T)
# Paired because the same patients are used for both drugs
```

This is slightly different to the previous t test example but both methods work just fine.