“Putting the R in ScHARR”

Robert Smith

01. October 2019

Contents

Background

This series of short courses are designed to equip the participant with a basic set of tools to undertake research using R. The aim is to create a strong foundation on which participants can build skills and knowledge specific to their research and consultancy objectives. The course makes use of the authors' experiences (many of which were frustrating) of working with R for data-science and statistical analysis. However there are many other resources available, and we would particularly recommend the freely available content at *R for Data Science* as a good place to recap the materials taught in this course. The hard copy of Hadley Wickham and Garrett Grolemund's book of the same name (and content) is available at *Amazon.com*. Alternatively, a user guide is available on the CRAN R-Project website here, although the author finds this less easy to follow than Hadley Wickham's book described above. Further details of where to go to answer more specific questions are provided throughout the course.

Requirements: It is assumed that all participants on the course have their own laptop, and have previously used software such as Excel or SPSS. Some basic understanding of statistics and mathematics is required (e.g. mean, median, minimum, maximum).

Who are we:

Robert Smith is a PhD Candidate at ScHARR, funded by the Wellcome Trust Doctoral Training Centre in Public Health Economics and Decision Science. His focus is on the methods used to estimate the costs and benefits of public health interventions, with a specific interest in microsimulation modelling (done in R). He has become increasingly intersted in the use of R-Markdown and R-Shiny to make research more transparent and to aid decision makers. While doing his PhD, Robert has been involved in projects with the WHO, Parkrun and Western Park Sports Medicine.

Paul Schneider joined ScHARR as a PhD candidate in 2018. He is working on conceptual and methodological problems in valuing health outcomes in economic evaluations. A medical doctor and epidemiologist by training, he has used R in various research projects, ranging from the modeling costs of breast cancer, and value of information analyses, to the monitoring of influenza in real-time using online data. He is a keen advocate of open science practices.

Sarah Bates ...

Thomas Bayley ...

Naomi Gibbs is a PhD Candidate at ScHARR, funded by the Wellcome Trust Doctoral Training Centre in Public Health Economics and Decision Science. Naomi is working on a health economic model to inform alcohol pricing policy options in South Africa. She is working closely with local stakeholders to conceptualise and validate the model. Naomi will be using R to create and communicate her modelling work and hopes to enable greater engagement from policy makers through interactive dashboards.

Our series of Short Courses in R.

We have put together content for a series of short courses in R. These courses are designed to provide the user with the necessary skills to utilise R to answer questions using data. By the end of the series a user should be able to manage data efficiently, analyse relationships between variables and display this in aesthetically pleasing graphs, run simulations, feed back on methods and create apps to facilitate the decision making process. The objective is to give the user an understanding of what is possible, and the foundation on which to achieve it.

Course 1 - Intro to R

By the end of the 1 day short course, the attendee should be able to:

- Install and navigate R Studio. Set working directory.
- Understand the types of objects and basic operations in R.
- Read in data from csv and excel files.
- Summarise data.
- Know where to find further information (StatsExchange/Google).

Course 2 - Beautiful Visualisations

By the end of the half day short course, the attendee should be able to:

- Create beautiful graphs using ggplot.
- Use the 'apply()' function to improve run-time.
- Create a custom function.
- Know where to find further information (StatsExchange/Google).

Course 3 - R for Health Economics

By the end of 1 day short course, the attendee should also be able to:

- Understand the strengths and limitations of R for HTA.
- Create a markov model from scratch given known parameters.
- Create a microsimulation model to incorporate hetrogeneity between groups.
- Understand the importance of tranparency of coding. In particular commenting.

Course 4 - R Shiny

By the end of the half day short course, the attendee should be able to:

- Understand the benefits and limitations of R-Shiny.
- Have a basic understanding of the principles behind R-Shiny.
- Create an R-Shiny application from scratch.
- Integrate beautiful plots into R-Shiny.
- Develop a user interface for an existing complex model in R-Shiny.

Course 5 - Collaboration in R

By the end of the half day short course, the attendee should be able to:

- Understand the strengths and limitations of R-Markdown.
- Create replicatable HTML, Word and PDF documents using R-Markdown.
- Include chunks of code, graphs, references and biblographies, links to websites and pictures.
- Change replicate analysis for new or updated datasets, or create templates for routine data.
- Create markdown presentations.

Course 1 - Introduction to R

Install and navigate R Studio.

R is a free software environment for statistical analysis and data science. It is a collaborative effort started by Robert Gentleman and Ross Ihaka in New Zealand, but now made up of hundreds of people. R is made freely available by the Comprehensive R archive Network (CRAN), in the UK it can be downloaded through the University of Bristol here.

Downloading R gives you the basic software environment, but an incredibly popular add-on called 'RStudio' is required to follow this course. You should download the free 'RStudio Desktop Open Source Licence' version for the laptop you will be attending the course with from RStudio.com.

If you have time before the course, it would be hugely beneficial to get familiar with RStudio.

Objectives:

Download R from https://www.stats.bris.ac.uk/R/.
Download RStudio from https://www.rstudio.com/products/rstudio/#Desktop.

Basics

Our course starts in the R console, which those of you who are familiar with R but not RStudio will recognise. We will enter commands as input into the console, and receive output from the console. We will start with some simple basic operations, for which R is clearly very excessive.

Basic operations

Entering 1+1, we get the output [1] 2. The output is 2, but the [1] lets us know that the number 2 is the first number of output. If we had an output that was particularly large (e.g. 100 seperate numbers) then r may let us know that the first row displayed starts with the first value [1] and the second row starts with the [x]th value.

```
rm(list = ls())

# add 1 to 1.
1 + 1
```

```
## [1] 2
```

```
# divide 12 by 4
12/4
```

```
## [1] 3
```

```
# times 3 by 7
3*7
```

```
## [1] 21
```

```r
# 10 to the power 3
10^3
```

```
## [1] 1000
```

```r
# root isn't a basic operation so we will look at later.
```

Objects

It is possible to create objects, an object can take a number of forms (e.g. a number, a vector of numbers, a matrix, a data-frame). We can then use these objects going forward rather than the values directly. Operations can be applied to these objects, and objects can be over-written. R is basically just objects and functions.

```r
# objects basics
x <-  3
y <-  5
x + y
```

```
## [1] 8
```

```r
x <-  4
x + y
```

```
## [1] 9
```

```r
z <- x + y
z
```

```
## [1] 9
```

Overwriting / Manipulating Objects

```r
a <- 10
a
```

```
## [1] 10
```

```r
a <- a + 1
a
```

```
## [1] 11
```

Exercises

set d equal to 10. divide d by 5 multiply d by 8 add 8 to d what is d?

Evaluations

We can perform evaluations, which provide a true or false answer. For example the input 4>2 returns "FALSE".

It can be very useful in cases where an outcome is binary (e.g. an individual dies or remains alive). Or where we want to change a continous variable to a binary.

```
# simple evaluations
# 4 is greater than 2
4 > 2
```

```
## [1] TRUE
```

```
# 4 is greater than 5
4 > 5
```

```
## [1] FALSE
```

```
# 4 is equal to 3, note double == for an evaluation
4 == 3
```

```
## [1] FALSE
```

```
# 4 is not equal to 3, note != is not equal to.
4 != 3
```

```
## [1] TRUE
```

```
# the character x is equal to the character x.
"dog" == "dog"
```

```
## [1] TRUE
```

```
"dog" == "cat"
```

```
## [1] FALSE
```

```
# the output from an evaluation can be stored as an object, x. This object can be subject to operations
b <- 4<2
b
```

```
## [1] FALSE
```

Different object classes and types

Objects don't have to take a single value. For example a single object may be the heights of each child in a group of children (in the example below a small class of 4).

We have been working with single values, which are vectors of 1. To illustrate the different classes we are going to create some vectors.

Object Classes

Different class include numeric, character, logical, integer & complex (ignore).

```
# numeric
height <- c(1.72,1.78,1.65,1.90) # 1:4
height
```

```
## [1] 1.72 1.78 1.65 1.90
```

```
# numeric
weight <- c(68,75,55,79)
weight
```

```
## [1] 68 75 55 79
```

```
class(weight)
```

```
## [1] "numeric"
```

```
# character
first_name <- c("Alice","Bob","Harry","Jane")
first_name
```

```
## [1] "Alice" "Bob"   "Harry" "Jane"
```

```
#first_name + 1 # error

# factor
sex <- factor(x = c("F","M","M","F"))
sex
```

```
## [1] F M M F
## Levels: F M
```

```
# logical
tall <- height > 1.8
```

Operations on different data structures

Operations on Vectors

```
#Adding:
c(1,2,3) + 1
```

```
## [1] 2 3 4
```

```r
c(1,2,3) + c(1,2,3)
```

```
## [1] 2 4 6
```

```r
#multiplication
heightft <- height*3.28
```

Exercise Create a vector called 'odds' with the numbers 1,3,5,7,9. Show what class odds is.

Evaluate which numbers in the odd vector are greater than 4.

Create a vector called 'fail' containing 1,3,5,'seven',9. Show what class fail is.

Create a vector that gives everyone's weight in pounds (2.2lbs to kg)

Basic object Types

There are multiple types of object in R. We can store objects together in a data-frame. In our example data-frame each column is a variable (height, weight, first_name), and each row is an individual.

Different object types include:

Vector - single variable a 1x1 vector. Vector all elements same data-type. Matrix - all same data-type. Dataframe - columns same data type. List - anything goes.

```r
# data frame- columns are variables, rows are observations.
df <- data.frame(height,weight,first_name,sex)
df
```

```
##   height weight first_name sex
## 1   1.72     68      Alice   F
## 2   1.78     75        Bob   M
## 3   1.65     55      Harry   M
## 4   1.90     79       Jane   F
```

```r
# we can select a single variable within the dataframe using the dollar sign.
df$height
```

```
## [1] 1.72 1.78 1.65 1.90
```

```r
# We can add a new variable easily, in this case based on other variables within the dataframe.
df$bmi <- df$weight / df$height^2
df
```

```
##   height weight first_name sex      bmi
## 1   1.72     68      Alice   F 22.98540
## 2   1.78     75        Bob   M 23.67125
## 3   1.65     55      Harry   M 20.20202
## 4   1.90     79       Jane   F 21.88366
```

Subsetting

We can subset our data, to reduce it to those we are interested in. This is useful when cleaning our data, and when changing a continuous variable to a categorical.

```
# Our data-frame contains the height, weight, first name and bmi of 4 individuals.
df
```

```
##   height weight first_name sex      bmi
## 1   1.72     68      Alice   F 22.98540
## 2   1.78     75        Bob   M 23.67125
## 3   1.65     55      Harry   M 20.20202
## 4   1.90     79       Jane   F 21.88366
```

```
#To subset a data frame we can use square brackets i.e df[row,column]
#Selecting a column(s)
df$height
```

```
## [1] 1.72 1.78 1.65 1.90
```

```
df[,"height"]
```

```
## [1] 1.72 1.78 1.65 1.90
```

```
df[,1]
```

```
## [1] 1.72 1.78 1.65 1.90
```

```
df[,1:3]
```

```
##   height weight first_name
## 1   1.72     68      Alice
## 2   1.78     75        Bob
## 3   1.65     55      Harry
## 4   1.90     79       Jane
```

```
df[,c(1,3)]
```

```
##   height first_name
## 1   1.72      Alice
## 2   1.78        Bob
## 3   1.65      Harry
## 4   1.90       Jane
```

```
#selecting a row(s)
df[1,]
```

```
##   height weight first_name sex     bmi
## 1   1.72     68      Alice   F 22.9854
```

```r
#We might also want to select observations (rows) based on characteristics of the data
#E.g. we might want to only look at the data for people who are taller than 1.75m

#create a logical variable called min_height which contains T/F for each individual being over 175cm.
min_height <- df$height >= 1.75
min_height
```

```
## [1] FALSE  TRUE FALSE  TRUE
```

```r
# Subset the data to include only those observations (rows) for which height > 175cm (using min_height)
df.at_least_175 <- df[min_height,]
df.at_least_175
```

```
##   height weight first_name sex      bmi
## 2   1.78     75        Bob   M 23.67125
## 4   1.90     79       Jane   F 21.88366
```

```r
#People smaller than 1.75m
# Subset the data to include only those who are not above min-height of 175cm.
smaller = df$height < 1.75
df[smaller,]
```

```
##   height weight first_name sex      bmi
## 1   1.72     68      Alice   F 22.98540
## 3   1.65     55      Harry   M 20.20202
```

```r
df[!min_height,]
```

```
##   height weight first_name sex      bmi
## 1   1.72     68      Alice   F 22.98540
## 3   1.65     55      Harry   M 20.20202
```

Note that there are other more advanced methods, which uses pipes and require less code (these are covered in more advanced courses).

Exercises select the 3rd row from the data frame

Select the weight variable from the data frame using your prefered method

Select alice's data from the data frame

Subset the data frame to show just the data for the females

type df[,-1] what does this give

Exercises

Exercise 1 Simple calculations

a) Calculate the following:

5*10 20/3

More complex calculations. b) Calculate x where a = 20 b = 9, c = 5, d = 1.2 *you are only allowed to type each number once*

$x = 4b + 7c + 3d$

$x = \frac{8b+4c-12d}{a}$

$\frac{12\times(a+b)}{x}$

Exercise 2 x <- c(10,30,4,52,60,7,8,10,12,15,14,17,19,20,25,30) a) Which numbers in x are above 8 b) Which numbers are equal to 10. c) Which numbers are below 8 or above 30.

d) Can you create a matrix with numbers and characters. names <- c("Anne","Tom","Jamie","Max","Claire") ages <- c(12,16,25,34,28) cbind(names,ages) What happens if you try to use the ages?

e) Create a dataframe for five individuals (Andrew, Betty, Carl, Diane and Elisa) who are aged (62,80,24,40,56) and have gender (male, female, male,female, female).

f) Use evaluations and subsetting to find the characteristics of the individual who can claim their free bus pass (age 65+).

g) Create a variable in the dataframe called life expectancy, set this to 83 for females and 80 for males.

h) Create another variable called lyr (life years remaining) which is the number of years to life expectancy for each individual

Working with Data in R

Keeping Track of progress in R

So far we have been working exclusively in the R console. This is useful for trialing code and doing quick intial analyses, however, the code we have typed is not saved for when we might look back at it in the future. If we want to keep a permanent record of our code, we can do this using a r-script. An r-script is basically a text-file containing lines of r-code. Usually we create them from scratch within R, though they can be created by importing a text file from text editor.

The easiest way to create an r-file is by clicking the button in the top left corner of RStudion that looks like a piece of paper with a greenplus over it. The use of # for commenting is common. For example below

```
getwd()       # this line of code sets the working directory.

paste("RRRRR")  # this line of code pastes RRRRR.

# One is enough, but sometimes I can use a few to make the code tidy, like below.

#====
# Section 1
#====
```

Setting Working Directory

When we use R, it is always associated with a specific directory within our computer. The place that R is associated with is known as the working directory. The working directory is the default place where R will look when we try to import (export) objects into (to) R as well as the place that files are saved to. We can find out which directory R is pointed at by using the getwd() function:

```
getwd()
```

```
## [1] "C:/Users/Robert/Google Drive/Teaching/R Course/Intro_to_R"
```

If you know that you will be reading and writing multiple files from and to the same folder, you can set the working directory to that folder. This can be useful when a project has many different r-files and associated items such as data, functions, plots etc. In this case, one can set the working directory to the folder containing the files to make sure that everything stays in one place. It is also useful for when projects are shared between individuals using different computers, as setting the working directory to the shared folder prevents any isses that could arise from people organising their files in different ways.

A new working directory can be set by clicking on the tab (Session) then (Set_Working Directory), or by the command setwd. Below I give the example of setting the working directory to my documents.

```
filename = "C:/Users/Robert/Google Drive/Teaching/R Course/Intro_to_R"

setwd(filename)
```

Importing Data

In almost every project, you will want to import some data to analyse. This data will come in a variety of formats. R studio has a nice feature in Environment>Import_Dataset which enables you to select the file you wish to download (similar to STATA) from your computer. The data is then imported into R and the code that R has used is displayed in the console.

It is possible to import files in the following formats:

Type	Suffix
R	.R
CSV	.csv
Excel	.xls/.xlsx
SPSS	.spv
Stata	.dta

If we want more control over the way that R imports the data we can also use the necessary commands in R directly. Some important examples of this are given in the next subsections.

In addition, packages can be installed to import data in almost any format. For example the readr package can read in spreadsheets from text files or tab delimited files.

CSV (Comma-seperated values)

A common format of data that you will likely import is comma-seperated values (CSV) data.Data is seperated by commas in rows. For example:

Age,Name,Sex,
30,Richard,Male,
27,Hazel,Female,
28,Louise,"",

Creates:

Age	Name	Sex
30	Richard	Male
27	Hazel	Female
28	Louise	

We can import the file using the full path with the file name and suffix included such as below. This will look in the working directory for the file specified, so given our working directory is "C:/Users/Robert/Documents" R will look in the Documents folder for the file "car_Data.csv".

It will then convert the first row to be the header of the data. There are numerous other options which we will skip for now.

```
# car_Data <- read.csv(file = "car_Data.csv", header = TRUE)

# if you couldn't get that to work don't worry, this is an example dataset from base R.
car_Data <- mtcars
```

Downloading files from the internet

Sometimes it is more practical to download files directly from the internet. There are lots of different packages out there to do this. The one I use was developed by Hadley Wickham, called readr. Here we are going to download some data from the github page for the course.

```
# load the readr package, if this is not installed then install it.

library(readr)

#use the function read_csv

car_Data <- read_csv("https://raw.githubusercontent.com/RobertASmith/Intro_to_R/master/car_Data.csv", h
```

Downloading files directly to R within the same script as the analysis can be useful since it reduces the risk of you accidently changing the file. Just be careful that the data will always be available.

Summarising Data

Once we have our data read into R, we want to ensure that the data is as we would expect, in the correct format etc.

We can use the function *head* to look at the first 6 number of lines of the data. We can specify a different number of lines by chancing the function input.

```
# head data with default 6 rows
head(car_Data)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
# head data with 10 rows
head(car_Data, n = 10)
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

We can summarise a dataset using the function *summary*. This shows us the length, class and Mode. If the class is numeric it will give some indication of the distribution by displaying min, median, mean, max.

```r
# summarise the data,

summary(car_Data)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

```r
# summarise single variable

summary(car_Data$mpg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   10.40   15.43   19.20   20.09   22.80   33.90
```

We can use the output of the summary function to create objects. The summary of the mpg variable gives the quantiles. These can be stored as an object, here called temp (temporary object). If we just want any one number from the vector of quantiles we can define this in brackets. The script below creates two new objects, median and range.

```
temp <- summary(car_Data$mpg)

Median  <- temp['Median']

Range   <- temp['Max.'] - temp['Min.']
```

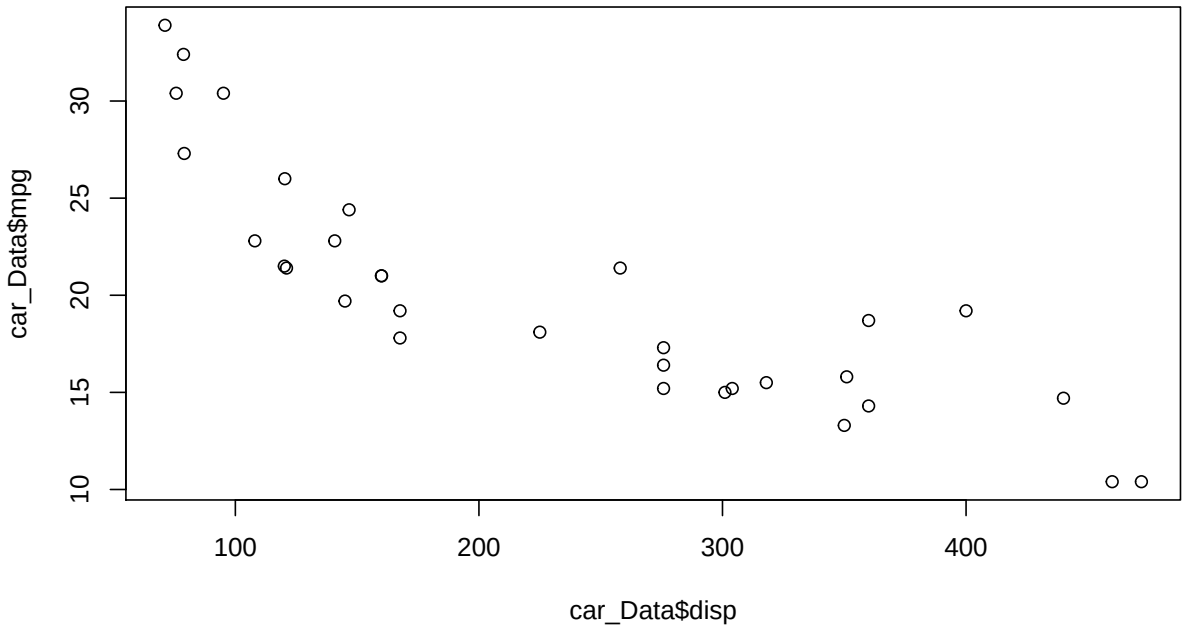
Plotting Data

Line Plot

R also has wide ranging plotting capabilites. For basic plotting we can use the *plot* function. In this next example, we will prodcue a simple plot of miles per gallon vs engine displacement in our data set to see what the relationship between the variables.

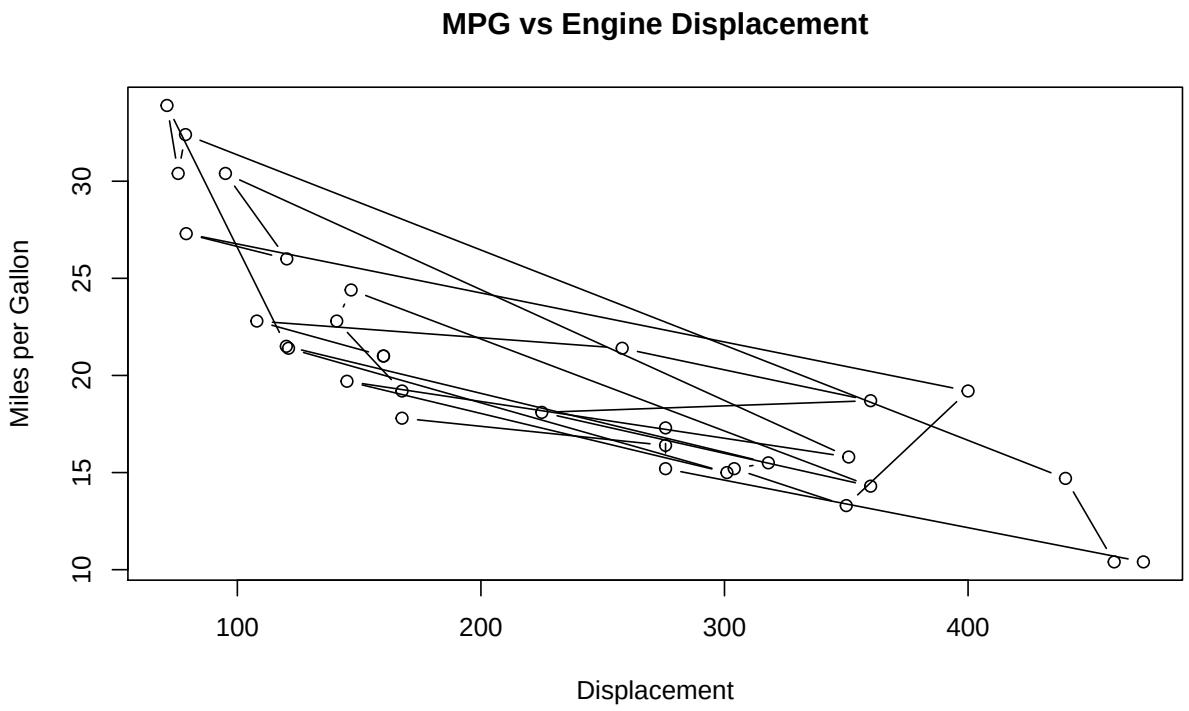
```
#plot of mpg vs disp
plot(x = car_Data$disp, y = car_Data$mpg)

#notice we can remove arguments and still get same result
plot(car_Data$disp, car_Data$mpg)
```



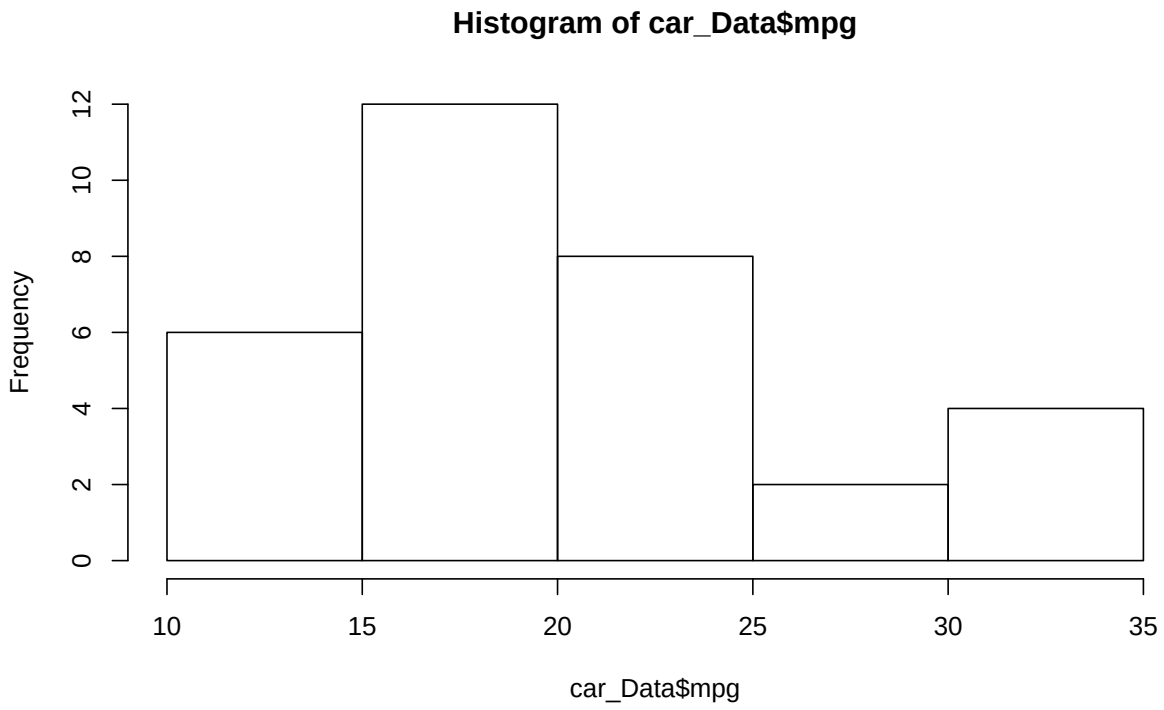
Whilst this plot is useful, it is quite basic. We make the plot more informative by specifying extra features that we want when we call the plot function. We can add labels, titles, lines of best fit and more.

```
plot(x = car_Data$disp, y = car_Data$mpg,
     type = "b",
     xlab = "Displacement",
     ylab = "Miles per Gallon",
     main = "MPG vs Engine Displacement")
```



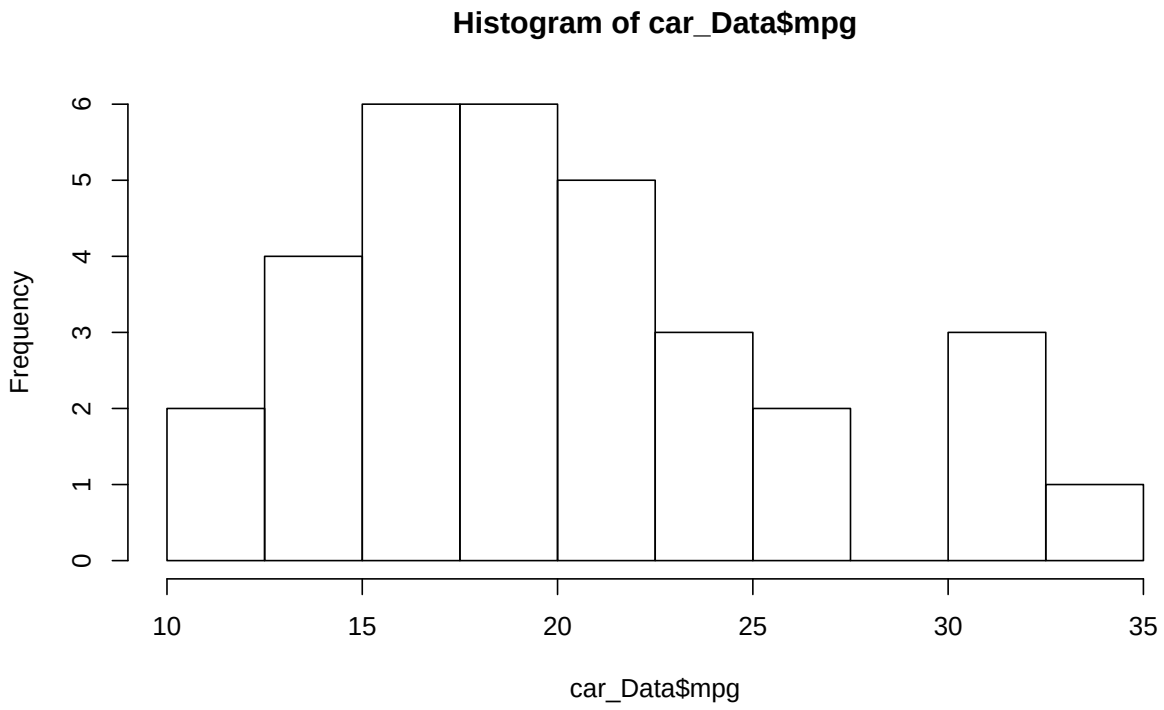
Sometimes we may just want to see the distribution of a single variable in the data. For numerical variables this is done easily by using plotting a histogram. To plot a histogram in R we use the command *hist*.

```
hist1 <- hist(car_Data$mpg)
```

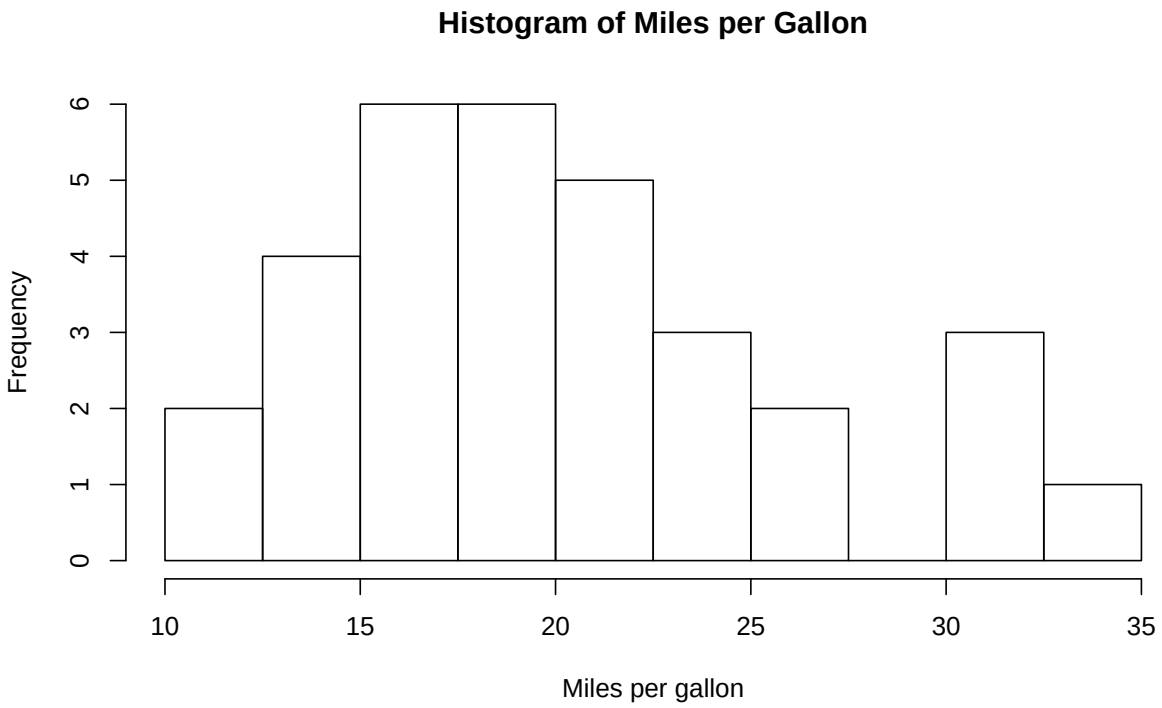



```r
#We can alter the 'bins' by specifying the additional argument 'breaks = ' in the hist function
hist(car_Data$mpg, breaks = c(10,12.5,15,17.5,20,22.5,25,27.5,30,32.5,35))

#a neater way of doing the same as above is to use seq
hist(car_Data$mpg, breaks = seq(10,35, by = 2.5))
```




```
#we can again edit the title etc by adding extra arguments
hist(car_Data$mpg,
     breaks = seq(10,35, by = 2.5),
     xlab = "Miles per gallon",
     main = "Histogram of Miles per Gallon")
```


Histogram of Miles per Gallon
Frequency
0
1
2
3
4
5
6
10
15
20
25
30
35
Miles per gallon