Introduction to R programming

Lusaka 9 - 11 July, 2025

Francis Edwardes

Day 1: Data handling and Descriptive statistics

1. R coding refresher

1.1 R Math

Let's start with the fundamentals. To get acquainted with the R programming environment, begin with some basic computations. The R Console can also function as an interactive calculator.

```
log(1) + sqrt(4)
```

```
[1] 2
```

```
1/3 + exp(1)
```

```
[1] 3.051615
```

```
log(100, base = 10)
```

```
[1] 2
```

```
4^2 + 2
```

```
[1] 18
```

```
sum(4^2, 2)
```

```
[1] 18
```

The *ceiling()* function rounds a number upwards to its nearest integer, and the *floor()* function rounds a number downwards to its nearest integer, and returns the result:

```
ceiling(1.4)
```

```
[1] 2
```

```
floor(1.4)
```

```
[1] 1
```

1.2 R comments

You can use the comments before a line code

```
#Print Good morning
"Good morning"
```

```
[1] "Good morning"
```

You can use it at the end of the line code

```
"Good morning" #Print Good morning
```

```
[1] "Good morning"
```

You can also use multiple line comments

```
# What is the value of
# square root of 3
print(sqrt(3))
```

```
[1] 1.732051
```

1.3 Variables

To create variables in R you can do the following

```
Surname <- "Alice"
Age <- 12
print(Surname)
```

```
[1] "Alice"
```

```
Age
```

```
[1] 12
```

To concatenate multiple words you can use the function *paste()*

```
text1 <- "Alice is"
text2 <- "beautiful"
paste(text1, text2)
```

```
[1] "Alice is beautiful"
```

How to assign a real value to a variable?

```
x <- 2
y <- 3
x+y
```

```
[1] 5
```

Can we concatenate a text with a numeric?

```
paste("mum", 6)
```

```
[1] "mum 6"
```

Can we assign multiple variables to a same value?

```r
x <- y <- z <- 16
x
```

```
[1] 16
```

```r
y
```

```
[1] 16
```

```r
z
```

```
[1] 16
```

1.4 R Data types

The *class()* function can be used to check the data types of a variable

```r
a <- 5L
class(a)
```

```
[1] "integer"
```

```r
b <- 5.5
class(b)
```

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

```r
c <- 1 + 1i
class(c)
```

```
[1] "complex"
```

```r
x <- "Hi everyone"
class(x)
```

```
[1] "character"
```

```r
y <- FALSE
class(y)
```

```
[1] "logical"
```

Can we convert a from numeric to complex?

```r
x <- 2
y <- as.complex(x)
y
```

```
[1] 2+0i
```

```r
class(y)
```

```
[1] "complex"
```

1.5 R string

Can we count the number of characters in a string?

```r
str <- "Brilliant"
nchar(str)
```

```
[1] 9
```

Can we check if a character or a sequence of characters are present in a string?

```r
str <- "Good Morning!"

grepl("d", str)
```

```
[1] TRUE
```

```r
grepl("s", str)
```

```
[1] FALSE
```

```
grepl("Good", str)
```

```
[1] TRUE
```

Can I write a text which contains word(s) in quotation marks?

```
str <- "Have you heard the song \"Don't Stop Believing in yourself\"?"
cat(str)
```

```
Have you heard the song "Don't Stop Believing in yourself"?
```

1.6 R Booleans

When you compare two values, the expression is evaluated and R returns the logical answer(TRUE or False):

```
x <- -1
y <- 0
x == y
```

```
[1] FALSE
```

1.6 The if statement

An "if statement" is created using the `if` keyword and is utilized to define a block of code that executes when a condition evaluates to TRUE.

```
x <- 15
y <- 3

if (x > y) {
  print("x is greater than y")
}
```

```
[1] "x is greater than y"
```

1.7 Else If

The `else if` keyword is R's way of saying "if the previous conditions were not true, then try this condition":

```
a <- 10
b <- 25

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print ("a and b are equal")
}
```

```
[1] "b is greater than a"
```

1.8 If Else

The `else` keyword catches anything which isn't caught by the preceding conditions:

```
a <- 10
b <- 15

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print("a and b are equal")
} else {
  print("a is greater than b")
}
```

```
[1] "b is greater than a"
```

You can also use `else` without `else if`:

```
a <- 213
b <- 310

if (b > a) {
  print("b is greater than a")
```

```
} else {
  print("b is not greater than a")
}
```

```
[1] "b is greater than a"
```

1.9 R While Loops

With the `while` loop we can execute a set of statements as long as a condition is TRUE. Example: Print `i` as long as `i` is less than 6:

```
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

1.10 Break

With the `break` statement, we can stop the loop even if the while condition is TRUE. Example: Exit the loop if `i` is equal to 4.

```
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
  if (i == 4) {
    break
  }
}
```

```
[1] 1
[1] 2
[1] 3
```

1.11 Next

With the `next` statement, we can skip an iteration without terminating the loop. Example: Skip the value of 3.

```
i <- 0
while (i < 6) {
  i <- i + 1
  if (i == 3) {
    next
  }
  print(i)
}
```

```
[1] 1
[1] 2
[1] 4
[1] 5
[1] 6
```

1.12 R For Loop

A `for` loop is used for iterating over a sequence.

Example1: Print the first 10 natural numbers.

```
for (x in 0:9) {
  print(x)
}
```

```
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

Example2: Print every item in a list

```
fruits <- list("apple", "banana", "cherry")

for (x in fruits) {
  print(x)
}
```

```
[1] "apple"
[1] "banana"
[1] "cherry"
```

Example3: Print every item in a vector

```
vec <- c(1, 2, 3, 4, 5, 6)

for (x in vec) {
  print(x)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
```

1.13 R Functions

1.13.1 Creating a function

To create a function, use the `function()` keyword:

```
my_function <- function() {      # create a function with the name my_function
  print("Hello World!")
}
```

1.13.2 Call a Function

To call a function, use the function name followed by parenthesis, like **my_function()**:

```r
my_function <- function() {
  print("Hello World!")
}

my_function() # call the function named my_function
```

```
[1] "Hello World!"
```

1.13.3 Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

```r
my_function <- function(fname) {
  paste(fname, "Denis")
}

my_function("Lara")
```

```
[1] "Lara Denis"
```

```r
my_function("Kirsten")
```

```
[1] "Kirsten Denis"
```

```r
my_function("Frank")
```

```
[1] "Frank Denis"
```

1.13.4 Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less:

```r
my_function <- function(fname, lname) {
  paste(fname, lname)
}

my_function("Kirsten", "Denis")
```

```
[1] "Kirsten Denis"
```

1.13.5 Return Values

To let a function return a result, use the return() function:

```r
my_function <- function(x) {
  return (3 * x)
}
print(my_function(5))
```

```
[1] 15
```

2. Descriptive Statistics

2.1 Importing CSV file into R

For importing data CSV file into R, it is important to specify the working directory (folder name) that the data is stored in - either an internal or external hard drive.

For example, this can be done with code that looks like the following (the code has been commented out to preserve the original working directory - it is shown for simple demonstration purposes).

```r
#setwd("C:/Users/user-name/My Documents/R Workshop/Day 1")
```

To check the working directory your R Session is working in you can use the following function.

```r
getwd()
```

```
[1] "D:/SUN PostDoc/R-Workshop/Day 1"
```

Now to read the data into your R environment you use the `read.csv()` function and store it into an object named 'chap1data1.

```
chap1data1 <-read.csv("chap1data1.csv", sep=",", header=T)
```

Alternatively, you can specify directory path to the stored data directly to the 'read.csv()' function. This helps when your working directory is set elsewhere, and you want to access data stored in a different directory. (The code has been commented out to to serve as a simple example.)

```
# chap1data1 <- read.csv(
#  "C:/Users/user-name/My Documents/R Workshop/Day 1/chap1data1.csv",
#  sep=",", header=T
#  )
```

2.2 Viewing the data

```
dim(chap1data1)
```

```
[1] 62 11
```

```
head(chap1data1)
```

```
  Male Married Runs.per.week  Age Income College Distance Treadmill
1    1       0             6 29.5   57.5       1        1         0
2    1       1             5 43.5   57.5       1        0         0
3    1       0             5 29.5   42.5       1        0         0
4    0       1             6 36.5   57.5       1        1         0
5    1       1             6 36.5     NA       1        1         0
6    1       1             5 43.5  112.5       0        0         0
  Miles.per.week Type.of.running Shoes
1           42.5       1Marathon     4
2           37.5          5Other     4
3           22.5          5Other     2
4           37.5  2Half marathon     4
5           47.5       1Marathon     5
6           32.5          5Other     3
```

```r
tail(chap1data1)
```

```
   Male Married Runs.per.week  Age Income College Distance Treadmill
57    0       0             4 29.5     NA       1        0         0
58    0       0             4 29.5   42.5       1        0         0
59    1       1             5 43.5   87.5       1        0         0
60    1       1             7 29.5   57.5       1        0         0
61    1       1             5 48.5   42.5       1        1         0
62    1       1             5 61.5   57.5       1        0         0
   Miles.per.week Type.of.running Shoes
57           27.5          5Other     2
58           12.5          5Other     2
59           12.5          5Other     1
60           47.5          5Other     1
61           37.5       1Marathon     2
62           27.5          5Other     3
```

2.3 Missing values

Finding and addressing missing values is a key step in preparing a clean dataset for analysis.

To identify missing values within the variables of a dataset, we can use the `apply()` function. This function lets us apply a custom operation, such as checking for `NA` values, across either rows or columns by specifying the `MARGIN` argument. Use `MARGIN = 1` to apply a function row-wise, and `MARGIN = 2` to apply it column-wise.

```r
apply(X = chap1data1, MARGIN = 2, FUN = is.na)
```

```
      Male Married Runs.per.week   Age Income College Distance Treadmill
 [1,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
 [2,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
 [3,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
 [4,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
 [5,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
 [6,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
 [7,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
 [8,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
 [9,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[10,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[11,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[12,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
```

```
[13,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[14,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[15,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[16,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[17,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[18,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[19,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[20,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[21,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[22,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[23,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[24,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[25,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[26,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[27,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[28,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[29,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[30,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[31,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[32,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[33,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[34,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[35,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[36,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[37,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[38,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[39,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[40,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[41,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[42,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[43,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[44,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[45,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[46,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[47,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[48,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[49,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[50,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[51,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[52,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[53,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[54,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[55,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
```

```
[56,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[57,] FALSE   FALSE         FALSE FALSE   TRUE   FALSE    FALSE     FALSE
[58,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[59,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[60,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[61,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
[62,] FALSE   FALSE         FALSE FALSE  FALSE   FALSE    FALSE     FALSE
      Miles.per.week Type.of.running Shoes
 [1,]          FALSE           FALSE FALSE
 [2,]          FALSE           FALSE FALSE
 [3,]          FALSE           FALSE FALSE
 [4,]          FALSE           FALSE FALSE
 [5,]          FALSE           FALSE FALSE
 [6,]          FALSE           FALSE FALSE
 [7,]          FALSE           FALSE FALSE
 [8,]          FALSE           FALSE FALSE
 [9,]          FALSE           FALSE FALSE
[10,]          FALSE           FALSE FALSE
[11,]          FALSE           FALSE FALSE
[12,]          FALSE           FALSE FALSE
[13,]          FALSE           FALSE FALSE
[14,]          FALSE           FALSE FALSE
[15,]          FALSE           FALSE FALSE
[16,]          FALSE           FALSE FALSE
[17,]          FALSE           FALSE FALSE
[18,]          FALSE           FALSE FALSE
[19,]          FALSE           FALSE FALSE
[20,]          FALSE           FALSE FALSE
[21,]          FALSE           FALSE FALSE
[22,]          FALSE           FALSE FALSE
[23,]          FALSE           FALSE FALSE
[24,]          FALSE           FALSE FALSE
[25,]          FALSE           FALSE FALSE
[26,]          FALSE           FALSE FALSE
[27,]          FALSE           FALSE FALSE
[28,]          FALSE           FALSE FALSE
[29,]          FALSE           FALSE FALSE
[30,]          FALSE           FALSE FALSE
[31,]          FALSE           FALSE FALSE
[32,]          FALSE           FALSE FALSE
[33,]          FALSE           FALSE FALSE
[34,]          FALSE           FALSE FALSE
[35,]          FALSE           FALSE FALSE
```

```
[36,]          FALSE          FALSE FALSE
[37,]          FALSE          FALSE FALSE
[38,]          FALSE          FALSE FALSE
[39,]          FALSE          FALSE FALSE
[40,]          FALSE          FALSE FALSE
[41,]          FALSE          FALSE FALSE
[42,]          FALSE          FALSE FALSE
[43,]          FALSE          FALSE FALSE
[44,]          FALSE          FALSE FALSE
[45,]          FALSE          FALSE FALSE
[46,]          FALSE          FALSE FALSE
[47,]          FALSE          FALSE FALSE
[48,]          FALSE          FALSE FALSE
[49,]          FALSE          FALSE FALSE
[50,]          FALSE          FALSE FALSE
[51,]          FALSE          FALSE FALSE
[52,]          FALSE          FALSE FALSE
[53,]          FALSE          FALSE FALSE
[54,]          FALSE          FALSE FALSE
[55,]          FALSE          FALSE FALSE
[56,]          FALSE          FALSE FALSE
[57,]          FALSE          FALSE FALSE
[58,]          FALSE          FALSE FALSE
[59,]          FALSE          FALSE FALSE
[60,]          FALSE          FALSE FALSE
[61,]          FALSE          FALSE FALSE
[62,]          FALSE          FALSE FALSE
```

As you can see, this produces a TRUE or FALSE output for each value of the dataset. This might be a bit much for a dataset that has many (i.e., thousands) of entries for us to check individually. We might be interest in understanding if at least 1 missing value is present.

In the example below, we use `apply(chap1data1, 2, function(x) any(is.na(x)))` to check whether each column in `chap1data1` contains any missing values. The result is a logical vector indicating which variables have at least one `NA`.

```
apply(X = chap1data1, MARGIN = 2, FUN = function(x) any(is.na(x)))
```

```
          Male        Married Runs.per.week            Age         Income
         FALSE          FALSE          FALSE          FALSE           TRUE
       College       Distance      Treadmill Miles.per.week Type.of.running
         FALSE          FALSE          FALSE          FALSE          FALSE
```

```
Shoes
FALSE
```

We find that the `Income` variable is the only variable to contain at least one missing value. If a basic criterion of your analysis is to keep complete observations only - the observations with missing values can be removed. To remove observations (i.e., the rows) corresponding to the missing values of the variable from the dataset we can use the code below.

```
chap1data1 <- chap1data1[which(!is.na(chap1data1$Income)), ]
```

Now, check how many observations exist in the updated dataset.

```
dim(chap1data1)
```

```
[1] 48 11
```

2.4 Outliers

For demonstrative purposes, let's use one variable, in this case `Age` to check if any outlines exist within it. Let's look at the range and quantiles of `Age`.

```
range(chap1data1$Age)
```

```
[1]  16.5 295.0
```

```
quantile(chap1data1$Age)
```

```
   0%   25%   50%   75%  100%
 16.5  22.5  29.5  36.5 295.0
```

Looks like there could be at least one outlying value - who can be 295 years old, and still running?!?!

Right, let's find the outlier(s) mathematically using the interquartile range method with the `IQR()`function.

```r
# Compute Q1 and Q3
Q1 <- quantile(chap1data1$Age, 0.25)
Q3 <- quantile(chap1data1$Age, 0.75)

# Compute IQR
IQR_value <- IQR(chap1data1$Age)

# Define bounds
lower_bound <- Q1 - 1.5 * IQR_value
upper_bound <- Q3 + 1.5 * IQR_value

# Identify outliers
outliers <- chap1data1$Age[chap1data1$Age < lower_bound |
                           chap1data1$Age > upper_bound]

# View result
outliers
```

```
[1] 295.0  61.5
```

Let's examine these observations a little closer.

```r
chap1data1[which(chap1data1$Age %in% outliers), ]
```

```
   Male Married Runs.per.week   Age Income College Distance Treadmill
46    0       1             1 295.0   57.5       1        0         0
62    1       1             5  61.5   57.5       1        0         0
   Miles.per.week Type.of.running Shoes
46          500.0          5Other     1
62           27.5          5Other     3
```

It seems there is one observation that is likely to be a data entry error, while the other entry seems to demonstrate natural variation in the sample. Based on our contextual understanding of the sample, we decide to remove the observation corresponding to the outlier in age likely due to a data entry error, while keeping the observation likely to represent natural variation.

```r
chap1data1 <- chap1data1[chap1data1$Age < 295, ]
dim(chap1data1)
```

```
[1] 47 11
```

2.5 Categorical variables

To prevent categorical variables from being treated as numerical data, we need to define them as factors.

```
chap1data1$Male <- as.factor(chap1data1$Male)
chap1data1$Married<- as.factor(chap1data1$Married)
chap1data1$College <- as.factor(chap1data1$College)
chap1data1$Distance <- as.factor(chap1data1$Distance)
chap1data1$Treadmill <- as.factor(chap1data1$Treadmill)
chap1data1$Shoes <- as.factor(chap1data1$Shoes)
```

2.6 Installing and loading required packages

To find out more of a specific package run the ??[package name]

```
options(repos = c(CRAN = "https://cloud.r-project.org"))
install.packages('RcmdrMisc')
```

```
Installing package into 'C:/Users/franc/AppData/Local/R/win-library/4.4'
(as 'lib' is unspecified)
```

```
package 'RcmdrMisc' successfully unpacked and MD5 sums checked
```

```
The downloaded binary packages are in
    C:\Users\franc\AppData\Local\Temp\Rtmpg5Ddgo\downloaded_packages
```

```
library(RcmdrMisc)
```

```
Loading required package: car
```

```
Loading required package: carData
```

```
Loading required package: sandwich
```

2.7 Summary statistics

We can get summary statistics of a specific variable, or all variables. The function numSummary can be used to get specific statistics

mean: measure of centrality sd: standard deviation measure of distribution skewness: measure of distribution and normality kurtosis: indication of form of frequency distribution

```
summary(chap1data1)
```

```
 Male   Married Runs.per.week        Age             Income       College
 0: 9   0:24    Min.   :2.000   Min.   :16.50   Min.   : 20.00   0:12
 1:38   1:23    1st Qu.:4.000   1st Qu.:22.50   1st Qu.: 42.50   1:35
                Median :5.000   Median :29.50   Median : 42.50
                Mean   :4.979   Mean   :31.61   Mean   : 55.37
                3rd Qu.:6.000   3rd Qu.:36.50   3rd Qu.: 57.50
                Max.   :7.000   Max.   :61.50   Max.   :112.50

 Distance Treadmill Miles.per.week  Type.of.running    Shoes
 0:40     0:45      Min.   : 5.00   Length:47          1:13
 1: 7     1: 2      1st Qu.:12.50   Class :character   2:13
                    Median :27.50   Mode  :character   3:11
                    Mean   :24.74                      4: 7
                    3rd Qu.:32.50                      5: 1
                    Max.   :57.50                      6: 1
                                                       8: 1
```

```
summary(chap1data1[,"Age"])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  16.50   22.50   29.50   31.61   36.50   61.50
```

```
sapply(chap1data1,summary)
```

```
$Male
 0  1
 9 38

$Married
 0  1
24 23
```

```
$Runs.per.week
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   4.000   5.000   4.979   6.000   7.000

$Age
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  16.50   22.50   29.50   31.61   36.50   61.50

$Income
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  20.00   42.50   42.50   55.37   57.50  112.50

$College
 0  1
12 35

$Distance
 0  1
40  7

$Treadmill
 0  1
45  2

$Miles.per.week
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   5.00   12.50   27.50   24.74   32.50   57.50

$Type.of.running
   Length     Class      Mode
       47 character character

$Shoes
 1  2  3  4  5  6  8
13 13 11  7  1  1  1
```

```
numSummary(chap1data1[,"Age"],statistics=c("mean","sd","skewness","kurtosis"))
```

```
     mean       sd  skewness  kurtosis  n
 31.60638 10.34037 0.6896707 0.3425462 47
```

2.8 Visualisation

Below is a list of different type of graphs used according to a given data type:

- Bar chart (for categorical or discrete numerical data)
- Line plot (similar to a bar chart, used to emphasize discreteness of the variable)
- Histogram (for continuous data)
- Pie chart (for categorical data measured on a nominal scale)
- Scatter plot (to inspect the relationship between two variables)
- Box plot (to inspect the distribution of a numerical variable measured on a continuous scale)

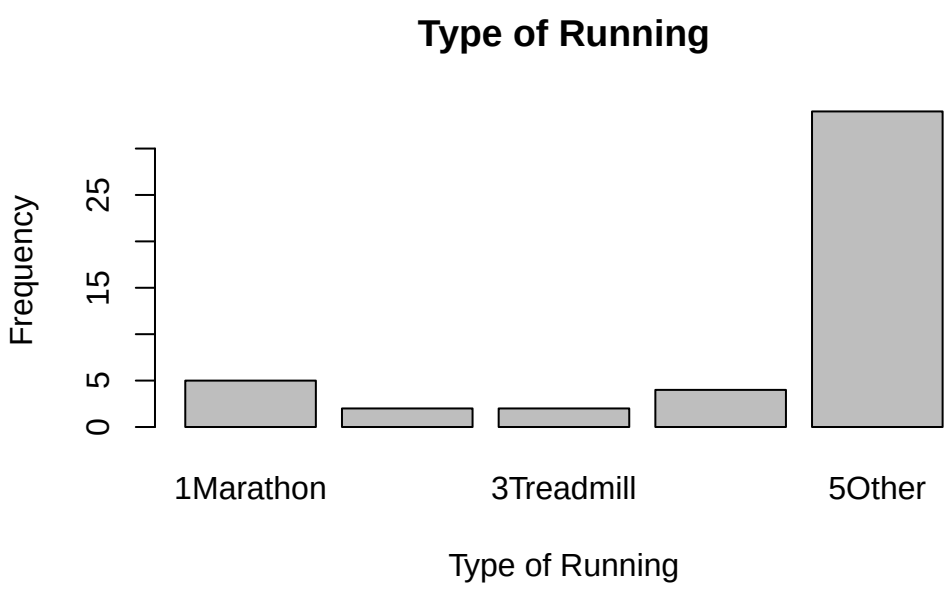
We will use two plotting systems in R in this presentation:

1. Base plotting
2. The ggplot system

Let us start with plotting the aforementioned list of different type of graphs with the R base plot

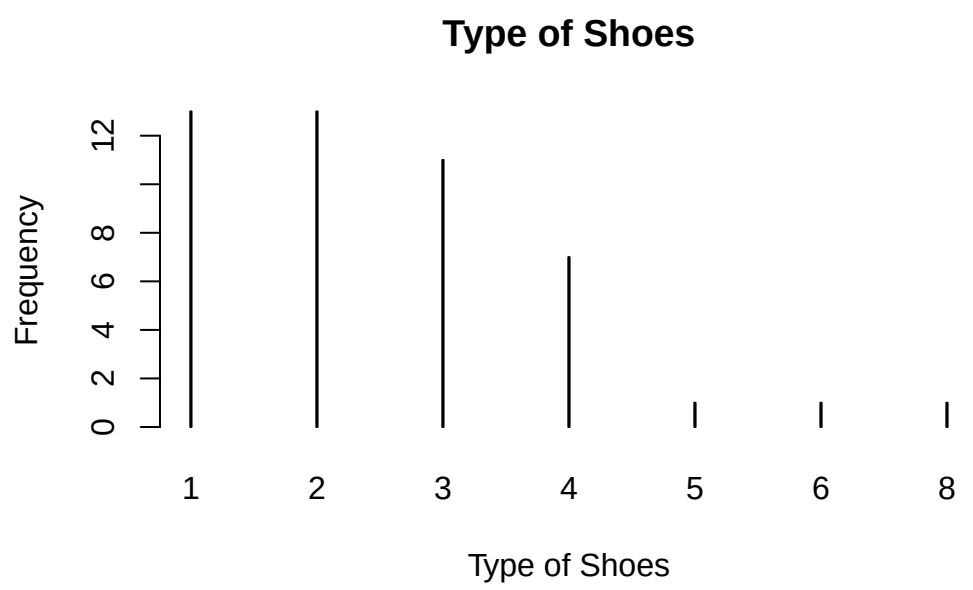
2.8.1 Bars plot

```
##Type of running
barplot(table(chap1data1$Type.of.running),
        main="Type of Running",ylab="Frequency",xlab="Type of Running")
```



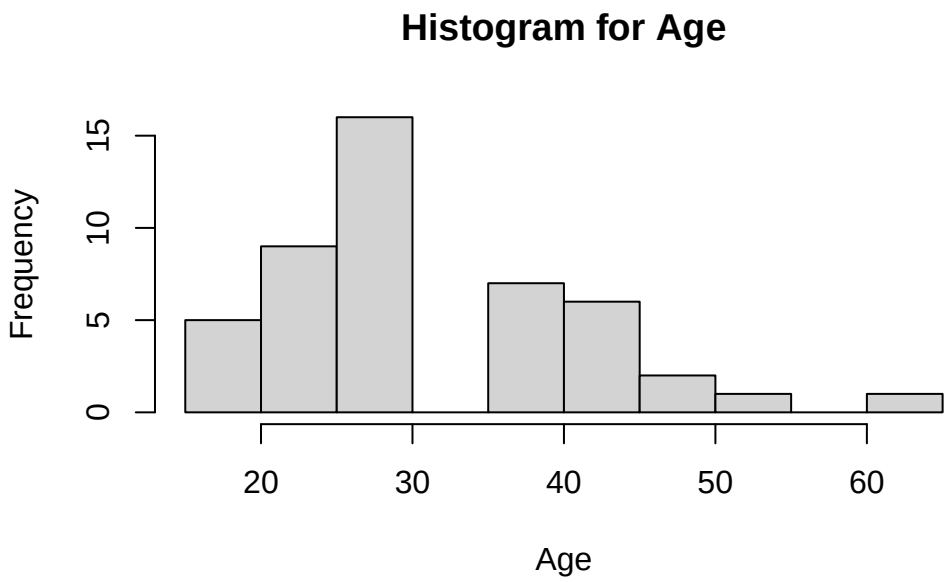
2.8.2 Line plot

```
###Shoes
barplot(table(chap1data1[,"Shoes"]),
        main="Type of Shoes",ylab="Frequency",xlab="Type of Shoes",
        col="green", width =1, space =100)
```



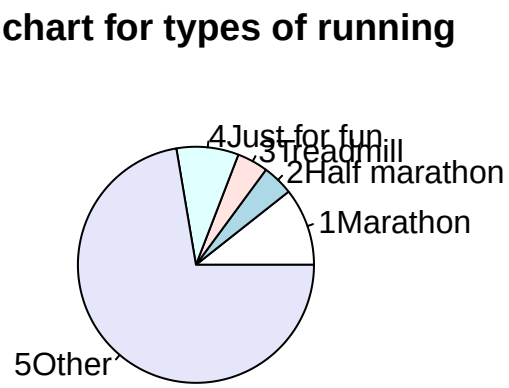
2.8.4 Histogram

```
##Age
hist(chap1data1[,"Age"],main="Histogram for Age",xlab="Age")
```



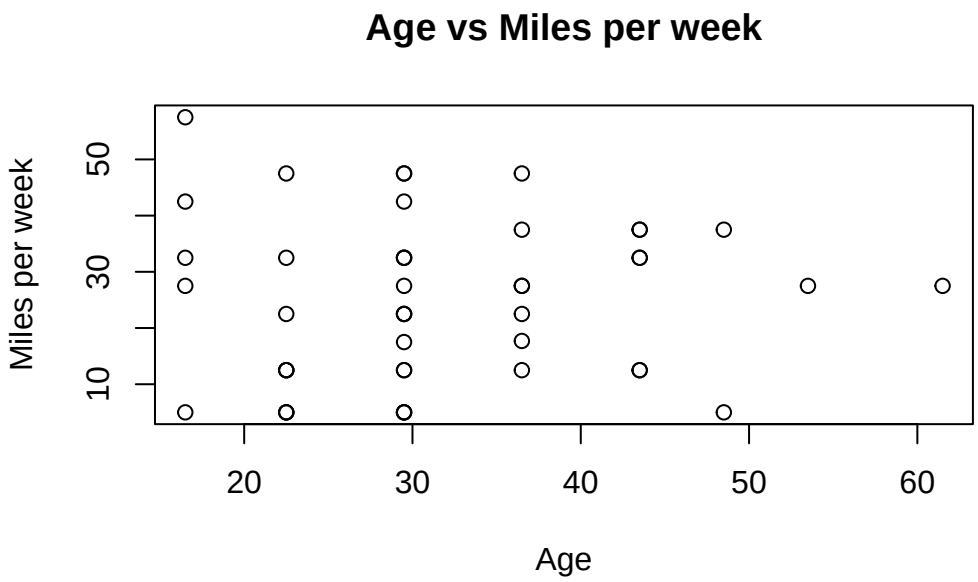
2.8.5 Pie chart

```
###Type of running
pie(table(chap1data1[,"Type.of.running"]),main="Pie chart for types of running")
```



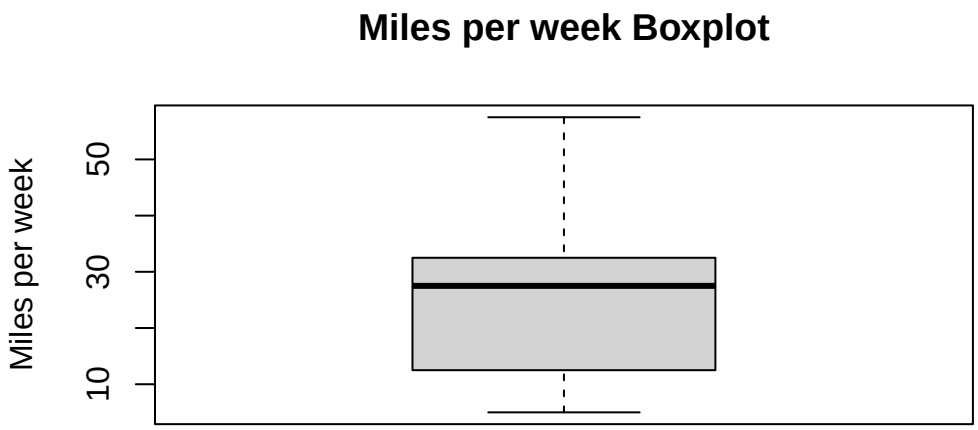
2.8.5 Scatterplot

```
## Age and Miles.per.week
plot(x=chap1data1[,"Age"],y=chap1data1[,"Miles.per.week"],
     main="Age vs Miles per week",ylab="Miles per week",xlab="Age")
```



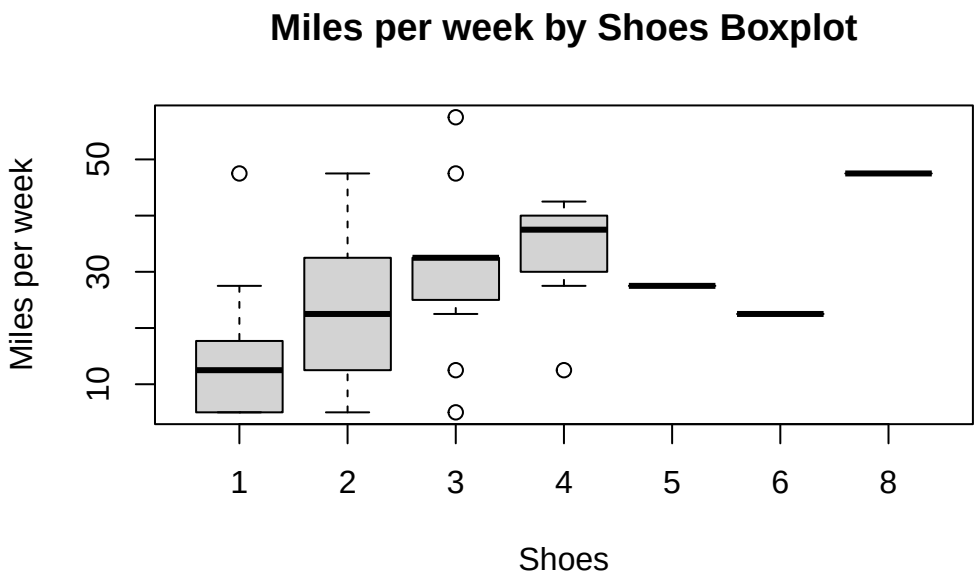
2.8.6 Box plot

```
## Miles.per.week
boxplot(chap1data1[,"Miles.per.week"],
        main="Miles per week Boxplot",ylab="Miles per week",xlab="")
```



2.8.7 Box plot by groups

```
## Miles.per.week by Shoes
boxplot(Miles.per.week~Shoes,data=chap1data1,
        main="Miles per week by Shoes Boxplot",ylab="Miles per week",xlab="Shoes")
```



To use the ggplot plotting system you need to install the package first and load it into R

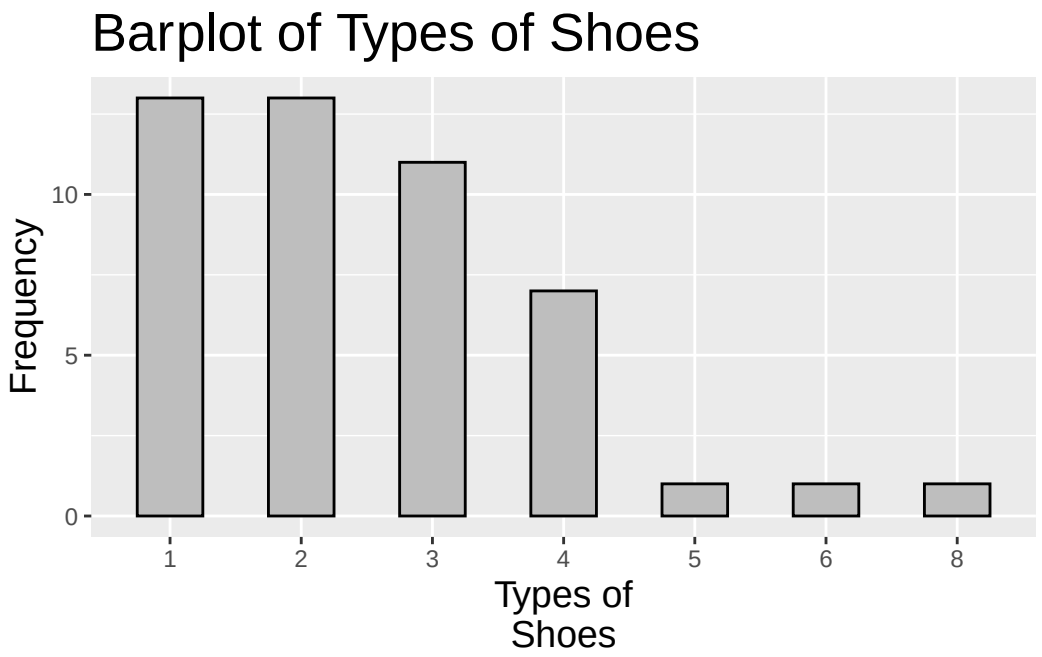
```
if(!"ggplot2" %in% installed.packages()[, "Package"])
{
 install.packages('ggplot2')
}

library(ggplot2)

#or do the following

#In the miscellaneous window, click on Packages.
#Click on Install, then type "ggplot2" beneath Packages (separate multiple with space or comm
#Make sure Install dependencies is selected, then click on Install
#After installation, load the package into R: package(ggplot2)
```

Construct the same graphs (except the line plot) with ggplot plotting system.

2.8.8 Bar plot – Shoes

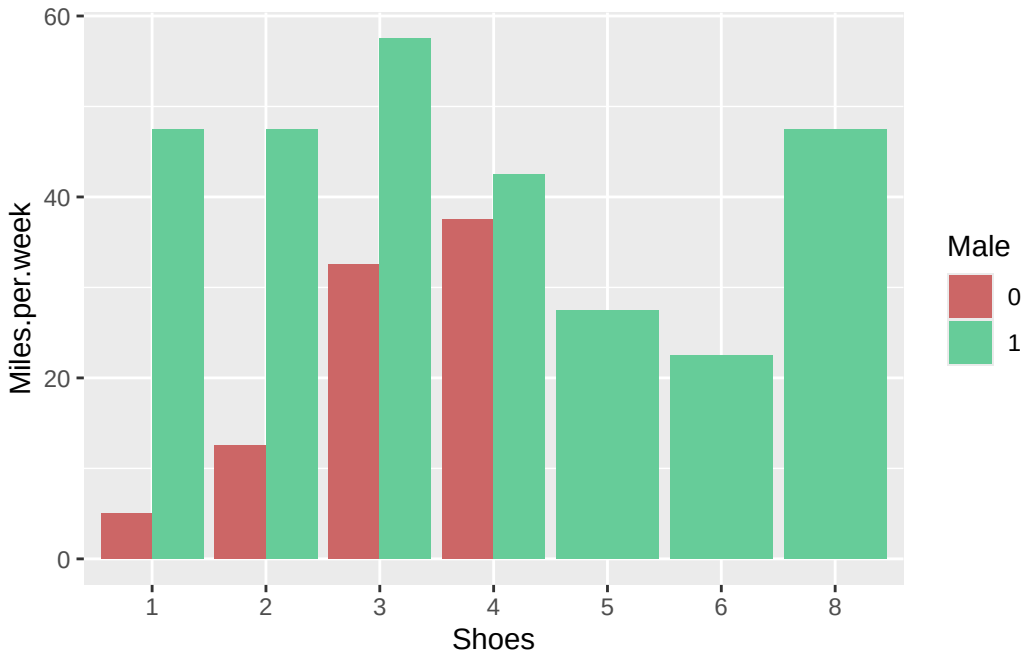
```
barplot <- ggplot(data=chap1data1,aes(x=Shoes)) +
geom_bar(fill="gray",color="black",width=.5)
```

```
barplot + ggtitle("Barplot of Types of Shoes") + labs(x="Types of
Shoes",y="Frequency") +
theme(plot.title=element_text(size=20),axis.title=element_text(size=
14))
```



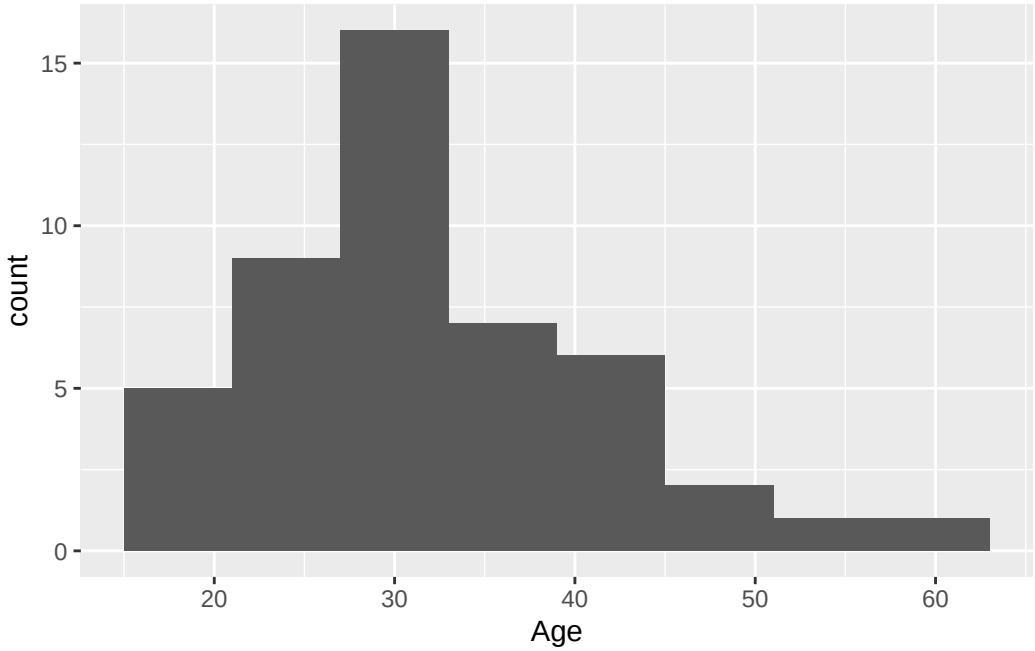
2.8.9 Bar plot by groups – Shoes by Male

```
barplot2 <- ggplot(data=chap1data1,aes(x=Shoes,y=Miles.per.week))
barplot2 + geom_bar(aes(fill=Male),
position=position_dodge(),stat="identity") +
scale_fill_manual(values=c("#CC6666","#66CC99"))
```



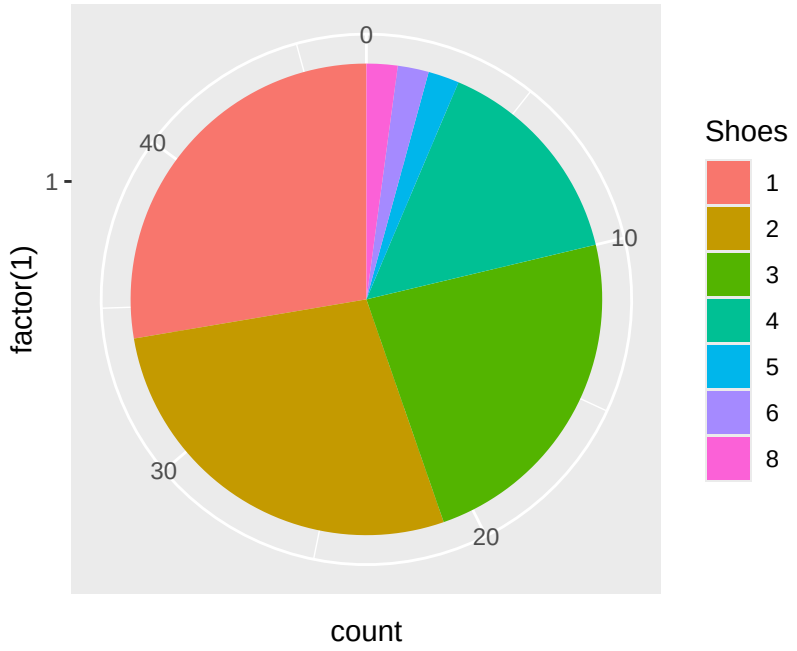
2.8.10 Histogram – Age

```
histo <- ggplot(chap1data1,aes(x=Age))
histo + geom_histogram(binwidth=6)
```



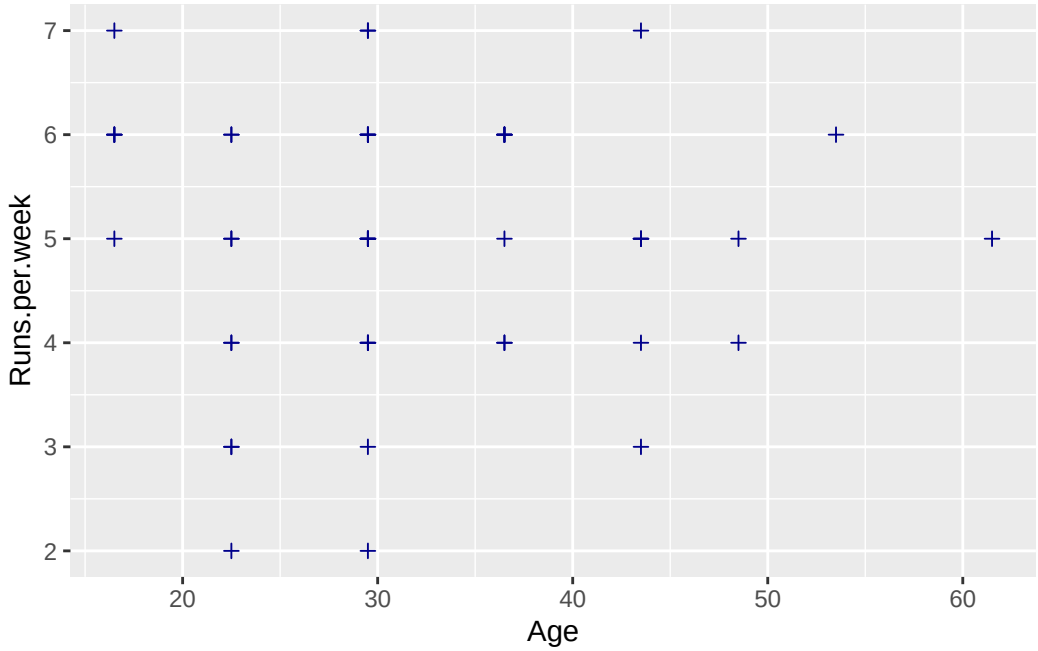
2.8.11 Pie chart – Shoes

```
pie <- ggplot(data=chap1data1,aes(x=factor(1),fill=Shoes))+
geom_bar(width = 1)
pie + coord_polar("y")
```



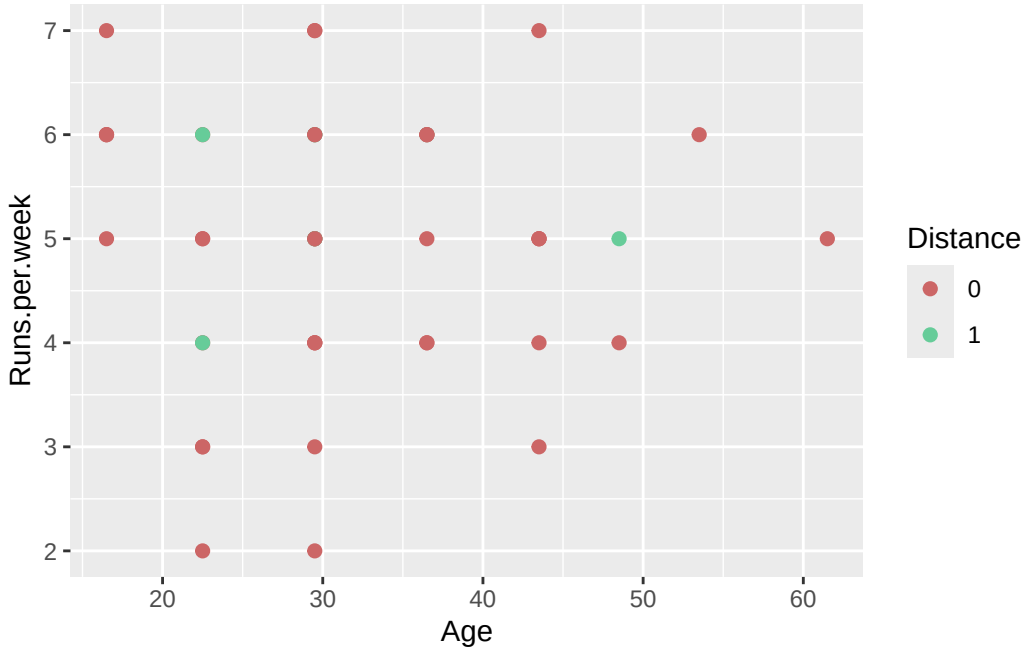
2.8.12 Scatter plot – Runs.per.week vs Age

```
ggplot(data=chap1data1,aes(y=Runs.per.week,x=Age)) +
geom_point(shape=3,color="darkblue")
```



2.8.13 Scatter plot – Runs.per.week vs Age by Distance

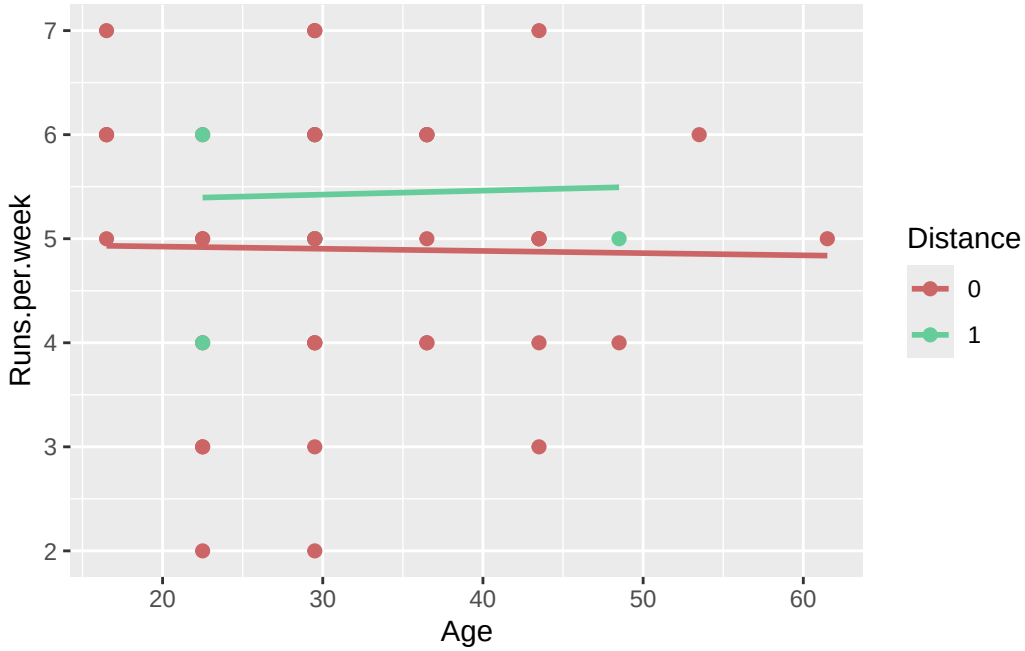
```
scatter <-
ggplot(data=chap1data1,aes(y=Runs.per.week,x=Age,color=Distance))
scatter + geom_point(size=2) +
scale_color_manual(values=c("#CC6666","#66CC99"))
```



2.8.14 Scatter plot – Runs.per.week vs Age by Distance (with trend lines)

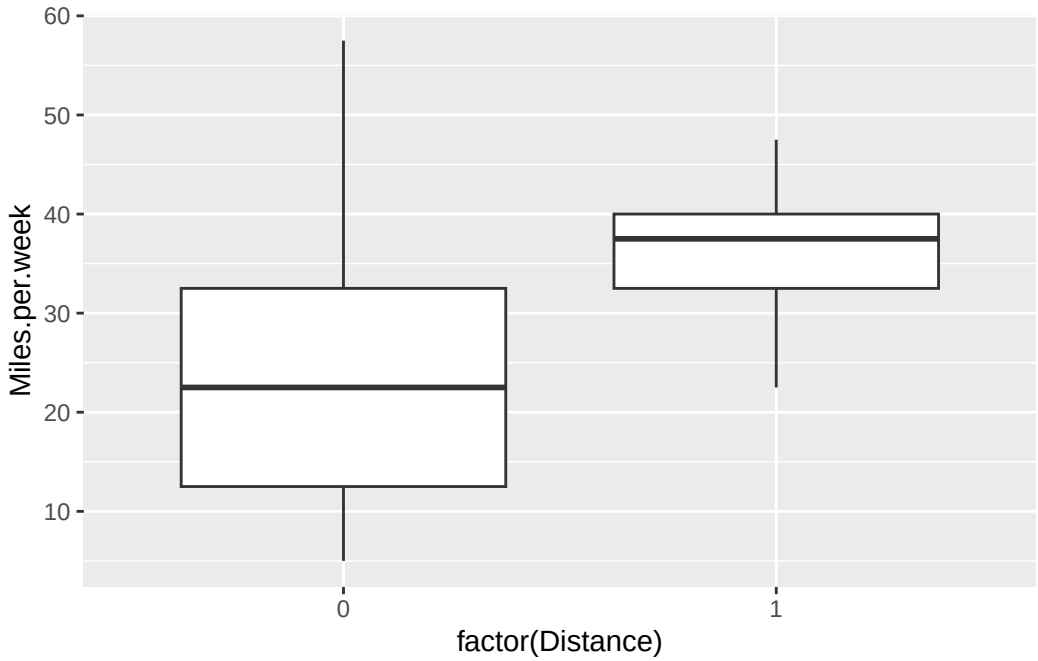
```
scatter2 <-
ggplot(data=chap1data1,aes(y=Runs.per.week,x=Age,color=Distance)) +
geom_point(size=2) +
scale_color_manual(values=c("#CC6666","#66CC99")) +
geom_smooth(method=lm,se=F)

scatter2
```

```
`geom_smooth()` using formula = 'y ~ x'
```



2.8.15 Box plot – Miles.per.week by Distance

```
ggplot(chap1data1,aes(factor(Distance),Miles.per.week)) +
geom_boxplot()
```



2.9 Exporting data visualisations

The `scatter2` object is used to demonstrate how to export a visualisation.

```
# Save the scatter2 plot to a PNG file
ggsave(
  filename = "scatter2_plot.png",  # name of the output file
  plot = scatter2,                 # the ggplot object to save
  width = 8,                       # width in inches
  height = 6,                      # height in inches
  dpi = 300                        # resolution (dots per inch)
)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
# Save the scatter2 plot to a PDF file
# This creates a vector graphic that scales well, ideal for reports or print.
ggsave("scatter2_plot.pdf", plot = scatter2, width = 8, height = 6)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

3. Exercises

Exercise 1

For the data in C&S Example 2.1 (p. 11) (chap1exer1.csv), calculate the following statistics regarding the sample plant height.

- Mean
- Median
- Variance
- Standard deviation
- Standard error of the mean
- Coefficient of variation

You can also try to import the data with the following function: "matrix(c(14.8,15.2,17.4,11.6,12.5),ncol=1)"

Exercise 2

See if you can import the data below with the following function: data.frame(Yield=c(8.1,8.7,9.2,7.8,8.4,9.4))

- Report the mean, median and mode.
- Report the variance and standard deviation.
- Produce a line plot.

Exercise 3

The data in C&S Example 3.2 (p. 18) (chap1exer3.csv), presents the yields (g) of small equal-sized plots of barley. Determine the following:

- mean
- mimimum and maximum
- standard deviation
- range
- first and third quartiles
- interquartile range and also
- produce a histogram
- produce a box plot.