Introduction to R programming:

R is a programming language and free software developed by Ross Ihaka and Robert Gentleman in 1993. R possesses an extensive catalog of statistical and graphical methods. It includes machine learning algorithms, linear regression, time series, statistical inference to name a few. Most of the R libraries are written in R, but for heavy computational tasks, C, C++ and Fortran codes are preferred. R is not only entrusted by academic, but many large companies also use R programming language, including Uber, Google, Airbnb, Facebook and so on.

Data analysis with R is done in a series of steps; programming, transforming, discovering, modeling and communicate the results.
Program: R is a clear and accessible programming tool

Transform: R is made up of a collection of libraries designed specifically for data science

Discover: Investigate the data, refine your hypothesis and analyze them

Model: R provides a wide array of tools to capture the right model for your data

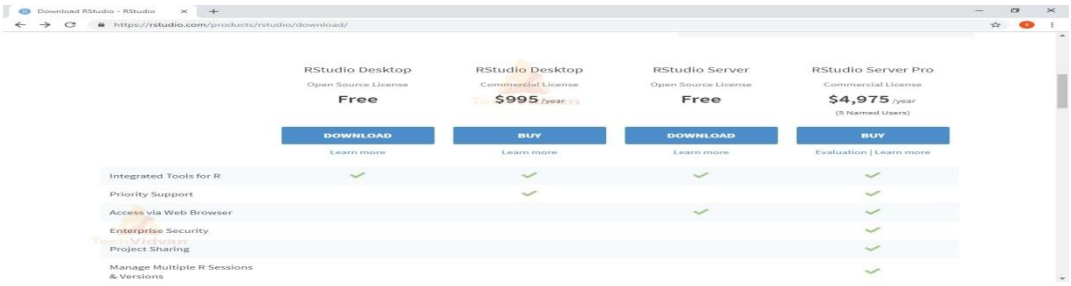
Communicate: Integrate codes, graphs, and outputs to a report with R Markdown or build Shiny apps to share with the world

What is R used for?

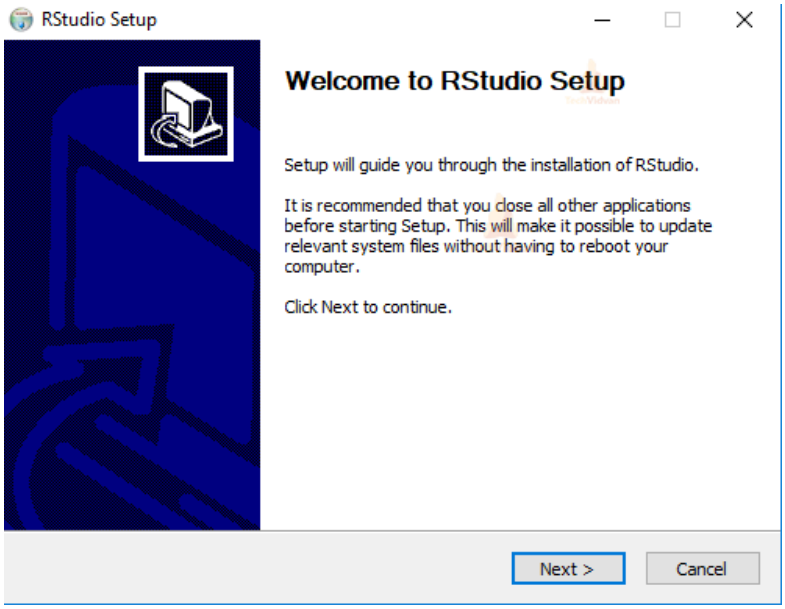
- Statistical inference
- Data analysis
- Machine learning algorithm

Installation of R-Studio on windows:

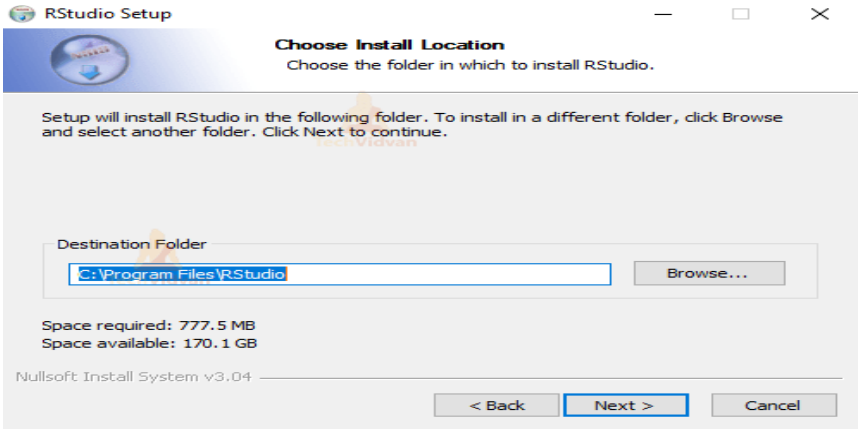
Step – 1: With R-base installed, let's move on to installing RStudio. To begin, goto download RStudio and click on the download button for RStudio desktop.



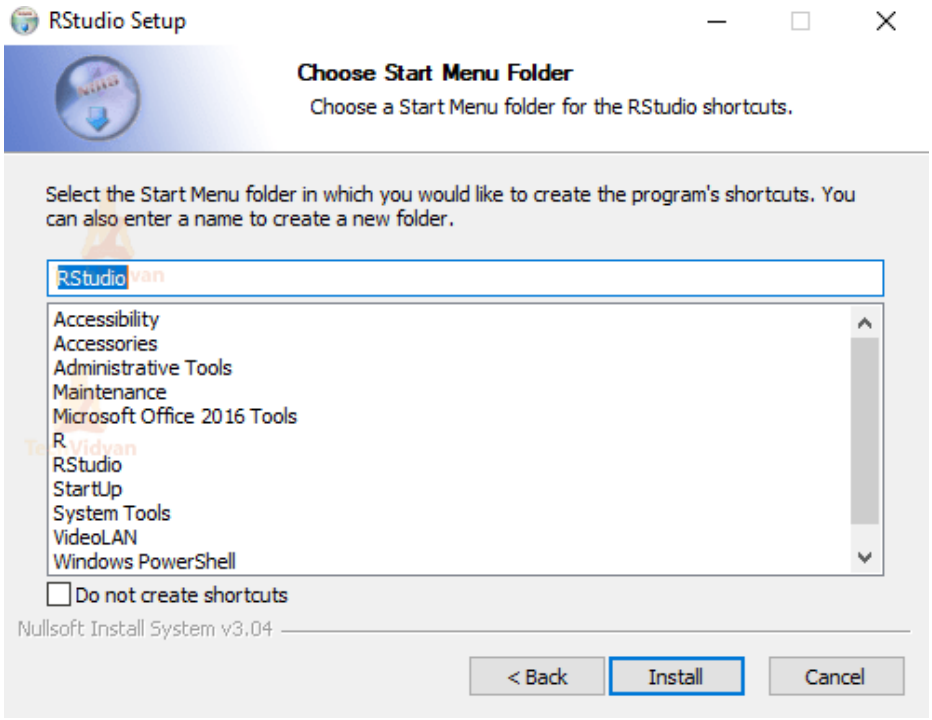
- Step – 2: Click on the link for the windows version of RStudio and save the .exe file.
- Step – 3: Run the .exe and follow the installation instructions.
- 3. Click Next on the welcome window.



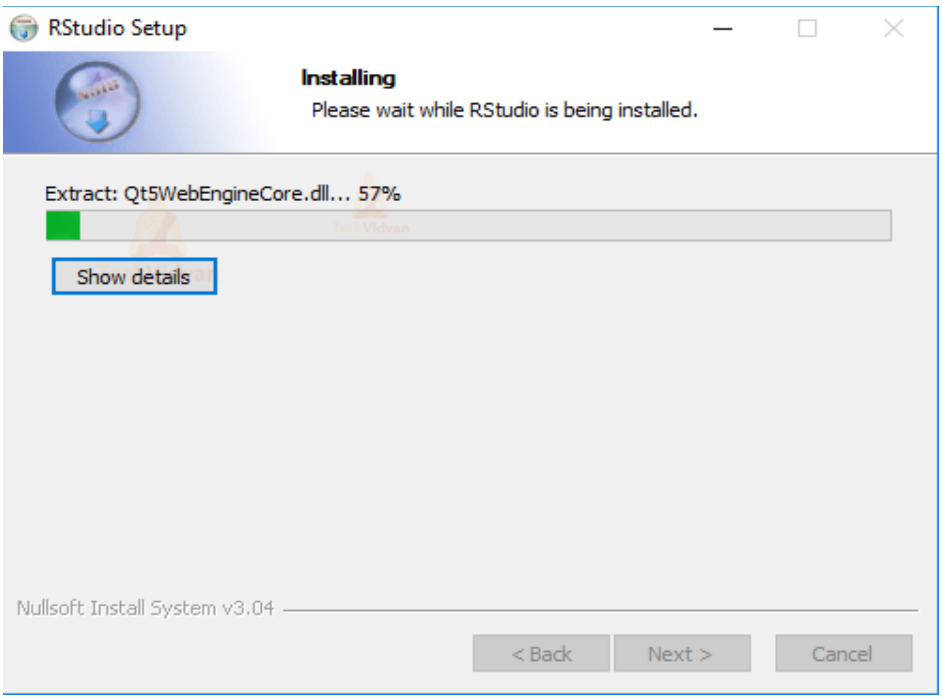
- Enter/browse the path to the installation folder and click Next to proceed.



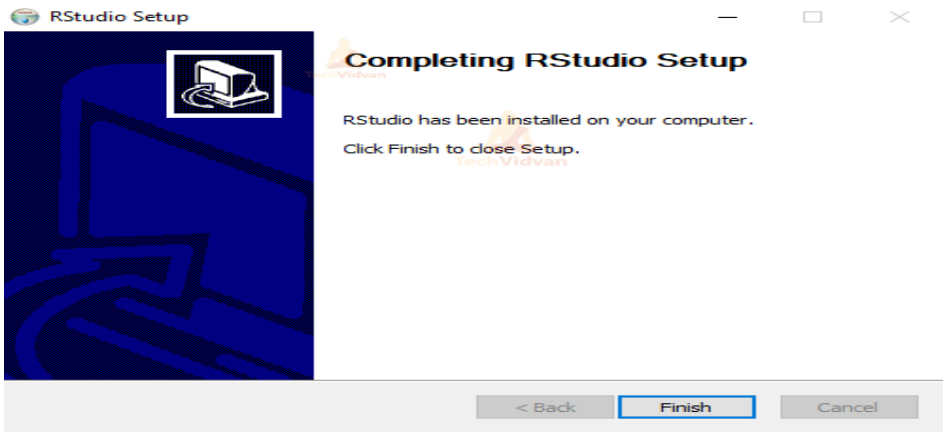
Select the folder for the start menu shortcut or click on do not create shortcuts and thenclick Next.



Wait for the installation process to complete.



Click Finish to end the installation.



Install the R Packages:-

In RStudio, if you require a particular library, then you can go through the following instructions:

- First, run R Studio.
- After clicking on the packages tab, click on install. The following dialog box will appear.
- In the Install Packages dialog, write the package name you want to install under the Packages field and then click install. This will install the package you searched for or give you a list of matching packages based on your package text.

Installing Packages:-

The most common place to get packages from is CRAN. To install packages from CRAN you use install.packages("package name"). For instance, if you want to install the ggplot2 package, which is a very popular visualization package, you would type the following in the console:-

Syntax:-

```
# install package from CRAN
install.packages("ggplot2")
```

Loading Packages:-

Once the package is downloaded to your computer you can access the functions and resources provided by the package in two different ways:

```
# load the package to use in the current R session
library(packagename)
```

Getting Help on Packages:-

For more direct help on packages that are installed on your computer you can use the help and vignette functions. Here we can get help on the ggplot2 package with the following:

```
help(package = "ggplot2")            # provides details regarding contents of a package
vignette(package = "ggplot2")        # list vignettes available for a specific package
vignette("ggplot2-specs")            # view specific vignette
vignette()                           # view all vignettes on your computer
```

Assignment Operators:-

The first operator you'll run into is the assignment operator. The assignment operator is used to assign a value. For instance we can assign the value 3 to the variable x using the <- assignment operator.

```
# assignment
x <- 3
```

Interestingly, R actually allows for five assignment operators:

```
# leftward assignment
x <- value
x = value
x <<- value
#
rightwardassignment
value -> x
value ->> x
```

The original assignment operator in R was <- and has continued to be the preferred among R users. The = assignment operator was added in 2001 primarily because it is the accepted assignment operator in many other languages and beginners to R coming from other languages were so prone to use it.

The operators <<- is normally only used in functions which we will not get into the details.

Evaluation

We can then evaluate the variable by simply typing x at the command line which will return the value of x. Note that prior to the value returned you'll see ## [1] in the command line. This simply implies that the output returned is the first output. Note that you can type any comments in your code by preceding the comment with the hash tag (#) symbol. Any values, symbols, and texts following # will not be evaluated.

```
# evaluation
x
## [1] 3
```

Case Sensitivity

Lastly, note that R is a case sensitive programming language. Meaning all variables, functions, and objects must be called by their exact spelling:

```
x <- 1
y <- 3
z <- 4
x * y * z
## [1] 12
x * Y * z
## Error in eval(expr, envir, enclos): object 'Y' not found
```

Basic Arithmetic

At its most basic function R can be used as a calculator. When applying basic arithmetic, the PEMDAS order of operations applies: parentheses first followed by exponentiation, multiplication and division, and final addition and subtraction.

```
8 + 9 / 5 ^ 2
```

```
## [1] 8.36

8 + 9 / (5 ^ 2)
## [1] 8.36
8 + (9 / 5) ^ 2
## [1] 11.24
(8 + 9) / 5 ^ 2
## [1] 0.68
```

By default R will display seven digits but this can be changed using options() as previously outlined.

```
1 / 7
## [1] 0.1428571
options(digits = 3)
1 / 7
## [1] 0.143
pi
## [1] 3.141592654
options(digits = 22)
pi
## [1] 3.141592653589793115998
```

We can also perform integer divide (%/%) and modulo (%%) functions. The integer divide function will give the integer part of a fraction while the modulo will provide the remainder.

```
42 / 4                  # regular division
## [1] 10.5
42 %/% 4                # integer division
## [1] 10
42 %% 4                 # modulo (remainder)
## [1] 2
```

Miscellaneous Mathematical Functions

There are many built-in functions to be aware of. These include but are not limited to the following. Go ahead and run this code in your console.

```
x <- 10
abs(x)    # absolute value
sqrt(x)   # square root
exp(x)    # exponential transformation
log(x)    # logarithmic transformation
cos(x)    # cosine and other trigonometric functions
```

Download the data set

Before we get rolling with the EDA, we want to download our data set. For this example, we are going to use the dataset produced by this recent science, technology, art and math (STEAM) project.

```
#Load the readr library to
bring in the dataset
library(readr)
```

#Download the data set
df=read_csv('https://raw.githubusercontent.com/lgellis/STEM/master/DATA-ART- 1/Data/FinalData.csv', col_names = TRUE)

Now that we have the data set all loaded, and it's time to run some very simple commands to preview the data set and it's structure.

ID <int>	Gender <chr>	Grade <int>	Horoscope <chr>	Subject <chr>	IntExt <chr>	OptPest <chr>	ScreenTime <dbl>	Sleep <dbl>	PhysActive <int>
1	male	4	Scorpio	Math	Extravert	Optimist	1	7	10
2	female	4	Capricorn	Gym	Extravert	Optimist	1	8	5
3	male	4	Taurus	Math	Introvert	Optimist	4	9	22
4	male	4	Aquarius	Math	Don't Know	Don't Know	3	9	9
5	male	4	Scorpio	Gym	Don't Know	Don't Know	1	9	10
6	male	4	Pisces	Gym	Extravert	Optimist	2	9	20
7	male	3	Scorpio	Art	Introvert	Optimist	1	11	4
8	male	6	Taurus	Math	Extravert	Optimist	4	9	12
9	male	6	Aries	Gym	Introvert	Pessimist	6	8	4
10	male	6	Pisces	Math	Introvert	Don't Know	3	9	12

1–10 of 10 rows | 1–10 of 17 columns

Head:-
To begin, we are going to run the head function, which allows us to see the first 6 rows by default. We are going to override the default and ask to preview the first 10 rows.
>head(df, 10)

Tail:-Tail function allows us to see the last n observations from a given data frame. The defult value for n is 6. User can specify value of n as per as requirements.

>tail(mtcars,n=5)

```
Observations: 185
Variables: 17
$ ID          <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 2...
$ Gender      <chr> "male", "female", "male", "male", "male", "male", "male", "male", "male", "male", "f...
$ Grade       <int> 4, 4, 4, 4, 4, 4, 3, 6, 6, 6, 4, 4, 4, 7, 8, 8, 8, 8, 8, 8, 5, 6, 5, 6, 6, 6, 6, 6, ...
$ Horoscope   <chr> "Scorpio", "Capricorn", "Taurus", "Aquarius", "Scorpio", "Pisces", "Scorpio", "Tauru...
$ Subject     <chr> "Math", "Gym", "Math", "Math", "Gym", "Gym", "Art", "Math", "Gym", "Math", "Gym", "M...
$ IntExt      <chr> "Extravert", "Extravert", "Introvert", "Don't Know", "Don't Know", "Extravert", "Int...
$ OptPest     <chr> "Optimist", "Optimist", "Optimist", "Don't Know", "Don't Know", "Optimist", "Optimis...
$ ScreenTime  <dbl> 1, 1, 4, 3, 1, 2, 1, 4, 6, 3, 1, 1, 0, 5, 6, 5, 8, 4, 2, 3, 2, 1, 1, 4, 1, 3, 1, 3, ...
$ Sleep       <dbl> 7, 8, 9, 9, 9, 9, 11, 9, 8, 9, 10, 10, 9, 8, 9, 7, 7, 8, 9, 8, 9, 8, 9, 9, 8, 6, 10,...
$ PhysActive  <int> 10, 5, 22, 9, 10, 20, 4, 12, 4, 12, 5, 5, 5, 14, 25, 6, 2, 10, 10, 10, 2, 5, 1, 6, 7...
$ HrsHomework <dbl> 10, 0, 1, 1, 1, 2, 14, 21, 6, 3, 0, 0, 0, 4, 2, 3, 0, 0, 3, 5, 1, 2, 8, 4, 4, 3, 2, ...
$ SpendTime1  <chr> "baseball", "playing outside", "video games", "video games", "reading", "sports", "w...
$ SpendTime2  <chr> "relaxing", "swimming", "soccer", "sports", "hanging out", "playing with friends", "...
$ Self1       <chr> "active", "kind", "active", "active", "intellegent", "funny", "joyful", "fun", "sad"...
$ Self2       <chr> "competitive", "active", "creative", "responsible", "strong", "active", "lazy", "con...
$ Career      <chr> "professional baseball player", "Teacher", "professional soccer player", "profession...
$ Superpower  <chr> "sonic speed", "power to grant wishes", "powerful kick", "teleportaion", "power to a...
```

Dim and Glimpse

Next, we will run the dim function which displays the dimensions of the table. The output takes theform of row, column.And then we run the glimpse function from the dplyr package. This will display a vertical previewof the dataset. It allows us to easily preview data type and sample data.

dim(df)
#Displays the type and a preview of all columns as a row so that it's very easy to
take in.library(dplyr)
glimpse(df)

```
       ID          Gender               Grade        Horoscope           Subject             IntExt
 Min.   :  1   Length:185         Min.   :3.000   Length:185         Length:185         Length:185
 1st Qu.: 47   Class :character   1st Qu.:5.000   Class :character   Class :character   Class :character
 Median : 93   Mode  :character   Median :6.000   Mode  :character   Mode  :character   Mode  :character
 Mean   : 93                      Mean   :5.735
 3rd Qu.:139                      3rd Qu.:7.000
 Max.   :185                      Max.   :8.000

   OptPest            ScreenTime        Sleep          PhysActive     HrsHomework     SpendTime1
 Length:185         Min.   : 0.000   Min.   : 2.000   Min.   : 0.00   Min.   : 0.00   Length:185
 Class :character   1st Qu.: 1.000   1st Qu.: 8.000   1st Qu.: 6.00   1st Qu.: 1.00   Class :character
 Mode  :character   Median : 3.000   Median : 9.000   Median : 9.00   Median : 3.00   Mode  :character
                    Mean   : 2.997   Mean   : 8.641   Mean   :11.52   Mean   : 4.17
                    3rd Qu.: 4.000   3rd Qu.:10.000   3rd Qu.:12.00   3rd Qu.: 6.00
                    Max.   :18.000   Max.   :12.000   Max.   :82.00   Max.   :35.00
                                                      NA's   :1
  SpendTime2           Self1              Self2             Career           Superpower
 Length:185         Length:185         Length:185         Length:185         Length:185
 Class :character   Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character
```

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are –

R Objects:-

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

Creating and Manipulating Objects:-

Vectors:-

R programming, the very basic data types are the R-objects called vectors which hold elements of different classes as shown above. Please note in R the number of classes is not confined to only the above six types. For example, we can use many atomic vectors and create an array whose class will become array.

When you want to create vector with more than one element, you should use c() function which means to combine the elements into a vector.

```
# Create a vector.
apple <- c('red','green',"yellow")
print(apple)

# Get the class of the vector.
print(class(apple))
```

When we execute the above code, it produces the following result −

```
[1] "red"    "green"  "yellow"
[1] "character"
```

R has five basic or “atomic” classes of objects:

• character

• numeric (real numbers)

• integer

• complex

• logical (True/False)

Lists:

A list allows you to store a variety of objects.

```
> mylist <- list(x, y, z, gender, mydata)
> mylist
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] 1 3 5 7 9

[[3]]
[1] 1 2 5 4 7

[[4]]
[1] "m" "f" "m" "m" "f"

[[5]]
  x y z gender
1 1 1 1      m
2 2 3 2      f
3 3 5 5      m
4 4 7 4      m
5 5 9 7      f
```

You can use subscripts to select the specific component of the list.

```
> mylist[[3]]
[1] 1 2 5 4 7
```

> x <- list(1:3, TRUE, "Hello", list(1:2, 5))

Here x has 4 elements: a numeric vector, a logical, a string and another list.

We can select an entry of x with double square brackets:

> x[[3]]

[1] "Hello"

To get a sub-list, use single brackets:

> x[c(1,3)]

[[1]]

[1] 1 2 3

[[2]]

[1] "Hello"

Notice the difference between x[[3]] and x[3].

We can also name some or all of the entries in our list, by supplying argument names to list().

Matrices:-

Matrices are much used in statistics, and so play an important role in R. To create a matrix use the function matrix(), specifying elements by column first:

> matrix(1:12, nrow=3, ncol=4)

[,1] [,2] [,3] [,4]

[1,] 1 4 7 10

[2,] 2 5 8 11

[3,] 3 6 9 12

This is called column-major order. Of course, we need only give one of the dimensions:

> matrix(1:12, nrow=3)

unless we want vector recycling to help us:

> matrix(1:3, nrow=3, ncol=4)

[,1] [,2] [,3] [,4]

[1,] 1 1 1 1

[2,] 2 2 2 2

[3,] 3 3 3 3

Sometimes it’s useful to specify the elements by row first

> matrix(1:12, nrow=3, byrow=TRUE)

There are special functions for constructing certain matrices:

> diag(3)

[,1] [,2] [,3]

[1,] 1 0 0

[2,] 0 1 0

[3,] 0 0 1

> diag(1:3)

[,1] [,2] [,3]

[1,] 1 0 0

[2,] 0 2 0

[3,] 0 0 3

> 1:5 %o% 1:5

[,1] [,2] [,3] [,4] [,5]

[1,] 1 2 3 4 5

[2,] 2 4 6 8 10

[3,] 3 6 9 12 15

[4,] 4 8 12 16 20

[5,] 5 10 15 20 25

The last operator performs an outer product, so it creates a matrix with (i, j)-th entry xiyj . The function outer() generalizes this to any function f on two arguments, to create a matrix with entries f(xi , yj). (More on functions later.)

> outer(1:3, 1:4, "+")

[,1] [,2] [,3] [,4]

[1,] 2 3 4 5

[2,] 3 4 5 6

[3,] 4 5 6 7

Array:

If we have a data set consisting of more than two pieces of categorical informationabout each subject, then a matrix is not sufficient. The generalization of matrices to higher

dimensions is the array. Arrays are defined much like matrices, with a call to the array() command. Here is a 2 × 3 × 3 array:

> arr = array(1:18, dim=c(2,3,3))

> arr

, , 1

[,1] [,2] [,3]

[1,] 1 3 5

[2,] 2 4 6

, , 2

[,1] [,2] [,3]

[1,] 7 9 11

[2,] 8 10 12

, , 3

[,1] [,2] [,3]

[1,] 13 15 17

[2,] 14 16 18

Each 2-dimensional slice defined by the last co-ordinate of the array is shown as a 2 × 3 matrix. Note that we no longer specify the number of rows and columns separately, but use a single vector dim whose length is the number of dimensions. You can recover this vector with the dim() function.

> dim(arr)

[1] 2 3 3

Note that a 2-dimensional array is identical to a matrix. Arrays can be

subsetted and modified in exactly the same way as a matrix, only using the appropriate number of co-ordinates:

```
> arr[1,2,3]
[1] 15
> arr[,2,]
```

[,1] [,2] [,3]

[1,] 3 9 15

[2,] 4 10 16

> arr[1,1,] = c(0,-1,-2) # change some values

> arr[,,1,drop=FALSE]

, , 1

[,1] [,2] [,3]

[1,] 0 3 5

[2,] 2 4 6

Factors:-

R has a special data structure to store categorical variables. It tells R that a variable is nominal or ordinal by making it a factor.

Simplest form of the factor function :

```
> gender <- c(1,2,1,2,1,2,1,2)
> gender <- factor(gender)
```

Ideal form of the factor function :

```
> gender <- c(1,2,1,2,1,2,1,2)
>
> gender <- factor(gender,
+ levels = c(1,2),
+ labels = c("male","female"))

> table(gender)
gender
  male female
     4      4
```

The factor function has three parameters:
1. Vector Name
2. Values (Optional)
3. Value labels (Optional)

Convert a column "x" to factor

data$x = as.factor(data$x)

Data Frames:-

A data frame is a table or a two-dimensional array-like structure in which each columncontains values of one variable and each row contains one set of values from each column.

- Data frames are tabular data objects.
- A Data frame is a list of vector of equal length.
- Data frame in R is used for storing data tables.

Characteristics of a data frame:

1. The column names should be non-empty.
2. The row names should be unique.
3. The data stored in a data frame can be of numeric, factor or character type.

Create Data Frame

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),
      start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-
11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)# Print the data frame.
print(emp.data)
```

When we execute the above code, it produces the following result –

emp_id	emp_name	salary	start_date
1	Rick	623.30	2012-01-01
2	Dan	515.20	2013-09-23
3	Michelle	611.00	2014-11-15
4	Ryan	729.00	2014-05-11
5	Gary	843.25	2015-03-27

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name =
   c("Rick","Dan","Michelle","Ryan","Gary"),salary =
   c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Print the summary.
print(summary(emp.data))
```

Get the Structure of the Data Frame

The structure of the data frame can be seen by using str() function.

```
# Create the data
frame.emp.data <-
data.frame(
   emp_id = c (1:5),
   emp_name =
   c("Rick","Dan","Michelle","Ryan","Gary"),salary
   = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
```

When we execute the above code, it produces the following result –

```
'data.frame':   5 obs. of  4 variables:
 $ emp_id    : int  1 2 3 4 5
 $ emp_name  : chr  "Rick" "Dan" "Michelle" "Ryan" ...
 $ salary    : num  623 515 611 729 843
 $ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
```

Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying summary()function.

When we execute the above code, it produces the following result −

```
 Median :3   Mode  :character   Median :623.3   Median :2014-05-11
 Mean   :3                      Mean   :664.4   Mean   :2014-01-14
 3rd Qu.:4                      3rd Qu.:729.0   3rd Qu.:2014-11-15
 Max.   :5                      Max.   :843.2   Max.   :2015-03-27
```

Expand Data Frame

A data frame can be expanded by adding columns and rows.0

1.Add Column:-Just add the column vector using a new column name.

2.Add Row:-To add more rows permanently to an existing data frame, we need to bring in the new rowsin the same structure as the existing data frame and use the rbind() function.

```
# Add the "dept" coulmn.

emp.data$dept <- c("IT","Operations","IT","HR","Finance")

v <- emp.data

print(v)
```

When we execute the above code, it produces the following result −

	emp_id	emp_name	salary	start_date	dept
1	1	Rick	623.30	2012-01-01	IT
2	2	Dan	515.20	2013-09-23	Operations
3	3	Michelle	611.00	2014-11-15	IT
4	4	Ryan	729.00	2014-05-11	HR
5	5	Gary	843.25	2015-03-27	Finance

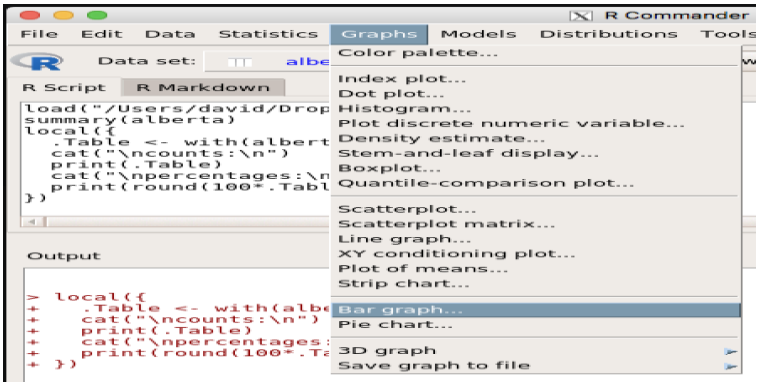
.

In the example below we create a data frame with new rows and merge it with the existingdata frame to create the final data frame.

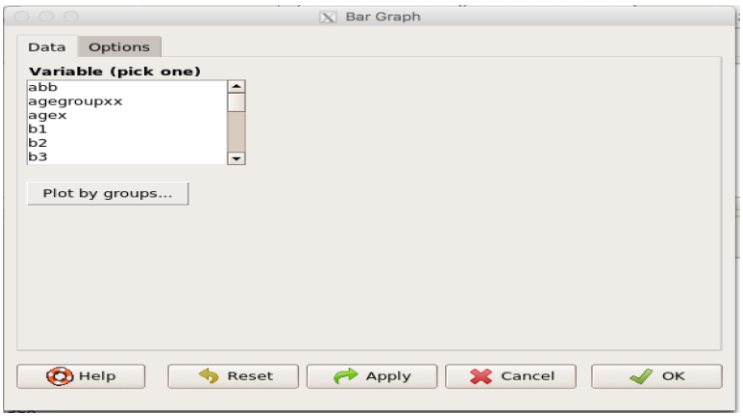
```
# Create the second data
frameemp.newdata <-
                    data.frame
(
  emp_id = c (6:8),
  emp_name =
  c("Rasmi","Pranab","Tusar"),salary =
  c(578.0,722.5,632.8),
  start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
  dept =
  c("IT","Operations","Fianance"),
  stringsAsFactors = FALSE
)
```

Charts

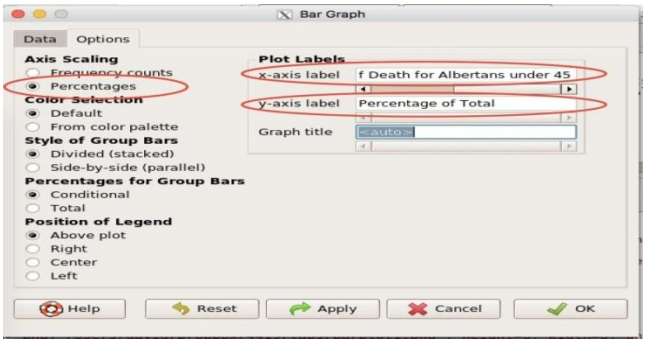
A bar chart is a way of summarizing a set of categorical data. It displays data using a number of rectangles of the same width, each of which represents a particular category. The length of each rectangle is proportional to the number of cases in the category it represents. Below, we will make a figure of the question "To thebest of your knowledge, what is the leading cause of death for Albertans under the age of 45?", question g1in the survey. You can make a bar chart in the R Commander by choosing Graphs Bar Graph from the R Commander menus.



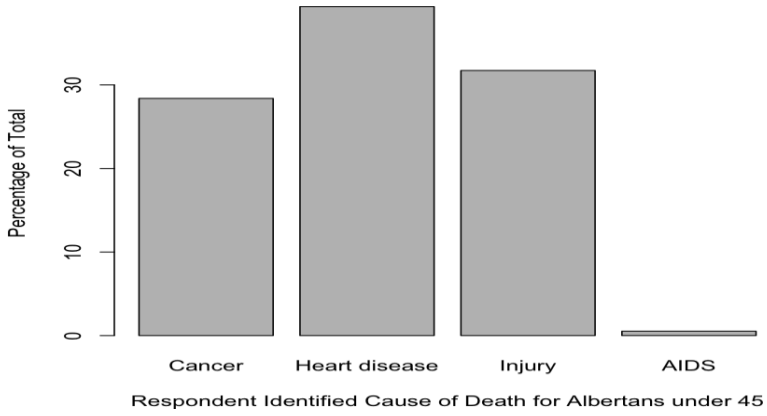
That will open up a dialog box that looks like the following:



You can also choose di↵erent options for how the figure is constructed (e.g., frequencies or percentages)by clicking on the percentages tab, which switches the dialog to the one below:



That should produce a bar plot that looks like the following:



Looping in R:-

"Looping", "cycling", "iterating" or just replicating instructions is an old practice that originated well before the invention of computers. It is nothing more than automating a multi-step process by organizing sequences of actions or 'batch' processes and by grouping the parts that need to be repeated.

All modern programming languages provide special constructs that allow for the repetition of instructions or blocks of instructions.

According to the R base manual, among the control flow commands, the loop constructs are for, while and repeat, with the additional clauses break and next.Remember that control flow commands are the commands that enable a program to branch between alternatives, or to "take decisions", so to speak.

If the condition is not met and the resulting outcome is False, the loop is never executed. This is indicated by the loose arrow on the right of the for loop structure. The program will then execute the first instruction found after the loop block.

If the condition is verified, an instruction -or block of instructions- i1 is executed. And perhaps this block of instructions is another loop. In such cases, you speak of a nested loop.

The initialization statement describes the starting point of the loop, where the loop variable is initialized with a starting value. A loop variable or counter is simply a variable that controls the flow of the loop. The test expression is the condition until when the loop is repeated

Syntax of for loop:-

```
for (val in sequence)

{

statement

}
```

Here, sequence is a vector and val takes on each of its value during the loop. In each iteration, statement is evaluated.

Flowchart of for loop

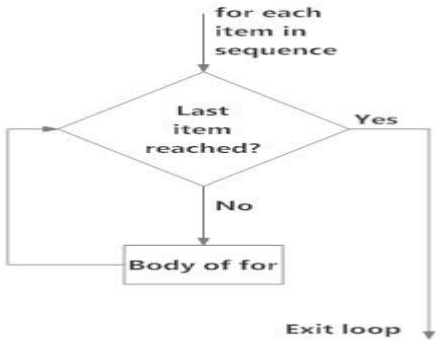


Fig: operation of for loop

Example: for loop

Below is an example to count the number of even numbers in a vector.

```
x <- c(2,5,3,9,8,11,6)

count <- 0

for (val in x) {
```

```
if(val %% 2 == 0)  count = count+1

}

print(count)
```

Output

```
[1] 3
```

In the above example, the loop iterates 7 times as the vector `x` has 7 elements.

In each iteration, `val` takes on the value of corresponding element of `x`.

We have used a counter to count the number of even numbers in `x`. We can see that `x` contains 3 even numbers.

R while Loop

Loops are used in programming to repeat a specific block of code. In this article, you will learn to create a while loop in R programming.In R programming, while loops are used to loop until a specific condition is met.

Syntax of while loop

```
while (test_expression)

{

statement

}
```

Here, `test_expression` is evaluated and the body of the loop is entered if the result is `TRUE`.

The statements inside the loop are executed and the flow returns to evaluate the `test_expression` again.

This is repeated each time until `test_expression` evaluates to `FALSE`, in which case, the loop exits.

Flowchart of while Loop

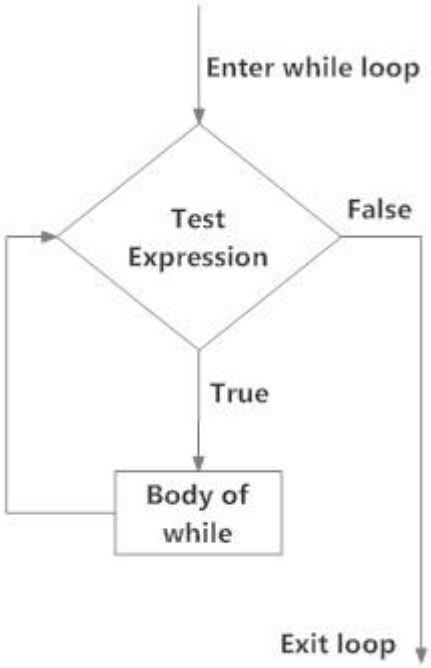


Fig: operation of while loop

Visualise the data using Bar chart anf box plot:-

A bar chart is a pictorial representation of data that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. In other words, it is the pictorial representation of dataset. These data sets contain the numerical values of variables that represent the length or height.

R uses the function barplot() to create bar charts. Here, both vertical and Horizontal bars can be drawn.

Syntax:
barplot(H, xlab, ylab, main, names.arg, col)

Parameters:

- H: This parameter is a vector or matrix containing numeric values which are used in bar chart.
- xlab: This parameter is the label for x axis in bar chart.
- ylab: This parameter is the label for y axis in bar chart.
- main: This parameter is the title of the bar chart.
- names.arg: This parameter is a vector of names appearing under each bar in bar chart.
- col: This parameter is used to give colors to the bars in the graph.

Creating a Simple Bar Chart

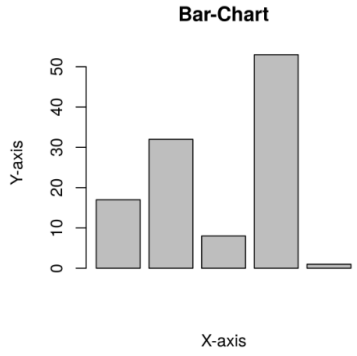
1. A vector (H <- c(Values…)) is taken which contain numeral values to be used.
2. This vector H is plot using barplot().

Example:

Create the data for the chart

```
A <- c(17, 32, 8, 53, 1)


# Plot the bar chart

barplot(A, xlab = "X-axis", ylab = "Y-axis", main ="Bar-Chart")
```

Output:



Horizontal Bar Chart:-

Creating a Horizontal Bar Chart

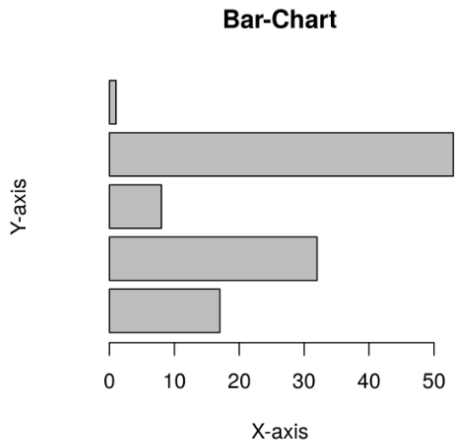
Approach: To create a horizontal bar chart:

1. Take all parameters which are required to make simple bar chart.
2. Now to make it horizontal new parameter is added.
   barplot(A, horiz=TRUE)

Example: Creating a horizontal bar chart

```
# Create the data for the chart

A <- c(17, 32, 8, 53, 1)


# Plot the bar chart

barplot(A, horiz = TRUE, xlab = "X-axis",

      ylab = "Y-axis", main ="Bar-Chart")
```

Output:



Adding Label, Title and Color in the BarChart

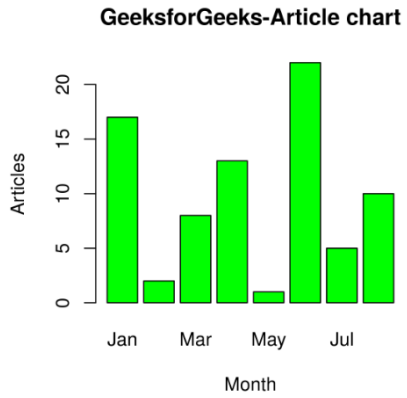
Label, title and colors are some properties in the bar chart which can be added to the bar by adding and passing an argument.
Approach:

1. To add the title in bar chart.
   barplot(A, main = title_name)
2. X-axis and Y-axis can be labeled in bar chart. To add the label in bar chart.
   barplot(A, xlab= x_label_name, ylab= y_label_name)
3. To add the color in bar chart.
   barplot(A, col=color_name)

Example :

```
# Create the data for the chart

A <- c(17, 2, 8, 13, 1, 22)

B <- c("Jan", "feb", "Mar", "Apr", "May", "Jun")



# Plot the bar chart

barplot(A, names.arg = B, xlab ="Month",

    ylab ="Articles", col ="green",

    main ="GeeksforGeeks-Article chart")
```

Output:



Histograms in R language:-
A histogram contains a rectangular area to display the statistical information which is proportional to the frequency of a variable and its width in successive numerical intervals. A graphical representation that manages a group of data points into different specified ranges. It has a special feature which shows no gaps between the bars and is similar to a vertical bar graph.We can create histogram in R Programming Language using hist() function.

Syntax: hist(v, main, xlab, xlim, ylim, breaks, col, border)
Parameters:

- v: This parameter contains numerical values used in histogram.
- main: This parameter main is the title of the chart.
- col: This parameter is used to set color of the bars.
- xlab: This parameter is the label for horizontal axis.
- border: This parameter is used to set border color of each bar.
- xlim: This parameter is used for plotting values of x-axis.
- ylim: This parameter is used for plotting values of y-axis.
- breaks: This parameter is used as width of each bar.

Creating a simple Histogram in R

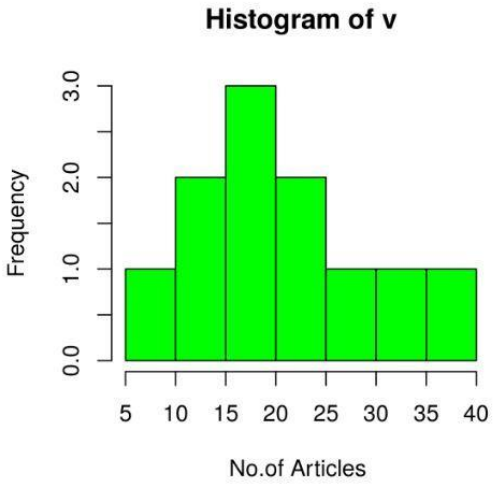
Creating a simple histogram chart by using the above parameter. This vector v is plot using hist().
Example:

- R

```
# Create data for the graph.

v <- c(19, 23, 11, 5, 16, 21, 32,

    14, 19, 27, 39)



# Create the histogram.

hist(v, xlab = "No.of Articles ",
```

col = "green", border = "black")

Output:



Range of X and Y values

To describe the range of values we need to do the following steps:

1. We can use the xlim and ylim parameter in X-axis and Y-axis.
2. Take all parameters which are required to make histogram chart.

Example

- R

Create data for the graph.

v <- c(19, 23, 11, 5, 16, 21, 32, 14, 19, 27, 39)

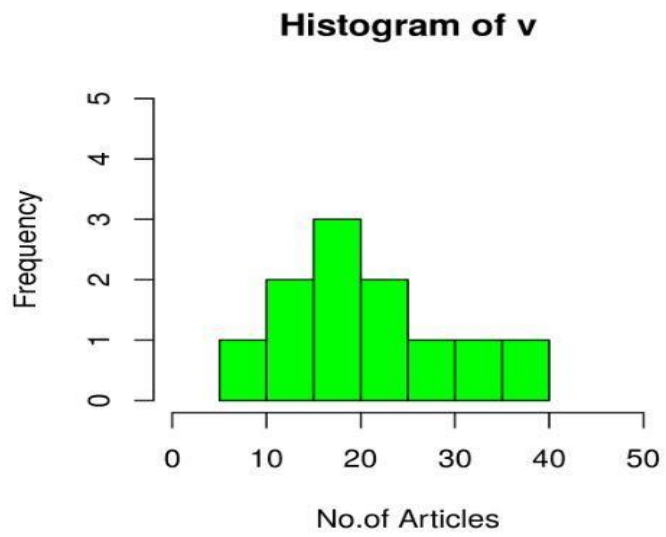
Create the histogram.

hist(v, xlab = "No.of Articles", col = "green",

border = "black", xlim = c(0, 50),

ylim = c(0, 5), breaks = 5)

Output:



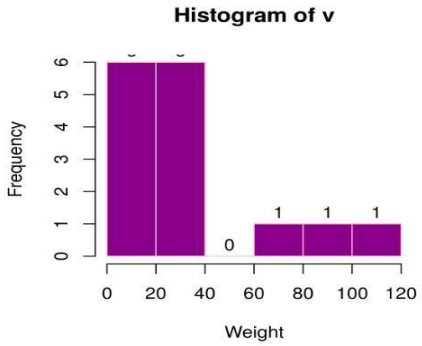
Using histogram return values for labels using text()

To create a histogram return value chart.

- R

```
# Creating data for the graph.
v <- c(19, 23, 11, 5, 16, 21, 32, 14, 19,
       27, 39, 120, 40, 70, 90)


# Creating the histogram.
m<-hist(v, xlab = "Weight", ylab ="Frequency",
        col = "darkmagenta", border = "pink",
        breaks = 5)


# Setting labels
text(m$mids, m$counts, labels = m$counts,
     adj = c(0.5, -0.5))
```

Output:



Boxplots in R Language:-
A box graph is a chart that is used to display information in the form of distribution by drawing boxplots for each of them. This distribution of data based on five sets (minimum, first quartile, median, third quartile, maximum).

Boxplots in R Programming Language
Boxplots are created in R by using the boxplot() function.

Syntax: boxplot(x, data, notch, varwidth, names, main)
Parameters:

- x: This parameter sets as a vector or a formula.
- data: This parameter sets the data frame.
- notch: This parameter is the label for horizontal axis.
- varwidth: This parameter is a logical value. Set as true to draw width of the box proportionate to the sample size.
- main: This parameter is the title of the chart.
- names: This parameter are the group labels that will be showed under each boxplot.

Creating a Dataset
To understand how we can create a boxplot:

- We use the data set “mtcars”.
- Let’s look at the columns “mpg” and “cyl” in mtcars.

- R

```
input <- mtcars[, c('mpg', 'cyl')]

print(head(input))
```

Output:

```
                   mpg cyl
Mazda RX4         21.0   6
Mazda RX4 Wag     21.0   6
Datsun 710        22.8   4
Hornet 4 Drive    21.4   6
Hornet Sportabout 18.7   8
Valiant           18.1   6
```

Creating the Boxplot

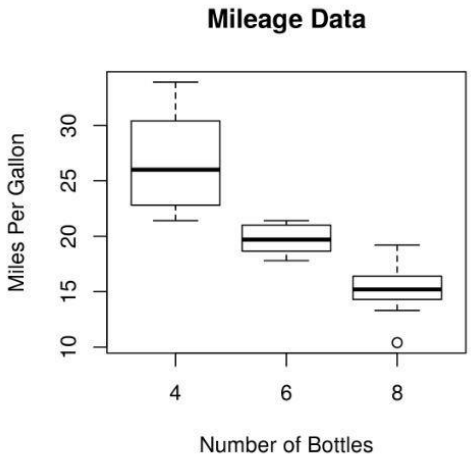
Creating the Boxplot graph.

- Take the parameters which are required to make boxplot.
- Now we draw a graph for the relation between “mpg” and “cyl”.

- R

```
# Plot the chart.

boxplot(mpg ~ cyl, data = mtcars,

    xlab = "Number of Cylinders",

    ylab = "Miles Per Gallon",

    main = "Mileage Data")
```

Output:



Scatter plot :-

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the plot() function.

Syntax

```
plot(x, y, main, xlab, ylab, xlim, ylim, axes)
```

Following is the description of the parameters used −

- x is the data set whose values are the horizontal coordinates.
- y is the data set whose values are the vertical coordinates.
- main is the tile of the graph.
- xlab is the label in the horizontal axis.
- ylab is the label in the vertical axis.
- xlim is the limits of the values of x used for plotting.
- ylim is the limits of the values of y used for plotting.
- axes indicates whether both axes should be drawn on the plot.

Example

We use the data set "mtcars" available in the R environment to create a basic scatterplot. Let's use the columns "wt" and "mpg" in mtcars.

```
input <- mtcars[,c('wt','mpg')]
print(head(input))
```

When we execute the above code, it produces the following result −

	wt	mpg
Mazda RX4	2.620	21.0
Mazda RX4 Wag	2.875	21.0
Datsun 710	2.320	22.8
Hornet 4 Drive	3.215	21.4
Hornet Sportabout	3.440	18.7
Valiant	3.460	18.1

Predictive modeling in R:-

In R programming, predictive models are extremely useful for forecasting future outcomes and estimating metrics that are impractical to measure. For example, data scientists could use predictive models to forecast crop yields based on rainfall and temperature, or to determine whether patients with certain traits are more likely to react badly to a new medication.

The lm() function estimates the intercept and slope coefficients for the linear model that it has fit to our data.

Whether we can use our model to make predictions will depend on:

1. Whether we can reject the null hypothesis that there is no relationship between our variables.
2. Whether the model is a good fit for our data.

The output of our model using summary(). The model output will provide us with the information we need to test our hypothesis and assess how well the model fits our data.