Data Science with R Programming

R is a programming language and software environment for statistical analysis, graphics representation and reporting.

Features of R

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

DAY - 1

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 –

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

Variables

In [1]:

```
Name <- "Sathish"
Age = 19
#Cat - cat converts its arguments to character strings, concatenates them and then p
cat("My Name is",Name,"and",Age,"years old")
```

```
My Name is Sathish and 19 years old
```

Dynamic User input

we can use readline() function to take input from the user (terminal).

This function will return a single element character vector.

In [2]:

```
name = readline("Enter you name:")
age = readline(prompt="Enter you age:")
cat("Your name is",name,"and you are",age,"years old")
```

```
Enter you name:Sathish
```

```
Enter you age:19
Your name is Sathish and you are 19 years old
```

Type Conversion

Adding a character string to a numeric vector converts all the elements in the vector to character.

In [3]:
```
age="18"
num_age=as.integer(age)
print(class(num_age))
```

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

MatrixSlicing

A matrix is a 2-dimensional array that has m number of rows and n number of columns. In other words, matrix is a combination of two or more vectors with the same data type.

In [4]:
```
#Matrix
marks = matrix( c('15','14','10','8','12','13','32',"34"), nrow = 2, ncol = 4, byrow
#Python Ca-2 marks
print(marks["Python","CA2"])
# FA Marks
print(marks[c(1,2),c(1,2,3)])
#Only R program
print(marks["R",c(1:4)])
```

```
[1] "14"
       CA1  CA2  CA3 
Python "15" "14" "10"
R      "12" "13" "32"
 CA1  CA2  CA3   FA 
"12" "13" "32" "34" 
```

Dataframes

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

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

In [5]:
```
name = c("Sathish","Karan","Sivant")
age = c(20,18,19)
class = c("AI","CS","ML")
df = data.frame(name,age,class)
print(df)
print(summary(df))
```

```
     name age class
1 Sathish  20    AI
2   Karan  18    CS
3  Sivant  19    ML
      name        age        class 
 Karan  :1   Min.   :18.0   AI:1  
```

```
 Sathish:1   1st Qu.:18.5   CS:1  
 Sivant :1   Median :19.0   ML:1  
             Mean   :19.0         
             3rd Qu.:19.5         
             Max.   :20.0         
```

DAY - 2

Vector

Vectors are the most basic R data objects and there are six types of atomic vectors. They are logical, integer, double, complex, character and raw.

In [6]:
```
apple <- c('red','green',"yellow")
print(apple)
```

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

Bypassing Scope

R provides many different “escape hatches”—ways to bypass lexical scoping.

In [7]:
```
num = 10
func<-function(){
  num<<- 25      #changing global value.<<- assigns as global irrespective of scope
  print(num)
}
func()
```

```
[1] 25
```

CumulativeSum

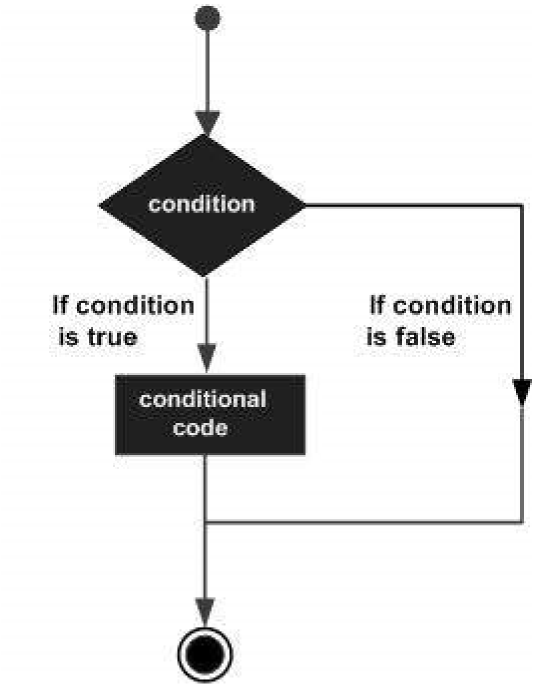
Used to calculate the cumulative sum of the vector passed as argument

In [8]:
```
#Cumulative Sum
numbers = c(1,5,7,85)
#Summation
print(sum(numbers))
#Cumulative summation
print(cumsum(numbers))
```

```
[1] 98
[1]  1  6 13 98
```

Decision Statements

Decision making structures requires to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.



In [9]:
```
#Decision Statements
v <- c(2,5.5,6,9)
t <- c(8,2.5,14,9)
print(v>t)
print(v == t)
print(v!=t)
print(v&&t)
```

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

Deleting Columns In DataFrame

In [10]:
```
#Dataframe creation
Name <- c("Jhon", "Lee", "Suzan", "Abhinav", "Brain", "Emma", "David")
#Variables
Subject <- c("Data Science", "Machine Learning", "Deep Learning", "Data Structures",
             "Database Managemnt System", "Operating Systems", "Python Programming")

Score <- c(56, 76, 86, 96, 73, 87, 47)

Rank <- c(5,8,6,7,9,2,1)

df = data.frame(Name,Subject,Score,Rank)
#Deleting Columns with subset and selcting -C
new_df = subset(df,select = -c(Name))
print(new_df)
```

```
                    Subject Score Rank
1              Data Science    56    5
2          Machine Learning    76    8
3             Deep Learning    86    6
4           Data Structures    96    7
5 Database Managemnt System    73    9
6         Operating Systems    87    2
7        Python Programming    47    1
```

Deleting Rows In DataFrame

In [11]:
```
#Dataframe creation
Name <- c("Jhon", "Lee", "Suzan", "Abhinav", "Brain", "Emma", "David")
#Variables
Subject <- c("Data Science", "Machine Learning", "Deep Learning", "Data Structures",
             "Database Managemnt System", "Operating Systems", "Python Programming")

Score <- c(56, 76, 86, 96, 73, 87, 47)

Rank <- c(5,8,6,7,9,2,1)

df = data.frame(Name,Subject,Score,Rank)
#Slicing rows from dataframe
df_new = df[-c(2,4),]
print(df_new)
```

```
   Name                   Subject Score Rank
1  Jhon              Data Science    56    5
3 Suzan             Deep Learning    86    6
5 Brain Database Managemnt System    73    9
6  Emma         Operating Systems    87    2
7 David        Python Programming    47    1
```

Factors

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. They are useful in data analysis for statistical modeling.

In [12]:
```
#Factors
names= c("Arjun","Ravi","Tharun","Ravi","Tharun","Arjun","Divi")
factor = factor(names)
#Printing unique values present in our vector
print(factor)
#Printing Count of unique values presnt in our vector
print(table(factor))
```

```
[1] Arjun  Ravi   Tharun Ravi   Tharun Arjun  Divi
Levels: Arjun Divi Ravi Tharun
factor
 Arjun   Divi   Ravi Tharun
     2      1      2      2
```

Filling NA Values

Missing values in data science arise when an observation is missing in a column of a data frame or contains a character value instead of numeric value. Missing values must be dropped or replaced in order to draw correct conclusion from the data.

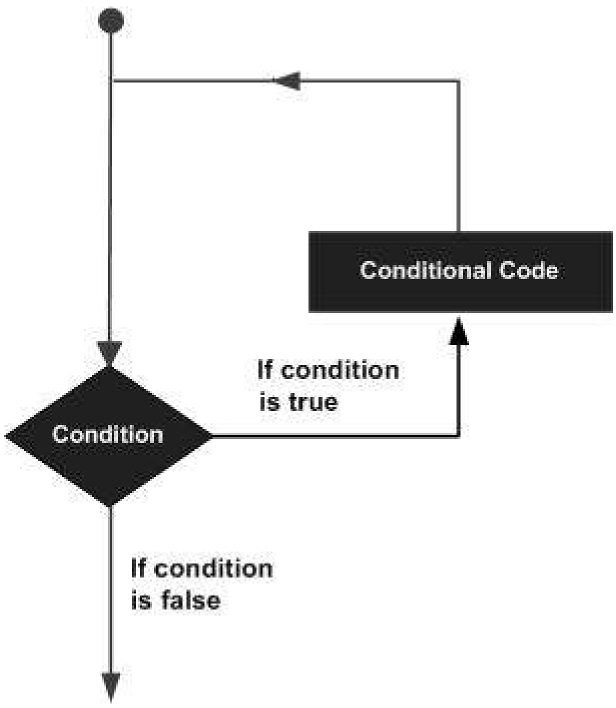
In [13]:
```
#Dataframe creation
Name <- c("Jhon", "Lee", "Suzan", "Abhinav", "Brain","Arun", "David")
#Variables
Subject <- c("Data Science","Database Managemnet System", "Deep Learning",
             "Data Structures","Cyber Security",
             "Operating Systems", "Python Programming")

Score <- c(56,NA, 86, 96, 73, 87, 47)

Rank <- c(5,8,6,NA,9,2,NA)

df = data.frame(Name,Subject,Score,Rank,stringsAsFactors=FALSE)
```

```
#filling Na values with o
df[is.na(df)] = 0
print(df)
```

```
     Name                    Subject Score Rank
1    Jhon               Data Science    56    5
2     Lee Database Managemnet System     0    8
3   Suzan              Deep Learning    86    6
4 Abhinav            Data Structures    96    0
5   Brain             Cyber Security    73    9
6    Arun          Operating Systems    87    2
7   David         Python Programming    47    0
```

For Looping

A loop statement allows us to execute a statement or group of statements multiple times and the following is the general form of a loop statement in most of the programming languages



In [14]:
```
#For Looping
numbers <- c(1:10)
for (i in numbers)
  {
  #Prints in same line
  cat(paste(i," "))
  }
```

```
1  2  3  4  5  6  7  8  9  10
```

Nested For Loops

The inner loop executes n-times of every execution of the outer for loop and also it’s a great tool to work with R Programming Language.

In [15]:
```
#Nested For Loops
res = matrix(nrow=4, ncol=4) # create a 4 x 4 matrix (of 4 rows and 4 columns)
for(i in 1:nrow(res))   #Assigned a variable  ‘i’for each row
{
  for(j in 1:ncol(res)) #Assigned a variable  ‘j for each column
  {
```

```
    res[i,j] = i*j      #calculating product of two indices
  }
}
print(res)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    4    6    8
[3,]    3    6    9   12
[4,]    4    8   12   16
```

Functions - Assigning Parameters

A function is a set of statements organized together to perform a specific task.

In [16]:

```
#Functions
EvenOdd<-function(num= 4,a=4,z=7){ #Defining a function with parameters
  #conditional statements
  if(num%%2==1)
  {
    return("Odd")
  }
  else
  {
    return("Even")
  }
}
#invoking function
EvenOdd()
```

'Even'

Functions - Passing Parameters

In [17]:

```
#Functions
EvenOdd<-function(num= 4){      #Defining a function
  #conditional statements
  if(num%%2==1)
  {
    return("Odd")
  }
  else
  {
      return("Even")
  }
}
#giving parameters to the function
EvenOdd(as.integer(readline("enter a number: ")))
```

```
enter a number: 25
```

'Odd'

IF statements

In [18]:

```
#IF condidtions
x <- c(15,8,9,45,89,23)
for (i in 1:6)
{
  print(x[i])
}
```

```
[1] 15
```

```
[1] 8
[1] 9
[1] 45
[1] 89
[1] 23
```

If - Else

An if statement can be followed by an optional else statement which executes when the boolean expression is false.

In [19]:
```
#If-Else statements
x <- -5
if(x > 0){
  #If condition
  print("Non-negative number")
} else {
  #Else Condition
  print("Negative number")
}
```

```
[1] "Negative number"
```

Inbuilt Functions

Simple examples of in-built functions are seq(), mean(), max(), sum(x) and paste(...) etc. They are directly called by user written programs. You can refer most widely used R functions

In [20]:
```
#Inbuild Functions
#Sequence of numbers
print(seq(47,89))           #print(47:89)
#Mean of all values in specified interval
print(mean(18:68))
#sum of all values
print(sum(41:68))
```

```
 [1] 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
[26] 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
[1] 43
[1] 1526
```

Max,Min Vectors

In [21]:
```
#Summary of data
numbers = c(5,85,97,14,65)
print(numbers)
#Maximum value in a vector
cat("Maximum Value is the vector is: ",max(numbers),"\n")
#Minimum value in a vector
cat("Minimum Value in the vector is : ",min(numbers),"\n")
#Length of values in a vector
cat("Length of values :",length(numbers))
```

```
[1]  5 85 97 14 65
Maximum Value is the vector is:  97
Minimum Value in the vector is :  5
Length of values : 5
```

Merging Dataframes

To merge two data frames (datasets) horizontally, use the merge function. In most cases, you join two data frames by one or more common key variables (i.e., an inner join).

In [22]:
```
#Dataframe creation
Name <- c("Jhon", "Lee", "Suzan", "Abhinav", "Brain", "Emma", "David")
#Variables
Subject <- c("Data Science", "Machine Learning", "Deep Learning", "Data Structures",
             "Database Managemnt System", "Operating Systems", "Python Programming")

Score <- c(56, 76, 86, 96, 73, 87, 47)

Rank <- c(5,8,6,7,9,2,1)

df_1 = data.frame(Name,Subject,Score,Rank)
#Cre
df_2 = data.frame(Name = c("Abi","Anu"),Subject = c("Cyber security"),Score = c(78,6
df = rbind(df_1,df_2)
#Merging Two DataFrames
print(df)
```

```
     Name                   Subject Score Rank
1    Jhon              Data Science    56    5
2     Lee          Machine Learning    76    8
3   Suzan             Deep Learning    86    6
4 Abhinav           Data Structures    96    7
5   Brain Database Managemnt System    73    9
6    Emma         Operating Systems    87    2
7   David        Python Programming    47    1
8     Abi            Cyber security    78    7
9     Anu            Cyber security    65    4
```

Random Numbers

In [23]:
```
#Random Numbers
print(sample(1:100,5))
#Sample integer values sample(start:end,frequency)
print(round(runif(10,1,100),2))
#Sample float values round(runif(frequency,start,end)frequency)
```

```
[1] 90 91 69 99 57
 [1] 69.59 64.41 99.43 65.91 71.14 54.86 59.82 29.63 15.56 96.34
```

Dataframe Operations

In [24]:
```
#Dataframe creation
Name <- c("Jhon", "Lee", "Suzan", "Abhinav", "Brain", "Emma", "David")
#Variables
Subject <- c("Data Science", "Machine Learning", "Deep Learning", "Data Structures",
             "Database Managemnt System", "Operating Systems", "Python Programming")

Score <- c(56, 76, 86, 96, 73, 87, 47)

Rank <- c(5,8,6,7,9,2,1)

df = data.frame(Name,Subject,Score,Rank)
#Renaming Column Name
colnames(df)[which(names(df) == "Name")] ="Candidate Name"
print(df)
```

```
  Candidate Name                   Subject Score Rank
1           Jhon              Data Science    56    5
2            Lee          Machine Learning    76    8
3          Suzan             Deep Learning    86    6
4        Abhinav           Data Structures    96    7
```

```
5          Brain Database Managemnt System    73    9
6           Emma            Operating Systems    87    2
7          David          Python Programming    47    1
```

Repeat

The Repeat Function(loop) in R executes a same block of code iteratively until a stop condition is met. repeat loop in R, is similar to while and for loop, it will execute a block of commands repeatedly till break.

In [25]:

```
#Repeat Looping
x <- 5
repeat {
  print(x)
  x = x+1#Incrementing values
  if (x == 10){
    break
}}
```

```
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

Repeat - Break

In [26]:

```
#repeat Break Loop
result <- c("Hello World")
i <- 1

# test expression
repeat {

  print(result)

  # update expression
  i <- i + 1

  # Breaking condition
  if(i >5) {
    break
  }
}
```

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

DAY - 3

Appending

In [27]:

```
#append to merge objects together
#random numbers
list1=sample(1:20,10)
#sample(start:end,numbers)
list2=sample(100:200,5)
```

```
#appending two lists
append(list1,list2)
```

1. 19
2. 4
3. 14
4. 17
5. 11
6. 7
7. 5
8. 12
9. 10
10. 15
11. 108
12. 140
13. 173
14. 122
15. 126

Substrings

Extract or replace substrings in a character vector.

In [28]:
```
#substring
#substring("string",start,end)
#substr("string",start,end)
print(substr("learn R program",4,8))
str <-"Sathish kumar"
len <- nchar("Hello")
print(substring(str,len-2,len))
```

```
[1] "rn R "
[1] "thi"
```

String Operations

Any value written within a pair of single quote or double quotes in R is treated as a string. Internally R stores every string within double quotes, even when you create them with single quote.

Rules Applied in String Construction

- The quotes at the beginning and end of a string should be both double quotes or both single quote. They can not be mixed.
- Double quotes can be inserted into a string starting and ending with single quote.
- Single quote can beinserted into a string starting and ending with double quotes.
- Double quotes can not be inserted into a string starting and ending with double quotes.
- Single quote can not be inserted into a string starting and ending with single quote.

In [29]:
```
#Length of String
#Importing package
library(stringr)
str_length("Hello")

a<-"sathish"
#Another way
nchar("h****elo'lo")
nchar(a)
```

5

11

7

In [30]:
```
#Types of expressing Strings
a <- 'Start and end with single quote'
print(a)

b <- "Start and end with double quotes"
print(b)

c <- "single quote ' in between double quotes"
print(c)

d <- 'Double quotes " in between single quote'
print(d)

#print(e <- 'Mixed" ) (It gives as error)

print("hello 1 ' hello how are you ' hello ")
print('hello 1 \' hello how are you \' hello ')
```

```
[1] "Start and end with single quote"
[1] "Start and end with double quotes"
[1] "single quote ' in between double quotes"
[1] "Double quotes \" in between single quote"
[1] "hello 1 ' hello how are you ' hello "
[1] "hello 1 ' hello how are you ' hello "
```

Sorting

The variable by which sort you can be a numeric, string or factor variable. You also have some options on how missing values will be handled: they can be listed first, last or removed.

In [31]:
```
#Built-in R features
#sorting
#Specifying to show in descending
sort(c(1,8,6,9),decreasing = TRUE)
#Default Ascending
sort(c(1,8,6,9))

names=c("Sathish","Arun","Ajay","Test","banana")
print(sort(names))
```

1. 9
2. 8
3. 6
4. 1

1. 1

2. 6
3. 8
4. 9

```
[1] "Ajay"    "Arun"    "banana"  "Sathish" "Test"
```

Switch Expressions

Switch case statements are a substitute for long if statements that compare a variable to several integral values. Switch case in R is a multiway branch statement. It allows a variable to be tested for equality against a list of values.

In [32]:
```
#Switch Expression
#switch()
switch(5,"one","two","three","four","five")
switch(2,colour = "green","shape"="square",length = "five")
```

'five'

'square'

Reverse

A generic function for reversing vector-like or list-like objects.

In [33]:
```
#reversing vector
rev(c(2,3,4,55))
#rev()
print(rev(c(1:78)))
```

1. 55
2. 4
3. 3
4. 2

```
 [1] 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54
[26] 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29
[51] 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4
[76]  3  2  1
```

Splicing datasets

In [34]:
```
#Dataframe creation
Name <- c("Jhon", "Lee", "Suzan", "Abhinav", "Brain", "Emma", "David")
#Variables
Subject <- c("Data Science", "Machine Learning", "Deep Learning", "Data Structures",
             "Database Managemnt System", "Operating Systems", "Python Programming")

Score <- c(56, 76, 86, 96, 73, 87, 47)

Rank <- c(5,8,6,7,9,2,1)

df = data.frame(Name,Subject,Score,Rank)
print(df)
#Dataframe Created Successfully
print(df$Name)
#Accessing columns with dOllar symbol
```

```
     Name                    Subject Score Rank
```

```
1     Jhon             Data Science    56    5
2      Lee         Machine Learning    76    8
3    Suzan            Deep Learning    86    6
4 Abhinav           Data Structures    96    7
5    Brain Database Managemnt System    73    9
6     Emma        Operating Systems    87    2
7    David       Python Programming    47    1
[1] Jhon    Lee     Suzan   Abhinav Brain   Emma    David
Levels: Abhinav Brain David Emma Jhon Lee Suzan
```

Sqrt - Log Functions

In [35]:
```
#sqrt,log functions

print(sqrt(5))

#log base10 100 multiplied by cos pi
print(log10(100)*cos(pi))
```

```
[1] 2.236068
[1] -2
```

Replace Strings

In [36]:
```
#replace string
#sub & gsub

sentences = c("I like maths", "you like apples")

#sub replaces only first instance
sub (pattern=" " , replacement="_",sentences)

#gsub replaces all instances
gsub (pattern=" " , replacement="_",sentences)
```

1. 'I_like maths'
2. 'you_like apples'

1. 'I_like_maths'
2. 'you_like_apples'

Regular Expression

A 'regular expression' is a pattern that describes a set of strings.
Two types of regular expressions are used in R, extended regular expressions (the default) and Perl-like regular expressions used by perl = TRUE .
There is also fixed = TRUE which can be considered to use a literal regular expression.

In [37]:
```
#Regular expressions
#grep("pattern","variable value")
vector<-c('a','b','b','c','c','d')
string <- "Sathish"
num <- "[1-9]"

#grepl in vector gives binary output of pattern
grepl('b',vector)

#grep in vector gives index of pattern
```

```
grep("b",vector)

#grepl in string gives binary output of pattern
grepl('t',string)

#grep in string gives whether it is present or not
grep("8",string)

#grep with index limit as pattern
grep(num,"sath45")

#grepl with index gives binary output of pattern
grepl(num,"sath45")
```

1. FALSE
2. TRUE
3. TRUE
4. FALSE
5. FALSE
6. FALSE

1. 2
2. 3

TRUE

1

TRUE

Paste Function

paste converts its arguments (via as. character) to character strings, and concatenates them (separating them by the string given by sep).

In [38]:

```
#Concatenating strings
#paste(variables,sep = "",collapse=NULL)
a<-"My Name is"
b<- "Sathish."
c<-"I am 19 years old"
paste(a,b,c,sep = " ")
```

'My Name is Sathish. I am 19 years old'

Next Function

In [39]:

```
#next operation
v=LETTERS[1:17]
for (i in v) {
  if(i=="D"){
  next                        #bypasses the value given with next
  }
print(i)
}
```

```
[1] "A"
[1] "B"
[1] "C"
[1] "E"
```

```
[1] "F"
[1] "G"
[1] "H"
[1] "I"
[1] "J"
[1] "K"
[1] "L"
[1] "M"
[1] "N"
[1] "O"
[1] "P"
[1] "Q"
```

Inbuilt Dataframe

In [40]:
```
#Dataframes
str(mtcars)#inbuilt dataframe
summary(mtcars)
```

```
'data.frame':	32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
      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  
```

Date-time

R provides a broad range of capabilities to deal with times and dates. While these ideas are quite fundamental to R programming, they have been left until now as some of the best ways of using them comes with in add-on packages.

In [41]:
```
#Date - time
Sys.Date()
#Time in system
Sys.time()

g<- "2020-11-4"
```

```
class(g)
#Converting String to date
y=as.Date(g)
#Here class will be date
class(y)
#Changing Date to specified format
my.date=as.Date("Nov-03-90",format="%b-%d-%y")
print(my.date)
#Converts Date-tiem to specified format
as.POSIXlt("11:02:03",format="%H:%M:%S")
```

2020-11-15

```
[1] "2020-11-15 18:05:12 IST"
```

'character'

'Date'

```
[1] "1990-11-03"
[1] "2020-11-15 11:02:03 IST"
```

Datatypes

In [42]:

```
#Datatype check
h <- "Sathish"
#Checking for Datatypes
is.double(h)
is.matrix(h)
is.array(h)
is.atomic(h)
is.call(h)
is.character(h)
is.complex(h)
is.data.frame(h)
```

FALSE

FALSE

FALSE

TRUE

FALSE

TRUE

FALSE

FALSE

Basic Calculator

In [43]:

```
#Simple Calculator
#add function
add <- function(x, y) {
  return(x + y)
}
#subtract
subtract <- function(x, y) {
  return(x - y)
}
#Multiply
multiply <- function(x, y) {
  return(x * y)
}
#Divide
divide <- function(x, y) {
  return(x / y)
```

```
}
# take input from the user
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
#Taking the choice of the user
choice = as.integer(readline(prompt="Enter choice[1/2/3/4]: "))
num1 = as.integer(readline(prompt="Enter first number: "))
num2 = as.integer(readline(prompt="Enter second number: "))
operator <- switch(choice,"+","-","*","/")
result <- switch(choice, add(num1, num2), subtract(num1, num2), multiply(num1, num2)
print(paste(num1, operator, num2, "=", result))
```

```
[1] "Select operation."
[1] "1.Add"
[1] "2.Subtract"
[1] "3.Multiply"
[1] "4.Divide"
Enter choice[1/2/3/4]: 2
Enter first number: 78
Enter second number: 9
[1] "78 - 9 = 69"
```

DAY - 4

Dataframe Slicing

In [44]:

```
#Reading a CSV File
df<-read.csv("E:/R Programs For Practice/DAY - 4/Class/Data/EmployeeDetails.csv")
#Data
print(df)
#Maximum value for a single column
print(paste("Maximum Salary: ",max(df$Salary)))
#Splicing the record with higer salary
details=subset(df,Salary==max(Salary))
print(details)
#Splicing person with age greater than 50
aged_ppl=subset(df,Age>=50)
print(aged_ppl)
#Satisying Two condition using and & operator
aged_ppl_and_high_salary=subset(df,Age>=50&Salary>5000)
print(aged_ppl_and_high_salary)
#Satisying Any one condition using and | operator
aged_ppl_or_high_salary=subset(df,Age>=50|Salary>5000)
print(aged_ppl_or_high_salary)
```

```
  ï..Name Age   phone Salary
1 sathish  45 9155525   1485
2   arjun  78   21561   5455
3   karan  89 4514444    955
4  sivant  21     556    255
5    Raja  64   54674   7555
6    siva  14     564   5552
[1] "Maximum Salary:  7555"
  ï..Name Age phone Salary
5    Raja  64 54674   7555
  ï..Name Age   phone Salary
2   arjun  78   21561   5455
3   karan  89 4514444    955
5    Raja  64   54674   7555
  ï..Name Age phone Salary
2   arjun  78 21561   5455
5    Raja  64 54674   7555
```

```
  ï..Name Age   phone Salary
2   arjun  78   21561   5455
3   karan  89 4514444    955
5    Raja  64   54674   7555
6    siva  14     564   5552
```

File location Locator

In [45]:

```
#File location
#gets the present file location
getwd()
#set the location for our file
setwd("E:/R Programs For Practice/DAY - 4/Class")
```

'E:/R Programs For Practice'

Reading Csv File

In [46]:

```
#Reading a CSV File
df<-read.csv("E:/R Programs For Practice/DAY - 4/Class/Data/EmployeeDetails.csv")
#Data
print(df)
#Checking if it is a dataframe
print(is.data.frame(df))
#Number of columns
print(ncol(df))
#Number of rows
print(nrow(df))
```

```
  ï..Name Age   phone Salary
1 sathish  45 9155525   1485
2   arjun  78   21561   5455
3   karan  89 4514444    955
4  sivant  21     556    255
5    Raja  64   54674   7555
6    siva  14     564   5552
[1] TRUE
[1] 4
[1] 6
```

Writing a Csv File

In [47]:

```
df<-read.csv("E:/R Programs For Practice/DAY - 4/Class/Data/EmployeeDetails.csv")
#Data
print(df)
#writig dataframe to csv
write.csv(df,"test.csv",row.names = TRUE)
#reading again the file for check
data = read.csv("test.csv")
print(data)
```

```
  ï..Name Age   phone Salary
1 sathish  45 9155525   1485
2   arjun  78   21561   5455
3   karan  89 4514444    955
4  sivant  21     556    255
5    Raja  64   54674   7555
6    siva  14     564   5552
  X ï..Name Age   phone Salary
1 1 sathish  45 9155525   1485
2 2   arjun  78   21561   5455
3 3   karan  89 4514444    955
4 4  sivant  21     556    255
```

```
5 5     Raja  64    54674    7555
6 6     siva  14      564    5552
```

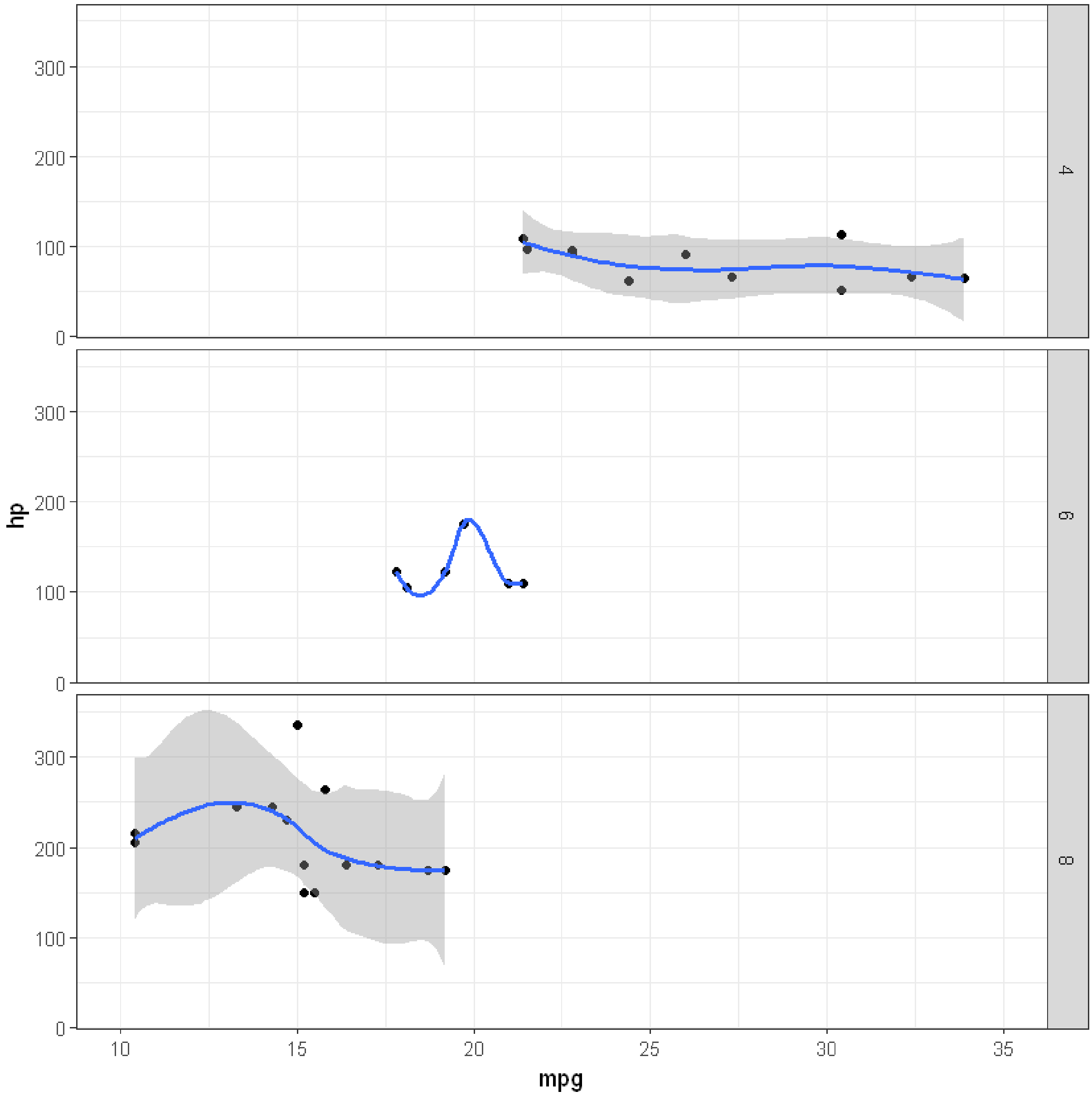
DAY - 5

ggplot layers

In [48]:

```
#ggplot-grammar of graphics
#data visualisation is the graphical representation of information
#import packages
library('ggplot2')
library('ggplot2movies')
#layers of ggplot:
#data,asthetics,geometry,customisationfacets,Statistics,Cordinates
#for first three layers  in ggplot we can use comma
pl=ggplot(data=mtcars,aes(x=mpg,y=hp))
pl+geom_point()+facet_grid(cyl ~ .)+
  stat_smooth()+coord_cartesian(xlim=c(10,35))+theme_bw()
#facet grid is applying the factor method to get the unique values
#coord cartion is used to limit the cordinates
```

```
Warning message:
"package 'ggplot2' was built under R version 3.6.3"`geom_smooth()` using method = 'l
oess' and formula 'y ~ x'
```



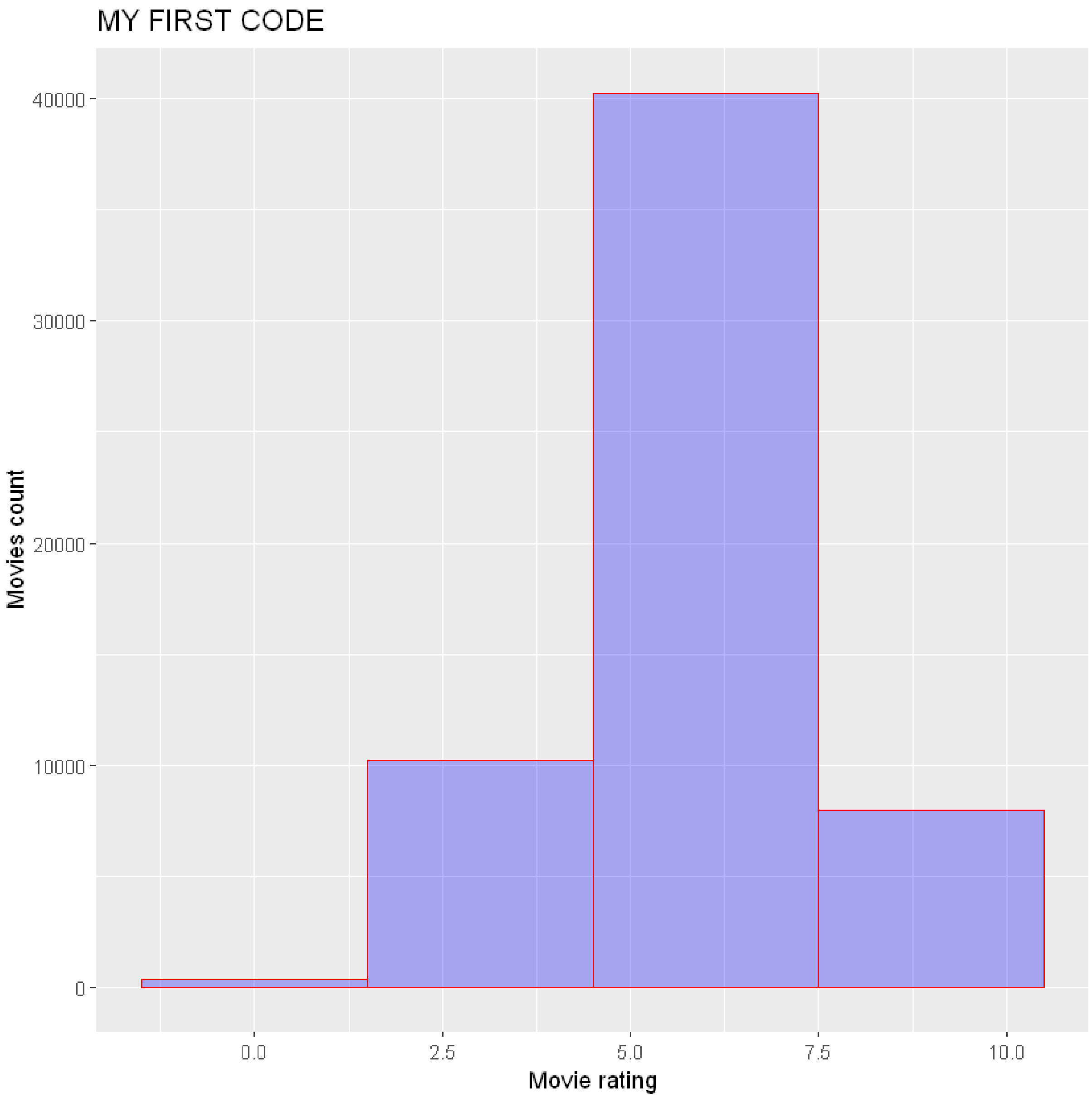
ggplot Histogram

In [49]:
```
#Histogram
#Importing Packages
library('ggplot2')
library('ggplot2movies')
colnames(movies)
head(movies)
#frequency of the rating column is done by histogram
pl=ggplot(data=movies,aes(x=rating))
#geometry
l=pl+geom_histogram(binwidth=3,color='red',fill="blue",alpha=0.3)
l+xlab("Movie rating")+ylab("Movies count")+ggtitle('MY FIRST CODE')
```

1. 'title'
2. 'year'
3. 'length'
4. 'budget'
5. 'rating'
6. 'votes'
7. 'r1'
8. 'r2'
9. 'r3'
10. 'r4'
11. 'r5'
12. 'r6'
13. 'r7'
14. 'r8'
15. 'r9'
16. 'r10'
17. 'mpaa'
18. 'Action'
19. 'Animation'
20. 'Comedy'
21. 'Drama'
22. 'Documentary'
23. 'Romance'
24. 'Short'

title	year	length	budget	rating	votes	r1	r2	r3	r4	...	r9	r10	mpaa	Actior
$	1971	121	NA	6.4	348	4.5	4.5	4.5	4.5	...	4.5	4.5		(
$1000 a Touchdown	1939	71	NA	6.0	20	0.0	14.5	4.5	24.5	...	4.5	14.5		(
$21 a Day Once a Month	1941	7	NA	8.2	5	0.0	0.0	0.0	0.0	...	24.5	24.5		(
$40,000	1996	70	NA	8.2	6	14.5	0.0	0.0	0.0	...	34.5	45.5		(
$50,000 Climax Show, The	1975	71	NA	3.4	17	24.5	4.5	0.0	14.5	...	0.0	24.5		(

title	year	length	budget	rating	votes	r1	r2	r3	r4	...	r9	r10	mpaa	Actio
$pent	2000	91	NA	4.3	45	4.5	4.5	4.5	14.5	...	14.5	14.5		(



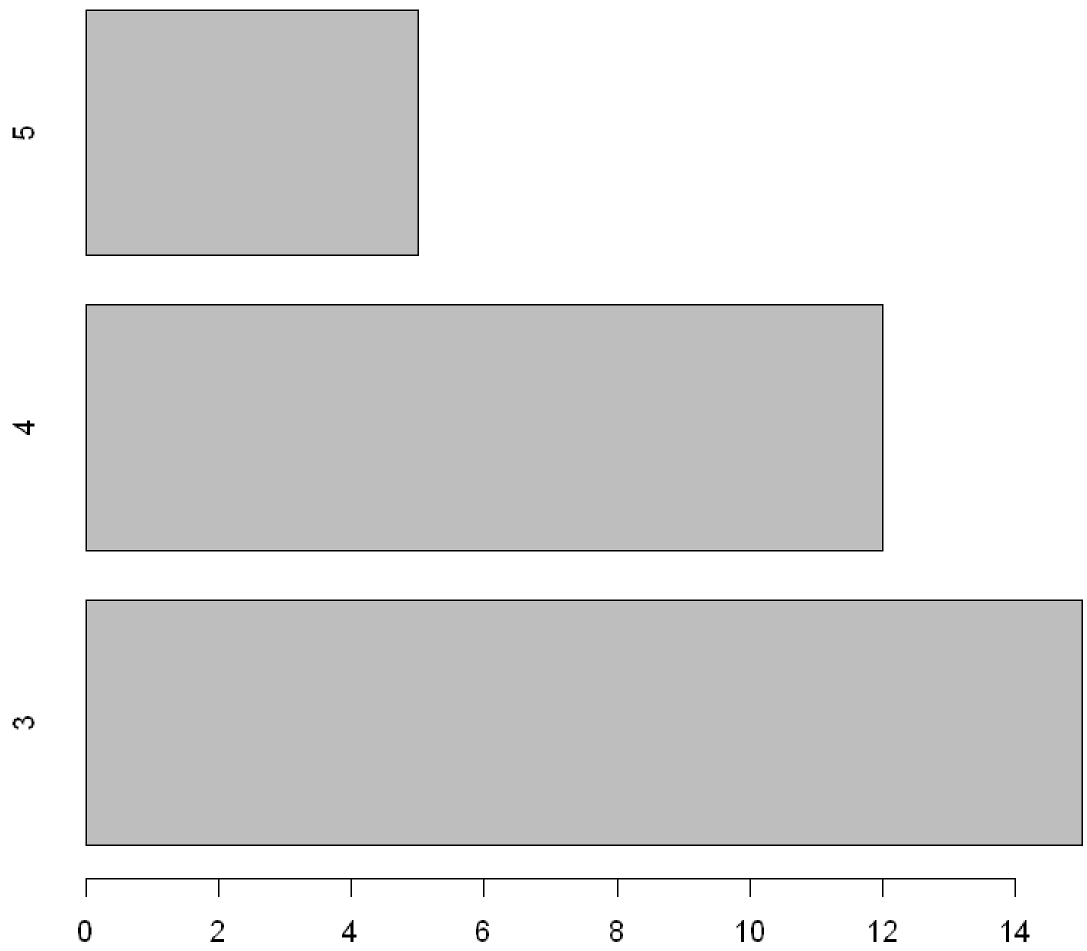
Bar plot

Bar plots can be created in R using the barplot() function. We can supply a vector or matrix to this function.

In [50]:

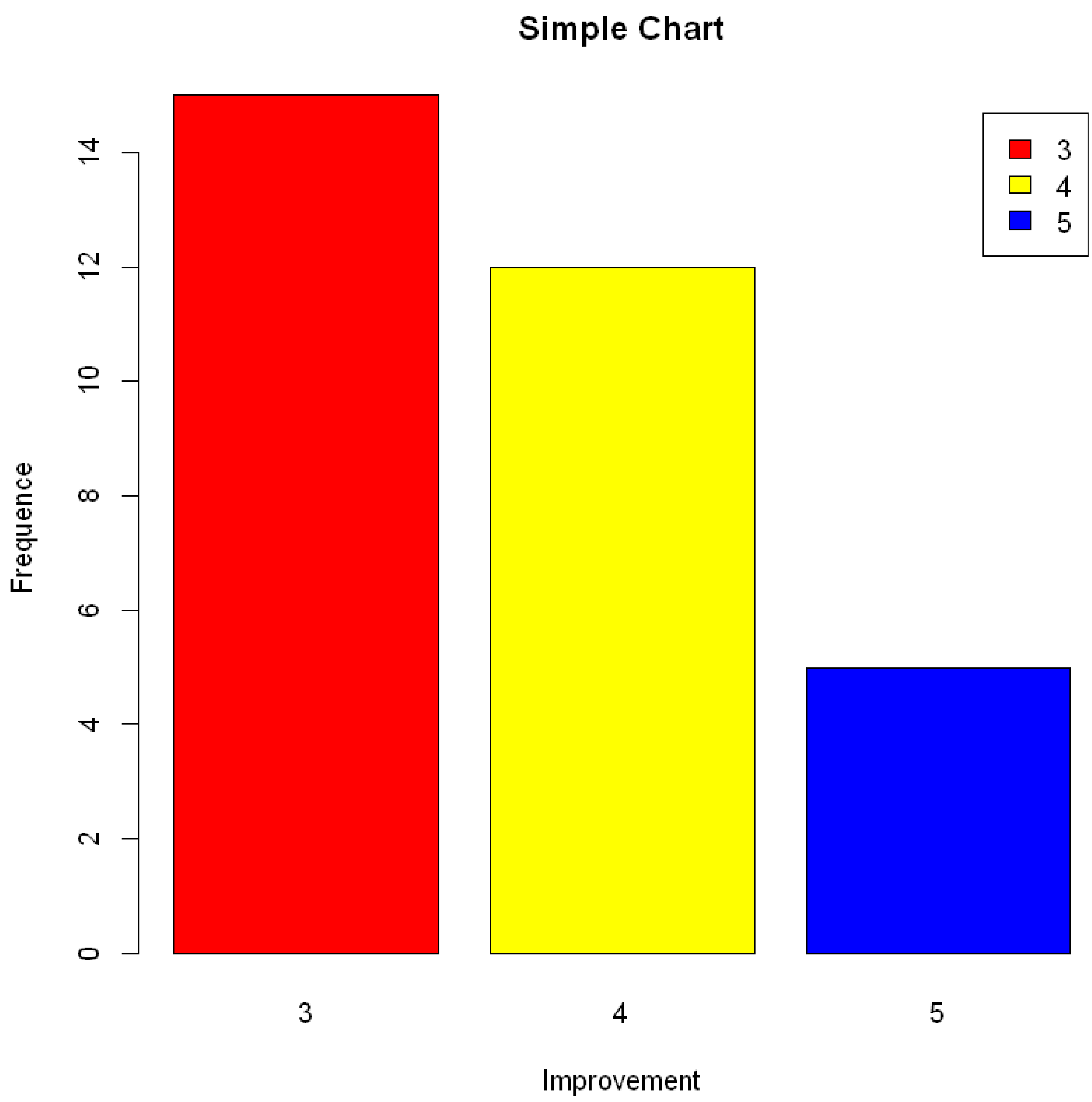
```
#taking gear column values from mtcars
count=mtcars$gear
print(count)
#changing raw data to frequency values
table=table(count)
#barplot(table name,horiz = true)
#deafult horiz is false
barplot(table,horiz = TRUE)
```

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



In [51]:
```
#taking gear column values from mtcars
count=mtcars$gear
print(count)
#changing raw data to frequency values
table=table(count)
#barplot(table name,horiz = true)
#deafult horiz is false
barplot(table,main="Simple Chart",
        xlab = "Improvement",
        ylab ="Frequence",
        legend= rownames(table),
        col = c("red","yellow","blue"))
#giving parameters for legend,colours,clable,ylabels
```

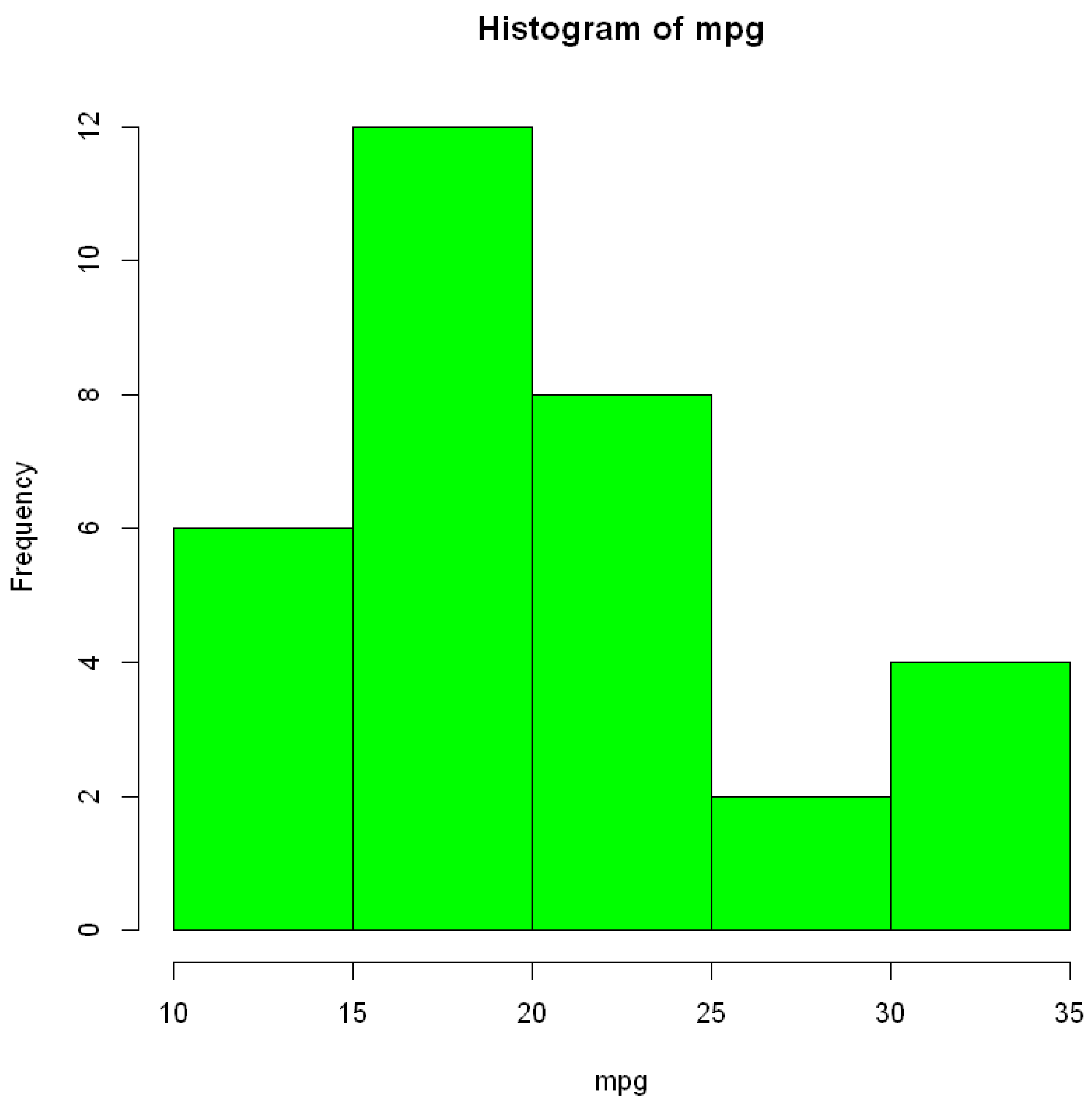
```
[1] 4 4 4 3 3 3 3 4 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 4 5 5 5 5 5 4
```



Hist Plot

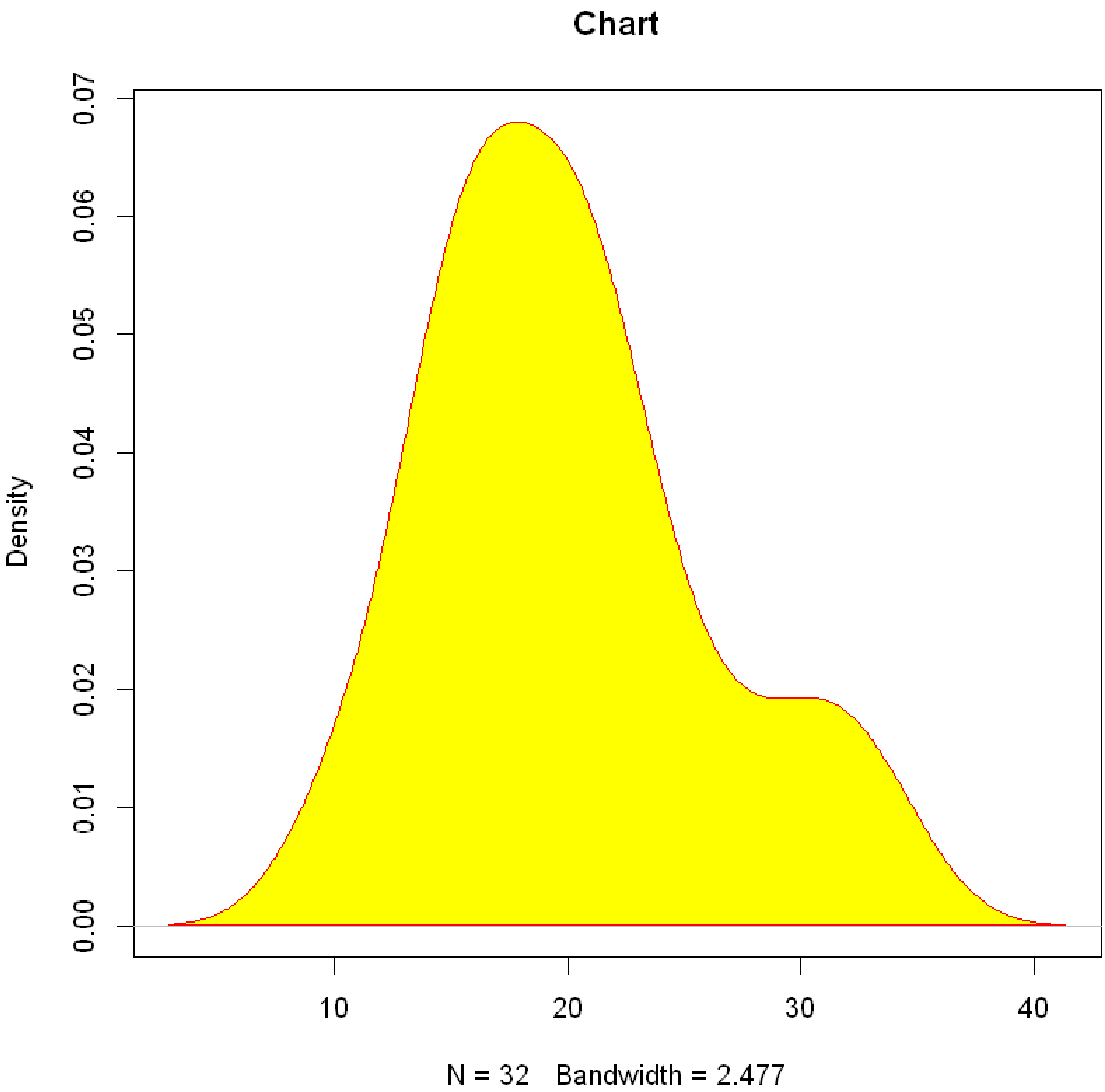
Histogram can be created using the hist() function in R programming language. This function takes in a vector of values for which the histogram is plotted.

In [52]:
```
mpg = (mtcars$mpg)
hist(mpg,col = "green")
```



Kernel Density Estimator

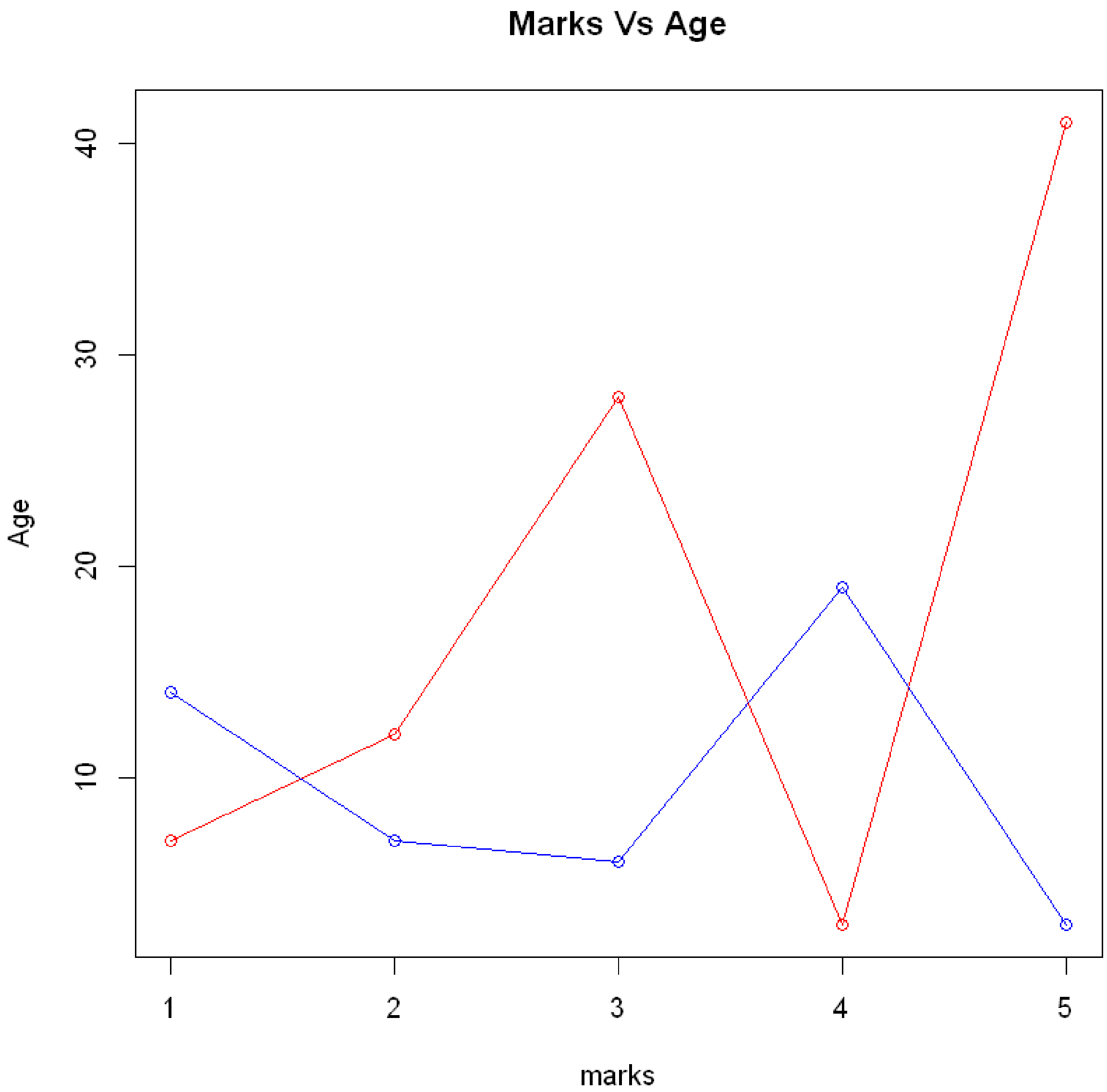
In [53]:
```
d <- density(mtcars$mpg)
plot(d, main="Chart")
polygon(d, col="yellow", border="red")
```



Line Chart

c is a vector containing the numeric values. type takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.

In [54]:
```
#Line plot
marks <- c(7,12,28,3,41)
age <- c(14,7,6,19,3)
#Line chart only accepts numbers(no string)
plot(marks,type = "o",col = "red", xlab = "marks", ylab = "Age",
     main = "Marks Vs Age")
#Mutiple lines in a single chart used for comparison
lines(age, type = "o", col = "blue")
```

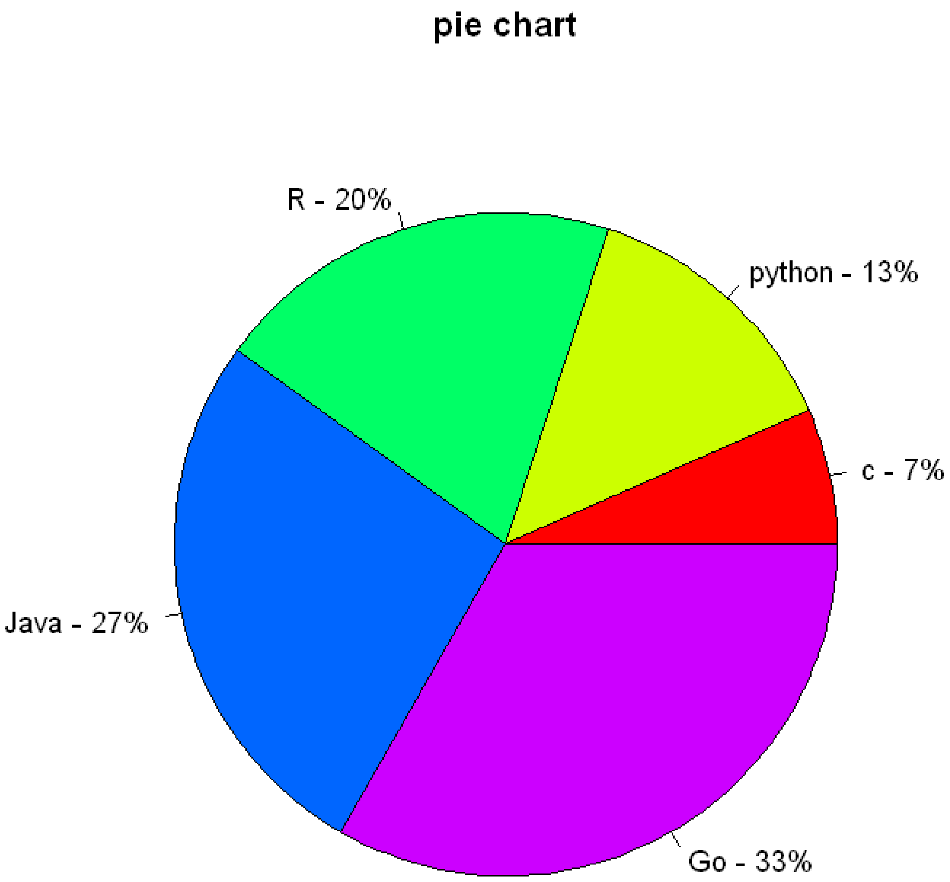


Pie Chart

The slices are labeled and the numbers corresponding to each slice is also represented in the chart. ... In R the pie chart is created using the pie() function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

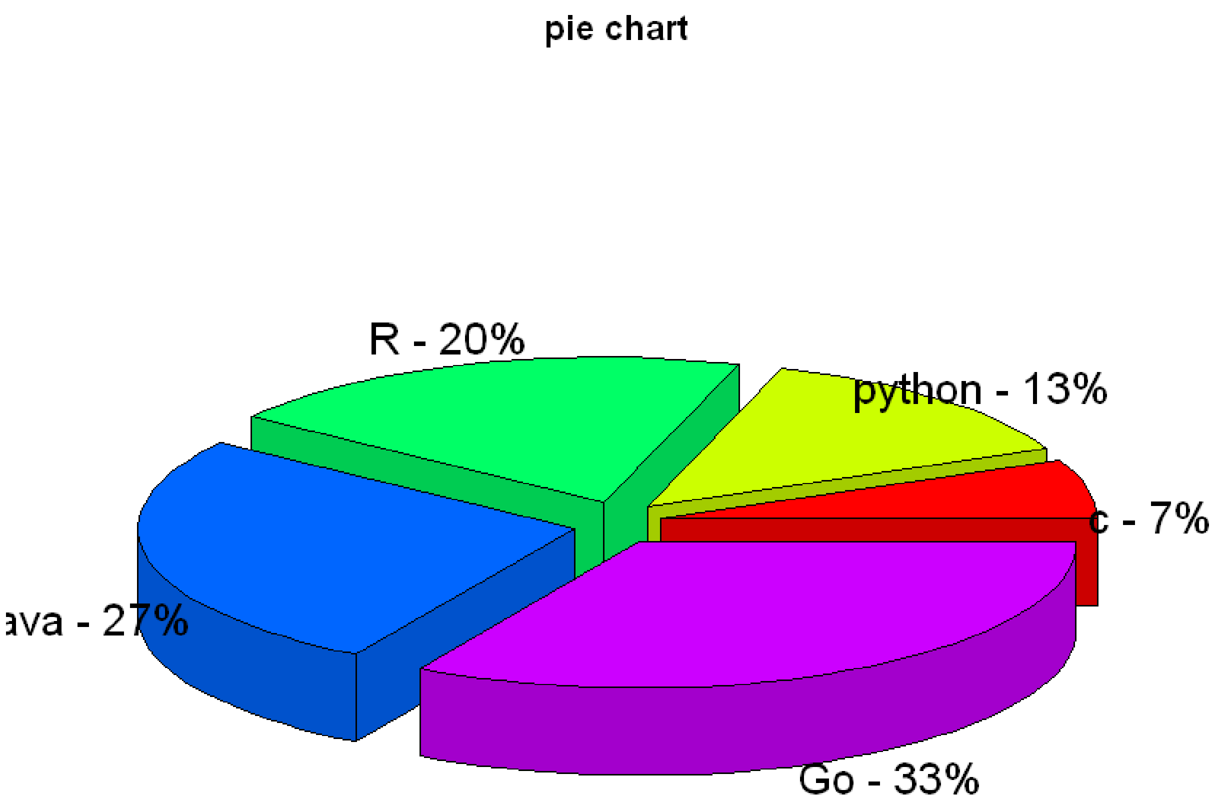
In [55]:
```
library(plotrix)
#Data
slices=c(10,20,30,40,50)
#taking percentage of frequency
pct=round(slices/sum(slices)*100)
lab=c("c","python","R","Java","Go")
#concatenating percentage and labels
lbl=paste(lab," - ",pct,"%",sep="")
#Plotting chart with color,header
pie(slices,labels=lbl,main="pie chart",col=rainbow(5))
```

```
Warning message:
"package 'plotrix' was built under R version 3.6.3"
```



Pie chart - 3D

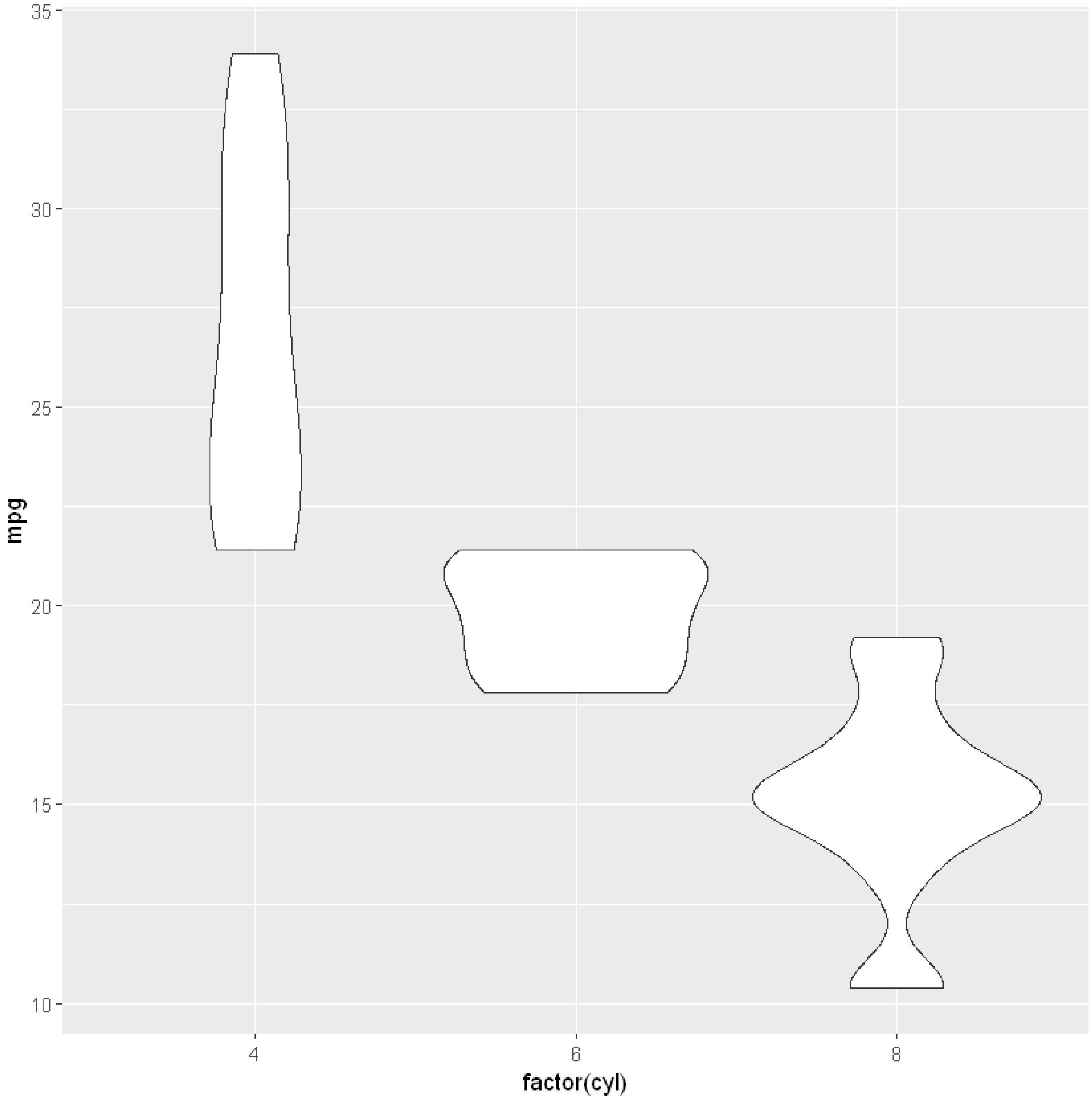
In [56]:
```
#importing Required Packages
library(plotrix)
#Data
slices=c(10,20,30,40,50)
#taking percentage of frequency
pct=round(slices/sum(slices)*100)
lab=c("c","python","R","Java","Go")
#concatenating percentage and labels
lbl=paste(lab," - ",pct,"%",sep="")
#Plotting chart with color,header
pie3D(slices,labels=lbl,main="pie chart",explode = 0.1,col=rainbow(5))
```



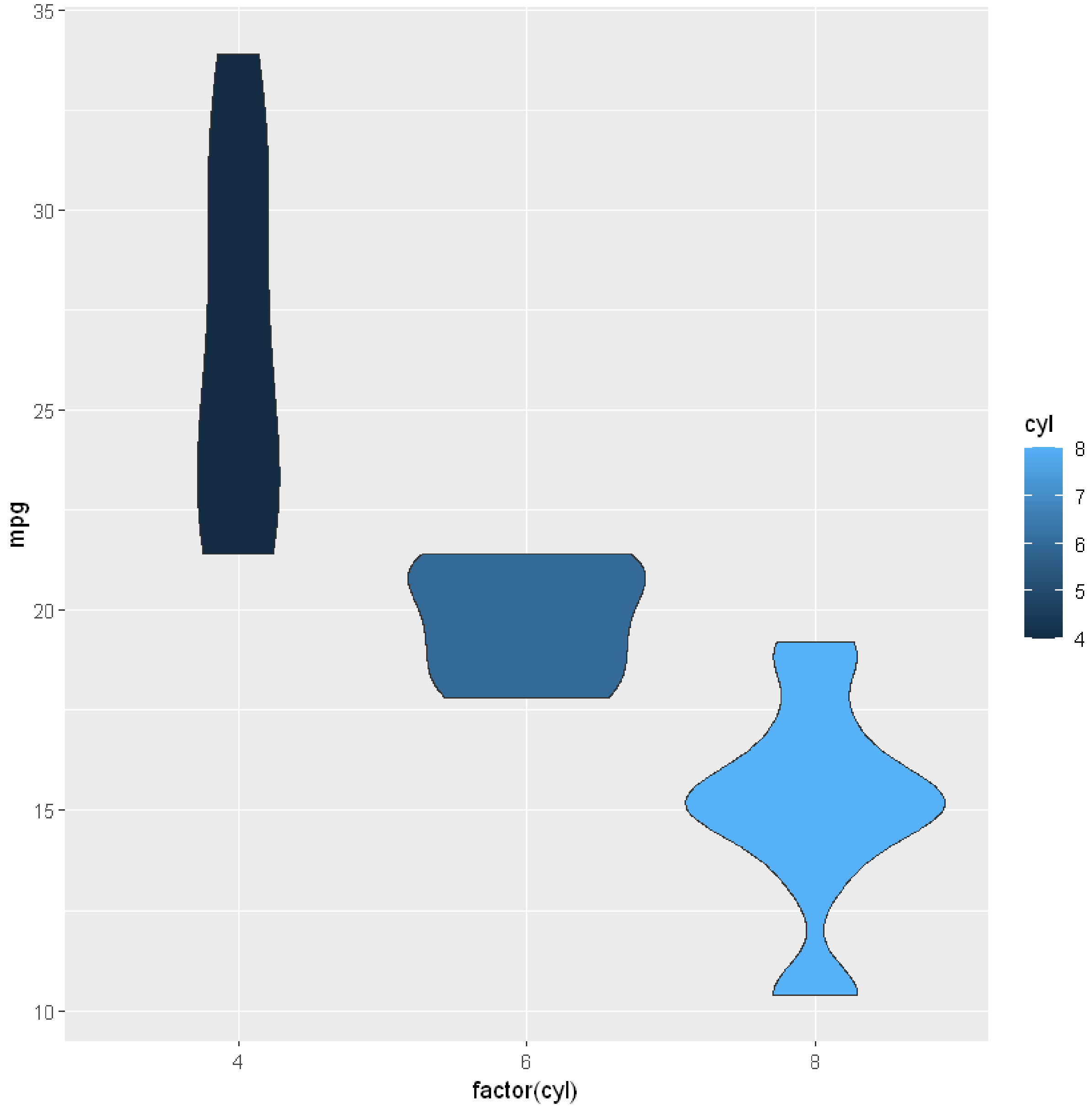
Violin Plot

violin plots are similar to box plots, except that they also show the kernel probability density of the data at different values.

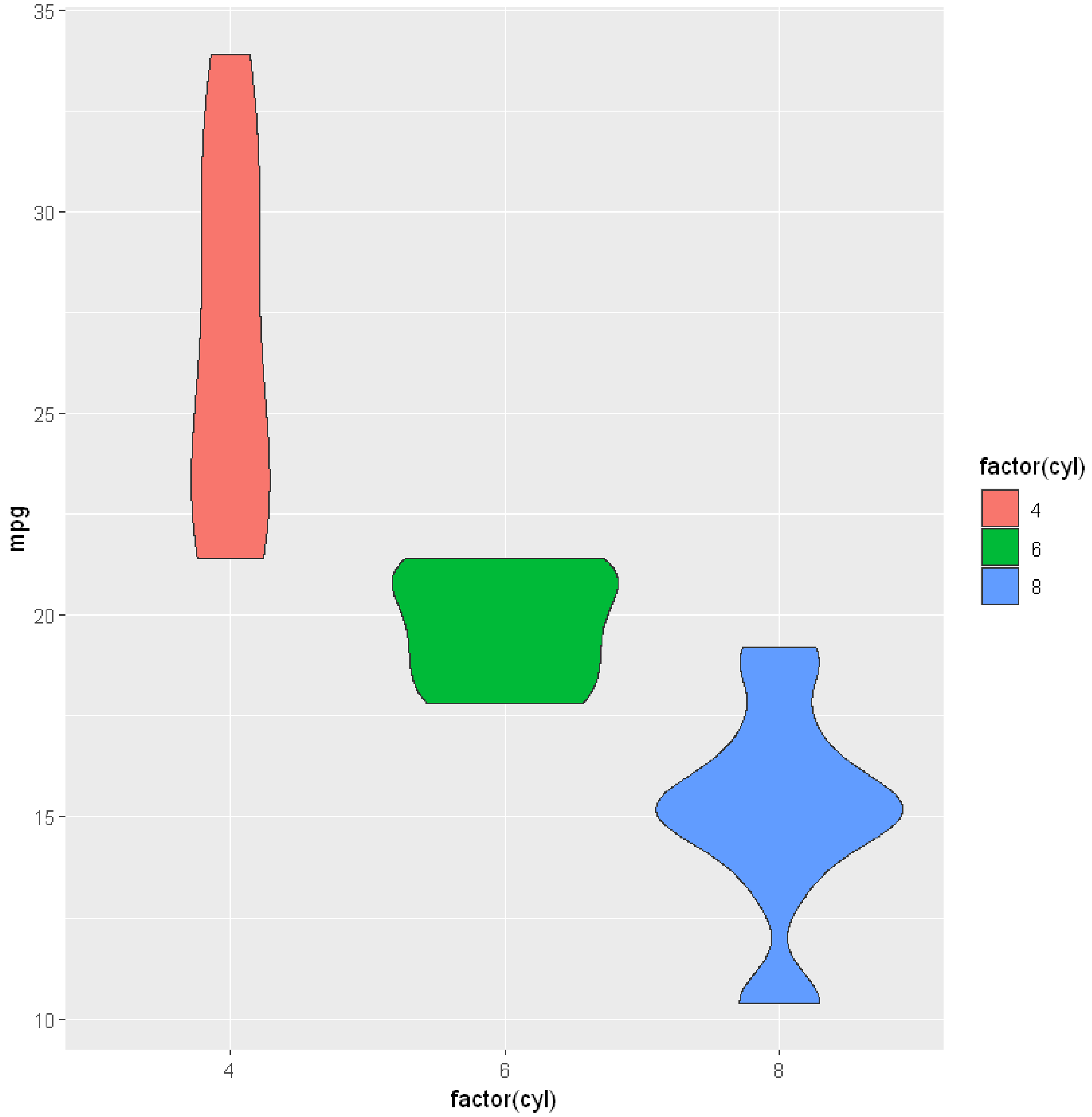
In [57]:
```
p <- ggplot(mtcars, aes(factor(cyl), mpg))
p + geom_violin()
```



In [58]:
```
p + geom_violin(aes(fill = cyl))
```



In [59]:
```
p + geom_violin(aes(fill = factor(cyl)))
```



Reading Excel File

In [60]:
```
#Importing Required Packages
library(readxl)
# reading the excel file
#sheeet attribute can also be added to give the sheet1
data=read_xlsx('E:/R Programs For Practice/DAY - 5/Class/Data/file.xlsx')
#data=read_xlsx('file path',sheet="sheet1")
print(data$word)
#gettting the number of sheets in excel file
excel_sheets('E:/R Programs For Practice/DAY - 5/Class/Data/file.xlsx')
sum(data$freq)
summary(data)
```

```
 [1] "hello"    "hey"      "test"     "photo"    "sathish"  "sret"
 [7] "yahoo"    "gmail"    "chennai"  "AI"       "ML"       "Data"
[13] "mining"   "Learning" "SBNA"
```

'file'

185

```
     word                freq
 Length:15          Min.   : 2.00
 Class :character   1st Qu.: 5.50
 Mode  :character   Median : 8.00
                    Mean   :12.33
```

```
3rd Qu.:14.00
Max.   :45.00
```

Writing Excel File

In [61]:

```
#Importing Required Packages
library(readxl)
# reading the excel file
#sheeet attribute can also be added to give the sheet1
data=read_xlsx('E:/R Programs For Practice/DAY - 5/Class/Data/file.xlsx')
#data=read_xlsx('file path',sheet="sheet1")
print(data$word)
#gettting the number of sheets in excel file
excel_sheets('E:/R Programs For Practice/DAY - 5/Class/Data/file.xlsx')
#to write a data from xlsx to txt or xls....
write.table(data,file="data.txt",row.names=F)
data1=read.table("data.txt",header=TRUE)
print(data1)
```

```
 [1] "hello"    "hey"      "test"     "photo"    "sathish"  "sret"
 [7] "yahoo"    "gmail"    "chennai"  "AI"       "ML"       "Data"
[13] "mining"   "Learning" "SBNA"
```

'file'

```
       word freq
1     hello    5
2       hey   14
3      test    8
4     photo    2
5   sathish    9
6      sret   36
7     yahoo    2
8     gmail    8
9   chennai    5
10       AI    7
11       ML   14
12     Data    9
13   mining    6
14 Learning   45
15     SBNA   15
```

WordCloud

In [62]:

```
#Importing Required Packages
library(wordcloud)
data=read.csv("E:/R Programs For Practice/DAY - 5/Class/Data/file.csv",header= TRUE)
head(data)
#Creates a word cloud with parameters value and frequency
#worcloud(word = dataframe_name$words,dataframe_name$words,min.freq="Least_legth,max
wordcloud(words=data$word,freq=data$freq,min.freq=2,max.words = 50,random.order = FA
```

```
Warning message:
"package 'wordcloud' was built under R version 3.6.3"Loading required package: RColo
rBrewer
```

word	freq
hello	5
hey	14
test	8
photo	2
sathish	9

word	freq
sret	36

S3_Class

In [63]:
```
#s3 class
#A list with its class attribute set to some class name, is an S3 object.
c=list(name="Siva",age=20,cgpa=9.5)
print(class(c))
print(c$name)
```

```
[1] "list"
[1] "Siva"
```

S4_Class

In [64]:
```
#s4 class
#S4 class is defined using the setClass() function.
setClass("student",slots=list(name="character",age="numeric",GPA="numeric"))
#Creating S4 objects
s=new("student",name="john",age=21,GPA=3.5)
print(s)
isS4(s)
```

```
An object of class "student"
Slot "name":
[1] "john"

Slot "age":
[1] 21

Slot "GPA":
[1] 3.5
```

TRUE

Reference CLass

In [65]:

```
#setref
# setRefClass("student")
s=setRefClass("details",fields=list(name="character",age="numeric",GPA="numeric"))
s1=s(name="john",age=21,GPA=3.5)
print(s1)
```

```
Reference class object of class "details"
Field "name":
[1] "john"
Field "age":
[1] 21
Field "GPA":
[1] 3.5
```

DAY - 6

Skewness & Kurtosis

In [66]:

```
#Importing Required Packages
library(e1071)
#Reading Csv file
df = read.csv("E:/R Programs For Practice/DAY - 6/Data/mark.csv")
#skewness
print(skewness(df$mark))
plot(density(df$mark))
print(mean(df$mark))
polygon(density(df$mark), col="red", border="blue")
#kurtosis
df$mark
```

```
Warning message:
"package 'e1071' was built under R version 3.6.3"
[1] -0.4277135
[1] 59.45326
```

1. 9.1
2. 6.7
3. 7.32
4. 7.54
5. 4.79
6. 10.28
7. 17.29
8. 13.39
9. 18.22
10. 11.09

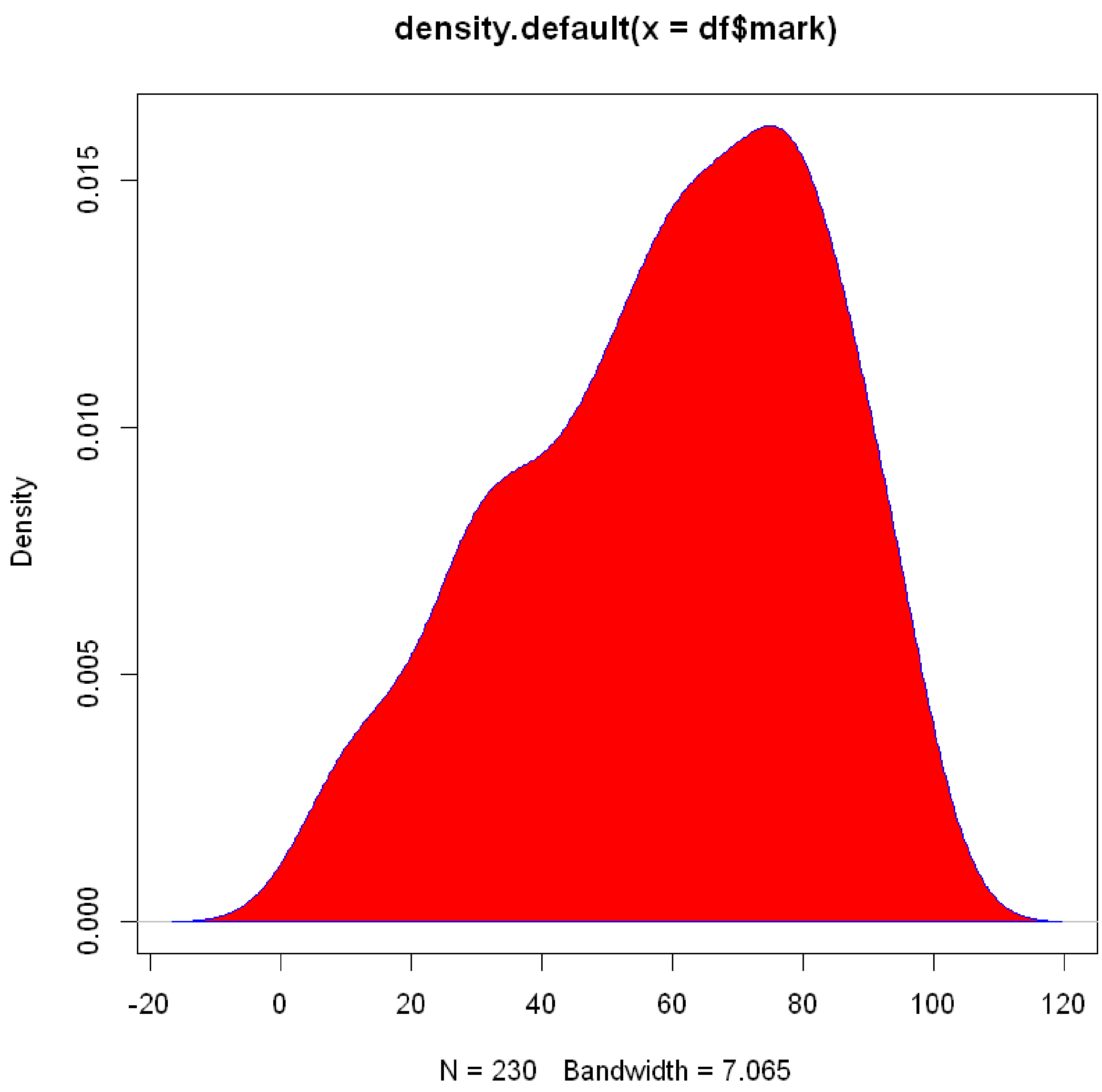
11. 15.05
12. 10.6
13. 14.77
14. 15.48
15. 15.48
16. 28.91
17. 29.51
18. 20.49
19. 28.98
20. 25.17
21. 20.6
22. 25.7
23. 26.73
24. 21.55
25. 27
26. 27.01
27. 25.6
28. 28.39
29. 28.86
30. 23.98
31. 34.19
32. 31.39
33. 32.86
34. 30.98
35. 31.45
36. 31.91
37. 30.04
38. 35.55
39. 39.77
40. 33.71
41. 36.83
42. 35.42
43. 30.6
44. 33.89
45. 37.31
46. 32.98
47. 37.1
48. 31.83
49. 30.89
50. 34.91
51. 49.05
52. 48.4
53. 47.29
54. 42.38
55. 45.74
56. 40.94
57. 47.71

58. 49.49
59. 42.31
60. 46.25
61. 42.91
62. 47.03
63. 43.82
64. 40.6
65. 45.85
66. 49.52
67. 41.68
68. 40.65
69. 42.35
70. 48.1
71. 46.25
72. 49.28
73. 40.59
74. 40.9
75. 40.49
76. 58.27
77. 51.4
78. 56.02
79. 59.48
80. 54.14
81. 53.82
82. 59.73
83. 53.82
84. 56.97
85. 56.53
86. 52.58
87. 57.11
88. 53.82
89. 59.46
90. 51.83
91. 53.4
92. 57.45
93. 53.12
94. 57.74
95. 50.67
96. 56.79
97. 50.91
98. 57.21
99. 51.52
100. 52.61
101. 51.68
102. 57.49
103. 51.91
104. 59.86

105. 58.06
106. 62.97
107. 69.8
108. 62.24
109. 65.36
110. 63.27
111. 62.5
112. 63.98
113. 68.54
114. 63.21
115. 67.77
116. 63.33
117. 62.53
118. 66.49
119. 62.38
120. 66.51
121. 65.31
122. 69.63
123. 65.28
124. 60.62
125. 62.11
126. 62.03
127. 67.26
128. 60.9
129. 64.5
130. 68.76
131. 67.68
132. 65.16
133. 65.11
134. 62.56
135. 60.23
136. 63.65
137. 69.15
138. 61.15
139. 64.98
140. 63.45
141. 78.38
142. 71.06
143. 78.15
144. 75.35
145. 79.72
146. 79.43
147. 76.66
148. 71.78
149. 74.15
150. 77.09
151. 76.14

152. 79.1
153. 74.51
154. 74.97
155. 78.92
156. 79.79
157. 74.91
158. 73.38
159. 78.42
160. 72.32
161. 73.35
162. 77.8
163. 77.36
164. 73.96
165. 71.35
166. 70.05
167. 71.56
168. 74.81
169. 74.18
170. 73.09
171. 74.46
172. 73.33
173. 79.58
174. 73.89
175. 79.49
176. 70.26
177. 73.97
178. 72.88
179. 74.19
180. 76.18
181. 88.59
182. 85.12
183. 87.8
184. 88.36
185. 80.95
186. 84.52
187. 84.94
188. 83.49
189. 81.18
190. 80.8
191. 81.91
192. 84.78
193. 84.72
194. 84.8
195. 80.64
196. 83.59
197. 84.66
198. 83.03

199. 85.99
200. 80.4
201. 80.6
202. 83.05
203. 85.36
204. 81.59
205. 84.13
206. 86.64
207. 80.44
208. 86.07
209. 80.62
210. 86.11
211. 94.4
212. 98.53
213. 91.96
214. 93.94
215. 92.23
216. 93.2
217. 93.75
218. 93.09
219. 91.69
220. 92.18
221. 93.51
222. 96
223. 98.26
224. 94.09
225. 92.48
226. 94.21
227. 92.84
228. 94.42
229. 92.93
230. 91.12



T-Test

In [67]:

```
#T-test
#data
Field1=c(15.2,15.3,16,15.8,15.6,14.9,15,15.4,15.6,15.7,15.5,15.2,15.5,15.1,15.3,15)
Field2=c(15.9,15.9,15.2,16.6,15.2,15.8,15.8,16.2,15.6,15.6,15.8,15.5,15.5,15.5,14.9,
#t.test(field1,field2)
print(t.test(Field1,Field2))
```

```
        Welch Two Sample t-test

data:  Field1 and Field2
t = -2.3388, df = 28.123, p-value = 0.02668
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.56269667 -0.03730333
sample estimates:
mean of x mean of y 
 15.38125  15.68125 
```

T- Test

Formula $= \frac{|\bar{X}_1 - \bar{X}_2|}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$

$\bar{X}_1 \rightarrow$ Sample 1 Mean

$\bar{X}_2 \rightarrow$ Sample 2 Mean

$n_1 \rightarrow$ Sample 1 Count

$n_2 \rightarrow$ Sample 2 Count

$S_1 \rightarrow$ Variance 1

$S_2 \rightarrow$ Variance 2

Data

Score 1 = 3, 3, 3, 12, 15, 16, 17, 19, 23, 24, 32

Score 2 = 20, 13, 13, 20, 29, 32, 23, 20, 25, 15, 30

$\bar{X}_1 = \frac{3+3+3+12+15+16+17+19+23+24+32}{11}$

$= 15.1818$

$\bar{X}_2 = \frac{20+13+13+20+29+32+23+20+25+15+30}{11}$

$= 21.8181$

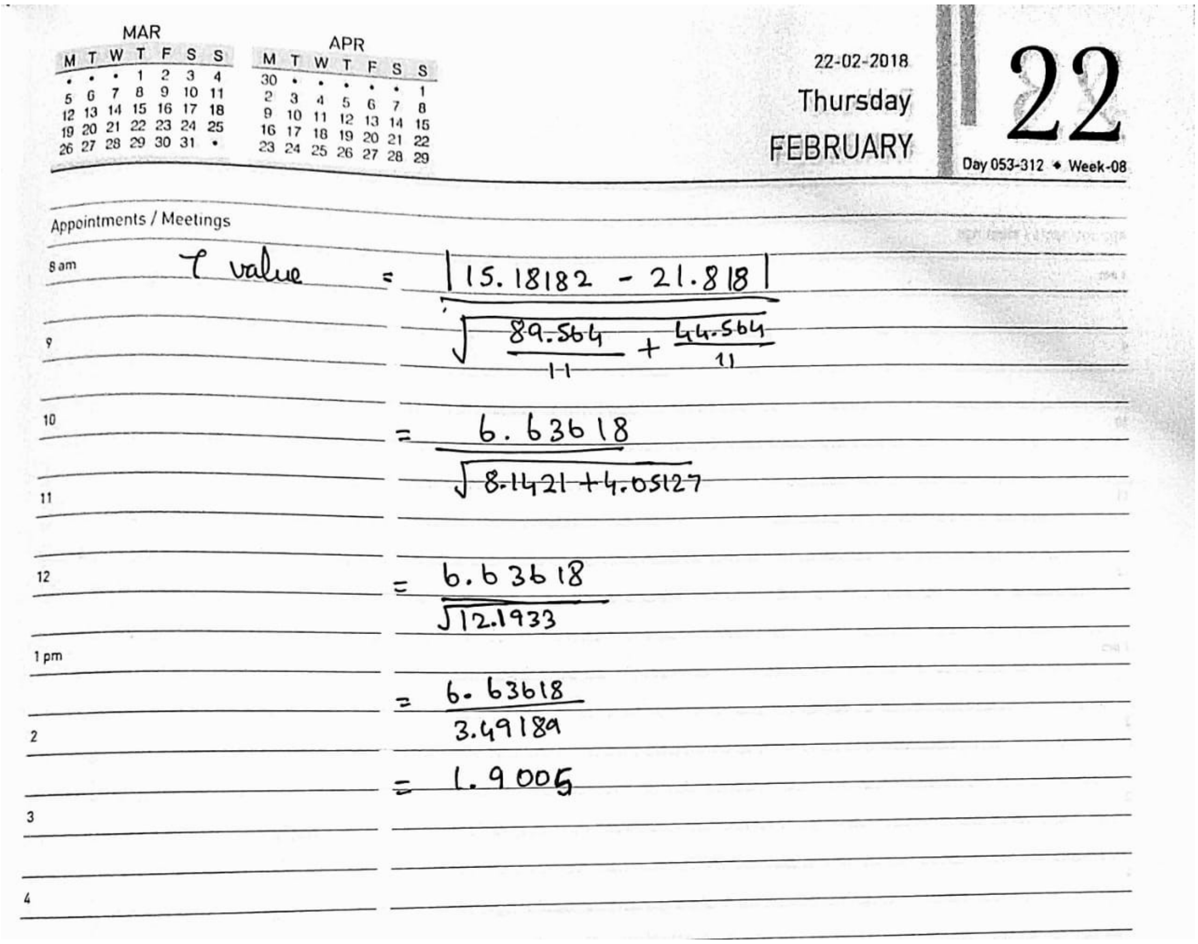
$S_1 = \frac{\Sigma (x_i - \bar{x})^2}{n}$

$= 89.56364$

$S_2 = \frac{\Sigma (x_i - \bar{x}_2)^2}{n_2}$

$= 44.56364$

$n_1 = 11$, $n_2 = 11$



Anova - Test

In [68]:
```
#Anova Test
group1<-c(2,3,7,2,6)
group2<-c(10,8,7,5,10)
group3<-c(10,13,14,13,15)
#Making data as datafrane
combined_groups <-data.frame(cbind(group1,group2,group3))
combined_groups
#stacking into one column
stack_group = stack(combined_groups)
stack_group
#applying aov anova function
anova  = aov(values~ind,data = stack_group)
summary(anova)
```

group1	group2	group3
2	10	10
3	8	13
7	7	14
2	5	13
6	10	15

values	ind
2	group1
3	group1

values	ind
7	group1
2	group1
6	group1
10	group2
8	group2
7	group2
5	group2
10	group2
10	group3
13	group3
14	group3
13	group3
15	group3

```
            Df Sum Sq Mean Sq F value   Pr(>F)    
ind          2  203.3   101.7   22.59 8.54e-05 ***
Residuals   12   54.0     4.5                     
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

DAY - 7

Probability Distribution

1.binomial distribution

2.poisson distribution

3.continuos uniform

4.exponential dis

5.normal

6.chi-square

7.t dis

8.f-distribution

Binomial Distribution

In [69]:
```
#binomial distribution
#Suppose there are twelve multiple choice questions in an English class quiz.
#Each question has five possible answers, and only one of them is correct.
```

```
#Find the probability of having four or less correct answers if a student attempts t
pbinom(7,size=10,prob=0.5)
```

0.9453125

Poisson Distribution

In [70]:

```
#Poisson distribution
#If there are twelve cars crossing a bridge per minute on average,
#find the probability of having seventeen or more cars crossing the bridge in a part
ppois(16,lambda=12,lower=FALSE)
```

0.101291007439838

Chi Square Distribution

In [71]:

```
#chi-square Distribution
#Find the 95th percentile of the Chi-Squared distribution with 7 degrees of freedom.
qchisq(.95, df=7)
```

14.0671404493402

Continuos uniform distribution

In [72]:

```
#Continuos uniform distribution
#Select ten random numbers between one and three.
runif(10, min=1, max=3)
```

1. 1.37538223853335
2. 2.56458860263228
3. 1.18718997342512
4. 1.93355808313936
5. 2.02301091980189
6. 2.19997791852802
7. 1.66564708063379
8. 1.97722606733441
9. 2.90894765499979
10. 1.9658047943376

Exponential Distribution

In [73]:

```
#Exponential Distribution
#Select ten random numbers between one and three.Suppose the mean checkout time of a
#Find the probability of a customer checkout being completed by the cashier in less
pexp(2,rate=1/3)
```

0.486582880967408

T-Distribution

In [74]:

```
#t Distribution
#Find the 2.5th and 97.5th percentiles of the Student t distribution with 5 degrees
qt(c(.025, .975), df=5)
```

1. -2.57058183563631
2. 2.57058183563631

F-Distribution

In [75]:
```
#F-Distribution
#Find the 95th percentile of the F distribution with (5, 2) degrees of freedom.
qf(.95, df1=5, df2=2)
```

19.2964096520172

Summary

In [76]:
```
# 1) The coin is flipped ten times. Find the probability of 7 heads occurring.
# 2.A card is selected three times (and replaced). Find the probability of 2 face ca
# 3. A student decides to guess on a section of his ACT test. The section contains 5
# a) Find the expected number of correct responses.
# 4. A company ships 5000 cell phones. They are expected to last an average of 10,00
#a) after 11,000 hours
#ex1
pbinom(7,size=10,prob=0.5)
#ex2
pbinom(2,size=3,prob=1/52)
#ex3
(pbinom(4,size=50,prob=0.2))
#ex4
pnorm(11000,mean=10000,sd=500,lower.tail=FALSE)
```

0.9453125

0.999992888029131

0.0184960150602093

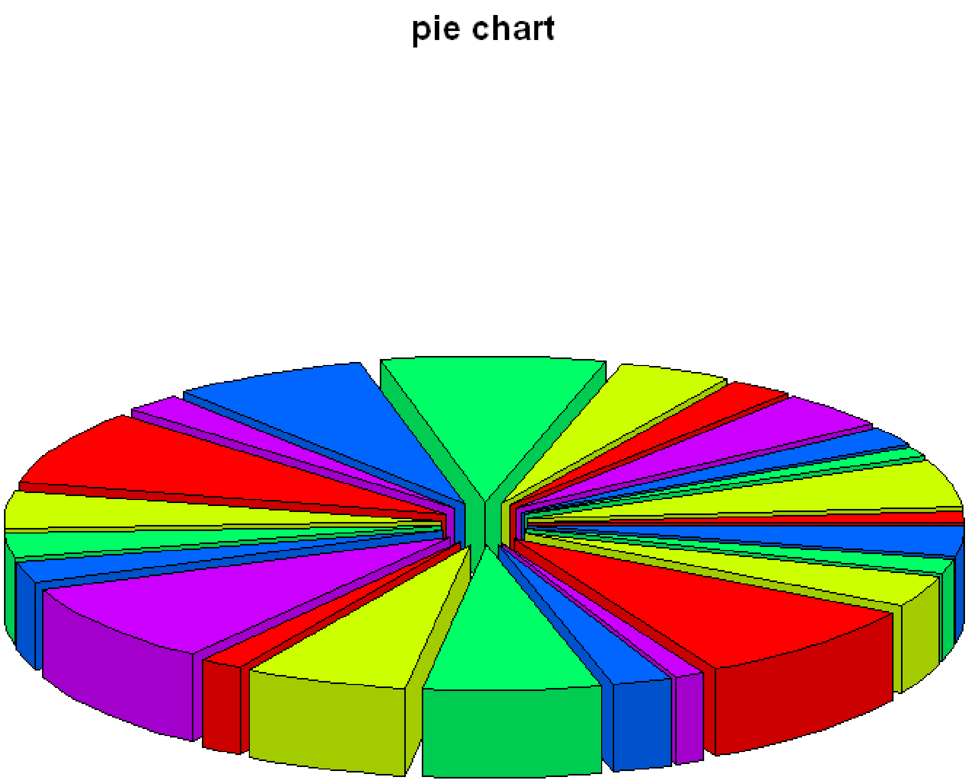
0.0227501319481792

Multiple - TimeSeriesGraph

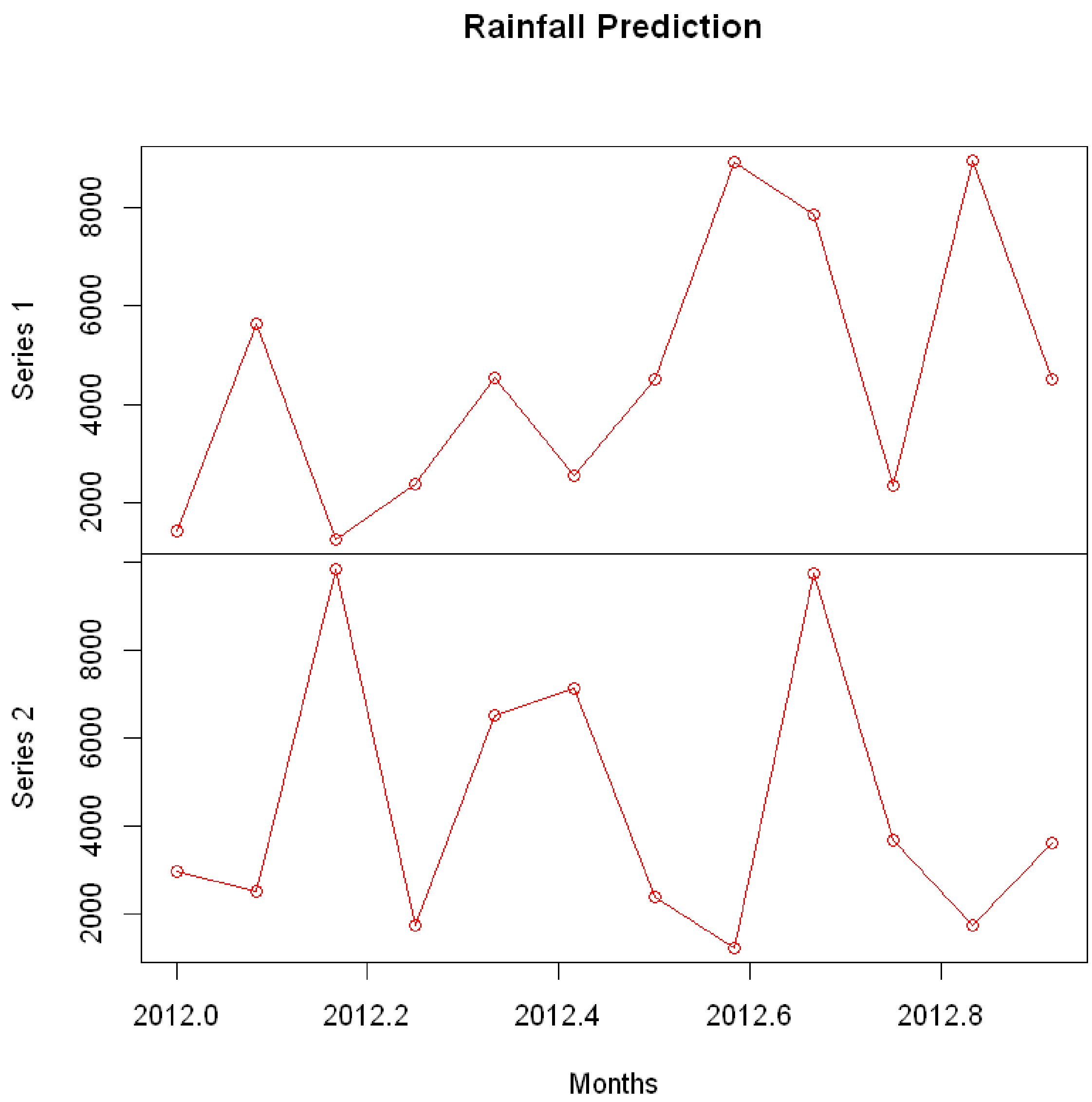
In [77]:
```
library(plotrix)
library(ggplot2)
vec1 <- c(1400,5642,1245,2356,4526,2536,4512,8945,7856,2345,8956,4512)
vec2 <- c(2980,2542,9845,1756,6526,7136,2412,1245,9756,3675,1756,3612)
mat=matrix(c(vec1,vec2),nrow=12)
result = ts(mat,c(2012,1),frequency = 12)
print(result)
```

```
         Series 1 Series 2
Jan 2012     1400     2980
Feb 2012     5642     2542
Mar 2012     1245     9845
Apr 2012     2356     1756
May 2012     4526     6526
Jun 2012     2536     7136
Jul 2012     4512     2412
Aug 2012     8945     1245
Sep 2012     7856     9756
Oct 2012     2345     3675
Nov 2012     8956     1756
Dec 2012     4512     3612
```

In [78]:
```
pie3D(result,main="pie chart",explode = 0.1,col=rainbow(5))
```



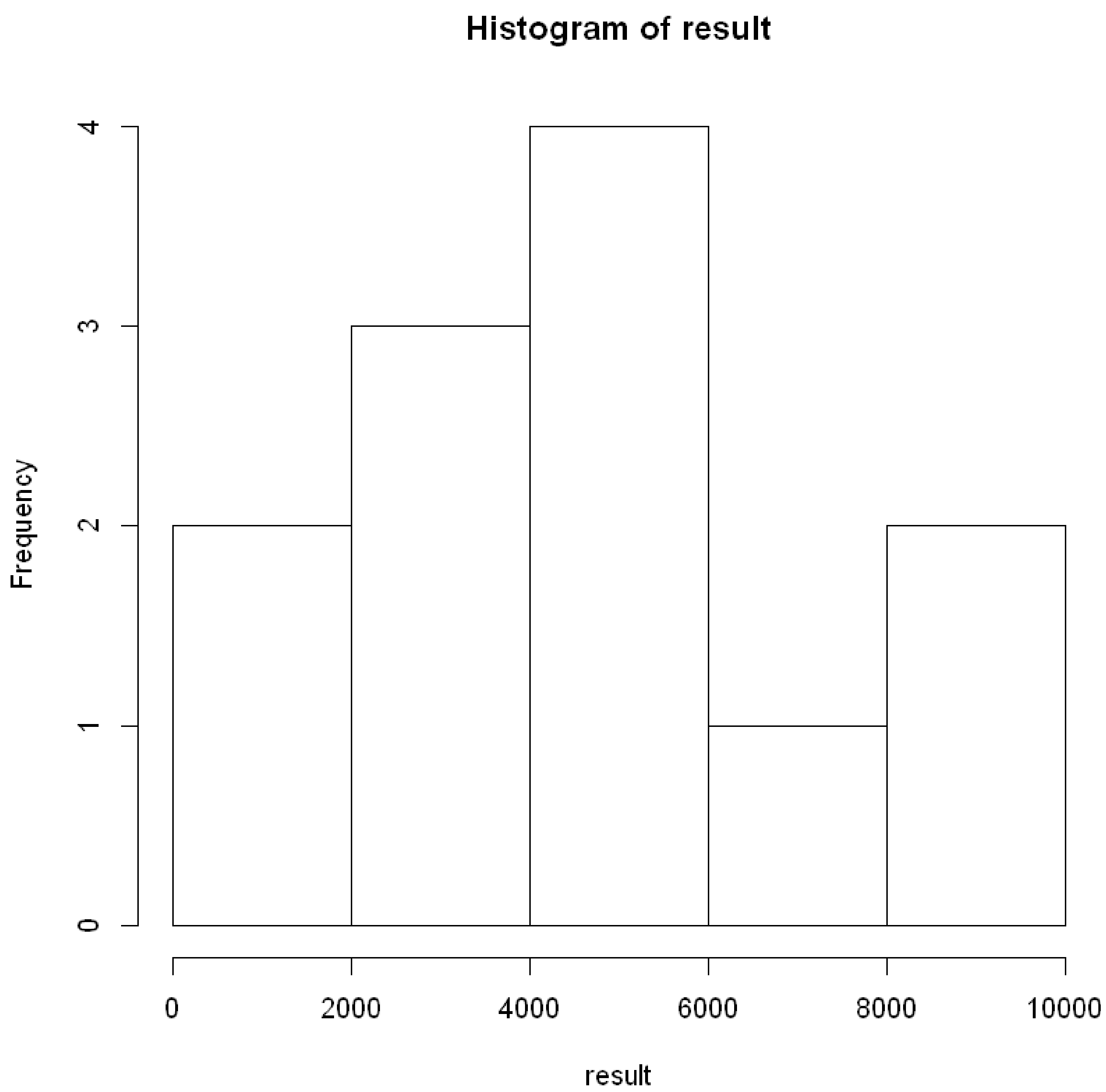

```
In [79]: plot(result,type = "o",col = "red", xlab = "Months", ylab = "Rainfal",
              main = "Rainfall Prediction")
```



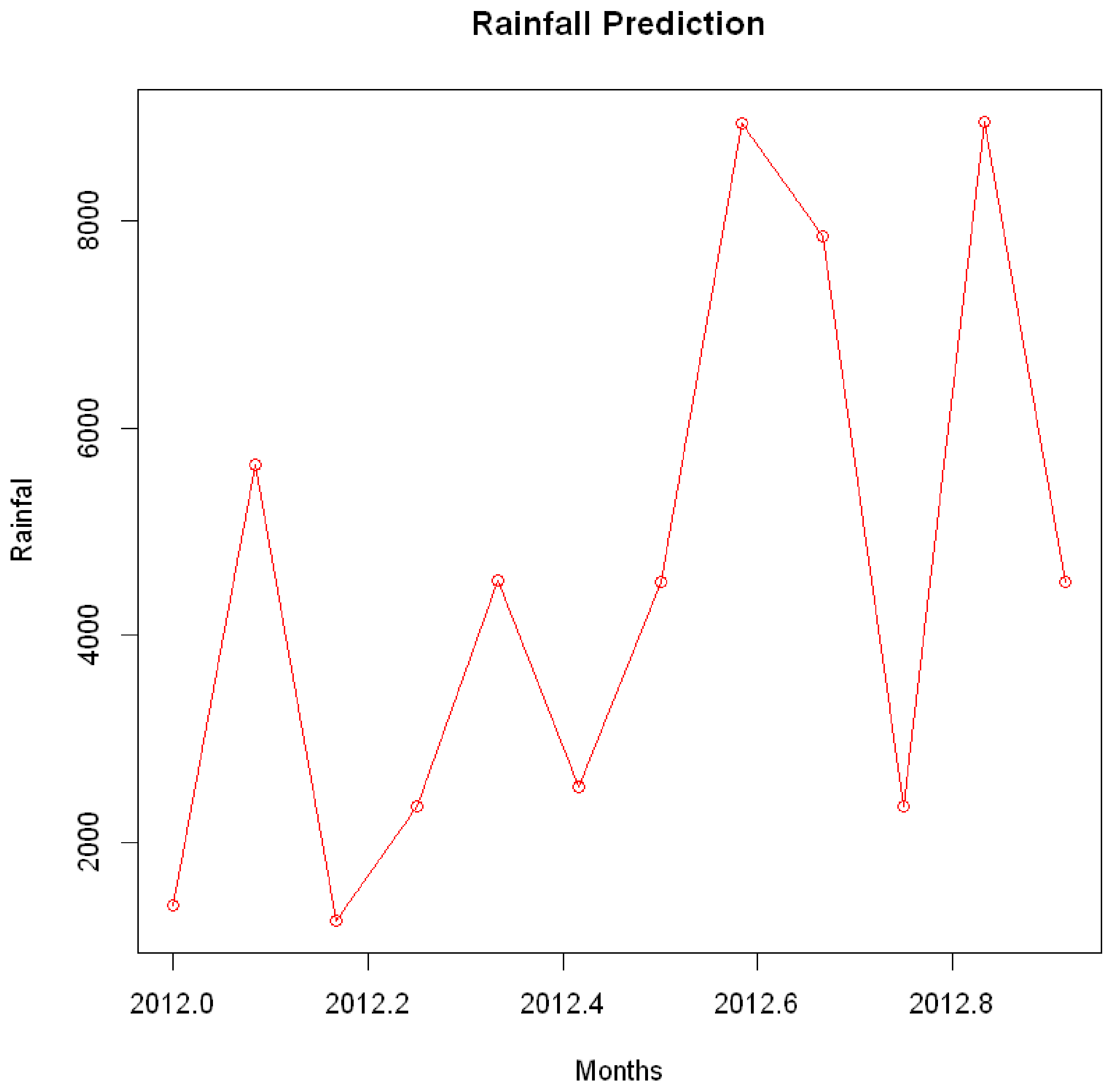
In [80]:
```
library(plotrix)
library(ggplot2)
vec1 <- c(1400,5642,1245,2356,4526,2536,4512,8945,7856,2345,8956,4512)
result = ts(vec1,c(2012,1),frequency = 12)
print(result)
```

```
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
2012 1400 5642 1245 2356 4526 2536 4512 8945 7856 2345 8956 4512
```

In [81]:
```
hist(result)
```



In [82]:
```
plot(result,type = "o",col = "red", xlab = "Months", ylab = "Rainfal",
     main = "Rainfall Prediction")
```



DAY - 8

Correlation

In [83]:

```
#Correlation
HoursStudied <- c(8,11,3,6,14,9,2,0,7,13,10,4,9)
ExamMark <- c(82,94,70,75,98,80,68,53,76,87,89,83,72)
#various types of correlation
cor(HoursStudied,ExamMark,method = "pearson")
cor(HoursStudied,ExamMark,method = "kendall")
cor(HoursStudied,ExamMark,method = "spearman")
#Gives a summary of pearson correlation
cor.test(HoursStudied,ExamMark, method=c("pearson", "kendall", "spearman"))
```

0.860462439748913

0.7096921893772

0.850069579975727

```
	Pearson's product-moment correlation

data:  HoursStudied and ExamMark
t = 5.6011, df = 11, p-value = 0.0001601
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
```

```
 0.5884739 0.9574966
sample estimates:
      cor
0.8604624
```

In [84]:

```
mtcars
#Input from mtcars
mpg = mtcars$mpg
cyl = mtcars$cyl
#Correlation Summary
cor.test(mpg,cyl)
```

	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
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

```
	Pearson's product-moment correlation

data:  mpg and cyl
t = -8.9197, df = 30, p-value = 6.113e-10
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.9257694 -0.7163171
sample estimates:
      cor
-0.852162
```

Linear Regression

In [85]:
```
#predict
HoursStudied <- c(8,11,3,6,14,9,2,0,7,13,10,4,9,10)
#response
ExamMark <- c(82,94,70,75,98,80,68,53,76,87,89,83,72,85.3494)
```

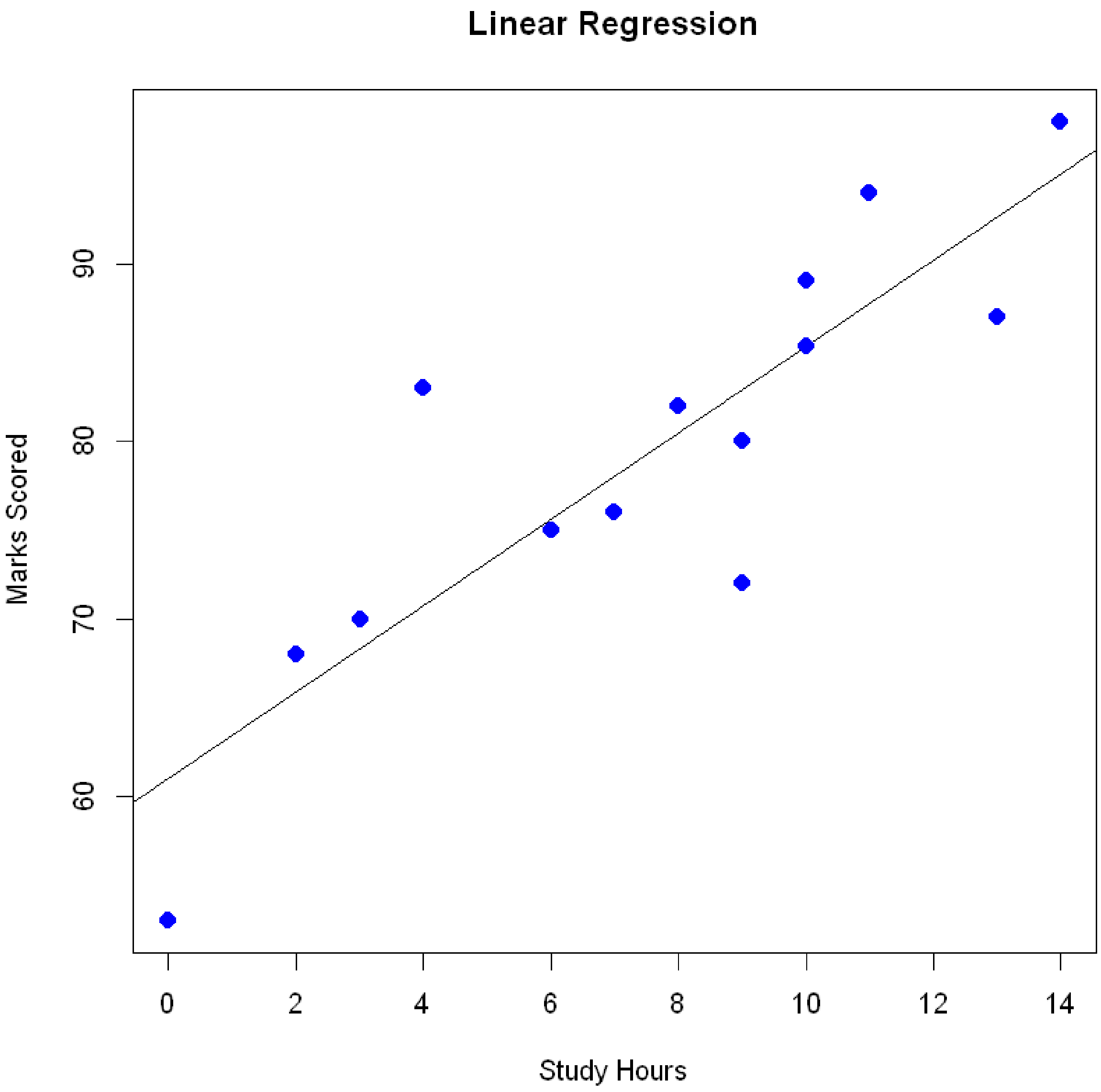
In [86]:
```
#Linear model
relation = lm(ExamMark~HoursStudied)
summary(relation)
```

```
Call:
lm(formula = ExamMark ~ HoursStudied)

Residuals:
    Min      1Q  Median      3Q     Max
-10.922  -2.708   0.753   2.723  12.217

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    61.072      3.502  17.441 6.85e-10 ***
HoursStudied    2.428      0.409   5.935 6.87e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.114 on 12 degrees of freedom
Multiple R-squared:  0.7459,	Adjusted R-squared:  0.7247
F-statistic: 35.23 on 1 and 12 DF,  p-value: 6.87e-05
```

In [87]:
```
#Assigning x value to our model
result = data.frame(HoursStudied = 10)
final = predict(relation,result)
final
```

1: 85.349397826087

In [88]:
```
#Visualizing
plot(HoursStudied,ExamMark,col = "blue",main = "Linear Regression",
     abline(lm(ExamMark~HoursStudied)),cex = 1.3,pch = 16,
     xlab = "Study Hours",ylab = "Marks Scored")
```



Multi Linear Regression

In [89]:
```
#mulitiple regression
weight=c(1.4,2.8,3.4)
size=c(1.2,2.4,3.4)
tail=c(0.9,1,0.8)
```

In [90]:
```
model=lm(size~weight+tail)
summary(model)
```

```
Call:
lm(formula = size ~ weight + tail)

Residuals:
ALL 3 residuals are 0: no residual degrees of freedom!

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)      1.6         NA      NA       NA
weight           1.0         NA      NA       NA
tail            -2.0         NA      NA       NA

Residual standard error: NaN on 0 degrees of freedom
Multiple R-squared:      1,     Adjusted R-squared:    NaN
F-statistic:   NaN on 2 and 0 DF,  p-value: NA
```

In [91]:
```
res=data.frame(weight=2.5,tail=0.8)
fin=predict(model,res)
print(fin)
```

```
  1
2.5
```

One Sampled & Two Sampled T-Test

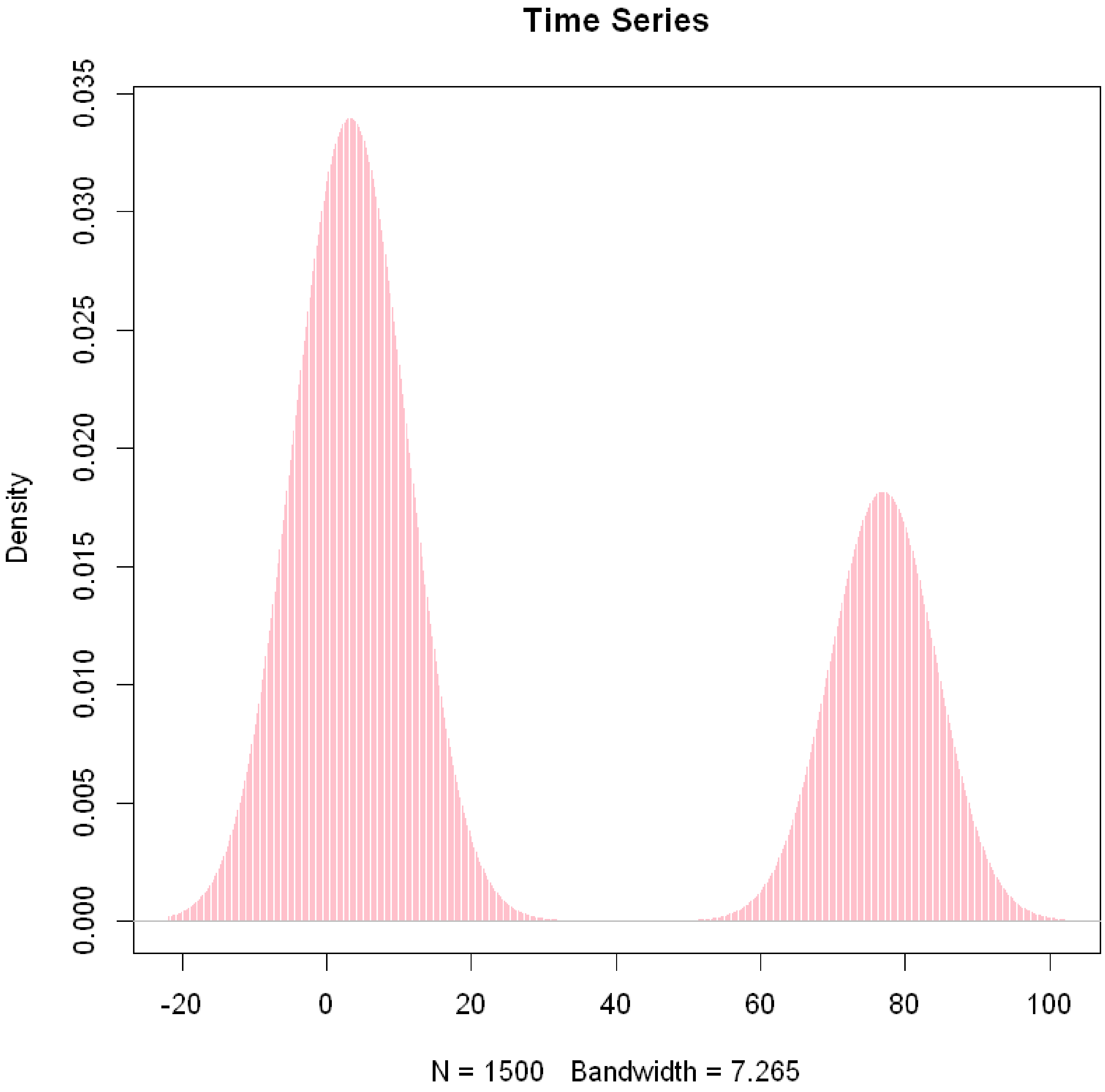
In [92]:
```
preferred<-c(12,7,11,13,10)
Nonpreffered<-c(7,9,8,10,9)
t.test(preferred,Nonpreffered,conf.level=0.95)

t.test(preferred,y = NULL,conf.level=0.95,mu = 0,paired = FALSE,var.equal = FALSE)

t.test(preferred)
```

```
	Welch Two Sample t-test

data:  preferred and Nonpreffered
t = 1.7408, df = 5.8509, p-value = 0.1336
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.8287445  4.8287445
sample estimates:
mean of x mean of y 
     10.6       8.6 
	One Sample t-test

data:  preferred
t = 10.296, df = 4, p-value = 0.000502
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  7.741475 13.458525
sample estimates:
mean of x 
     10.6 
	One Sample t-test

data:  preferred
t = 10.296, df = 4, p-value = 0.000502
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  7.741475 13.458525
sample estimates:
mean of x 
     10.6 
```

Time Series

In [93]:
```
set.seed(13)

mydata1 = rnorm(500,6)
mydata2 = rnorm(500,77)
mydata3 = runif(500)

mydata=data.frame(mydata1,mydata2,mydata3)

mydataMatrix = as.matrix(mydata)
# print(mydataMatrix)

myts = ts(mydata,start=c(1980,5),frequency=12)
print(head(myts))

print(plot(density(myts),col="pink",type="h",main="Time Series"))
```

```
      mydata1  mydata2    mydata3
[1,] 6.554327 77.91682 0.27864352
[2,] 5.719728 75.31737 0.01656612
[3,] 7.775163 76.93840 0.86765504
[4,] 6.187320 78.83783 0.47517723
[5,] 7.142526 75.75849 0.89428873
[6,] 6.415526 78.30384 0.93067461
NULL
```



Train_Test_Split

In [94]:
```
set.seed(10)
#Package for test-test split
library(caTools)
#Splitting for test-train 0.8/0.2
s=sample.split(mtcars,SplitRatio = 0.8)
```

```
Warning message:
"package 'caTools' was built under R version 3.6.3"
```

In [95]:
```
#Gives output as True False
train=subset(mtcars,split=TRUE)
test=subset(mtcars,split=FALSE)
```

In [96]:
```
#linear model
model=lm(mtcars$mpg~.,data=train)
```

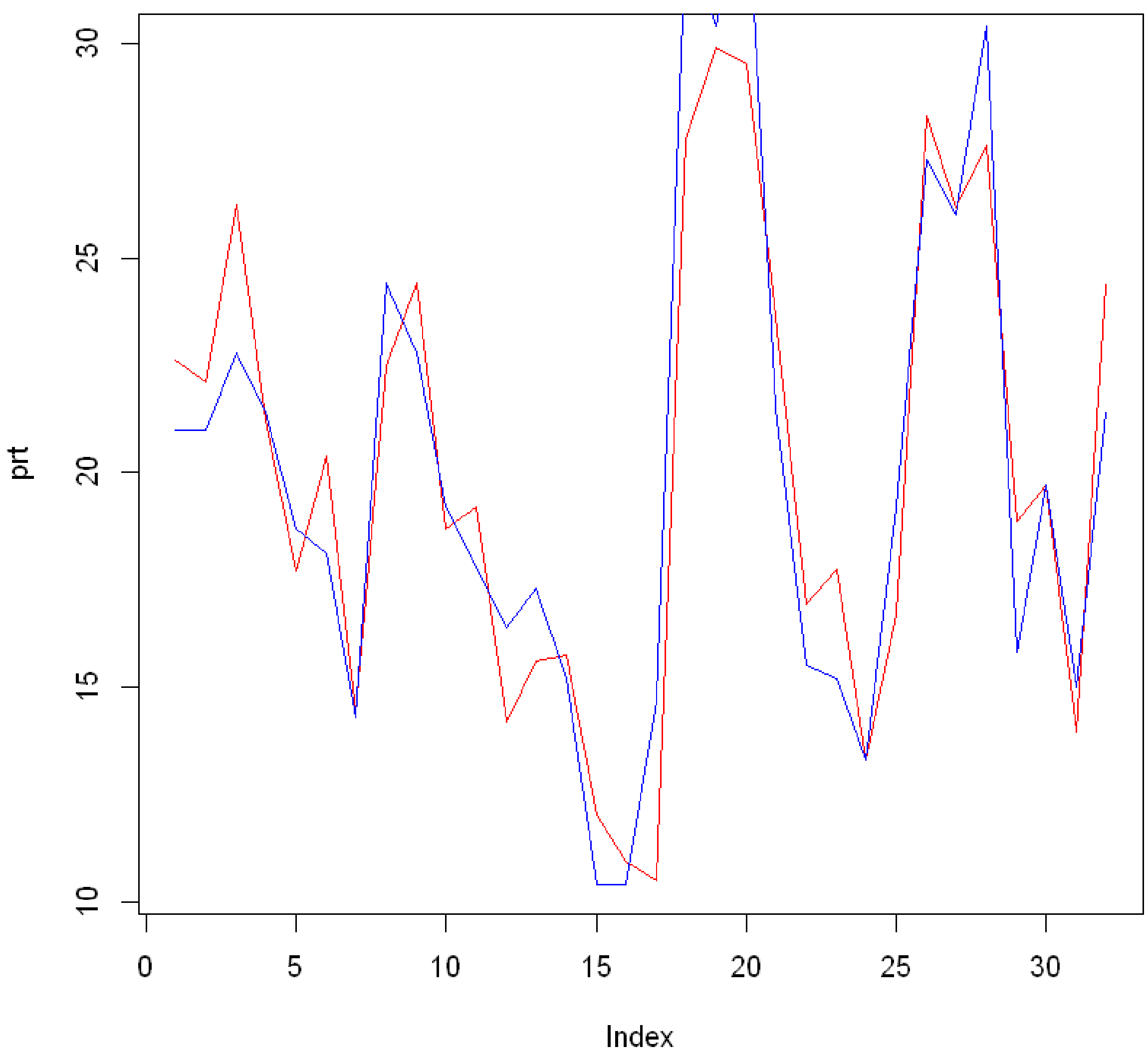
```
prt=predict(model,test)
print(prt)
```

```
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive 
           22.59951            22.11189            26.25064            21.23740 
  Hornet Sportabout             Valiant          Duster 360           Merc 240D 
           17.69343            20.38304            14.38626            22.49601 
           Merc 230            Merc 280           Merc 280C          Merc 450SE 
           24.41909            18.69903            19.19165            14.17216 
         Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln Continental 
           15.59957            15.74222            12.03401            10.93644 
  Chrysler Imperial            Fiat 128         Honda Civic      Toyota Corolla 
           10.49363            27.77291            29.89674            29.51237 
      Toyota Corona    Dodge Challenger         AMC Javelin          Camaro Z28 
           23.64310            16.94305            17.73218            13.30602 
   Pontiac Firebird           Fiat X1-9       Porsche 914-2        Lotus Europa 
           16.69168            28.29347            26.15295            27.63627 
     Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E 
           18.87004            19.69383            13.94112            24.36827 
```

In [97]:

```
#plotting line chart
plot(prt,type='l',col="red")
lines(mtcars$mpg,type="l",col="blue")

print(mtcars$mpg~.)
```

```
mtcars$mpg ~ .
```



DAY - 9

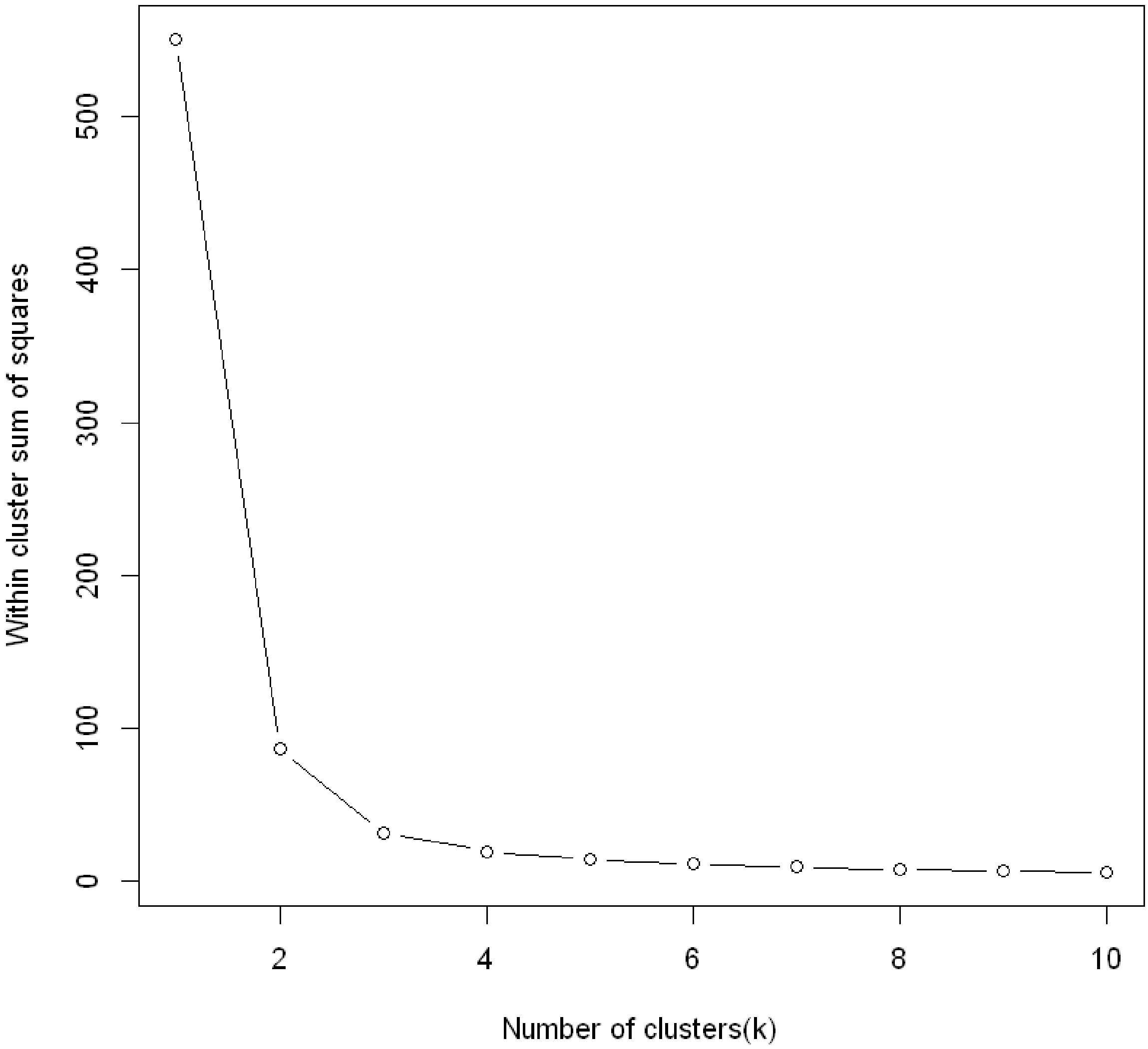
Elbow Method

In [98]:

```
df <- iris
head(df)
#Plotting
set.seed(27)
set.seed(200)
k.max <- 10
#nstart       - No. of  random sets should be chosen?
#Total within - cluster sum of squares, i.e. sum(withinss).
#sapply       - takes list, vector or data frame as input and gives output in vector
wss<- sapply(1:k.max,function(k){kmeans(iris[,3:4],k,nstart = 20,iter.max = 20)$tot.
class(wss)
plot(1:k.max,wss, type= "b", xlab = "Number of clusters(k)",ylab = "Within cluster s
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

'numeric'



K - Means Clustering

Introduction

****

- **K - Means Clustering make inferences from data using only input vectors without labelled outcomes.**

- **K - means clustering makes partition of "n" observations into "k" clusters in which each observation belongs to the cluster with the nearest centroid**

- **A cluster refers to a collection of data points aggregated together because of certain similarities**
  **
**

Applications Of K - Means Clustering

- **Academic performance - Based on the scores, students are categorized into grades like A, B, or C.**

- **Search engines - Clustering forms a backbone of search engines. When a search is performed, the search results need to be grouped, and the search engines use clustering to do this.**

- **Customer segmentation - It helps marketers improve their customer base, work on target areas, and segment customers based on purchase history, interests, or activity monitoring.**
  ****

Example For clustering

```
</b>

A bank wants to give credit card offers to its customers. Currently, they
look at the details of each customer and based on this information they
decide which offer should be given to which customer.

The bank will have millions of customers. So here clustering plays a vital
role in segregating customers into different groups. For instance, the bank
can group the customers based on their income:
```

Steps to perform K - Means Clustering

1. **Specify the number of clusters (K) to be created**

2. **Select randomly (K) objects from the dataset as the initial cluster centroids**

3. **Assign each observation to their closest centroid, based on the Euclidean distance between the object and the centroid**

4. For each of the k clusters update the cluster centroid by calculating the new mean values of all the data points in the cluster.

5. Iteratively minimize the total within sum of square. That is, iterating steps 3 and 4 until the cluster assignments stop changing

Importing Packages

In [99]:
```
library(factoextra)
library(ggplot2)
library(cluster)
```

```
Warning message:
"package 'factoextra' was built under R version 3.6.3"Welcome! Want to learn more? S
ee two factoextra-related books at https://goo.gl/ve3WBa
Warning message:
"package 'cluster' was built under R version 3.6.3"
```

Loading the Dataset

USArrests - This data set contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas.

In [100…
```
#Inbuilt Dataset
df = USArrests
df
```

	Murder	Assault	UrbanPop	Rape
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0
Arkansas	8.8	190	50	19.5
California	9.0	276	91	40.6
Colorado	7.9	204	78	38.7
Connecticut	3.3	110	77	11.1
Delaware	5.9	238	72	15.8
Florida	15.4	335	80	31.9
Georgia	17.4	211	60	25.8
Hawaii	5.3	46	83	20.2
Idaho	2.6	120	54	14.2
Illinois	10.4	249	83	24.0
Indiana	7.2	113	65	21.0
Iowa	2.2	56	57	11.3
Kansas	6.0	115	66	18.0

	Murder	Assault	UrbanPop	Rape
Kentucky	9.7	109	52	16.3
Louisiana	15.4	249	66	22.2
Maine	2.1	83	51	7.8
Maryland	11.3	300	67	27.8
Massachusetts	4.4	149	85	16.3
Michigan	12.1	255	74	35.1
Minnesota	2.7	72	66	14.9
Mississippi	16.1	259	44	17.1
Missouri	9.0	178	70	28.2
Montana	6.0	109	53	16.4
Nebraska	4.3	102	62	16.5
Nevada	12.2	252	81	46.0
New Hampshire	2.1	57	56	9.5
New Jersey	7.4	159	89	18.8
New Mexico	11.4	285	70	32.1
New York	11.1	254	86	26.1
North Carolina	13.0	337	45	16.1
North Dakota	0.8	45	44	7.3
Ohio	7.3	120	75	21.4
Oklahoma	6.6	151	68	20.0
Oregon	4.9	159	67	29.3
Pennsylvania	6.3	106	72	14.9
Rhode Island	3.4	174	87	8.3
South Carolina	14.4	279	48	22.5
South Dakota	3.8	86	45	12.8
Tennessee	13.2	188	59	26.9
Texas	12.7	201	80	25.5
Utah	3.2	120	80	22.9
Vermont	2.2	48	32	11.2
Virginia	8.5	156	63	20.7
Washington	4.0	145	73	26.2
West Virginia	5.7	81	39	9.3
Wisconsin	2.6	53	66	10.8
Wyoming	6.8	161	60	15.6

In [101…

```
summary(df)
```

```
     Murder          Assault          UrbanPop          Rape
```

```
Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30
1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07
Median : 7.250   Median :159.0   Median :66.00   Median :20.10
Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23
3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18
Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00
```

Data Preprocessing

Scaling data - To avoid biased results

In [102…

```r
df <- scale(USArrests) # Scaling the data
head(df)
```

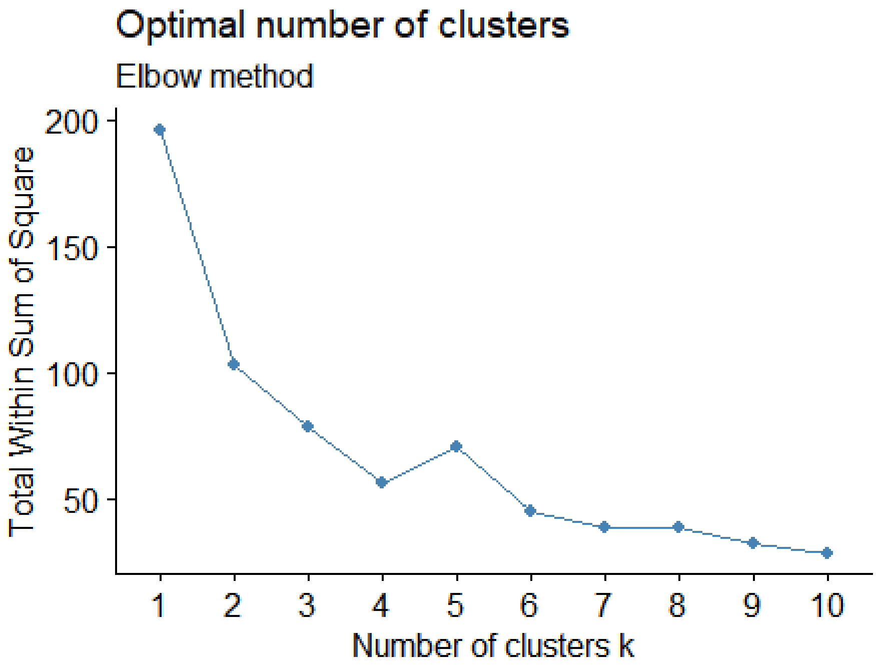
	Murder	Assault	UrbanPop	Rape
Alabama	1.24256408	0.7828393	-0.5209066	-0.003416473
Alaska	0.50786248	1.1068225	-1.2117642	2.484202941
Arizona	0.07163341	1.4788032	0.9989801	1.042878388
Arkansas	0.23234938	0.2308680	-1.0735927	-0.184916602
California	0.27826823	1.2628144	1.7589234	2.067820292
Colorado	0.02571456	0.3988593	0.8608085	1.864967207

STEP 1 : FINDING NUMBER OF K VALUES

To find the appropriate suitable K value:

1. **Compute k-means clustering using different values of clusters k.**
2. **Next, the wss (within sum of square) is drawn according to the number of clusters.**
3. **The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.**

After Performing Elbow Method

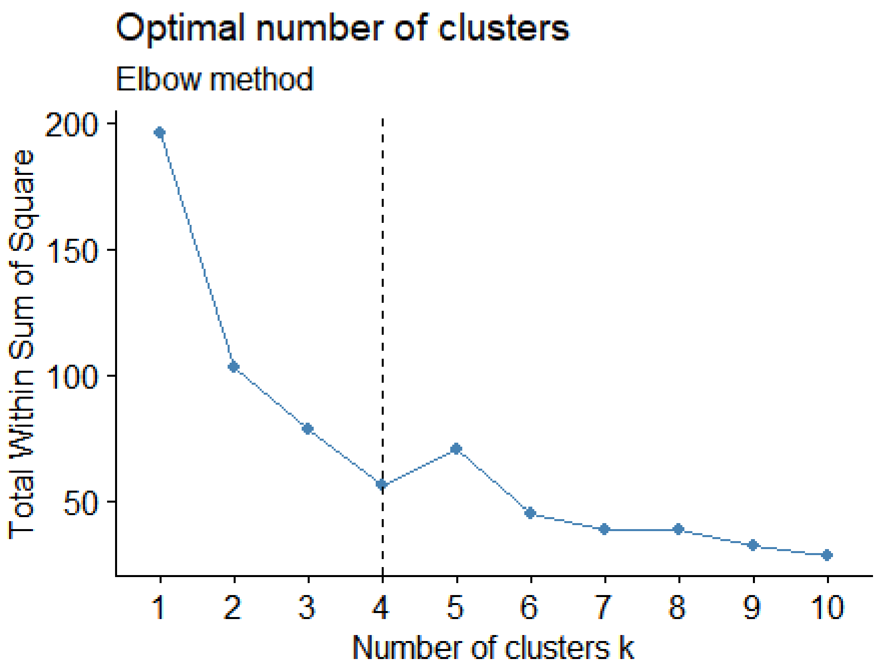


The Elbow method looks at the total within-cluster sum of square (WSS) as a function with respect to the number of clusters.

The location of a knee in the plot is usually considered as an indicator of the appropriate number of clusters because it means that adding another cluster does not improve much better the partition

So Here k = 4 is a best fit and it points out position of elbow

code : fviz_nbclust(df, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 3)+ labs(subtitle = "Elbow method")



Compute k-means with k = 4

In [103…

```
set.seed(27)
kmcluster <- kmeans(df, 4, nstart = 25)
```

set.seed() function in order to set a key for random number generator.As k-means clustering algorithm starts with k randomly selected centroids
Syntax <- kmeans("dataframe_name",Number of clusters,nstart = no. of different random starting choices)

In [104…

```
kmcluster
```

```
K-means clustering with 4 clusters of sizes 16, 13, 8, 13

Cluster means:
      Murder    Assault   UrbanPop        Rape
1 -0.4894375 -0.3826001  0.5758298 -0.26165379
2  0.6950701  1.0394414  0.7226370  1.27693964
3  1.4118898  0.8743346 -0.8145211  0.01927104
4 -0.9615407 -1.1066010 -0.9301069 -0.96676331

Clustering vector:
       Alabama         Alaska        Arizona       Arkansas     California 
             3              2              2              3              2 
      Colorado    Connecticut       Delaware        Florida        Georgia 
             2              1              1              2              3 
        Hawaii          Idaho       Illinois        Indiana           Iowa 
             1              4              2              1              4 
        Kansas       Kentucky      Louisiana          Maine       Maryland 
             1              4              3              4              2 
 Massachusetts       Michigan      Minnesota    Mississippi       Missouri 
             1              2              4              3              2 
       Montana       Nebraska         Nevada  New Hampshire     New Jersey 
             4              4              2              4              1 
    New Mexico       New York North Carolina   North Dakota           Ohio 
             2              2              3              4              1 
      Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina 
             1              1              1              1              3 
  South Dakota      Tennessee          Texas           Utah        Vermont 
             4              3              2              1              4 
      Virginia     Washington  West Virginia      Wisconsin        Wyoming 
             1              1              4              4              1 
```

```
Within cluster sum of squares by cluster:
[1] 16.212213 19.922437  8.316061 11.952463
 (between_SS / total_SS =  71.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

In [105…

```
table(kmcluster$tot.withinss)
```

```
56.4031734582928
               1
```

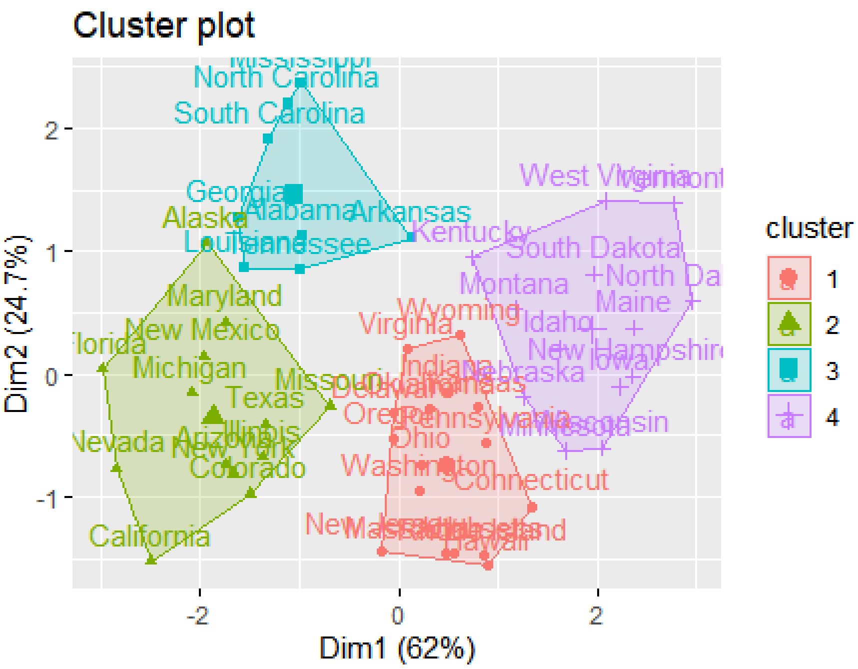
In [106…

```
table(kmcluster$cluster)
```

```
 1  2  3  4
16 13  8 13
```

Plotting & Visulaization

Code :fviz_cluster(kmcluster, data = df)



Iris Dataset

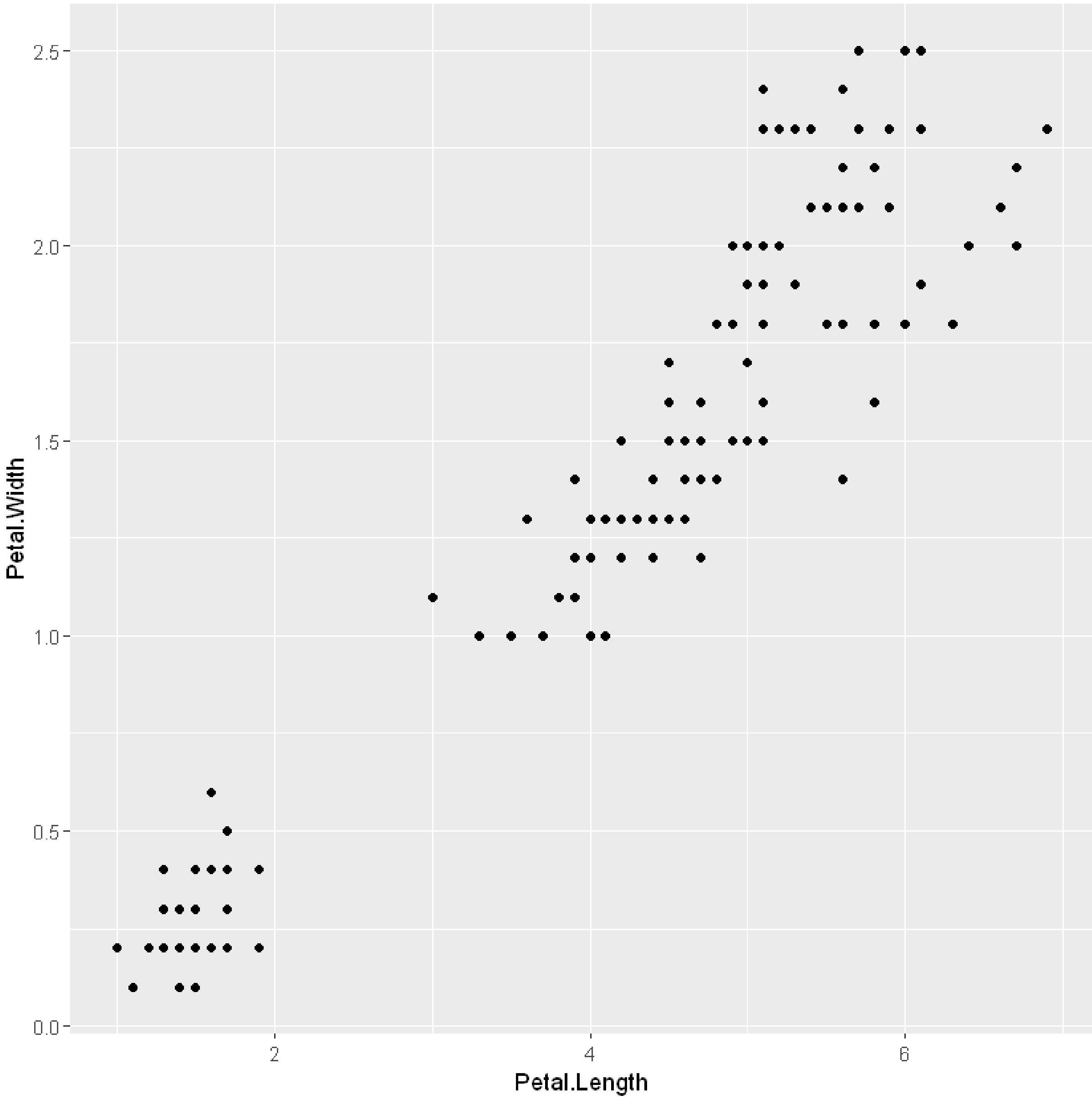
In [107…

```
head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

In [108…

```
ggplot(iris, aes(Petal.Length, Petal.Width)) + geom_point()
```

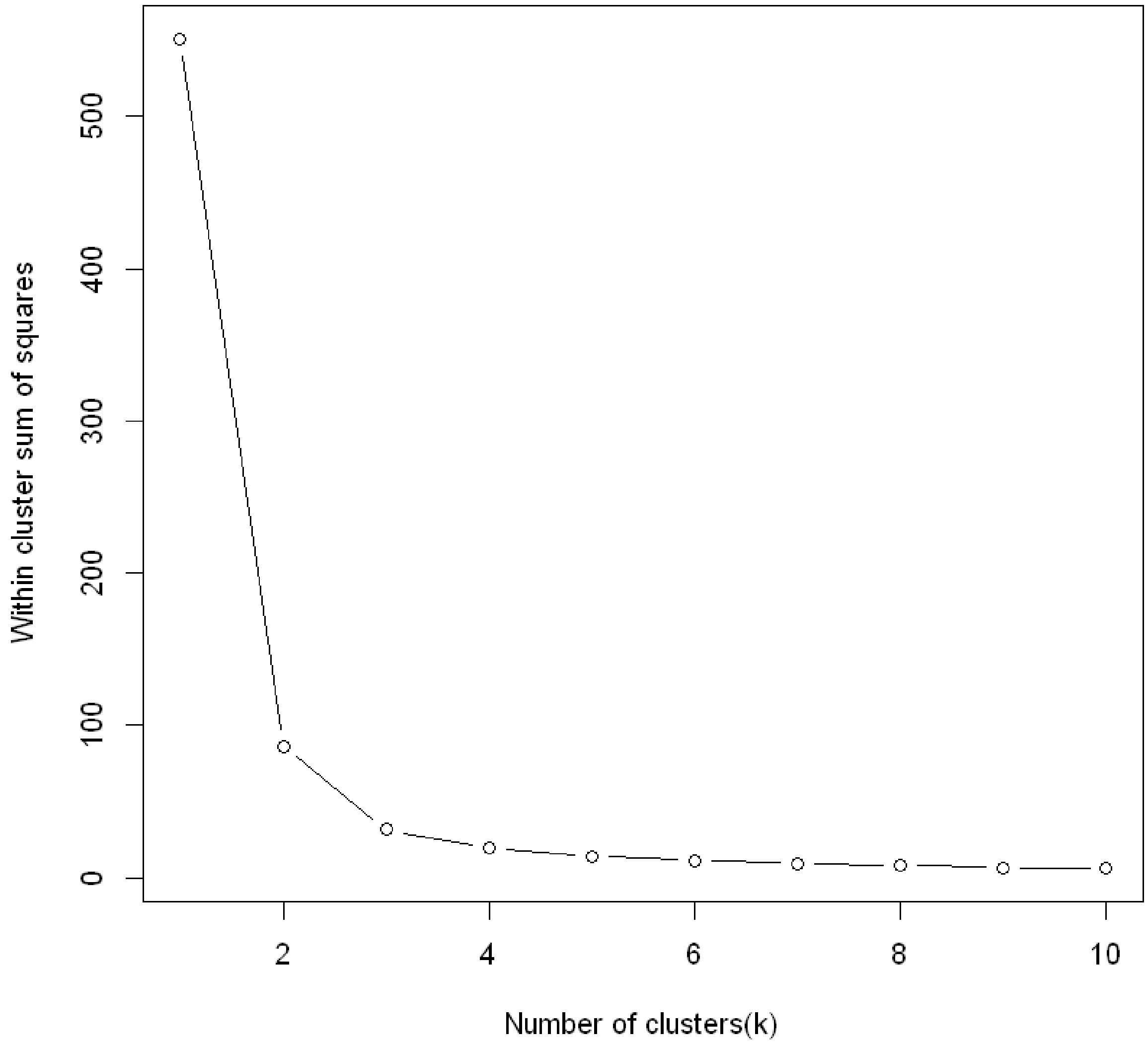


To Find optimal number of clusters

Elbow Method

In [109…

```
set.seed(27)
k.max <- 10

#nstart       - No. of  random sets should be chosen?
#Total within - cluster sum of squares, i.e. sum(withinss).
#sapply       - takes list, vector or data frame as input and gives output in vector
#iter. max is the number of times the algorithm will repeat the cluster assignment a
#nstart is the number of times the initial starting points are re-sampled.

wss<- sapply(1:k.max,function(k){kmeans(iris[,3:4],k,nstart = 20,iter.max = 20)$tot.
plot(1:k.max,wss, type= "b", xlab = "Number of clusters(k)",ylab = "Within cluster s
```



In [110…
```
set.seed(20)
irisCluster <- kmeans(iris[, 1:4], 3, nstart = 20)
irisCluster
```

```
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [75] 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3
[112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3
[149] 3 2

Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

In [111…
```
irisCluster$tot.withinss
```

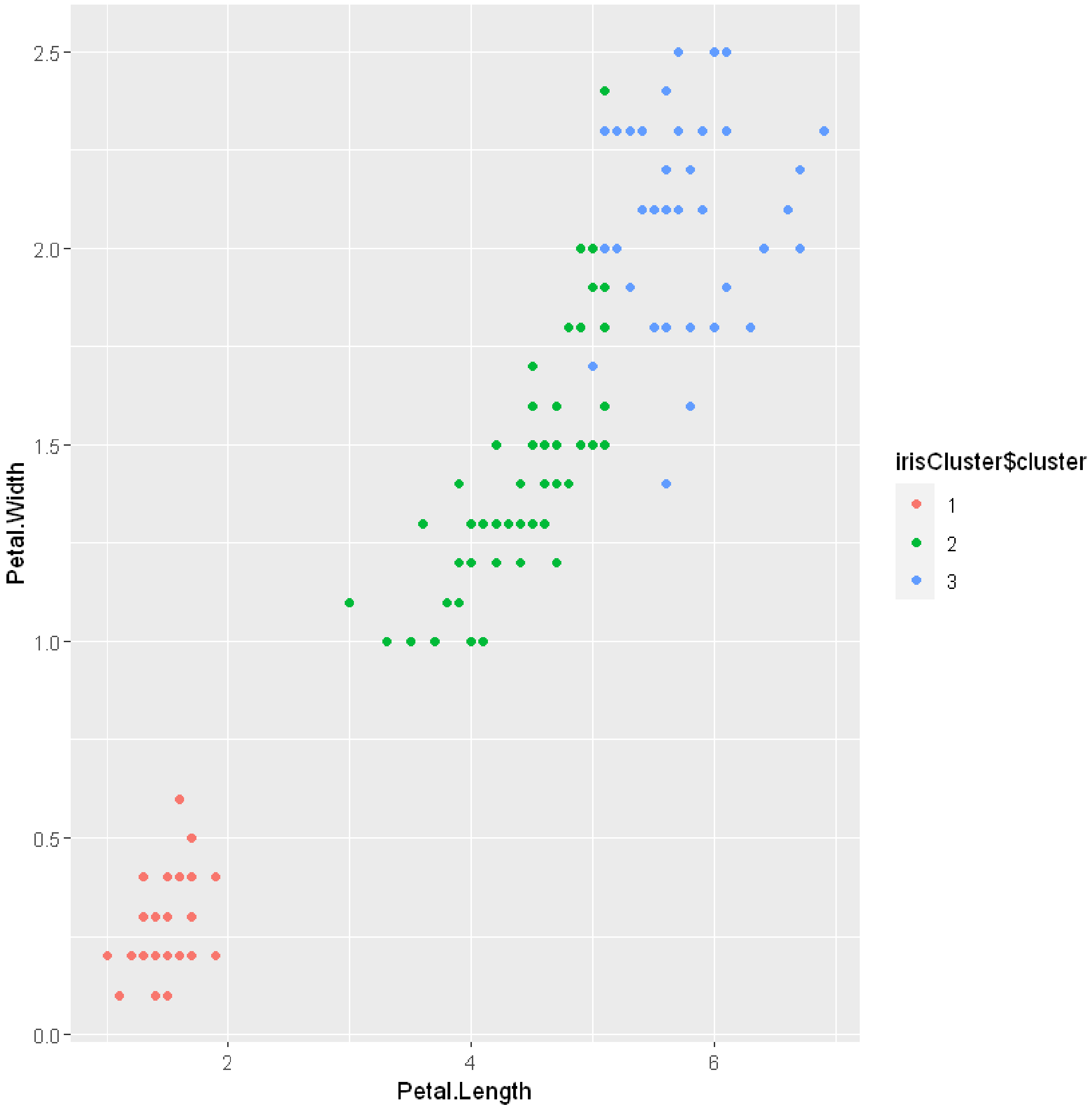
78.851441426146

In [112…
```
table(irisCluster$cluster, iris$Species)
```

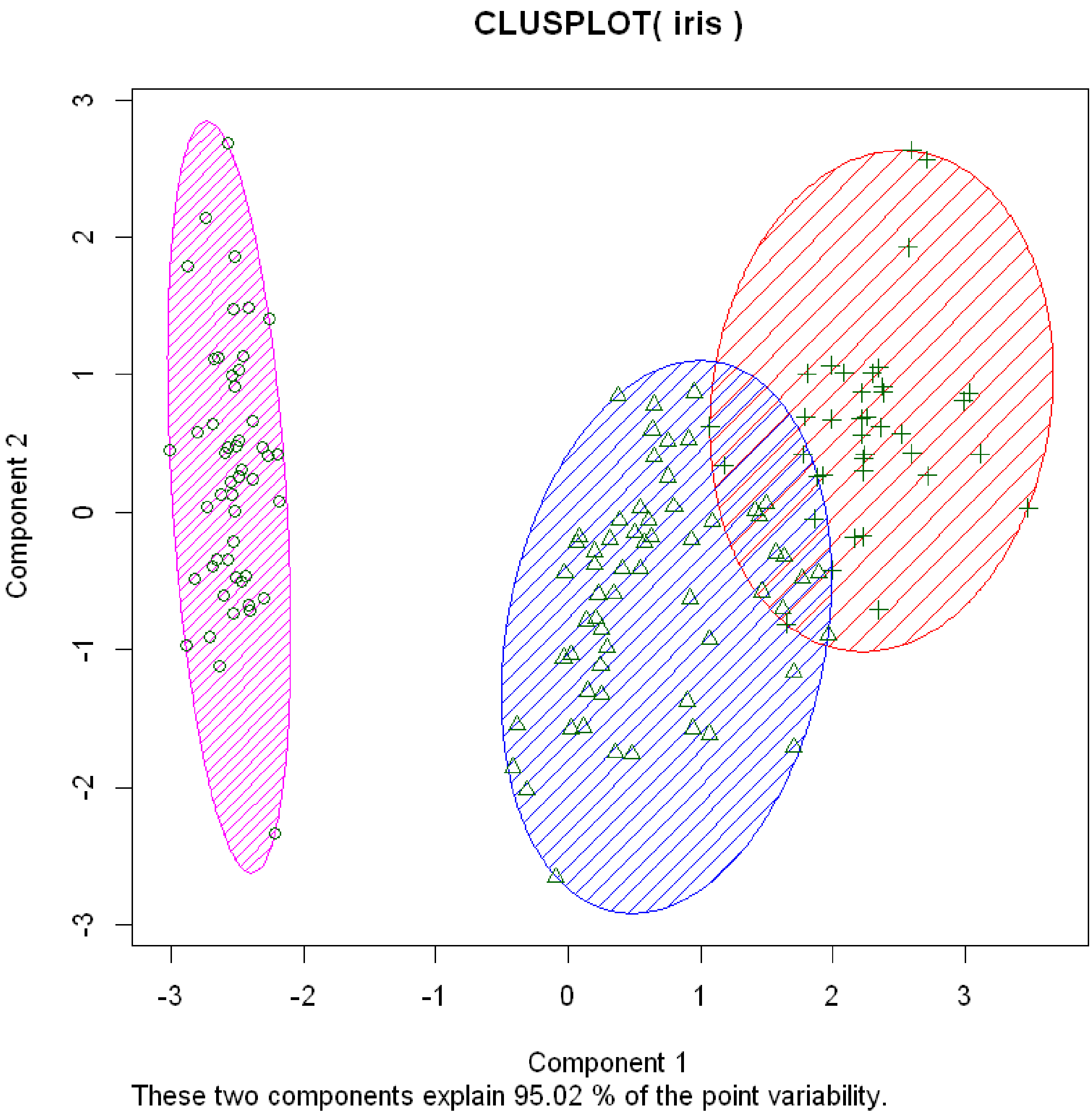
```
    setosa versicolor virginica
  1     50          0         0
  2      0         48        14
  3      0          2        36
```

In [113…
```
irisCluster$cluster <- as.factor(irisCluster$cluster)
```

In [114…
```
ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom_poi
```



In [115…
```
clusplot(iris, irisCluster$cluster, color=T, shade=T,lines=0)
```



Some Takeaways

- **Scale/standardize the data when applying kmeans algorithm.**
- **Elbow method for selecting number of clusters**
- **Kmeans gives more weight to the bigger clusters.**
- **Kmeans may still cluster the data even if it can't be clustered**
- **Different initial partitions can result in different final clusters.**
- **It can not handle noisy data and outliers.**
- **It is not suitable to identify clusters with non-convex shapes(irregular shapes - elliptical).**

Thank you !!