VISION

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

MISSION

To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice.

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices.

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION & MISSION

VISION

To become a leader in providing Computer Science & Engineering education with emphasis on knowledge and innovation.

MISSION

- To offer flexible programs of study with collaborations to suit industry needs.
- To provide quality education and training through novel pedagogical practices.
- To expedite high performance of excellence in teaching, research and innovations.
- To impart moral, ethical values and education with social responsibility.

DEPARTMENT OF COMPTER SCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES

After 3-5 years of graduation, the graduates will be able to

PEO1: Apply technical concepts, Analyze, Synthesize data to Design and create novel products and solutions for the real life problems.

PEO2: Apply the knowledge of Computer Science Engineering to pursue higher education with due consideration to environment and society.

PEO3: Promote collaborative learning and spirit of team work through multidisciplinary projects

PEO4: Engage in life-long learning and develop entrepreneurial skills.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

Engineering Graduates will be able to:

PO1. Engineering knowledge: Apply the basic knowledge of mathematics, science and engineering fundamentals along with the specialized knowledge of mechanical engineering to understand complex engineering problems.

PO2. Problem analysis: Identify, formulate, design and analyze complex mechanical engineering problems using knowledge of science and engineering.

PO3. Design/development of solutions: Develop solutions for complex engineering problems, design and develop system components or processes that meet the specified needs with appropriate consideration of the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Formulate engineering problems, conduct investigations and solve using research-based knowledge.

PO5. Modern tool usage: Use the modern engineering skills, techniques and tools that include IT tools necessary for mechanical engineering practice.

PO6. The engineer and society: Apply the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities during professional practice.

PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities to various groups, ability to write effective reports and make effective presentations.

PO11. Project management and finance: Demonstrate and apply the knowledge to understand the management principles and financial aspects in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in Independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

At the end of 4 years, Computer Science and Engineeringgraduates at MCET will be able to:

PSO1: Apply the knowledge of Computer Science and Engineering in various **domains** like networking and data mining to manage projects in multidisciplinary environments.

PSO2: Develop software applications with open-ended programming environments**.**

PSO3: Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LIST OF PROGRAMS

SI.No.	Name of the Program	Date of Program	Date of Submission	Page No	Faculty Signature
1	**R AS CALCULATOR APPLICATION** a. Using with and without R objects onconsole b. Using mathematical functions onconsole c. Write an R script, to create R objects for calculator application and save in a specified location indisk.				
2	**DESCRIPTIVE STATISTICS IN R** a.Write an R script to find basic descriptive statistics using summary, str, quartile function on mtcars& cars datasets. b.Write an R script to find subset of dataset by using subset (), aggregate () functions on iris dataset.				
3	**READING AND WRITING DIFFERENT TYPES OF DATASETS** a. Reading different types of data sets (.txt, .csv) from web and disk and writing in file in specific disklocation. b. Reading Excel data sheet inR.				

	c. Reading XML dataset inR.				
4	**VISUALIZATIONS** a. Find the data distributions using box and scatterplot. b. Find the outliers usingplot. c. Plot the histogram, bar chart and pie chart on sampledata.				
5	**CORRELATION AND COVARIANCE** a. Find the correlation matrix. b. Plot the correlation plot on dataset and visualize giving an overview of relationships among data on irisdata. c. Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.				
6	**REGRESSION MODEL** Import a data from web storage. Name the dataset and now do Logistic Regression to find out relation between variables that are affecting the admission of a student in a institute based on his or her GRE score, GPA obtained and rank of the student. Also check the model is fit or not. require (foreign), require(MASS).				
7	**CLASSIFICATION MODEL** a. Install relevant package for classification. b. Choose classifier for classification problem. c. Evaluate the performance of classifier.				
8	**CLUSTERING MODEL** a. Clustering algorithms for unsupervised classification. b. d. Plot the cluster data using R visualizations.				

ADDITIONAL PROGRAMS

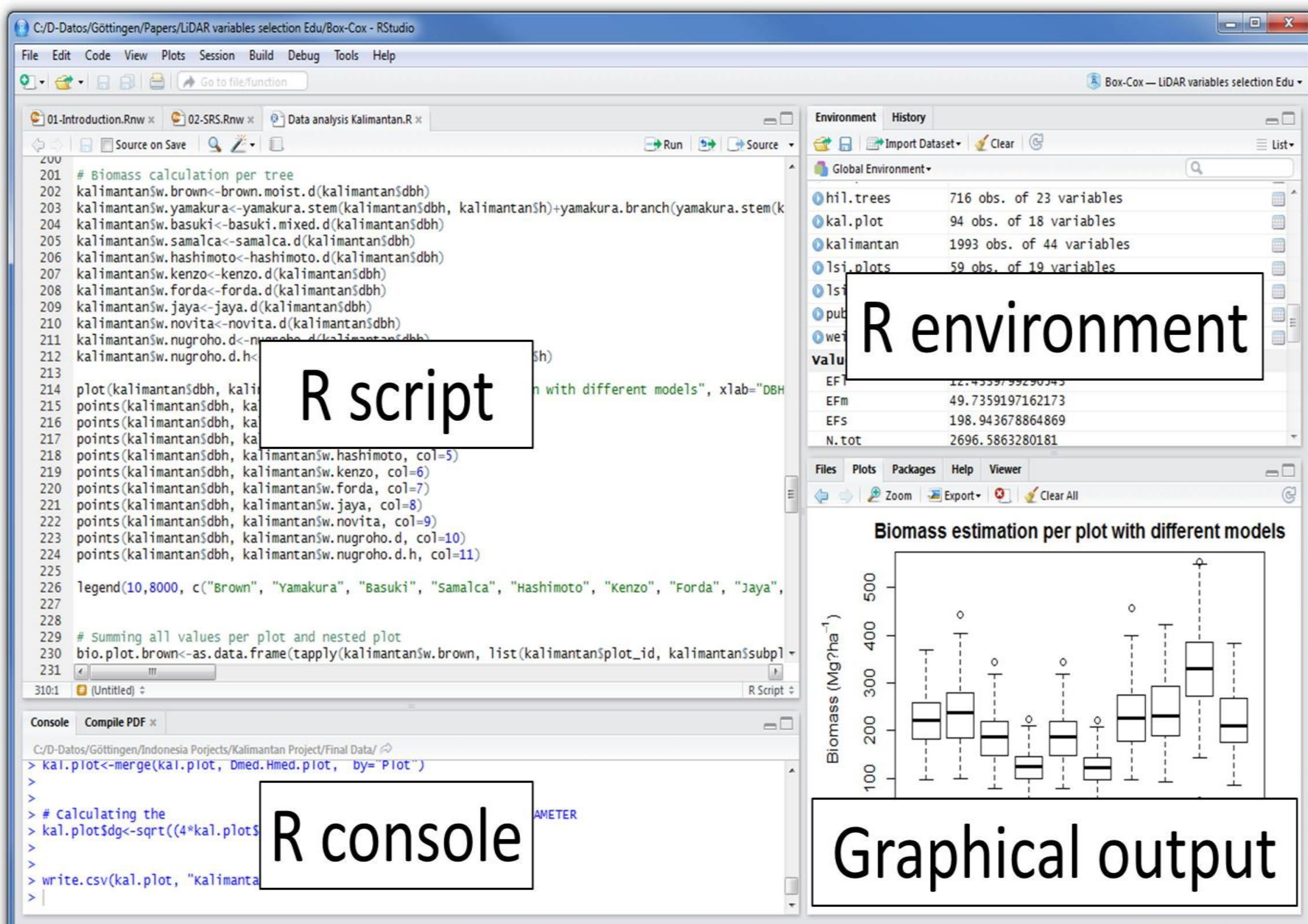
SI.No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
9.	write a program to find given no is even or odd				
10.	write a program to find given year is leap or not				
11.	write a program to find greatest of four numbers				
12.	write a program to find the sum of the digits of the number.				
13	write a program to find the frequency of a digit in the number.				
14	write a program to find the sum of squares of a given series of numbers. Sum = $1^2+2^2+\ldots+N^2$				
15	Consider the annual rainfall details at a place starting from January 2012. Create an R time series obiect				

Setting up R

Before start need to install R on r personal machine. For doing this, visit: https://cran.r-project.org where will find instructions on how to download and install R on Windows, MAC and Linux devices.

It is important that also use an editor for r scripts. One option is to use the default editors that existfor the Windows and MAC versions of R or to download other editors like:

- RStudio is a more full fledged editor (http://www.rstudio.com/). This is probably the best choice if are not very experienced with R. One useful feature is its LaTeX integration.
- can use the general purpose editors Emacs or XEmacs together with the extension ESS (http://ess.r-project.org/). Getting to use Emacs efficiently requires a bit of effort but once have mastered it it can be very powerful (and as it is general purpose will be able to use it for (most) other programming languages).



R basics

R can be used as a calculator that can perform simple arithmetic operations. Type the

```
4+5

9-7
```

following R commands into the console, and check that their outcome is what would expect:

Or it can be used to print messages like:

```
print("Hello World")
```

Objects

There are two equivalent ways of defining objects in R, either using the = or the <- operators.

```
x<-9
x=9
```

By typing either of the two commands below will create an object in R named x that takes the integer value 9.

In the same way that the object x is defined to take the value 9 (an integer value), it can be defined to have a different class, e.g.:

- Numeric representing real values e.g. x=10.343
- Character representing string values e.g. x="Monday"
- Logical representing boolean values e.g. x=TRUE
- Complex representing complex values e.g. x=2+3i

The function class(x) can be used for finding the class of an object x. The as. followed by the class name functions can be used for defining the class of the object x, for example, as.integer(x), as.numeric(x) and as.character(x). Equivalently, can check if an object is of a specific class by using the is. followed by the class name functions, for example, is.integer(x) and is.character(x).

Type the following R commands into the console, and check that they do what expect:

By typing the command ls() can see all the objects that are defined in the current R

```
x=10. 3
y=3  z=as.integer(11)
class(y)  class(z)

is.numeric(y) is.numeric(x)

w1=x>y; w1
w2=y>z; w2
class(w1)
class(w2)
```

environment(https://cran.r-project.org/doc/manuals/r-release/R-intro.html #The-R-environment). The ; separates commands on a single line.

Data Structures

Vectors

The simplest data structure in R is the numeric vector. The following R command creates a numeric vector named vec that has 5 numbers {2.3, 4, 1, 3, 7}:

```
vec=c(2.3,4,1,3,7)
```

Simple operations can be done on the vectors for example to select individual elements

```
vec[1]
vec[2:3]
```

or subsets of the vector using square brackets, as follows:

```
vec[-1]
vec[-c(3, 4)]
```

Or can omits elements of the vector:

It is worth noting that indexing in R starts from one instead of zero as with some other programming langugages. Simple calculations can be done on a vector. The commands below compute the length of the vector and the average value of the vector and of subsets of it.

```
length(vec) # The length function computes the length of the vector
```

```
## [1] 5
```

```
length(c(vec,vec))
```

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

```
mean(vec) # The mean function computes the average value of the vector
```

```
## [1] 3.46
```

```
mean(vec[-c(1:2)])
```

```
## [1] 3.666667
```

Anything listed after the '#' symbol is regarded as a comment in R and it is not computed. Other types of objects in R include: matrices, factors, lists an data frames.

Factors

Factors in R are objects that are used to define a selection of categories that can be both ordered and unordered.

For example, suppose that we are recording pets in three categories: “dogs”,“cats” and “other”; and a factorvector, *pet*, has been observed with the following entries:

```
pet=as.factor(c("dogs","dogs","other","cats","cats")
pet
```

```
## [1] dogs   dogs   other cats   cats  dogs
## Levels: cats dogs other
```

```
levels(pet)
```

```
## [1] "cats"   "dogs"   "other"
```

```
pet[1]
```

```
## [1] dogs
## Levels: cats dogs other
```

```
table(pet) #The table function produces a frequency table of the observations
```

```
## pet
##   cats   dogs other
##      2      3      1
```

Factors can be modified to include additional levels. For example, we can have an additional

```
levels(pet)=c(levels(pet),"goldfish")
```

level “goldfish”. Before adding any new observations we need to add the “goldfish” level to the levels of the vector pet:

Matrices

Matrices are a two-dimensional representation of data elements in R. The columns and the rows of a matrix canrepresent different observations, for example:

```
A=matrix(1:9, nrow=3, ncol=3)
```

Similarly to vectors, we can do basic arithmetic calculations on a matrix, including matrix multiplication andsubsetting of elements. Type the following R commands into the console, and check their outcome:

```
B=matrix(1:9,nrow=3,ncol=3)

##Arithmetic calculations  of  a  matrix
A+B A%*%B

##Print columns and rows of a matrix
A[,3]
B[2,]
B[1,3]
diag(A) #The diagonal of matrix A is printed
```

Vectors and matrices can be combined to form new matrices using the cbind and rbind commands that columnand row bind the objects together, respectiv

Type the following R commands into the console, and check their outcome:

```
x=c(1:3)

y=c(4:6)

z=cbind(x,y); is.matrix(z); dim(z) #Column bind

z=rbind(x,y); is.matrix(z); dim(z);#Row bind
```

Data Frames

A data frame in R is a list of vectors of equal length. Each vector can have different types of data stored in it. Wecan index data frames like a matrix or like a list. Type the following R commands on r console, and check thatthey do what expect:

```
x=c(4,3,2,1)
y=c(20,10,20,10)
df=as.data.frame(cbind
(x,y))df

df[1,1]
df[,1]

class(df);  dim(df)
colnames(df)
rownames(df)

is.numeric(df[,1]);  class(df[,1])
is.numeric(df[,2]);
```

As the column names of the data frame df are: x and y we can use the $ operator to subset entries of the dataframe. For example:

```
df$x
df$y[1:2]
```

R has several built-in data frames, for example:

```
mtcars
```

(see ?mtcars for more infomation)

Lists

A list can contain a combination of anything want, including nested lists:

```
x=list(1,list("hat","coat"),c(3,4,5),list(q="?",answer=24)) x
```

List indexing can be used to extract a sub-list

```
x[2]
x[2:4]
x[[2]]
```

Functions

R has a number of built-in functions for performing different types of calculations or other operations. Functionsin R can be called using functionName(argument_1, argument_2, ...). So far we have discussed the use ofthe class, mean, which, cbind,table functions amongst others.

As discussed R provides comprehensive help with all built-in functions that can access by typing a questionmark followed by the function's name, for example, ?functionName. This will bring up a help window whereinformation about the syntax of the function, available options for the input arguments and examples applying the function are provided. It is recommended that when working with unfamiliar functions check their help pagesfor further information and the best places to use them.

R offers the flexibility of creating r own functions. New functions in R can be defined by:

functionName = function(argument1, argument2, ...) { expression }

For example we can write our own function for computing the average value of a numeric vector:

```
new_average=function(V){ new_ave=sum(V)/length(V)return(new_ave)

      }
```

We can check if our new function has the same output as the mean function in R:

```
x=c(4,3,2,1); new_average(V=x); mean(x)
```

[1] 2.5
[1] 2.5

```
new_average(3:20); mean(3:20)
```

[1] 11.5
[1] 11.5

Multiple arguments can be passed in functions. Some of which might want to prespecify, in the example belowy is prespecified to take the value 1. We can call

```
new_add=function(x,y=1){ x+y
}
new_add(x=5)
```

the function with or without the y statement and we will not have an error:

[1] 6

```
new_add(x=5,y=1)
```

[1] 6

```
new_add(x=c(2,3),y=2)
```

[1] 4 5

If the y value was not prespecified in the definition of the function, the first command: new_add(x=5) would haveled to an error.

should note a few things about creating r own functions:

1. Any new variables created within a function are local and are not available within the global R environment. For example, in the new_average function the object new_ave is only available within the function and not globally. can check this by running ls() on r console.
2. can allow for one function to pass on arguments to another function by adding the ... argument (https: //cran.r-project.org/doc/manuals/r-release/R-intro.html#The-three-dots-argument). The ... argument is a useful argument when do not want to prespecify any arguments that are used by other function within r creted function. For example check the R output of the following R command

```
col_plot=function(x,y,...){
print(x+y) plot(x,y,...)
}

col_plot(1:5,1:5,col="red")col_plot(1:5,1:5,cex=2)
```

Loops

If would like to perform the same set of actions repeatedly it is a good idea to use loops for this. R has anumber of loop constructions available: for, while and repeat. Depending on the nature of the problem that are working a different construction will be more suitable. The examples below illustrate how to code the three loops in R:

```
# for loop

for(i in 1:4){

  print(i)
```

```
# while
loopi=1
while(i<4){

  i=i+1
  print(i)
```

```
# repeat loop

repeat{
  print(i)
  i=i+1
  if(i>4){

    break
```

The if command in the repeat loop checks if a statement is TRUE or FALSE. In the case that it is TRUE then the expression within the brackets is evaluated. The break command terminates the repeat loop.

Further information about conditional execution (if, if else, else) statements and loop constructions can be found at: https://cran.r-project.org/doc/manuals/r-release/R-intro.html#Conditional-execution

Other R built-in functions for evaluating expressions on vectors, matrices, data frames and lists include the sapply,apply and lapply functions.

The apply function can be used to compute a specific function on either the columns or the rows of a matrix. Theoutcome of the function is a vector of values. For

```
A=matrix(1:9, nrow=3, ncol=3)

apply(A,2,mean) #The average value of each column of matrix A is returned
```

example:

```
## [1] 2 5 8
```

```r
# Instead of a pre-defined function, we can use our own functions.

# Below the average value of each row of matrix A is returned computed our own function#
that has as an input statement vector i
apply(A,1,function(i){return(sum(i)/length(i))})
```

```
## [1] 4 5 6
```

Check ?sapply and ?lapply for further information on how to use them to repetitively perform the actions of a function on the elements of a vector.

Plotting

R can produce a wide variety of plots and graphics. This section presents some common types of plots that arisewhen working with data, and how to tweak the output so that it looks presentable.

Input a small height and weight dataset by pasting the two lines below.

```r
weight=c(60, 72, 57, 90, 95, 72)
height=c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
```

To get a simple scatter plot illustrating these variables, use the plot command:

```r
plot(h)
```

R plots can be customised extensively. Try modifying the arguments of the plot function (e.g.cex, col) to checktheir effect on the plot:

```r
plot(height, weight,pch=2,col="red",cex=2, main="scatter plot")
```

Add the xlab and ylab arguments to the command above to change the labels on the two axes.The hist built-in function can be used to produce a histogram of the height data.

```r
hist(height)
```

A nicer-looking histogram can be obtained by specifying the number of bins manually. Check the argument
breaks=k in the hist function.

For looking at experimental data, the *box and whisker plot*, often just called a box plot, is a useful tool. It showsthe first and third quantiles of a numerical dataset as the bottom and top of a box, and a line inside the boxrepresents the median of the data. Whiskers at the top and bottom of the box show a multiple of the interquartilerange - this is where “most” of the data should be. Any data points outside the whisker region are displayed asindividual points. These individually displayed points may be outliers (depending on the situation).

A box plot of the height data can be produced using the boxplot function:

```r
boxplot(height)
```

can get a quick glimpse of R's plotting capabilities by calling the graphics demo command

```
demo(graphics)
```

For more advanced visualisation techniques in R can explore the ggplot2 package (https://cran.r-project. org/web/packages/ggplot2/index.html). Further guidance on how to use the ggplot2 can be found at: https: //r4ds.had.co.nz/data-visualisation.html.

Importing/ Exporting Data

R can handle data in a wide variety of formats. For the smallest datasets, can enter the variables directly asvectors, as we have done above. Most commonly, will have data in a file that needs to be read in to R.

```
head(mtcars)
```

To read data in a text file can use the read.table command, and to save data created in R can use the write.table command. For example:

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

```
write.table(mtcars[,1:5],file="my_mtcars.txt",row.names=FALSE,col.names=TRUE
,quote=FALSE)
new.mtcars=read.table("my_mtcars.txt",header=TRUE)
head(new.mtcars)
```

## mpg cyl disp	hp	drat
## 1 21.0 6 160	110	3.90
## 2 21.0 6 160	110	3.90
## 3 22.8 4 108	93	3.85
## 4 21.4 6 258	110	3.08
## 5 18.7 8 360	175	3.15
## 6 18.1 6 225	105	2.76

If the data files that need to read are comma-separated files can use the read.csv command to read the file or can amend the read.table to include the argument sep=",". In addition, if there are character values in the file that are importing in R can choose whether the character entries will be read as factors

using the argument stringsAsFactors=TRUE. For further information about the functions check ?read.table and ?read.csv.

Also can save r current R environment by exiting R with q("yes") or can save a subset of the defined objects of the environment using:

```
save(x,y, file = "filename.RData")
```

and can retrieve previously saved data using:

```
load("FILEDIRECTORY/filename.RData")
```

need to be careful when importing/ exporting data as need to properly specify the directory where the files (data) are located.

R Packages

R functions are stored in R packages. The standard R functions, like print, plot, etc are included in the base R packages. For using functions and/or datasets that

```
install.packages("packageName")
```

are not stored in the standard R packages will need to install the relevant R package.

and for using the relevant package after installation will need to run:

```
library("packageName")
```

Statistics and R

R as a statistical programming language has a number of built-in functions for conducting statistical tests and computations.

Such functions include evaluating the cumulative distribution function ($P(X \le x)$; CDF), density and quantile functions of a number of distributions (e.g. Normal, Beta, Binomial, Poisson) and for simulating data from these distributions. R functions for computing the probability density of a distribution begin with a d, for the CDF with a p, for the quantile function with q and for simulating data with r. For further information and for the list of available distributions see https://cran.r-project.org/doc/manuals/r-release/R-intro.html#Probability-distributions.

Let's look at the examples below:

Suppose that $X \sim Poisson(4)$. We can compute $P(X \le 3)$ using:

```
ppois(3,lambda=5,lower.tail=TRUE)
```

```
## [1] 0.2650259
```

Or can draw 5 values from a Uniform(0,1) distribution using:

```
runif(5,min=0,max=1)
```

[1] 0.051457463 0.179099351 0.008170631 0.967664950 0.362049386

Binomial Distribution

This section looks at the binomial distribution and binomial test in R.

Suppose that a coin is thrown 20 times and the number of heads is recorded. The number of heads obtained is considered to follow a Binomial distribution with $n =$

```
pbinom(15,20,0.5)
```

20 trials and with probability of success $p = 0.5$. Using R we can compute the P (Number of heads obtained ≤ 15):

[1] 0.994091

Let's look at a different experiment. Suppose we have observed 15 heads (successes) out of 40 flips of a coin (trials). We want to know whether we have evidence that the true probability p of success for each trial is smaller than 0.5. We might also say that we want to know whether the difference between the observed probability and 0.5 is *statistically significant*. The significance level (often called the p value) is a measure of how likely our observed result would be, if the *null hypothesis* of $p = 0.5$ were true. Specifically, the significance level is the probability of obtaining as extreme a result as we did, if 0.5 is the true success probability. The following exercises attempt to make this notoriously tricky concept a bit clearer.

First, simulate a large number of draws from the null distribution, i.e. many sets of 40 trials, where each trial is a success with probability 0.5. Each dataset is then just

a number between 0 and 40 - the number of successes.

We can review the null distribution using a histogram. Also we can add a vertical line that presents the value observed in the real data 'x.test'.

```
n=1e+6 x.empirical=rbinom(size=40,n=n,p=0.5)
```

```
x.test=15
hist(x.empirical,main="Empirical null distribution")
abline(v=x.test,col="red",lty=2,lwd=2)
```

x.empirical

Does the observed value look plausible if the null distribution is indeed the true distribution? We can evaluate how plausible it is by computing:

```
sum(x.empirical<=x.test)/n
```

[1] 0.077166

Normal Distribution

This section is devoted on Normally distributed data and statistical analysis for testing the distribution of such data.

Suppose we have two populations one drawn from a N(1,1.6) distribution and the second one from a N(0.5,1.5)distribution. Each population has 100 observations. We are interested in testing whether the two populations follow the same distribution.

```
x=rnorm(100,mean=0.5,sd=1.5)

y=rnorm(100,mean=1,sd=1.6)
```

Before starting any statistical analysis we should visualise the data. Similarly to before we can plot the samplesusing a histogram and/or we can use the Quantile-Quantile plot to examine the distribution of the samples:

Normal Q−Q Plot

```
qqnorm(x)
qqline(x) #Check ?qqline
```

Theoretical Quantiles

In addition, we can plot the empirical cumulative density function (ECDF) of the data:

```
plot(ecdf(y))
```

ecdf(y)

1. R AS CALCULATOR APPLICATION

a. Using without R objects onconsole

> 2587+2149

Output:-
[1] 4736

> 287954-135479

Output:-
[1] 152475

> 257*52

[1] 13364

> 257/21

Output:-

[1]12.2381

Using with R objects on console:

>A=1000
>B=2000
>c=A+B
>c

Output:-

[1] 3000

b. Using mathematical functions onconsole

Function	What It Does
abs(x)	Takes the absolute value of *x*
log(x,base=y)	Takes the logarithm of *x* with base *y*; if base is not specified, returns the natural logarithm
exp(x)	Returns the exponential of *x*
sqrt(x)	Returns the square root of *x*
factorial(x)	Returns the factorial of *x* (*x*!)
choose(x,y)	Returns the number of possible combinations when drawing *y* elements at a time from *x* possibilities

```
>x=-50.567
> abs(x)
```

Output:-

```
[1] 50.567
```

```
> log(1:3,base=6)
```

Output:-

```
[1] 0.0000000 0.3868528 0.6131472
```

```
>exp(x)
```

Output:-

```
[1] 1.094034e-22
```

```
> x=50
>sqrt(x)
```

Output:-

```
[1] 7.071068
> factorial(x)
```

Output:-

```
[1] 3.041409e+64
```

c. **Write an R script, to create R objects for calculator application and save in a specified location in disk.**

```
cat("1) For Addition\n")
cat("2) For Subtraction\n")
cat("3) For Division\n")
cat("4) For multiplication\n")
a<-readline(prompt="Enter first number:")
b<-readline(prompt="Enter second number:")
choice<-readline(prompt="Enter r choice:")
a<- as.integer(a)
b<- as.integer(b)
choice<- as.integer(choice)
switch(
    choice,
    "1"=cat("Addition=",a+b),
    "2"=cat("Subtraction =",a-b),
    "3"=cat("Division= ",a/b),
    "4"=cat("multiplication =",a*b)
)
```

2. DESCRIPTIVE STATISTICS IN R

a. Write an R script to find basic descriptive statistics using summary, str, quartile function on mtcars& carsdatasets.

>mtcars

```
> mtcars
                     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
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
>
```

>summary(mtcars)

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

>str(mtcars)

```
> str(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 ...
> 
```

>quantile(mtcars$mpg)

```
> quantile(mtcars$mpg)
    0%    25%    50%    75%   100% 
10.400 15.425 19.200 22.800 33.900 
> 
```

>cars

```
> cars
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
```

>summary(cars)

```
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
> 
```

>str(cars)

```
> str(cars)
'data.frame':   50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
> 
```

>quantile(cars$speed)

```
> quantile(cars$speed)
  0%  25%  50%  75% 100% 
   4   12   15   19   25 
> 
```

b. Write an R script to find subset of dataset by using subset (), aggregate () functions on iris dataset.

> iris

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
7	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.5	2.3	4	1.3	versicolor
6.5	2.8	4.6	1.5	versicolor
5.7	2.8	4.5	1.3	versicolor
6.3	3.3	4.7	1.6	versicolor
4.9	2.4	3.3	1	versicolor
7.3	2.9	6.3	1.8	virginica
6.7	2.5	5.8	1.8	virginica
7.2	3.6	6.1	2.5	virginica
6.5	3.2	5.1	2	virginica
6.4	2.7	5.3	1.9	virginica

>subset(iris,iris$Sepal.Length==5.0)

```
> subset(iris,iris$Sepal.Length==5.0)
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
5             5         3.6          1.4         0.2     setosa
8             5         3.4          1.5         0.2     setosa
26            5         3.0          1.6         0.2     setosa
27            5         3.4          1.6         0.4     setosa
36            5         3.2          1.2         0.2     setosa
41            5         3.5          1.3         0.3     setosa
44            5         3.5          1.6         0.6     setosa
50            5         3.3          1.4         0.2     setosa
61            5         2.0          3.5         1.0 versicolor
94            5         2.3          3.3         1.0 versicolor
> 
```

>aggregate(. ~ Species, data = iris, mean)

```
> aggregate(. ~ Species, data = iris, mean)
     Species Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa        5.006       3.428        1.462       0.246
2 versicolor        5.936       2.770        4.260       1.326
3  virginica        6.588       2.974        5.552       2.026
> 
```

3. READING AND WRITING DIFFERENT TYPES OF DATASETS

a. Reading different types of data sets (.txt, .csv) from web and disk and writing in file in specific disk location.

Reading csv file:

```
library(utils)

#Reading .csv file
CSVfile<- read.csv(file.choose())
View(CSVfile)


print(is.data.frame(CSVfile))
print(ncol(CSVfile))
print(nrow(CSVfile))

# Get the max salary from data frame.
sal<- max(CSVfile$Salary)
sal

# Get the person detail having max salary.
retval<- subset(CSVfile, Salary == max(Salary))
retval

#Get all the people working in IT department
retval1<- subset(CSVfile, Dept == "IT")
retval1
```

Output:

	Id.No	Name.of.Employee	Salary	Dept
1	1011	Raj Sharma	64300	IT
2	1012	Sharad Gandhi	30000	Operations
3	1013	Danish D'Souza	18560	IT
4	1014	Pawan Patil	23200	HR
5	1015	Rijo Paul	22720	Finance
6	1016	Joseph P	15290	IT
7	1017	Aakash Patel	13920	Operations
8	1018	Ganesh Rahu	12990	Finance
9	1019	Vinudas K.S	14490	IT
10	1020	Divya Kumar	7420	Operations
11	1021	Shilpa R	7194	IT
12	1022	Sindhu J.P	7420	HR
13	1023	Deepthi P.S	7419	Finance
14	1024	Lijin k c	14060	IT
15	1025	Sayad K M	33200	Operations
16	1026	Ajil k Mohanan	20420	Finance
17	1027	Edison ML	12140	IT
18	1028	Basil P E	25417	Operations
19	1029	Jobin George	16350	IT
20	1030	Jismon Tomy	11280	HR
21	1031	Sharafali P	12990	Finance

```
> library(utils)
>
> #Reading .csv file
> CSVfile<- read.csv(file.choose())
> View(CSVfile)
>
>
> print(is.data.frame(CSVfile))
[1] TRUE
> print(ncol(CSVfile))
[1] 4
> print(nrow(CSVfile))
[1] 21
>
> # Get the max salary from data frame.
> sal<- max(CSVfile$Salary)
> sal
[1] 64300
>
> # Get the person detail having max salary.
> retval<- subset(CSVfile, Salary == max(Salary))
> retval
  Id.No Name.of.Employee Salary Dept
1  1011       Raj Sharma  64300   IT
`

> #Get all the people working in IT department
> retval1<- subset(CSVfile, Dept == "IT")
> retval1
   Id.No Name.of.Employee Salary Dept
1   1011       Raj Sharma  64300   IT
3   1013   Danish D'Souza  18560   IT
6   1016         Joseph P  15290   IT
9   1019      Vinudas K.S  14490   IT
11  1021         Shilpa R   7194   IT
14  1024        Lijin k c  14060   IT
17  1027       Edison ML   12140   IT
19  1029     Jobin George  16350   IT
>
```

Reading txt file:

```
# Read a txt file
TXTfie <- read.delim(file.choose())
View(TXTfie)

print(is.data.frame(TXTfie))
print(ncol(TXTfie))
print(nrow(TXTfie))

# Get the max salary from data frame.
sal<- max(data$Salary)
sal

# Get the person detail having max salary.
retval<- subset(TXTfie, Salary == min(Salary))
retval

#Get all the people working in HR department
retval2<- subset(TXTfie, Dept == "HR")
retval2
```

Output:

	Id.No	Name.of.Employee	Salary	Dept
1	1011	Raj Sharma	64300	IT
2	1012	Sharad Gandhi	30000	Operations
3	1013	Danish D'Souza	18560	IT
4	1014	Pawan Patil	23200	HR
5	1015	Rijo Paul	22720	Finance
6	1016	Joseph P	15290	IT
7	1017	Aakash Patel	13920	Operations
8	1018	Ganesh Rahu	12990	Finance
9	1019	Vinudas K.S	14490	IT
10	1020	Divya Kumar	7420	Operations
11	1021	Shilpa R	7194	IT
12	1022	Sindhu J.P	7420	HR
13	1023	Deepthi P.S	7419	Finance
14	1024	Lijin k c	14060	IT
15	1025	Sayad K M	33200	Operations
16	1026	Ajil k Mohanan	20420	Finance
17	1027	Edison ML	12140	IT
18	1028	Basil P E	25417	Operations
19	1029	Jobin George	16350	IT
20	1030	Jismon Tomy	11280	HR
21	1031	Sharafali P	12990	Finance

```
> # Read a txt file
> TXTfie <- read.delim(file.choose())
> View(TXTfie)
>
> print(is.data.frame(TXTfie))
[1] TRUE
> print(ncol(TXTfie))
[1] 7
> print(nrow(TXTfie))
[1] 21
>
> # Get the max salary from data frame.
> sal<- max(data$Salary)

> sal
[1] 64300
>
> # Get the person detail having max salary.
> retval<- subset(TXTfie, Salary == min(Salary))
> retval
   Id.No Name.of.Employee Salary Dept  X X.1 X.2
11  1021         Shilpa R   7194   IT NA  NA  NA
>
> #Get all the people working in HR department
> retval2<- subset(TXTfie, Dept == "HR")
> retval2
   Id.No Name.of.Employee Salary Dept  X X.1 X.2
4   1014      Pawan Patil  23200   HR NA  NA  NA
12  1022       Sindhu J.P   7420   HR NA  NA  NA
20  1030      Jismon Tomy  11280   HR NA  NA  NA
> |
```

Writing in a file in a Specific Location

```
# Write filtered data into a new file.
write.csv(retval1,"D:\\output.csv")
newdata<- read.csv("D:\\output.csv")
newdata
```

Output:

```
> # Write filtered data into a new file.
> write.csv(retval1,"D:\\output.csv")
> newdata<- read.csv("D:\\output.csv")
> newdata
  X Id.No Name.of.Employee Salary Dept
1  1  1011       Raj Sharma  64300   IT
2  3  1013   Danish D'Souza  18560   IT
3  6  1016        Joseph P  15290   IT
4  9  1019      Vinudas K.S  14490   IT
5 11  1021        Shilpa R   7194   IT
6 14  1024        Lijin k c  14060   IT
7 17  1027       Edison ML  12140   IT
8 19  1029     Jobin George  16350   IT
> 
```

C.**Reading Excel data sheet in R.**

```
install.packages("xlsx")
library("xlsx")
XLSXfile<- read.xlsx("D:\\input.xlsx", sheetIndex = 1)
XLSXfile
```

Output:

```
> library("xlsx")
> XLSXfile<- read.xlsx("D:\\input.xlsx", sheetIndex = 1)
> XLSXfile
   Id.No Name.of.Employee Salary       Dept NA. NA..1 NA..2
1   1011       Raj Sharma  64300         IT  NA    NA    NA
2   1012   Sharad Gandhi  30000 Operations  NA    NA    NA
3   1013   Danish D'Souza  18560         IT  NA    NA    NA
4   1014     Pawan Patil  23200         HR  NA    NA    NA
5   1015       Rijo Paul  22720    Finance  NA    NA    NA
6   1016        Joseph P  15290         IT  NA    NA    NA
7   1017    Aakash Patel  13920 Operations  NA    NA    NA
8   1018     Ganesh Rahu  12990    Finance  NA    NA    NA
9   1019     Vinudas K.S  14490         IT  NA    NA    NA
10  1020     Divya Kumar   7420 Operations  NA    NA    NA
11  1021        Shilpa R   7194         IT  NA    NA    NA
12  1022     Sindhu J.P   7420         HR  NA    NA    NA
13  1023    Deepthi P.S   7419    Finance  NA    NA    NA
14  1024       Lijin k c  14060         IT  NA    NA    NA
15  1025      Sayad K M  33200 Operations  NA    NA    NA
16  1026  Ajil k Mohanan  20420    Finance  NA    NA    NA
17  1027       Edison ML  12140         IT  NA    NA    NA
18  1028      Basil P E  25417 Operations  NA    NA    NA
19  1029    Jobin George  16350         IT  NA    NA    NA
20  1030     Jismon Tomy  11280         HR  NA    NA    NA
21  1031    Sharafali P  12990    Finance  NA    NA    NA
> 
```

4 . VISUALIZATIONS

a. Find the data distributions using box and scatter plot

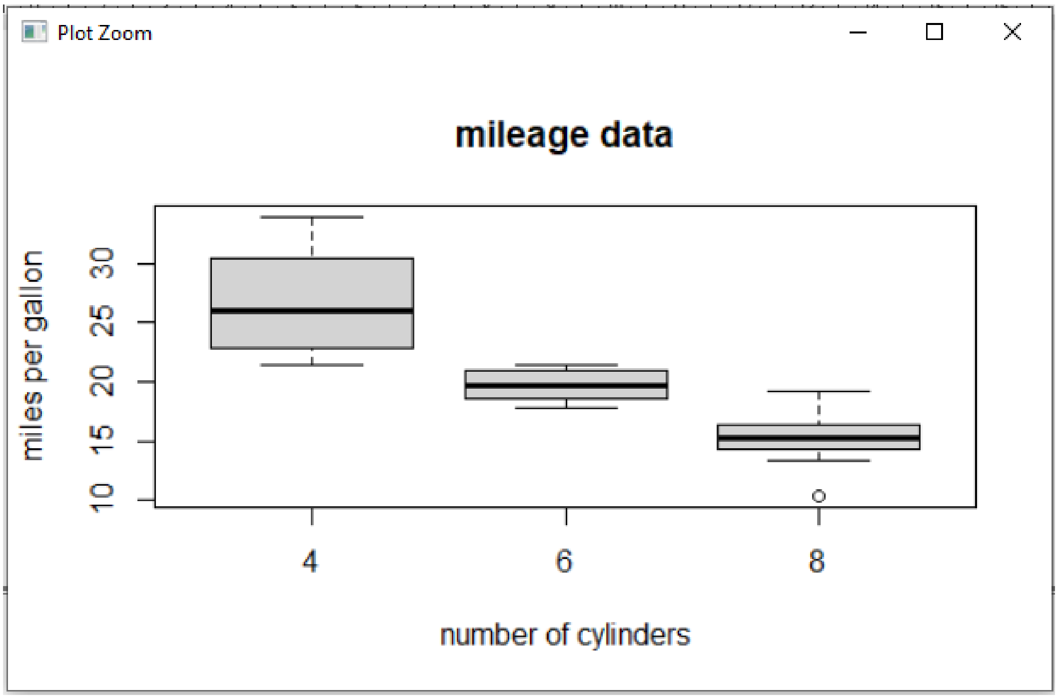
BoxPlot:

install.packages("ggplot2")
Library(ggplot2)
input <- mtcars[,c('mpg','cyl')]
input

boxplot(mpg ~ cyl, data = mtcars, xlab = "number of cylinders", ylab = "miles per gallon",
main = "mileage data")

Output:

```
> input <- mtcars[,c('mpg','cyl')]
> input
                     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
Duster 360          14.3   8
Merc 240D           24.4   4
Merc 230            22.8   4
Merc 280            19.2   6
Merc 280C           17.8   6
Merc 450SE          16.4   8
Merc 450SL          17.3   8
Merc 450SLC         15.2   8
Cadillac Fleetwood  10.4   8
Lincoln Continental 10.4   8
Chrysler Imperial   14.7   8
Fiat 128            32.4   4
Honda Civic         30.4   4
Toyota Corolla      33.9   4
Toyota Corona       21.5   4
Dodge Challenger    15.5   8
AMC Javelin         15.2   8
Camaro Z28          13.3   8
Pontiac Firebird    19.2   8
Fiat X1-9           27.3   4
Porsche 914-2       26.0   4
Lotus Europa        30.4   4
Ford Pantera L      15.8   8
Ferrari Dino        19.7   6
Maserati Bora       15.0   8
Volvo 142E          21.4   4
>
> boxplot(mpg ~ cyl, data = mtcars, xlab = "number of cylinders",
+         ylab = "miles per gallon", main = "mileage data")
> 
```



ScatterPlot:

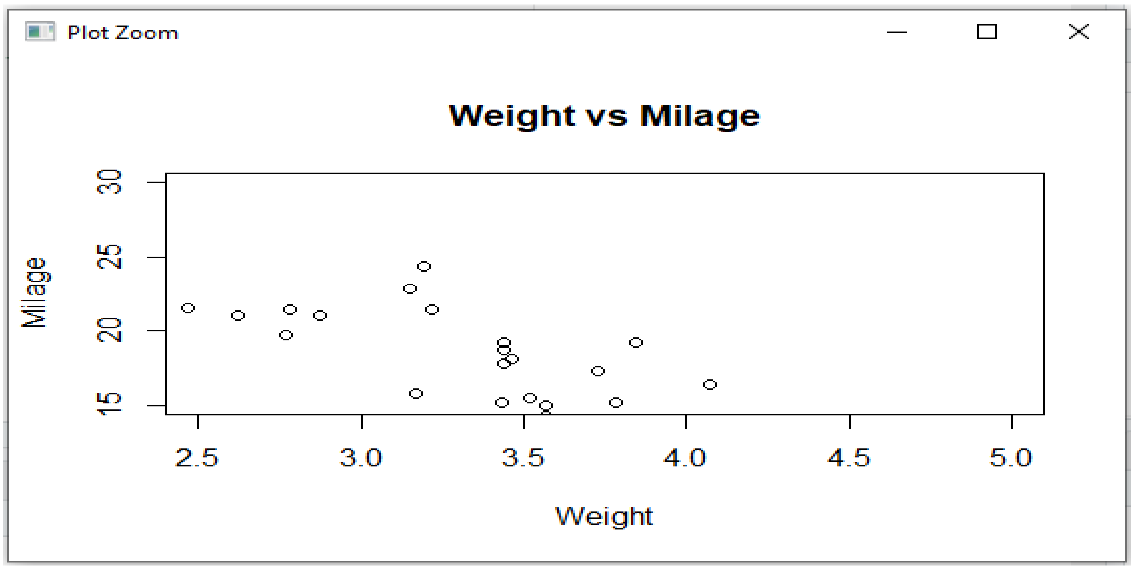
input1 <- mtcars[,c('wt','mpg')]
input1

Output:

```
> input1 <- mtcars[,c('wt','mpg')]
> input1
                       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
Duster 360          3.570 14.3
Merc 240D           3.190 24.4
Merc 230            3.150 22.8
Merc 280            3.440 19.2
Merc 280C           3.440 17.8
Merc 450SE          4.070 16.4
Merc 450SL          3.730 17.3
Merc 450SLC         3.780 15.2
Cadillac Fleetwood  5.250 10.4
Lincoln Continental 5.424 10.4
Chrysler Imperial   5.345 14.7
Fiat 128            2.200 32.4
Honda Civic         1.615 30.4
Toyota Corolla      1.835 33.9
Toyota Corona       2.465 21.5
Dodge Challenger    3.520 15.5
AMC Javelin         3.435 15.2
Camaro Z28          3.840 13.3
Pontiac Firebird    3.845 19.2
Fiat X1-9           1.935 27.3
Porsche 914-2       2.140 26.0
Lotus Europa        1.513 30.4
Ford Pantera L      3.170 15.8
Ferrari Dino        2.770 19.7
Maserati Bora       3.570 15.0
Volvo 142E          2.780 21.4
```

```
# Plot the chart for cars with weight between 2.5 to 5 and
#mileage between 15 and 30.

plot(x = input1$wt,y = input1$mpg,
     xlab = "Weight",
     ylab = "Milage",
     xlim = c(2.5,5),
     ylim = c(15,30),
     main = "Weight vs Milage"
)
```

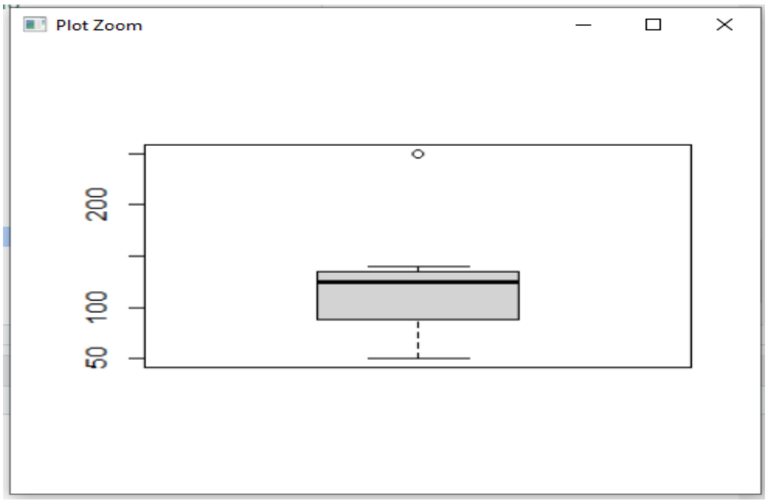
Output:



b. Find the outliers using plot.

v=c(50,75,100,125,130,140,250)
boxplot(v)

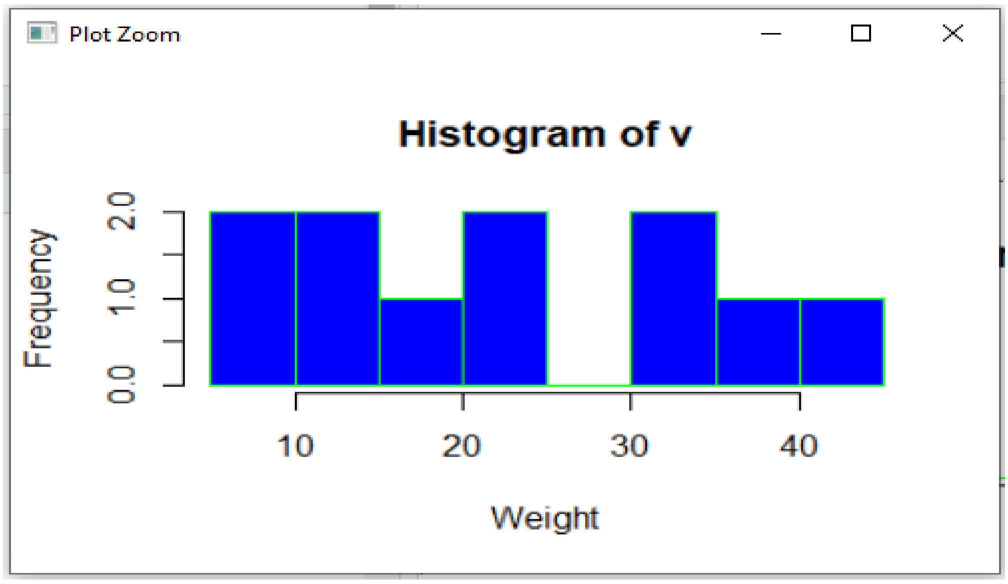
Output:



c. **Plot the histogram, bar chart and pie chart on sample data.**

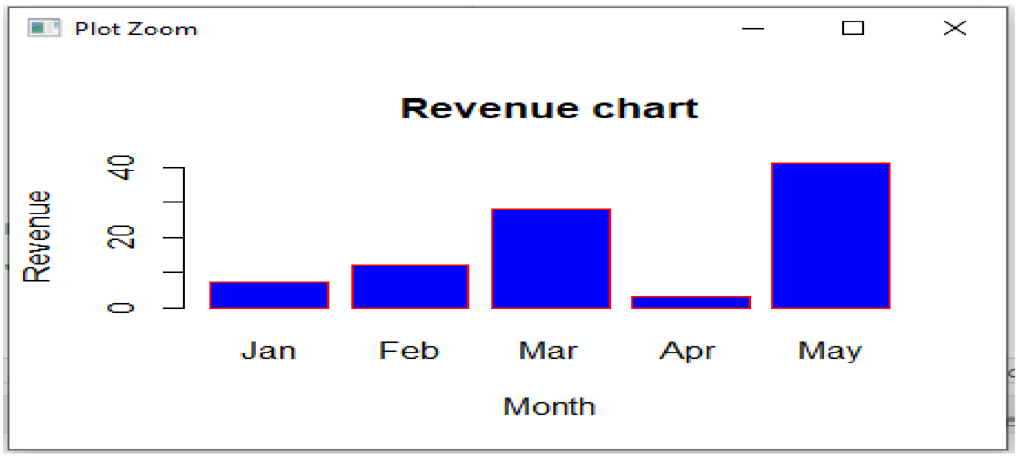
Histogram:

```
library(graphics)
v <- c(9,13,21,8,36,22,12,41,31,33,19)

# Create the histogram.
hist(v,xlab = "Weight",col = "blue",border = "green")
```



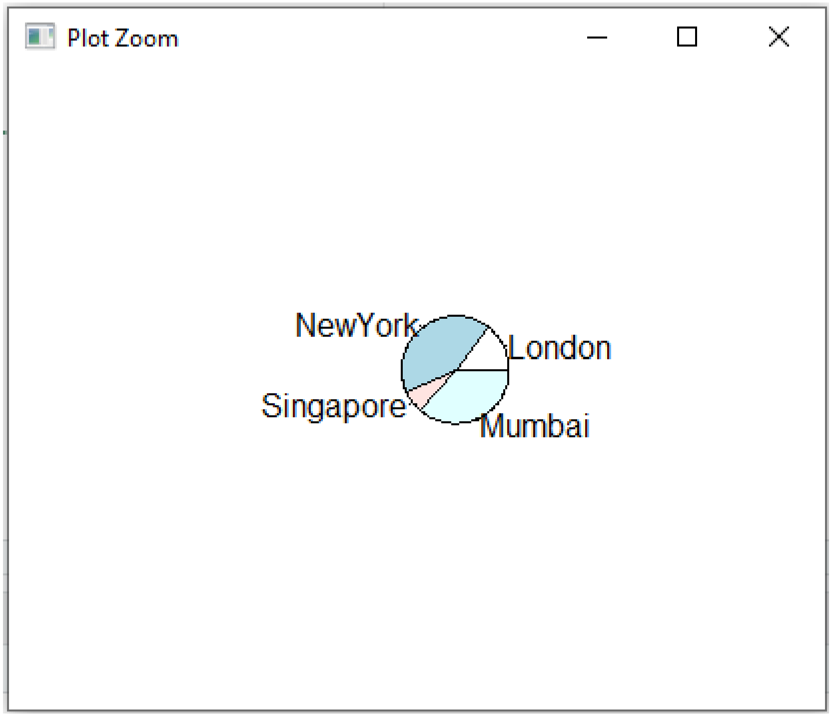
Barplot:

```
library(graphics)
H <- c(7,12,28,3,41)
M <- c("Jan","Feb","Mar","Apr","May") # Plot the bar chart.
barplot(H,names.arg = M,xlab = "Month",ylab = "Revenue",
        col = "blue",main = "Revenue chart",border= "red")
```



PieChart:

```
#Pie Chart
library(graphics)
x <- c(21, 62, 10, 53)
labels<- c("London", "NewYork", "Singapore", "Mumbai")
# Plot the Pie chart.
pie(x,labels)
```



5. CORRELATION AND COVARIANCE

a. Find the corelationmatrix on iris data set

```
d<-data.frame(x1=rnorm(10),x2=rnorm(10),x3=rnorm(10))
cor(d)
m<-cor(d)

 #get correlations
corrplot(m,method=”square”)
x<-matrix(rnorm(2),nrow=5,ncol=4)
y<-matrix(rnorm(15),nrow=5,ncol=3)
COR<-cor(x,y)
COR
```

Output:

```
> x<-matrix(rnorm(2),nrow=5,ncol=4)
> y<-matrix(rnorm(15),nrow=5,ncol=3)
> COR<-cor(x,y)
> COR
           [,1]       [,2]       [,3]
[1,] -0.8714279  0.3814612  0.2585345
[2,]  0.8714279 -0.3814612 -0.2585345
[3,] -0.8714279  0.3814612  0.2585345
[4,]  0.8714279 -0.3814612 -0.2585345
>
```

b.Plot the correlation plot on dataset and visualize giving an overview of relationships among data on irisdata.

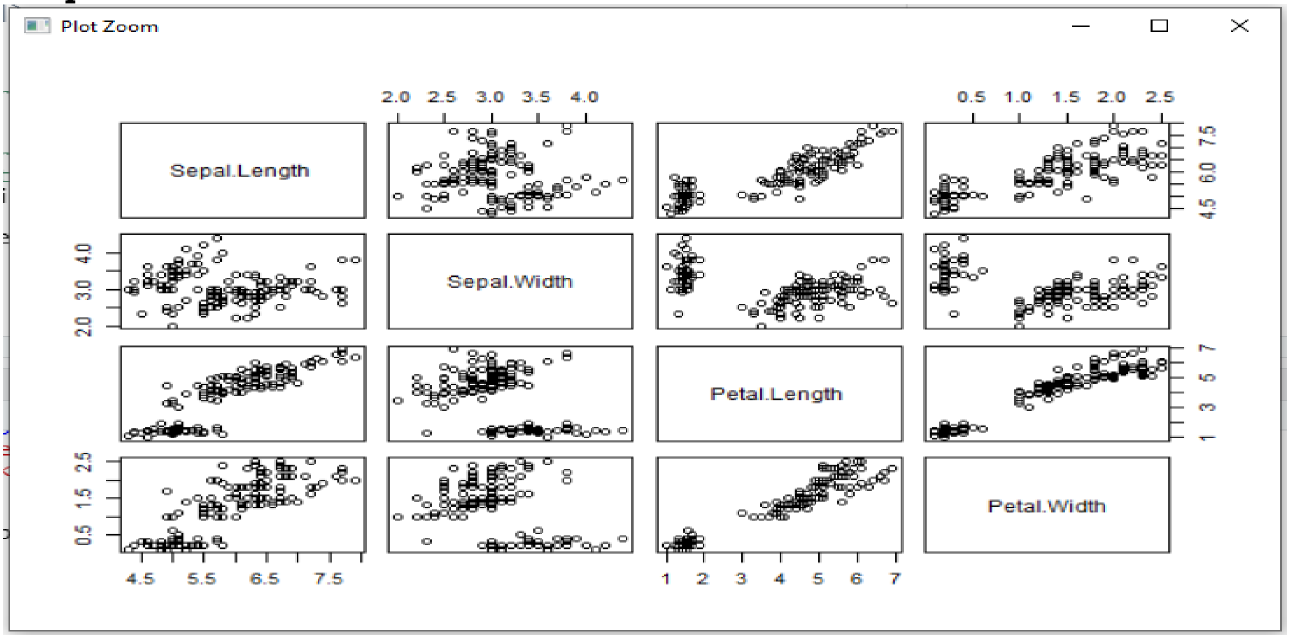
```
#+1 means variables are correlated, -1 inversely correlated.
corr<- cor(iris[,1:4])
round(corr,3)
```

Output:

```
> corr <- cor(iris[,1:4])
> round(corr,3)
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length        1.000      -0.118        0.872       0.818
Sepal.Width        -0.118       1.000       -0.428      -0.366
Petal.Length        0.872      -0.428        1.000       0.963
Petal.Width         0.818      -0.366        0.963       1.000
```

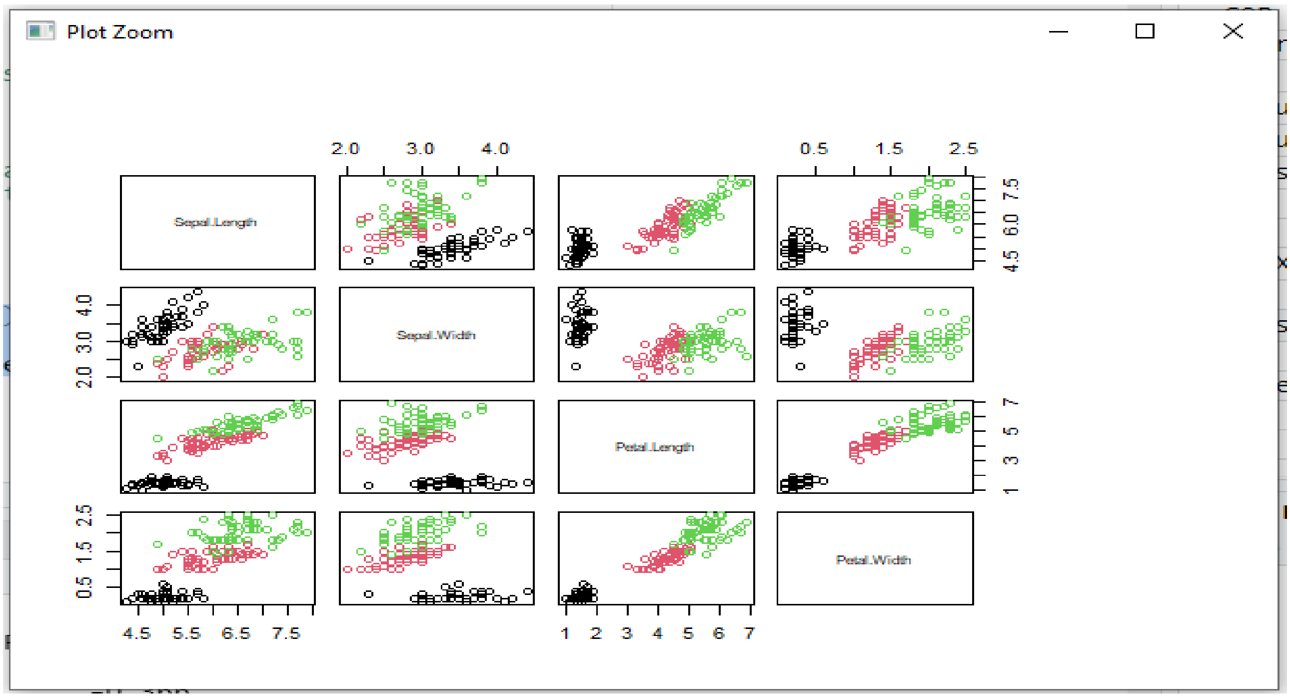
#Scatterplot matrices are very good visualization tools and
#may help identify correlations or lack of it:
pairs(iris[,1:4])

Output:



#Are the (visual) correlations different for each class?
#Let's color the points by the classes.
pairs(iris[,1:4],col=iris[,5],oma=c(4,4,6,12))
par(xpd=TRUE)
legend(0.85,0.6, as.vector(unique(iris$Species)),fill=c(1,2,3))

Output:



6. Import a data from web storage. Name the dataset and now do Logistic Regression to find out relation between variables that are affecting the admission of a student in a institute based on his or her GRE score, GPA obtained and rank of the student. Also check the model is fit or not. require (foreign), require(MASS).

Reading Data into R

>data <- read.csv("binary.csv")

>View(data)

	ADMIT	GRE	GPA	RANK
1	0	380	3.61	3
2	1	660	3.67	3
3	1	800	4.00	1
4	1	640	3.19	4
5	0	520	2.93	4
6	1	760	3.00	2
7	1	560	2.98	1

Data Cleaning

Looking at the structure of the dataset

```
str(data)
```

```
## 'data.frame':    400 obs. of  4 variables:
##  $ ADMIT: num  0 1 1 1 0 1 1 0 1 0 ...
##  $ GRE  : num  380 660 800 640 520 760 560 400 540 700 ...
##  $ GPA  : num  3.61 3.67 4 3.19 2.93 ...
##  $ RANK : num  3 3 1 4 4 2 1 2 3 2 ...
##  - attr(*, "label")= chr "LOGIT"
```

Variables **ADMIT** and **RANK** are of type numeric but they should be factor variables since were are not going to perform any mathematical operations on them.

```
data$ADMIT <- as.factor(data$ADMIT)

data$RANK <- as.factor(data$RANK)

str(data)
```

```
## 'data.frame':    400 obs. of  4 variables:
##  $ ADMIT: Factor w/ 2 levels "0","1": 1 2 2 2 1 2 2 1 2 1 ...
##  $ GRE  : num  380 660 800 640 520 760 560 400 540 700 ...
##  $ GPA  : num  3.61 3.67 4 3.19 2.93 ...
##  $ RANK : Factor w/ 4 levels "1","2","3","4": 3 3 1 4 4 2 1 2 3 2 ...
```

```
## - attr(*, "label")= chr "LOGIT"
```

Looking at the summary of the dataset

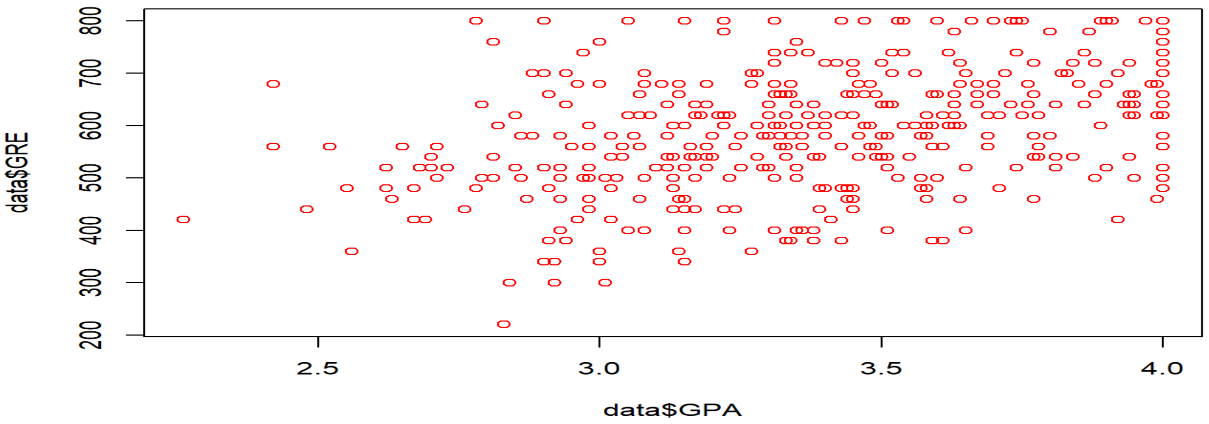
```
summary(data)
## ADMIT      GRE            GPA          RANK
## 0:273  Min.   :220.0  Min.   :2.260  1: 61
## 1:127  1st Qu.:520.0  1st Qu.:3.130  2:151
##        Median :580.0  Median :3.395  3:121
##        Mean   :587.7  Mean   :3.390  4: 67
##        3rd Qu.:660.0  3rd Qu.:3.670
##        Max.   :800.0  Max.   :4.000
```

From the summary statistics we observe

- Most of students did not get admitted
- There are no missing data values(NAs).

Checking for multicollineality

```
plot(data$GPA,data$GRE,col="red")
```




```
cor(data$GRE,data$GPA)
## [1] 0.3842659
```

From the plot we can infer that the two numeric variables are not correlated which is confirmed by low correlation value of **0.3842659**.

Exploratoty Data Analysis.

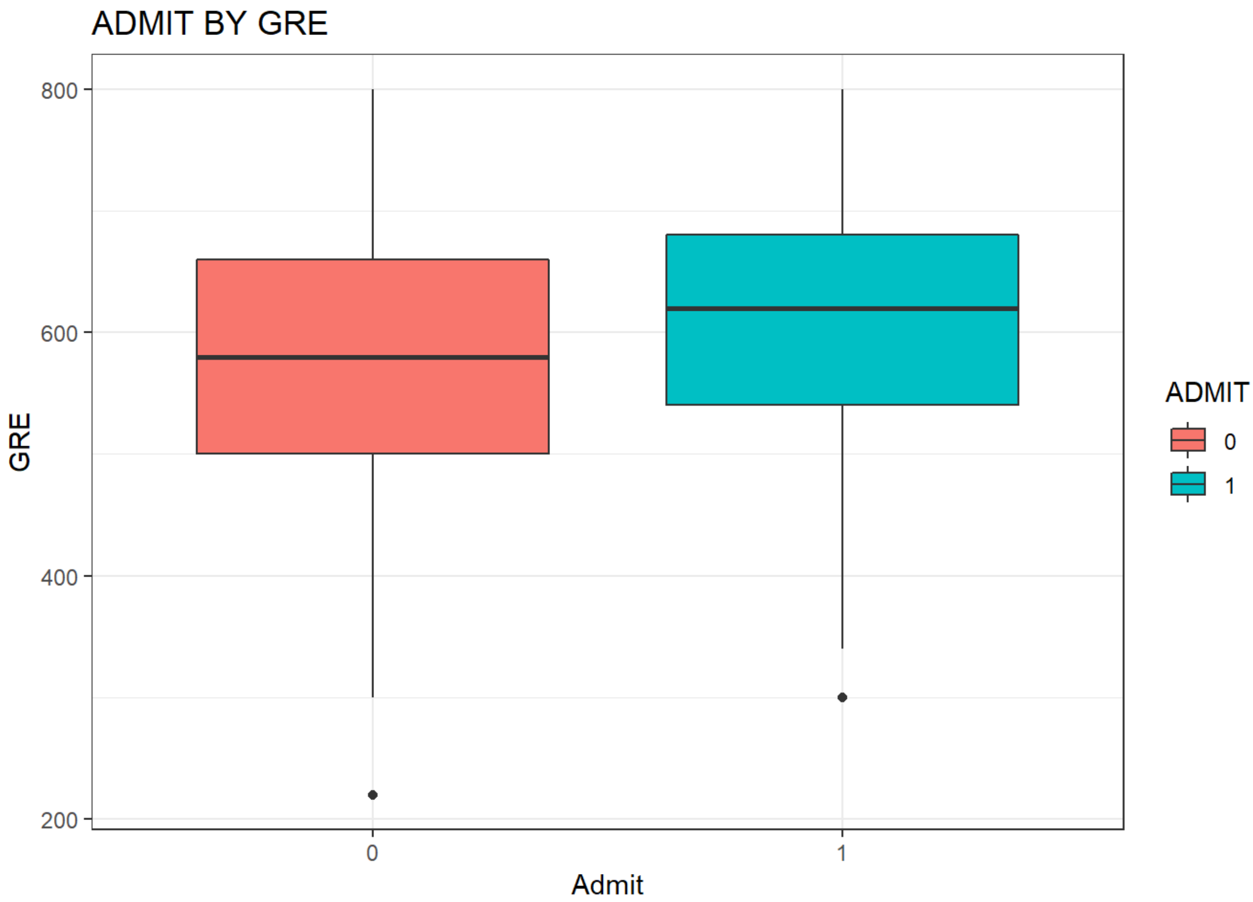
We will explore the relationship between dependent and independent variables by way of visualization.

GRE

Since GRE is numeric variable and dependent variable is factor variable, we plot a box plot.

```
library(ggplot2) #For plotting
ggplot(data,aes(ADMIT,GRE,fill=ADMIT))+
  geom_boxplot()+
```

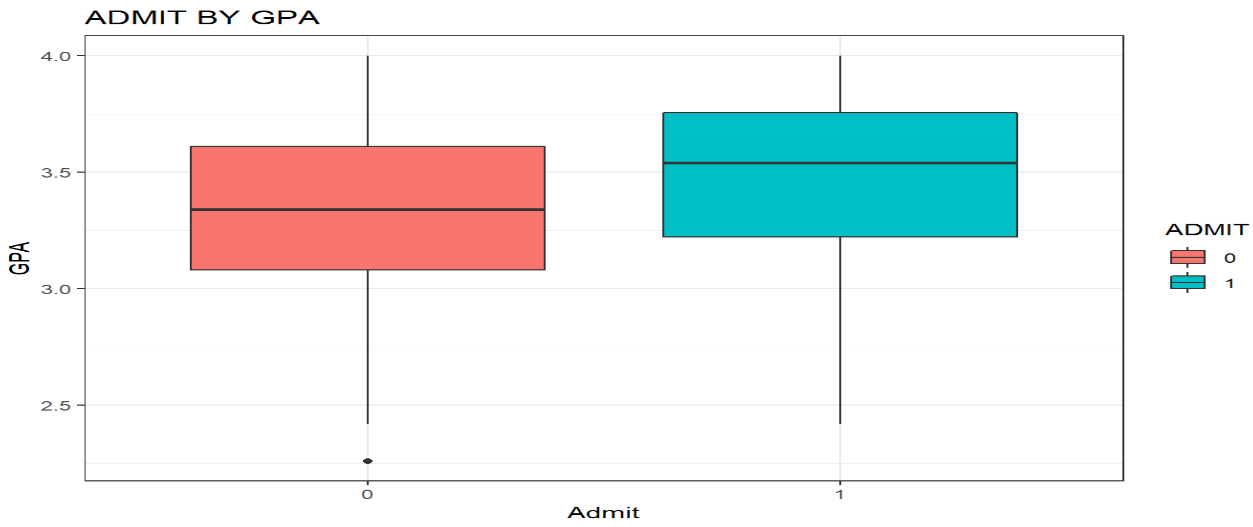
```
  theme_bw()+
  xlab("Admit")+
  ylab("GRE")+
  ggtitle("ADMIT BY GRE")
```



The two box plots are differents in terms of displacement, and hence *GRE* is significant variable.

GPA

```
ggplot(data,aes(ADMIT,GPA,fill=ADMIT))+
  geom_boxplot()+
  theme_bw()+
  xlab("Admit")+
  ylab("GPA")+
  ggtitle("ADMIT BY GPA")
```

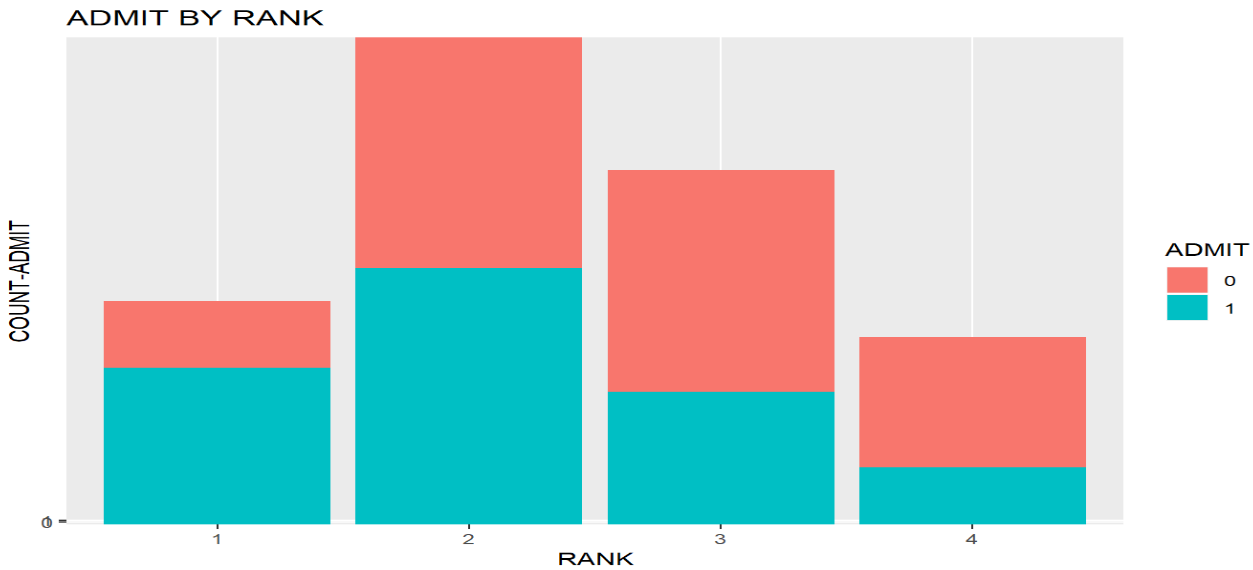


There is clear difference in displacement between the two box plots, hence *GPA* is an important predictor.

RANK

RANK is a factor variable and since the dependent variable is a factor variable we plot a bar plot.

```
ggplot(data,aes(RANK,ADMIT,fill=ADMIT))+
  geom_col()+
  xlab("RANK")+
  ylab("COUNT-ADMIT")+
  ggtitle("ADMIT BY RANK")
```



There is a clear pattern; as *RANK* increases the possibility of a student being admitted decreases.

Modelling

Data Splitting

Before we fit a model, we need to split the dataset into training and test dataset to be able to assess the performance of the model with the unseen test dataset.

```
library(caret)  #For data spliting
set.seed(125)   #For reproducibiity
ind <- createDataPartition(data$ADMIT,p=0.80,list = FALSE)
training <- data[ind,] #training data set
testing <- data[-ind,] #Testing data set
```

Fitting a logistic regression model

We will exclude the insignificant variable *GRE* we saw during EDA.

```
set.seed(123)
mymodel <- glm(ADMIT~GPA + RANK,data=training,family=binomial(link = "logit")
)
summary(mymodel)
```

```
##
## Call:
## glm(formula = ADMIT ~ GPA + RANK, family = binomial(link = "logit"),
##     data = training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4258  -0.8728  -0.6686   1.1715   2.0585
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.7638     1.1877  -2.327 0.019965 *
## GPA           0.8328     0.3358   2.480 0.013148 *
## RANK2        -0.5532     0.3531  -1.567 0.117152
## RANK3        -1.3937     0.3867  -3.604 0.000313 ***
## RANK4        -1.6086     0.4757  -3.381 0.000721 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 401.36  on 320  degrees of freedom
## Residual deviance: 374.80  on 316  degrees of freedom
## AIC: 384.8
##
## Number of Fisher Scoring iterations: 4
```

From the model summary, we can see that all the predictor variables are significant as we expected.

Making Predictions on Testing dataset

We evaluate the accurancy of the model by making predictions on the testing dataset.

```
pred <- predict(mymodel,testing,type = "response")
pred <- ifelse(pred>=0.5,1,0)
```

Creating a confusion matrix

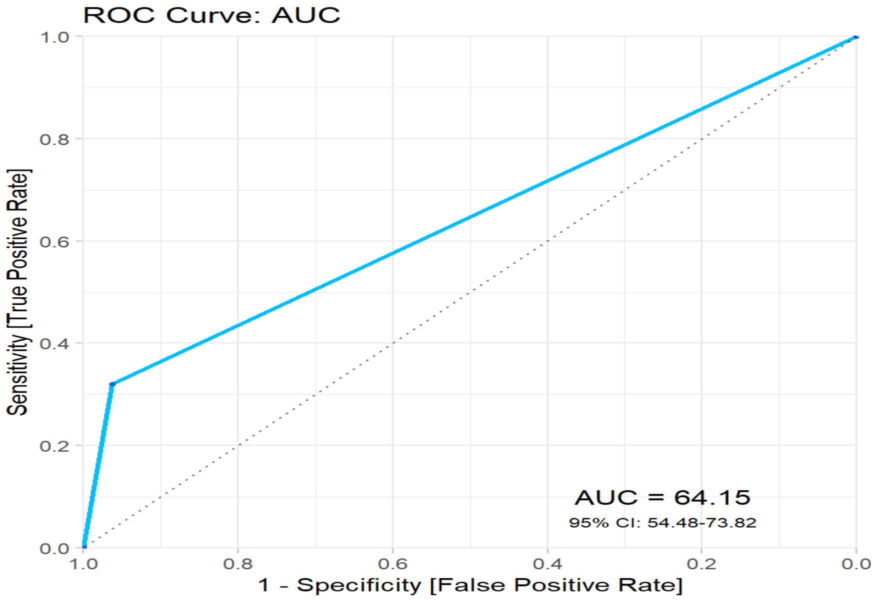
```
pred <- as.factor(pred)
confusionMatrix(pred,testing$ADMIT)
```

```
## Confusion Matrix and Statistics
 
##           Reference
## Prediction  0  1
##          0 52 17
##          1  2  8
##                                           
##                Accuracy : 0.7595          
##                  95% CI : (0.6502, 0.8486)
##     No Information Rate : 0.6835          
##     P-Value [Acc > NIR] : 0.089451        
##                                           
##                   Kappa : 0.3373          
##  Mcnemar's Test P-Value : 0.001319        
##                                           
##             Sensitivity : 0.9630          
##             Specificity : 0.3200          
##          Pos Pred Value : 0.7536          
##          Neg Pred Value : 0.8000          
##              Prevalence : 0.6835          
##          Detection Rate : 0.6582          
##    Detection Prevalence : 0.8734          
##       Balanced Accuracy : 0.6415          
##                                           
##        'Positive' Class : 0               
## 
```

From the confusion matrix we have overall accurracy of *76%*.

Calculating Area under the curve

```
library(lares)
tag.1 <- as.numeric(testing$ADMIT)
score.1 <- as.numeric(pred)
mplot_roc(tag=tag.1,score=score.1)
```



We have an *AUC* of about 64%, we which considerably good bearing in mind that our data set was small and imbalanced response variable

Results and Communication

The following equation fits our model.

$$\log(p/1-p)=y=-2.7638+(0.8328*GPA)+(-0.5532*RANK2)+(-1.3937*RANK3)+(-1.6086*RANK4)$$

$$\log(p/1-p)=y=-2.7638+(0.8328*GPA)+(-0.5532*RANK2)+(-1.3937*RANK3)+(-1.6086*RANK4)$$

Where **p** is the probability of a student being admitted.

Goodness-of-fit test

We test the fitness of our model by calculating the p-value

```
with(mymodel,pchisq(null.deviance-deviance,df.null-df.residual,lower.tail = F))
## [1] 2.447434e-05
```

With a **p-value of 0.00002447434** which is less than **0.05**, we conclude that our model is statistically significant.

7. **CLASSIFICATION MODEL**
   a. Install relevant package for classification.
   b. Choose classifier for classification problem.
   c. Evaluate the performance of classifier.

To predict if the car requires servicing or not using knn algorithm.

knn algorithm is a supervised learning algorithm which is primarily used as a classification algorithm.

- knn stands for K nearest neighbour.
- It is a lazy learning algorithm.
- It works on majority voting method

Problem Statement

The data sets contains 6 variables in it. Each column contains a particular information which would help in knowing if service for a particular vehicle is needed or not.

The first 5 columns gives us reading about the vehicle and the 6th tells us if the vehicle requires service or not Two data sets are given :-

1. Train Data : This data set contains 6 variables. The first 5 being the data of the vehicle and the 6th variable being the result whether the car requires servicing or not.
2. Test Data : This data set is to check the quality of the algorithm built using train data.

```
library(caret)# package used to create confusion matrix
## Loading required package: lattice
## Loading required package: ggplot2
library(readr)# package used to read/import files
library(class)# package used for knn algorithm
serviceTestData <- read.csv("serviceTestData.csv")# assigning the
test data to a variable
serviceTrainData <- read.csv("serviceTrainData.csv")# assigning th
e train data to a variable
#Viewing the data imported
View(serviceTestData)
View(serviceTrainData)
#Sructure of the data
str(serviceTestData)
```

'data.frame':   135 obs. of 6 variables:

```
##  $ OilQual    : num  45.77 4.99 4.99 106.39 104.39 ...
##  $ EnginePerf : num  49.94 7.89 4.89 104.45 103.74 ...
##  $ NormMileage: num  49.78 6.59 7.31 103.05 103.05 ...
##  $ TyreWear   : num  48.26 9.49 8.37 106.28 106.13 ...
##  $ HVACwear   : num  50.95 3.24 2.78 105.54 105.78 ...
##  $ Service    : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 2 2 1 ...
```

```
str(serviceTrainData)
```

```
## 'data.frame':    315 obs. of  6 variables:
##  $ OilQual    : num  103.4 26.8 62.4 45.5 104.4 ...
##  $ EnginePerf : num  103.5 26.2 63.7 49.9 103.3 ...
##  $ NormMileage: num  103.1 31.3 59.7 48.8 103.1 ...
##  $ TyreWear   : num  106.2 29.2 64.7 48.1 105.8 ...
##  $ HVACwear   : num  105.7 31.3 58.6 48 106.5 ...
##  $ Service    : Factor w/ 2 levels "No","Yes": 1 2 2 1 1 1 1 1 1 1 ...
```

```
#summarising the data
summary(serviceTrainData)
```

```
##     OilQual           EnginePerf       NormMileage        TyreWear
##  Min.   :  0.9872   Min.   :  1.891   Min.   :  3.359   Min.   :  6.213
##  1st Qu.: 26.7655   1st Qu.: 27.418   1st Qu.: 31.260   1st Qu.: 29.036
##  Median : 59.6633   Median : 59.741   Median : 57.221   Median : 60.304
##  Mean   : 59.6493   Mean   : 60.306   Mean   : 60.297   Mean   : 61.759
##  3rd Qu.:104.3888   3rd Qu.:103.744   3rd Qu.:103.051   3rd Qu.:106.173
##  Max.   :106.4288   Max.   :105.744   Max.   :105.051   Max.   :108.173
##     HVACwear      Service
##  Min.   : -1.72   No :232
##  1st Qu.: 31.34   Yes: 83
##  Median : 60.62
##  Mean   : 60.39
##  3rd Qu.:105.54
##  Max.   :107.54
summary(serviceTestData)
##     OilQual          EnginePerf       NormMileage        TyreWear
##  Min.   :  2.597   Min.   :  1.891   Min.   :  3.589   Min.   :  6.143
##  1st Qu.: 26.696   1st Qu.: 27.418   1st Qu.: 31.260   1st Qu.: 28.901
##  Median : 61.023   Median : 61.501   Median : 59.351   Median : 61.304
##  Mean   : 58.629   Mean   : 59.077   Mean   : 59.118   Mean   : 60.864
##  3rd Qu.:104.229   3rd Qu.:103.744   3rd Qu.:103.051   3rd Qu.:106.173
##  Max.   :106.389   Max.   :105.744   Max.   :105.051   Max.   :108.173
##     HVACwear      Service
##  Min.   : -1.72   No :99
##  1st Qu.: 31.31   Yes:36
##  Median : 62.62
##  Mean   : 58.99
##  3rd Qu.:105.33
##  Max.   :105.83
```

```
#Applying the knn algorithm and assigning it to a variable
```

```r
#In the train and test part of the data assigned in the algorithm, 6th column is removed because it contains the result that if a ser vice is required or not

predictedknn <- knn(train=serviceTrainData[,-6],
                    test=serviceTestData[,-6],
                    cl=serviceTrainData$Service,
                    k=3)

predictedknn
```

```
##   [1] No  No  No  No  No  No  No  Yes Yes No  No  No  No  No  No  No  Yes
##  [18] No  No  Yes Yes No  Yes No  No  No  No  No  No  No  No  No  No  No
##  [35] No  Yes No  No  No  No  No  Yes No  Yes No  No  No  Yes Yes No  Yes
##  [52] No  Yes No  No  No  No  No  No  No  No  No  No  No  Yes Yes Yes No
##  [69] Yes No  No  Yes No  No  No  No  No  No  Yes Yes Yes Yes No  Yes No
##  [86] No  Yes Yes Yes No  No  No  Yes No  Yes No  No  No  No  No  No  No
## [103] No  No  No  Yes No  No  No  No  No  Yes No  Yes No  Yes Yes No  Yes
## [120] No  No  No  No  No  Yes No  No  No  No  No  No  No  Yes No  No
## Levels: No Yes
```

```r
#Creating a confusion matrix to know how many right predictions ar e done

conf_matrix <- confusionMatrix(data=predictedknn,serviceTestData$S ervice)

conf_matrix
```

```
## Confusion Matrix and Statistics
##           Reference
## Prediction No Yes
##        No  99   0
##        Yes  0  36
##              Accuracy : 1
##                95% CI : (0.973, 1)
##     No Information Rate : 0.7333
##     P-Value [Acc > NIR] : < 2.2e-16
##                 Kappa : 1
##  Mcnemar's Test P-Value : NA
##           Sensitivity : 1.0000
##           Specificity : 1.0000
##        Pos Pred Value : 1.0000
##        Neg Pred Value : 1.0000
##            Prevalence : 0.7333
##        Detection Rate : 0.7333
##    Detection Prevalence : 0.7333
##       Balanced Accuracy : 1.0000
##        'Positive' Class : No
```

The confusion matrix shows that the predictions done is all correct and hence the accuracy is 1.

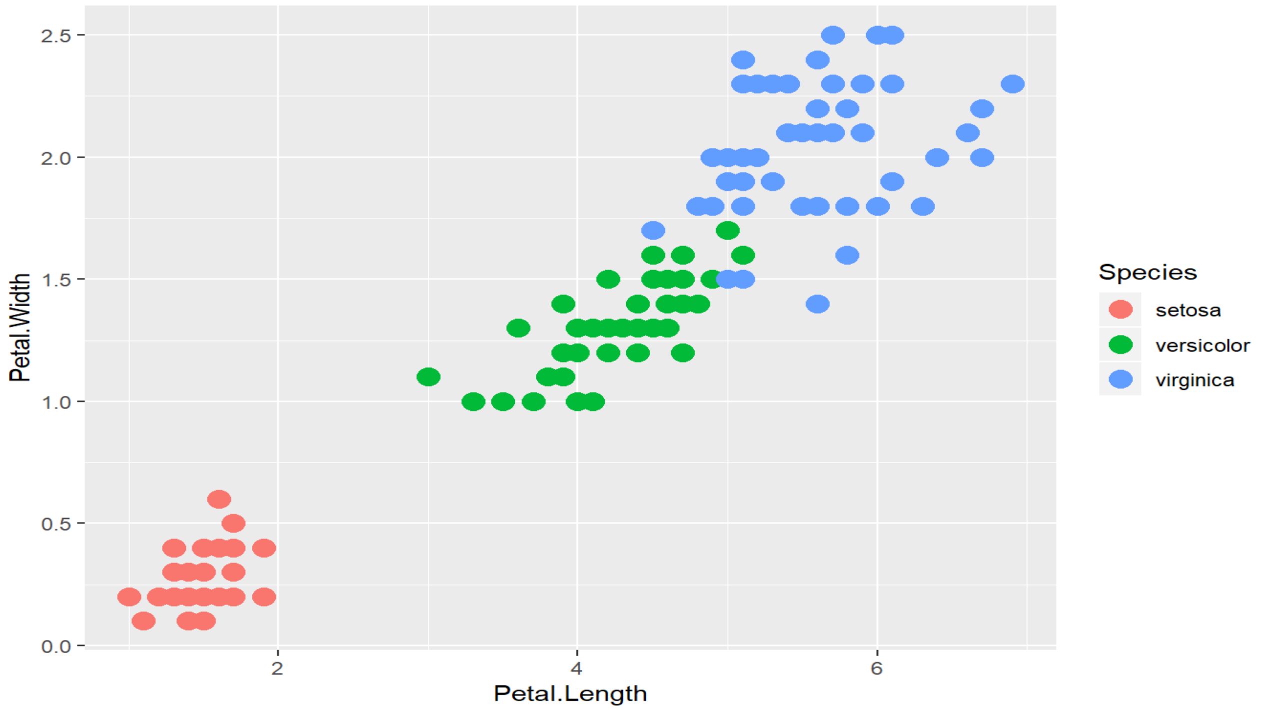
8. **CLUSTERING MODEL**
   a. Clustering algorithms for unsupervised classification.
   b. Plot the cluster data using R visualizations.

   c. K-means Clustering is used with unlabeled data, but in this case, we have a labeled dataset so we have to use the iris data without the Species column. In this way, algorithm will cluster the data and we will be able to compare the predicted results with the original results, getting the accuracy of the model.

```
library(ggplot2)
df <- iris
head(iris)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Let's make a scatterplot.

```
ggplot(df, aes(Petal.Length, Petal.Width)) + geom_point(aes(col=Species), size=4
)
```



As we can see, setosa is going to be clustered easier. Meanwhile, there is noise between versicolor and virginica even when they look like perfectly clustered.

Let's run the model. kmeans is installed in the base package from R, so we don't have to install any package.

In the kmeans function, it is necessary to set center, which is the number of groups we want to cluster to. In this case, we know this value will be 3. Let's set that.

, but let's see how we would build the model if we didn't know it.

```
set.seed(101)

irisCluster <- kmeans(df[,1:4], center=3, nstart=20)

irisCluster
```

```
## K-means clustering with 3 clusters of sizes 38, 62, 50
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     6.850000    3.073684     5.742105    2.071053
## 2     5.901613    2.748387     4.393548    1.433871
## 3     5.006000    3.428000     1.462000    0.246000
##
## Clustering vector:
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1
## [106] 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 1 1 2 1
## [141] 1 1 2 1 1 1 2 1 1 2
##
## Within cluster sum of squares by cluster:
## [1] 23.87947 39.82097 15.15100
##  (between_SS / total_SS =  88.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

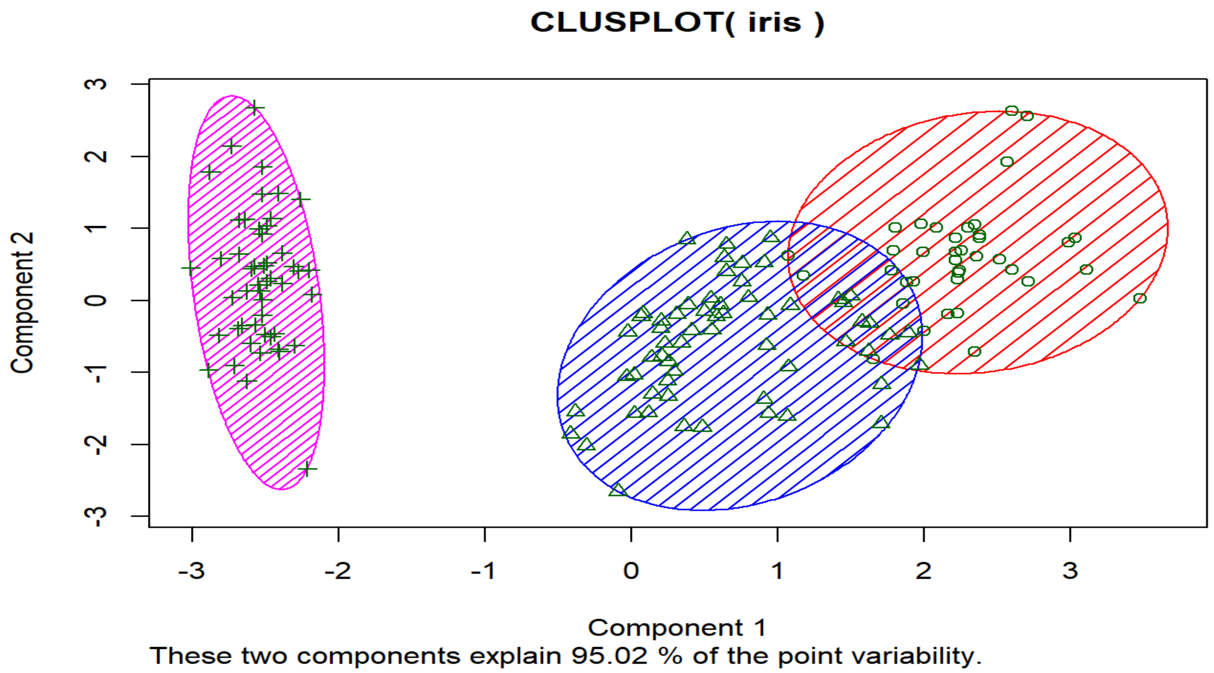
We can compare the predicted clusters with the original data.

```
table(irisCluster$cluster, df$Species)
```

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

We can plot out these clusters.

```
library(cluster)

clusplot(iris, irisCluster$cluster, color=T, shade=T, labels=0, lines=0)
```



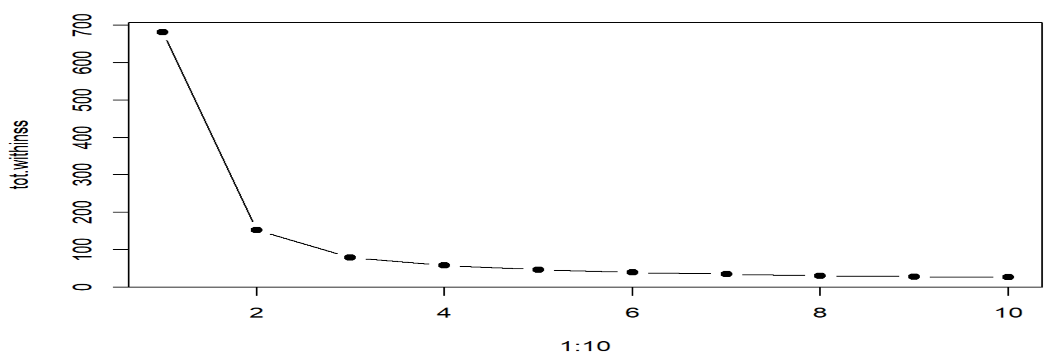
We can see the setosa cluster perfectly explained, meanwhile virginica and versicolor have a little noise between their clusters.

As said before, we will not always have the labeled data. If we would want to know the exactly number of centers, we should have built the elbow method.

```
tot.withinss <- vector(mode="character", length=10)
for (i in 1:10){
  irisCluster <- kmeans(df[,1:4], center=i, nstart=20)
  tot.withinss[i] <- irisCluster$tot.withinss
}
```

Let's visualize it.

```
plot(1:10, tot.withinss, type="b", pch=19)
```



As we saw, the optimal number of clusters is 3.

ADDITIONAL PROGRAMS

9) write a program to find given no is even or odd

```
x <-24
if(x%%2==0){
    cat(x," is an even number")
}
if(x%%2!=0){
    cat(x," is an odd number")
}
```

10) write a program to find given year is leap or not

```
year1 = 2011
if(year1 %% 4 == 0) {
 if(year1 %% 100 == 0) {
     if(year1 %% 400 == 0) {
         cat(year,"is a leap year")
        } else {
         cat(year,"is not a leap year")
        }
    } else {
     cat(year,"is a leap year")
    }
} else {
 cat(year,"is not a leap year")
}
```

11) write a program to find greatest of four numbers

```
n1=4
n2=87
n3=43
n4=74
if(n1>n2){
    if(n1>n3&&n1>n4){
        largest=n1
    }
}else if(n2>n3){
    if(n2>n1&&n2>n4){
        largest=n2
    }
}else if(n3>n4){
    if(n3>n1&&n3>n2){
        largest=n3
    }
}else{
    largest=n4  }
cat("Largest number is =",largest)
```

12)write a program to find the sum of the digits of the number.

```
n<-readline(prompt="please enter any integer value: ")
please enter any integer value: 12367906
n <- as.integer(n)
sum<-0
while(n!=0){
    sumsum=sum+(n%%10)
    n=as.integer(n/10)
}
cat("sum of the digits of the numbers is=",sum)
```

13)write a program to find the frequency of a digit in the number.

```
num = as.integer(readline(prompt="Enter a number: "))
digit = as.integer(readline(prompt="Enter digit: "))
n=num
count = 0
while(num > 0) {
        if(num%%10==digit){
            countcount=count+1
        }
        num=as.integer(num/10)
}
print(paste("The frequency of",digit,"in",n,"is=",count))
```

14) **write a program to find the sum of squares of a given series of numbers using recursion**

Sum = $1^2+2^2+...+N^2$

```
sum_series <- function(vec){
     if(length(vec)<=1)
     {
          return(vec^2)
     }
     else
     {
          return(vec[1]^2+sum_series(vec[-1]))
     }
}
series <- c(1:10)
sum_series(series)
```

15) **Consider the annual rainfall details at a place starting from January 2012. Create an R time series object for a period of 12 months and plot it.**

Time series is a series of data points in which each data point is associated with a timestamp. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called time-series object. It is also a R data object like a vector or data frame.

The time series object is created by using the ts() function.

Syntax

The basic syntax for ts() function in time series analysis is –

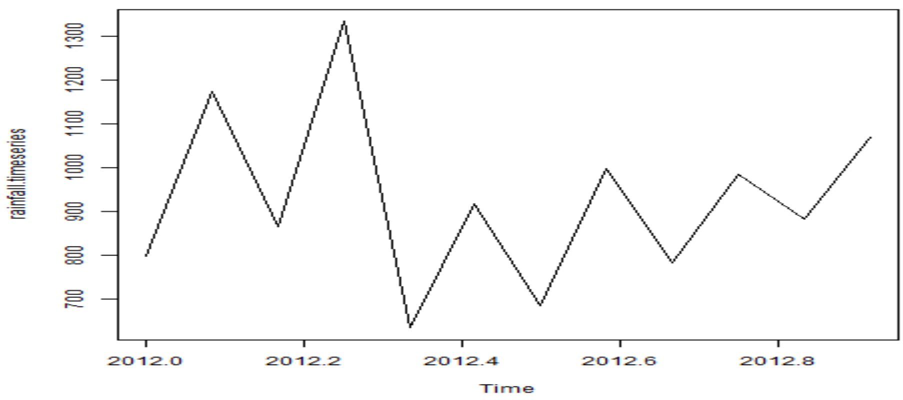
timeseries.object.name <- ts(data, start, end, frequency)

Following is the description of the parameters used −

data is a vector or matrix containing the values used in the time series.
start specifies the start time for the first observation in time series.
end specifies the end time for the last observation in time series.
frequency specifies the number of observations per unit time.

Except the parameter "data" all other parameters are optional.

```
Library(ggplot2)
# Get the data points in form of a R vector.
rainfall <-
c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,107
1)

# Convert it to a time series object.
rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)

# Print the timeseries data.
print(rainfall.timeseries)

# Give the chart file a name.
png(file = "rainfall.png")

# Plot a graph of the time series.
plot(rainfall.timeseries)

# Save the file.
dev.off()
getwd() # Find graph in this locations
```


rainfall.timeseries
700
800
900
1000
1100
1200
1300
2012.0
2012.2
2012.4
2012.6
2012.8
Time