

## INTRODUCTION:

1. R is a language and environment for statistical computing and graphics.
2. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues.
3. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.
4. R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.
5. One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed.
6. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.
7. R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.
8. R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes
  9. an effective data handling and storage facility,
  10. a suite of operators for calculations on arrays, in particular matrices,
  11. a large, coherent, integrated collection of intermediate tools for data analysis,
  12. graphical facilities for data analysis and display either on-screen or on hardcopy, and
  13. a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.
14. R can be extended (easily) via *packages*. There are about eight packages supplied with the R distribution and many more are available through the CRAN family of Internet sites covering a very wide range of modern statistics.

# EXPERIMENT1:

1. R operates on named data structures. The simplest such structure is the numeric vector, which is a single entity consisting of an ordered collection of numbers.
2. To set up a vector named x, say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7, use the R command `> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)`.
3. This is an assignment statement using the function `c()` which in this context can take an arbitrary number of vector arguments and whose value is a vector got by concatenating its arguments end to end.
4. A number occurring by itself in an expression is taken as a vector of length one. Notice that the assignment operator (`<=`), which consists of the two characters `<` and `=` (`assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))`)
5. The usual operator, `<=`, can be thought of as a syntactic short-cut to this.
6. Assignments can also be made in the other direction, using the obvious change in the assignment operator.
7. So the same assignment could be made using `> c(10.4, 5.6, 3.1, 6.4, 21.7) -> x`
8. Generating regular sequences
9. For example `> seq(-5, 5, by=.2) -> s3` generates in s3 the vector `c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0)`.

## Experiment1: (vectors & operations)

→ In r programming we can directly assign without data type. All arithmetic operations can perform directly on variables. (+, -, /, \*, ^, etc) **mod %%**

**X=5 → [5]**

**X=x+6 → [11]**

**Y=24%%5→[4]**

operation	example	result
initialization	<code>X=c(1,2,5)</code>	<code>[1,2,5]</code>
Add element	<code>X=append(x,5)</code>	<code>[1,2,5,5]</code>
Add elements	<code>X=append(x,c(6,8))</code>	<code>[1,2,5,5,6,8]</code>
Append 2 vectors	<code>Y=c(11,12,13)</code> <code>X=append(x,y)</code>	<code>[1,2,5,5,6,8,11,12,13]</code>
Add at any position	<code>X[3]=85</code>	<code>[1,2,5,85,5,6,8,11,12,13]</code>
delete	<code>X=x[-2]</code>	<code>[1,2,5,6,8,11,12,13]</code>
Summary of X	<code>summary(x)</code>	Min. 1st Qu. Median Mean 3rd Qu. Max. 1.00 5.75 9.50 17.62 12.25 85.00
Sum of X	<code>sum(x)</code>	141

Printing X	print(x)	1 85 5 6 8 11 12 13
Factor of X	factor(x)	num=factor(x)
Is.factor of X (Returns A True Boolean Value if it is a Factor)	is.factor(x)	Is.factor(x) TRUE
Mean of X	mean(x)	17.65
Median of X	median(x)	9.5
Variance of X	var(x)	757.145
Standard Deviation of X	sd(x)	27.5159
Scale of X	scale(x)	[,1] [1,] -0.6041960 [2,] 2.4485839 [3,] -0.4588255 [4,] -0.4224829 [5,] -0.3497977 [6,] -0.2407698 [7,] -0.2044272 [8,] -0.1680846 attr(,"scaled:center") [1] 17.625 attr(,"scaled:scale") [1] 27.5159
Quantile of X	quantile(x)	0% 25% 50% 75% 100% 1.00 5.75 9.50 12.25 85.00

## Experiment2: Import & export .csv files:

- A CSV (comma-separated values) file is a text file in which information is separated by commas. CSV files are most commonly encountered in spreadsheets and databases. You can use a CSV file to move data between programs that aren't ordinarily able to exchange data.
- We can read and write on csv files using r commands like read and write. The data of csv can be placed into a data frame. A data frame is similar to matrix which has no of rows and columns.

```
X<-read.csv("C:\\Users\\mvs\\Desktop\\marks.csv")
```

```
  Names Marks
1    Ram    99
2    Sita   98
3 Lakshman 100
```

```
Print(x)
```

It will display the contents of x which is similar to given marks.csv file.

In the similar manner we can create or write the data into csv file using write command.

```
df <-data.frame(Column1 = c("Value 1", "Value 2") ,
                Column2 =c("Value 1", "Value 2", ...))

print (df)

write.csv(df,"Path")

write.csv(df,"Path", row.names = TRUE)
```

Example:

```
X=data.frame(names=c("Ram","Sita","lakshman"),Marks=c(99,98,100))
```

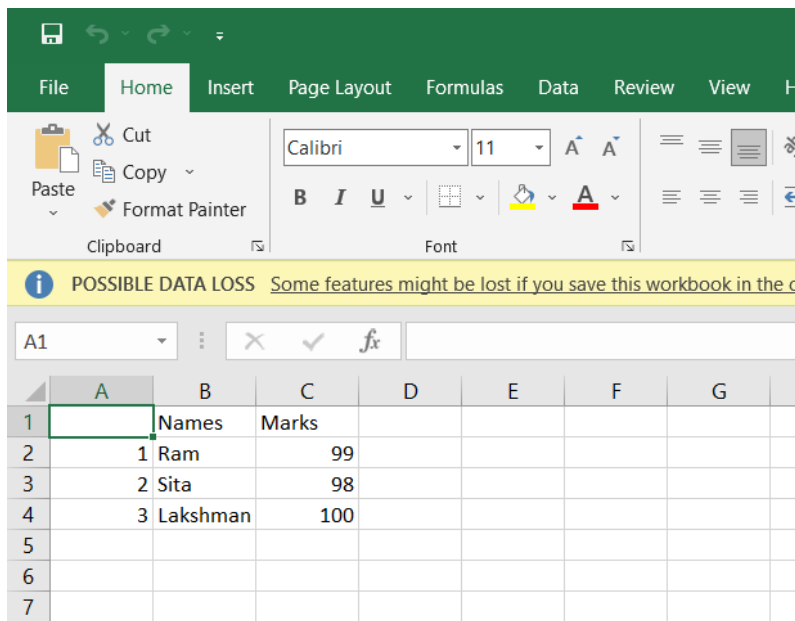
X.

By opening that csv file we can find the data of that data frame.

**Write function can use rownames arguments as either TRUE OR FALSE.**

```
write.csv(x,"c:\r1ab\marks2.csv", row.names = TRUE)
```

then a csv file is created and data is stored in that csv file.



	A	B	C	D	E	F	G
1		Names	Marks				
2	1	Ram	99				
3	2	Sita	98				
4	3	Lakshman	100				
5							
6							
7							

```
write.csv(df,"Path", row.names = FALSE)
```

	A	B	C	D	E	F	G
1		Names	Marks				
2	1	Ram	99				
3	2	Sita	98				
4	3	Lakshman	100				
5							
6							
7							

But row names will not present in the above file.

### to add the rows to the existing csv file

```
row <- data.frame('1', 'A', 'Manager', '24')
```

```
# sample csv name
csv_fname = "sample.csv"
```

### to add the rows to the existing csv file

```
# writing row in the csv file
write.table(row, file = csv_fname, sep = ",",
            append = TRUE, quote = FALSE,
            col.names = FALSE, row.names = FALSE)
```

```
data_frame <- data.frame(ID = c(8:9),
                        Name = c("K", "L"),
                        Post= c("IT", "Writer"),
                        Age = c(18, 27))
```

```
# writing contents of the file
content <- write.table(data_frame , path, append = T ,
                      col.names = FALSE, sep = ",",
                      row.names = F)
```

```
print(content)
```

To actually merge multiple CSV/Excel files as one data frame first the required packages are imported and then list of files are read and joined together.

```
data1 <- data.frame(id = 1:6,  
# Create first example data frame  
                    x1 = c(5, 1, 4, 9, 1, 2),  
                    x2 = c("A", "Y", "G", "F", "G", "Y"))  
  
data2 <- data.frame(id = 4:9,  
# Create second example data frame  
                    y1 = c(3, 3, 4, 1, 2, 9),  
                    y2 = c("a", "x", "a", "x", "a", "x"))  
  
data3 <- data.frame(id = 5:6,  
# Create third example data frame  
                    z1 = c(3, 2),  
                    z2 = c("K", "b"))  
  
write.csv(data1, "C:/...Your Path.../my_folder/data1.csv")  
# Write first example data frame  
write.csv(data2, "C:/...Your Path.../my_folder/data2.csv")  
# Write second example data frame  
write.csv(data3, "C:/...Your Path.../my_folder/data3.csv")  
# Write third example data frame
```

The required packages are

```
install.packages("dplyr")  
# Install dplyr package  
install.packages("plyr")  
# Install plyr package  
install.packages("readr")  
# Install readr package  
  
library("dplyr")  
# Load dplyr package  
library("plyr")  
# Load plyr package  
library("readr")
```

to create the data frame that copies the data from multiple csv files

```
data_all <- list.files(path = "C:/...Your Path.../my_folder",  
pattern = "*.csv", full.names = TRUE) %>%lapply(read_csv)  
%>%bind_rows
```

```
> data_all  
# A tibble: 17 x 12  
  ...1 id x1 x2 y1 y2 z1 z2 `1` A Manager `24`  
  <dbl> <dbl> <dbl> <chr> <dbl> <chr> <dbl> <chr> <dbl> <chr> <chr> <dbl>  
1 1 1 1 5 A NA NA NA NA NA NA NA NA  
2 2 2 1 Y NA NA NA NA NA NA NA NA  
3 3 3 4 G NA NA NA NA NA NA NA NA  
4 4 4 9 F NA NA NA NA NA NA NA NA  
5 5 5 1 G NA NA NA NA NA NA NA NA  
6 6 6 2 Y NA NA NA NA NA NA NA NA  
7 1 4 NA NA 3 a NA NA NA NA NA NA NA  
8 2 5 NA NA 3 x NA NA NA NA NA NA NA  
9 3 6 NA NA 4 a NA NA NA NA NA NA NA  
10 4 7 NA NA 1 x NA NA NA NA NA NA NA  
11 5 8 NA NA 2 a NA NA NA NA NA NA NA  
12 6 9 NA NA 9 x NA NA NA NA NA NA NA  
13 1 5 NA NA NA NA 3 K NA NA NA NA NA  
14 2 6 NA NA NA NA 2 b NA NA NA NA NA  
15 NA NA NA NA NA NA 1 A Manager 24  
16 NA NA NA NA NA NA 8 K IT 18  
17 NA NA NA NA NA NA 9 L writer 27  
> |
```



## EXPERIMENT3:

### Visualize Data Using Boxplot, Scatter plot, Bar plot, Histograms:

- The statistical summary of the given data is presented graphically using a boxplot. A boxplot depicts information like the minimum and maximum data point, the median value, first and third quartile, and interquartile range.
- A scatter plot is composed of many points on a Cartesian plane. Each point denotes the value taken by two parameters and helps us easily identify the relationship between them.
- There are two types of bar plots- horizontal and vertical which represent data points as horizontal or vertical bars of certain lengths proportional to the value of the data item. They are generally used for continuous and categorical variable plotting. By setting the **horiz** parameter to true and false, we can get horizontal and vertical bar plots respectively.
- A histogram is like a bar chart as it uses bars of varying height to represent data distribution. However, in a histogram values are grouped into consecutive intervals called bins. In a Histogram, continuous values are grouped and displayed in these bins whose size can be varied.

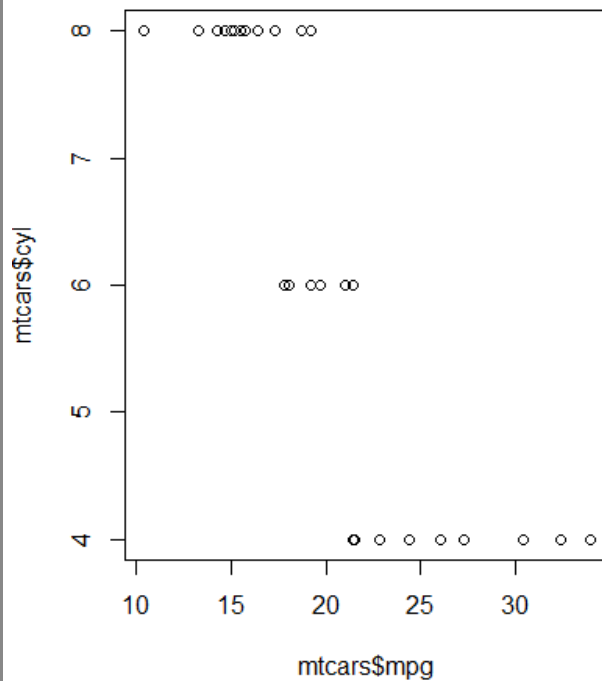
Example;

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

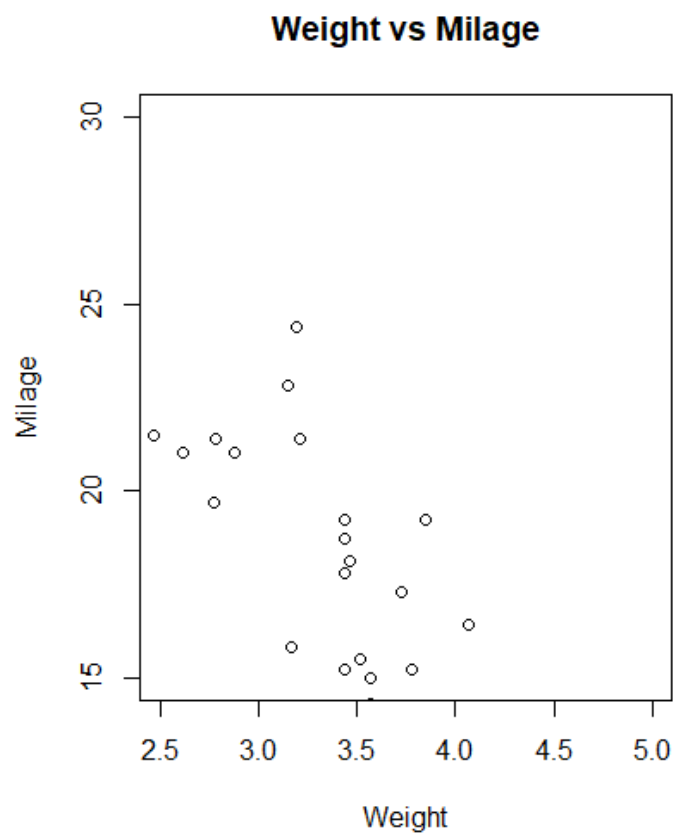
	mpg	cyl
Mazda RX4	21.0	6
Mazda RX4 Wag	21.0	6
Datsun 710	22.8	4
Hornet 4 Drive	21.4	6
Hornet Sportabout	18.7	8
Valiant	18.1	6

```
#scatter plot:
```

```
plot(x=mtcars$mpg,y=mtcars$cyl)
```

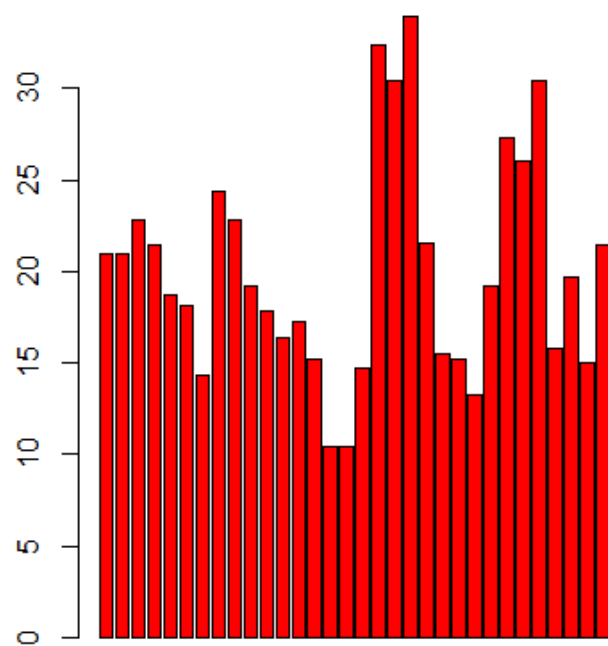


```
plot(x = mtcars$wt,y = mtcars$mpg,  
      xlab = "Weight",  
      ylab = "Milage",  
      xlim = c(2.5,5),  
      ylim = c(15,30),  
      main = "Weight vs Milage")
```



```
plot(mtcars$mpg,type='b')
```

```
> barplot(mtcars$mpg,type='h',col='red')
```



Plot the matrices between 4 variables giving 12 plots.

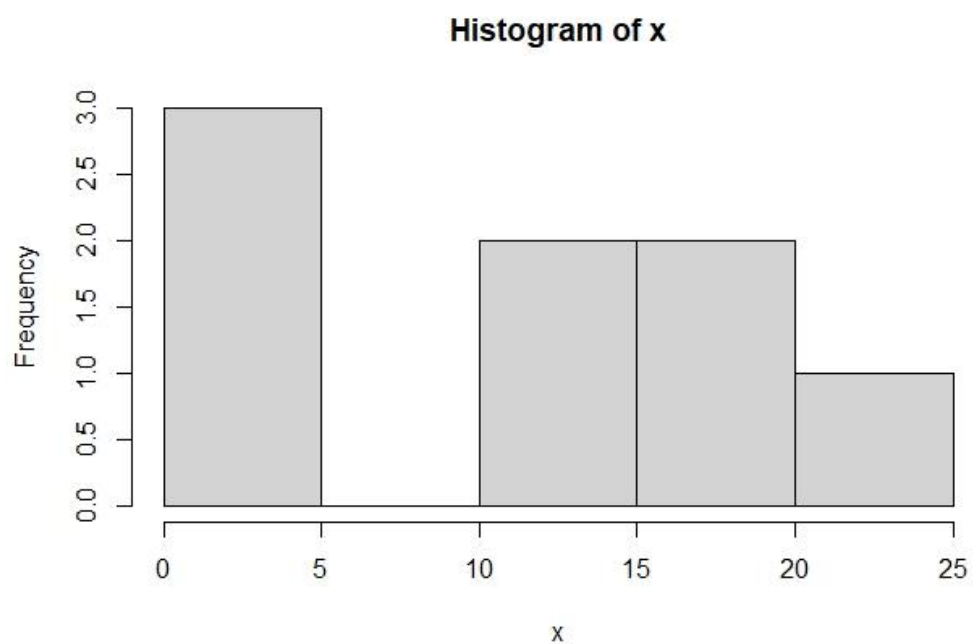
# One variable with 3 others and total 4 variables.

```
pairs(~wt+mpg+disp+cyl,data = mtcars,  
      mainx=hi = "Scatterplot Matrix")
```

#histograms

```
x=c(1,2,3,13,15,17,16,22)
```

```
> hist(x)
```

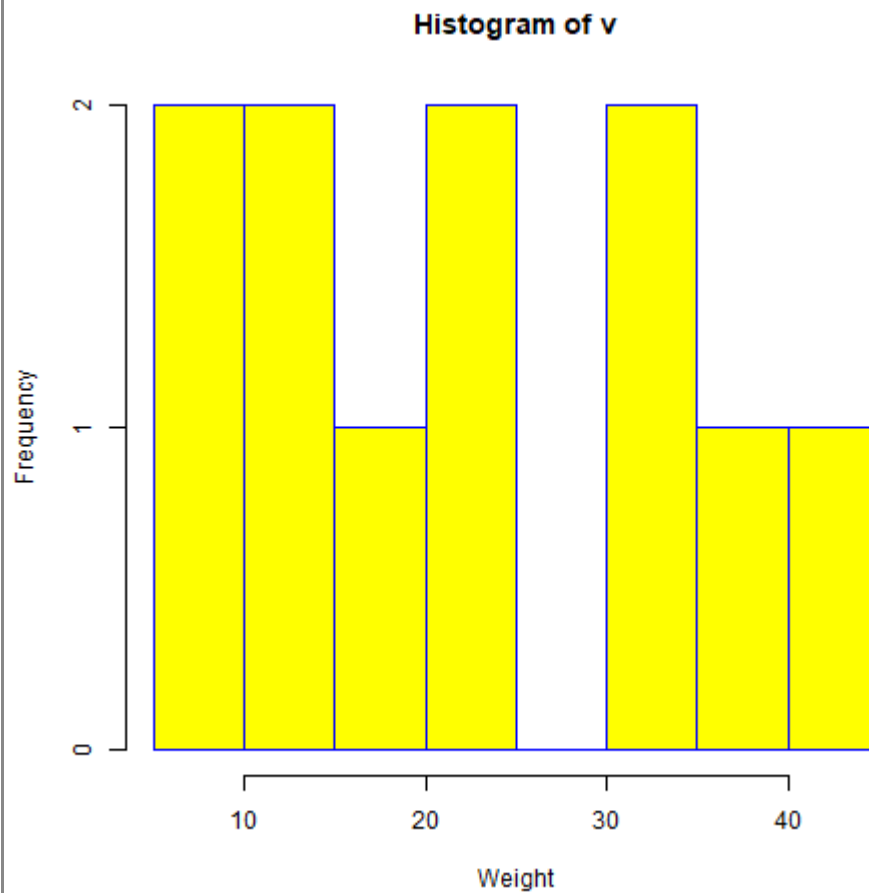


```
v <- c(9,13,21,8,36,22,12,41,31,33,19)
```

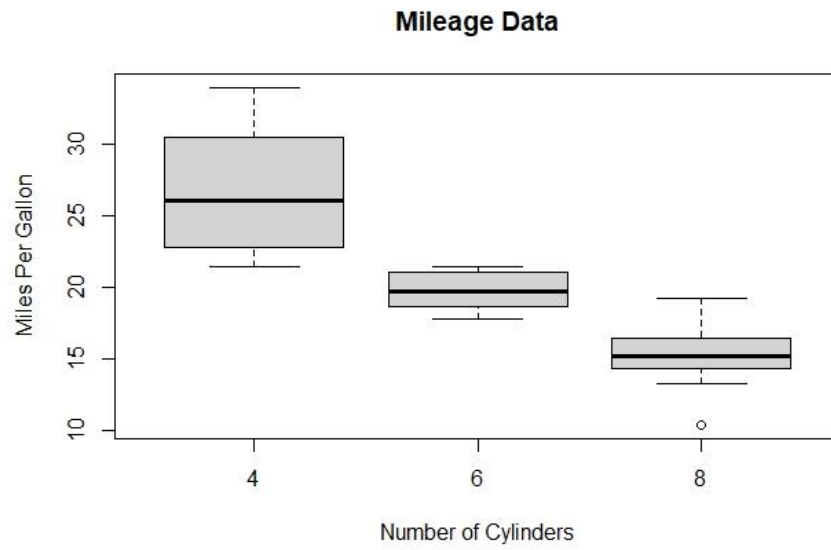
```
png(file = "histogram.png")
```

```
hist(v,xlab = "Weight",col = "yellow",border = "blue")
```

```
dev.off()
```



```
boxplot(mpg ~ cyl, data = mtcars, xlab = "No of Cylinders", ylab = "Miles Per Gallon",  
        main = "Mileage Data")
```



## EXPERIMENT4:

### Data frames in R:

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.

```
empdata <- data.frame(  
  emp_id = c(1:5),  
  emp_name = c("Rick","Dan","Michelle","Ryan"),  
  salary = c(623.3,515.2,611.0,729.0,843.25,45),  
  start_date = as.Date(c("2012-01-01", "2013-09-23",  
    "2014-11-15", "2014-05-11", "2015-03-27")),  
  stringsAsFactors = FALSE)
```

```
print(empdata)
```

```
  emp_id emp_name salary start_date  
1      1     Rick  623.3 2012-01-01  
2      2      Dan  515.2 2013-09-23  
3      3 Michelle  611.0 2014-11-15  
4      4     Ryan  729.0 2014-05-11
```

```
str(empdata) # It specifies structure
```

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

```
print(summary(empdata)) # summary information
```

```

      emp_id      emp_name      salary
Min.   :1.00   Length:4   Min.   :515.2
1st Qu.:1.75   Class :character 1st Qu.:587.0
Median :2.50   Mode  :character Median :617.1
Mean   :2.50                      Mean   :619.6
3rd Qu.:3.25                      3rd Qu.:649.7
Max.   :4.00                      Max.   :729.0

      start_date
Min.   :2012-01-01
1st Qu.:2013-04-18
Median :2014-01-16
Mean   :2013-09-27
3rd Qu.:2014-06-27
Max.   :2014-11-15

```

# Extract Specific columns.

```

result <- data.frame(empdata$emp_name,empdata$salary)
print(result)

```

```

      empdata.emp_name empdata.salary
1           Rick      623.3
2            Dan      515.2
3      Michelle      611.0
4            Ryan      729.0

```

# Extract first two rows.

```

result <- emp.data[1:2,] #rows or objects
print(result)

```

```

      empdata.emp_name empdata.salary
1           Rick      623.3
2            Dan      515.2
3      Michelle      611.0
4            Ryan      729.0

```

# Extract 3rd and 5th row with 2nd and 4th column.

```

result <- emp.data[c(3,5),c(2,4)] # specific rows with specific columns
print(result)

```

```

      emp_id emp_name salary start_date
1          1    Rick  623.3 2012-01-01
2          2     Dan  515.2 2013-09-23

```

```

empdata$dept <- c("IT","Operations","IT","HR","Finance")#add col

```

```

v <- empdata

```

```

print(v)

```

```

      emp_id emp_name salary start_date      dept
1          1    Rick  623.3 2012-01-01         IT
2          2     Dan  515.2 2013-09-23 Operations
3          3 Michelle  611.0 2014-11-15         IT
4          4     Rvan  729.0 2014-05-11         HR

```

# Create the second data frame & add a record or object or row

```

empnewdata <-      data.frame(
  emp_id = c (6:8),
  emp_name = c("Rasmi","Pranab","Tusar"),

```



```
salary = c(578.0,722.5,632.8),
start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
dept = c("IT","Operations","Fianance"),
stringsAsFactors = FALSE
)
```

# Bind the two data frames.

```
empdata <- rbind(empdata,empnewdata)
```

```
print(empdata)
```

	emp_id	emp_name	salary	start_date	dept
1	1	Rick	623.3	2012-01-01	IT
2	2	Dan	515.2	2013-09-23	operations
3	3	Michelle	611.0	2014-11-15	IT
4	4	Ryan	729.0	2014-05-11	HR
5	6	Rasmi	578.0	2013-05-21	IT
6	7	Pranab	722.5	2013-07-30	operations
7	8	Tusar	632.8	2014-06-17	Fianance

## EXPERIMENT5:

## .Add rows and columns to data frame using rbind and cbind

Dataframe with 10 observations & 3 variables:

>Add rows(rbind)

>Add cols (cbind)

```
df <-data.frame(a = c(1:10) ,b =c(10:20),x=c(21:30))
```

```
df
```

	a	b	x
1	1	11	21
2	2	12	22
3	3	13	23
4	4	14	24
5	5	15	25
6	6	16	26
7	7	17	27
8	8	18	28
9	9	19	29
10	10	20	30

```
df=rbind(df,c(11,20,30))
```

```
df
```

```
df
```

	a	b	x
1	1	11	21
2	2	12	22
3	3	13	23
4	4	14	24
5	5	15	25

```
6    6 16 26
7    7 17 27
8    8 18 28
9    9 19 29
10   10 20 30
11   11 20 30
```

**Cbind is used to add columns to dataframe.**

```
df=cbind(df,c=c(31:41))
```

```
df
```

```
      a  b  x  c
1     1  1 11 21 31
2     2  2 12 22 32
3     3  3 13 23 33
4     4  4 14 24 34
5     5  5 15 25 35
6     6  6 16 26 36
7     7  7 17 27 37
8     8  8 18 28 38
9     9  9 19 29 39
10    10 10 20 30 40
```

## EXPERIMENT6:

### Linear Regression:

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1.

Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is –

$$y = ax + b$$

Following is the description of the parameters used –

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.

### Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is –

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()** function in R.

### Linear Regression in r

We use “lm” function to carry the linear regression

```
lr=lm(formula=mpg~cyl+disp+hp,data=mtcars)
```

```
Summary(lr)
```

```
> summary(lr)

Call:
lm(formula = mpg ~ cyl + disp + hp, data = mtcars)

Residuals:
    Min       1Q   Median       3Q      Max
-4.0889 -2.0845 -0.7745  1.3972  6.9183

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  34.18492    2.59078   13.195 1.54e-13 ***
cyl          -1.22742    0.79728   -1.540  0.1349
disp         -0.01884    0.01040   -1.811  0.0809 .
hp           -0.01468    0.01465   -1.002  0.3250
---
Signif. codes:
  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

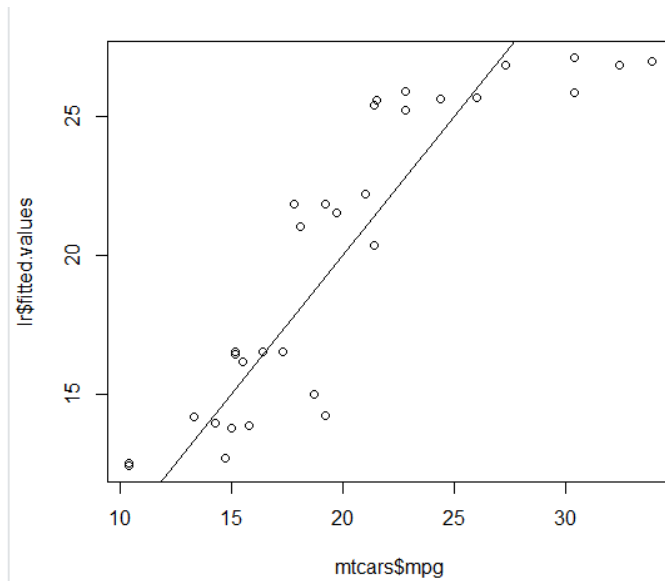
Residual standard error: 3.055 on 28 degrees of freedom
Multiple R-squared:  0.7679,    Adjusted R-squared:  0.743
F-statistic: 30.88 on 3 and 28 DF,  p-value: 5.054e-09
```

```
> names(lr)
[1] "coefficients" "residuals"    "effects"      "rank"
[5] "fitted.values" "assign"        "qr"           "df.residual"
[9] "xlevels"      "call"          "terms"        "model"
```

```
> lr$coefficients
(Intercept)      cyl      disp      hp
34.18491917 -1.22741994 -0.01883809 -0.01467933
```

```
Plot(x=mtcars$mpg,y=lr$fitted.values,xlab="true
values",ylab="predictedvalues",main="regression diagram")

abline(b=1,a=0)
```



```
# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The resposne vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)

# Find weight of a person with height 170.
a <- data.frame(x = 170)
result <- predict(relation,a)
print(result)
```

## How To Implement K-Means Clustering:

- The centroids have stabilized — there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.

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

## EXPERIMENT8:

## How To Implement K-Medoids Clustering:

1. Find a set of  $k$  Medoids ( $k$  refers to the number of clusters, and  $M$  is a collection of medoids) from the data points of size  $n$  ( $n$  being the number of records).
2. Using any distance metric (say  $d(\cdot)$ , could be euclidean, manhattan, etc.), try and locate Medoids that minimize the overall distance of data points to their closest Medoid.
3. Finally, swap Medoid and non-Medoid pairs that reduce the loss function  $L$  among all possible  $k(n-k)$  pairs. The loss function is defined as:

$$L(M) = \sum_{i=1}^n \min_{m \in M} d(m, x_i)$$

```
i1=iris[,-5]
```

**kms=pam(i1,3)**

**kms**

[illegible]



## EXPERIMENT9

### DECISION TREE :

- Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any Boolean function on discrete attributes using the decision tree.

In Decision Tree the major challenge is to identification of the attribute for the root node in each level. This process is known as attribute selection. We have two popular attribute selection measures:

#### 1. Information Gain

When we use a node in a decision tree to partition the training instances into smaller subsets the entropy changes. Information gain is a measure of this change in entropy.

#### 2. Entropy

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content.

## Procedure to implement DECISION TREE using reading skills data set

```
install.packages("party")
```

```
library(party)

# Print some records from data set readingSkills.
print(head(readingSkills))
# Load the party package. It will automatically load other
# dependent packages.
library(party)

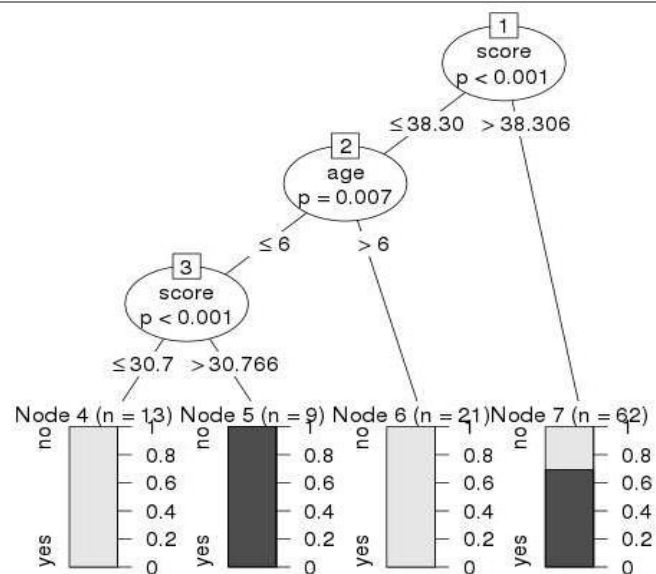
# Create the input data frame.
input.dat <- readingSkills[c(1:105),]

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

# Create the tree.
output.tree <- ctree(
  nativeSpeaker ~ age + shoeSize + score,
  data = input.dat)

# Plot the tree.
plot(output.tree)

# Save the file.
dev.off()
```



# EXPERIMENT10:

## CORPUS CREATION AND DOCUMENT TERM MATRIX

```
load(url("https://cbail.github.io/Trump_Tweets.Rdata"))
head(trumptweets$text)
install.packages("tm")
library(tm)
trump_corpus <- Corpus(VectorSource(as.vector(trumptweets$text)))
trump_corpus
```

```
<<SimpleCorpus>>
Metadata: corpus specific: 1, document level (indexed): 0
Content: documents: 3196
> |
```

### Tidy-Text

```
install.packages("tidytext")
library(tidytext)
library(dplyr)
tidy_trump_tweets<- trumptweets %>% select(created_at,text) %>%
  unnest_tokens("word", text)
head(tidy_trump_tweets)
tidy_trump_tweets %>%
  count(word) %>%
  arrange(desc(n))
```

## Text Pre-Processing

**Stopwords** (Common words such as “the”, “and”, “but”, “for”, “is”, etc. are often described as “stop words.”)

**We can remove stopwords**

```
trump_corpus <- tm_map(trump_corpus, removeWords, stopwords("english"))
```

In tidytext we can remove stopwords as follows:

```
data("stop_words")
tidy_trump_tweets <- tidy_trump_tweets %>%
  anti_join(stop_words
```

And now we can repeat the count of top words above:

```
tidy_trump_tweets %>%
  count(word) %>%
  arrange(desc(n))
```

To create a document-term matrix from a `Corpus` object, use the following code:

```
trump_DTM <- DocumentTermMatrix(trump_corpus, control = list(wordLengths =
c(2, Inf)))
```

The end of the code above specifies that we only want to include words that are at least two characters long.

We can view the first five rows of the DTM and two of its columns as follows:

```
inspect(trump_DTM[1:5, 3:8])
```