NAGARJUNA DEGREE COLLEGE FOR WOMEN
KADAPA
AFFILIATED TO

Yogi Vemana University

Kadapa, ANDHRA PRADESH - 516 003

B.SC COMPUTER SCIENCE

III YEAR VI SEMESTER

CLUSTER ELECTIVE – A1

FOUNDATIONS OF DATA SCIENCE

Prepared by D.SatyanarayanaMurthy, Dept of Computer Science/Applications
Nagarjuna Degree College for Women Kadapa

Paper-VIII–A1 : Foundations of Data Science Syllabus

Course Objectives

Modern scientific, engineering, and business applications are increasingly dependent on data, existing traditional data analysis technologies were not designed for the complexity of the modern world. Data Science has emerged as a new, exciting, and fast-paced discipline that explores novel statistical, algorithmic, and implementation challenges that emerge in processing, storing, and extracting knowledge from Big Data.

Course Outcomes

1. Able to apply fundamental algorithmic ideas to process data.
2. Learn to apply hypotheses and data into actionable predictions.
3. Document and transfer the results and effectively communicate the findings using visualization techniques.

UNIT I

INTRODUCTION TO DATA SCIENCE : Data science process – roles, stages in data science project – working with data from files – working with relational databases – exploring data – managing data – cleaning and sampling for modeling and validation – introduction to NoSQL.

UNIT II

MODELING METHODS : Choosing and evaluating models – mapping problems to machine learning, evaluating clustering models, validating models – cluster analysis – K-means algorithm.

UNIT III

INTRODUCTION TO R Language: Reading and getting data into R – ordered and unordered factors – arrays and matrices – lists and data frames.

UNIT IV

MAP REDUCE: Introduction – distributed file system – algorithms using map reduce, Matrix-Vector Multiplication by Map Reduce – Hadoop - Understanding the Map Reduce architecture.

UNIT V

DELIVERING RESULTS : Documentation and deployment – producing effective presentations– Introduction to graphical analysis – plot() function – displaying multivariate data.

Reference Books

1.Nina Zumel, John Mount, "Practical Data Science with R", Manning Publications, 2014.
2.Jure Leskovec, Anand Rajaraman, Jeffrey D.Ullman, "Mining of Massive Datasets", Cambridge University Press, 2014.
3.Mark Gardener, "Beginning R - The Statistical Programming Language", John Wiley & Sons, Inc., 2012.
4.W. N. Venables, D. M. Smith and the R Core Team, "An Introduction to R", 2013.
5.Tony Ojeda, Sean Patrick Murphy, Benjamin Bengfort, Abhijit Dasgupta, "Practical Data Science Cookbook", Packt Publishing Ltd., 2014.
6.Nathan Yau, "Visualize This: The FlowingData Guide to Design, Visualization, and Statistics", Wiley, 2011.
7.Boris lublinsky, Kevin t. Smith, Alexey Yakubovich, "Professional Hadoop Solutions", Wiley, ISBN: 9788126551071, 2015.

Student Activity:

1. **Collect data from any real time system and create clusters using any clustering algorithm**
2. **Read the student exam data in R perform statistical analysis on data and print results.**

Paper-VIII- A1 Foundations of Data Science Lab

Objectives :

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

Outcomes:

1) At end student will learn to handle the data through R.
2) Student will familiar with loading and unloading of packages.

I. Installing R and R studio

II. Basic Operations in r

1. Arithmetic Operations
2. Comments and spacing
3. Logical Operators - <, <=, >, >=, = , !=, &&, 1

III.
1. Getting data into R, Basic data manipulation
2. Vectors, Materials, operation on vectors and matrices.

IV.
1. Basic Plotting
2. Quantitative data
3. Frequency plots
4. Box plots
5. Scatter plot
6. 6.Categorial data
7. Bar charts
8. Pie charts

V. Loops and functions

1. if, if else, while, for break, next, repeat.
2. Basic functions- Print(), exp( ), Log( ), sqrt( ), abs( ), sin( ), Cos( ), tan( ), factorial( ), rand ()

UNIT I

Working with Data from Files:

- Data exists in different forms and sizes, but most of it can be presented as Structured Data and Unstructured Data.

Working with Structured Data:

- Structured data is the type of data that is well-organized and accurately formatted. This data exists in a format of relational databases (RDBMSs), which means that the information is stored in tables with rows and columns. Structured data is arranged and recorded neatly, so it can be easily found and processed.
- As data is arranged in the form of tables with rows and columns Data analysis can be easily done. In this first row indicates column names and each column represents a fact or measurement and each row represents an example for the fact

  Example:

Student Name	Course	Percentage	Result
Kishore	B.Tech	78	Pass
Anil	M.Tech	85	Pass
Ashok	BCom	77	Pass

- In R language it is very easy to load structured data from a file. We can use read. table( ) function to load any data which is in structured format. We can also use read.csv( ) to load data from a .csv file (Comma Separated Value), read.tsv( ) to load data from a .tsv file (Tab Separated Value) and read.xls( ) to load data from a excel file.

Working with Un-Structured Data:

- Data which is not available in a structured format can be called as Unstructured Data. .
- Traditional methods and tools can't be used to analyze and process Unstructured Data. One of the ways to manage unstructured data is to use non-relational databases, also known as NoSQL.

  Unstructured data examples:

  Unstructured data is available in variety of forms such as emails, text files, social media posts, videos, images, audio file, sensor data, and so on.

- Unstructured data can also be loaded in to R. Before using such data it should be reformatted and then loaded into R. To transform unstructured data into structured it should be stored in a file and then by using R we can arrange data as columns and rows along with column names.

Working with relational databases:

- In many organizations data is always stored in relational databases. Relational databases easily store millions of records and provide important features such as security, integrity etc.
- Data can be easily transported from relational databases in to R. To load data from a RDBMS we require an ODBC (Open Database Connectivity).
- SQL is the most common database querying language which allows Joins, Aggregations etc. by establishing an ODBC we can execute SQL commands directly from R.

Exploring Data:

- When we have huge volumes of data sometimes there can be missing data or incorrect data. Many problems may arise because of missing data or incorrect data. When such kinds of problems are not solved the data may give incorrect results during data exploration.
- Problems which are raised because of missing values or incorrect values should be solved before data exploration.
- To find out the problems which are raised because of missing values or incorrect values, Data exploration uses a combination of summary statistics like mean, median, count, variance etc. Exploration also uses visualization techniques in the form of graphs.

Dealing with Missing Values:

- When few values are missing in a large volume of data then it may not be a big problem. But if important values of a field or column are missing then it creates problems.
- Using R language missing values can be easily identified. All the missing values are represented as NA which means that the values are Not Available.

Dealing with Incorrect Values:

- When invalid values are more it leads to wrong results. Before data exploration the values in a field or column have to be checked for their correctness.
- Using R language incorrect values can be easily identified. The columns which have incorrect values are indicated as –Ve.

Using Summary function to identify problems:

- In R language we can use summary ( ) to identify missing values and invalid values. It is a statistical function which generates a detailed report of variety of statistical details like mean, median, minimum, maximum, 1st quarter, 3rd quarter of a given numerical set of data.

Using Graphs to identify problems:

- For some type of problems graphs are more useful when compared with summary statistics. Using graphs to examine data is called as Visualization.
- A graph can display more information with less strain. Graphs are also used to identify data characteristics and issues in it.
- When a single column is to be examined for missing values or invalid values then bar chart or histograms are more useful. When more than one column is to be examined for missing values or invalid values then line graph or scatter graph is more useful.

Managing Data:

Managing data include solving problems that are discovered during exploration. This includes cleaning data and sampling data.

Cleaning Data:

- This involves finding missing values and converting them into meaningful values. When small portion of data is missing it is better to delete the rows which have missing values as small portion of data which is missing does not affect the entire data. If missing values are more this method is not suitable.
  Ex: If a data set of 1000 customers have missing data of 56 customers then delete the data of 56 customers as it represents only 6% of entire data.
- Another method to convert missing values into meaningful values is to replace all the missing values with mean values. This method is suitable only when the values are missing randomly
  Ex: In an Employ dataset, if salary values are missing they can be replace by mean salary values.
- If the values are missing in a systematic way then the values are replaced with zero.

Sampling Data:

- Sampling is a process of selecting a subset of data which represents entire data. During analysis and modeling it is easier to test and find the errors in small samples rather than the entire data set.
- It is important to collect sample data which is meaningful and easy for modeling. The collected sample should be an accurate representation of entire data set.
- Another reason for creating sample data is to develop training data set and testing data set.
  Ex: When we are collecting customers data and preparing it as a data set it is important to collect sample customers data from all the regions of the country rather than one part of the country.

Training & Testing Data sets:

- When building a model we require data to build the model and also we require data to test the model. The data which is used to build the model is called as Training data set and the data which is used to check the performance of the model is called as Testing data set.

- Training data set is used as input in the model building and Testing data set is used as input in model verification.
- R-Language uses a function called sample( ) which is used to create a random sample from a data set.

Introduction to NoSQL

- The term NoSQL was coined by Carlo Strozzi in the year 1998. NoSQL is a non-relational database management system which did not have an SQL interface
- It is designed for distributed data stores where very large scale of data storing needs for example Google or Facebook which collects terabits of data every day for their users.

RDBMS vs NoSQL

RDBMS

- Structured and organized data
- Use specific softwares like Structured query language (SQL)
- Data and its relationships are stored in separate tables.
- Data Manipulation Language, Data Definition Language are used

NoSQL

- Stands for Not Only SQL
- Unstructured and unpredictable data
- No specific software's
- Data is stored as Key-Value pairs, Column Oriented, Document Oriented and Graphs

Advantages :

Following are some advantages of No-Sql

- High scalability
- Distributed Computing
- Lower cost
- Schema flexibility, semi-structure data
- No complicated Relationships

NoSQL Categories

There are four general types of NoSQL databases.

- Key-value stores
- Column-oriented
- Graph
- Document oriented

Key-value stores

- Key-value stores are most basic types of NoSQL databases.
- Designed to handle huge amounts of data.
- Key value stores allow developer to store table-less data.

Column-oriented databases

- Column-oriented databases primarily work on columns and every column is treated individually.
- Values of a single column are stored continuously
- Column stores data in column specific files.
- High performance on aggregation queries (e.g. COUNT, SUM, AVG, MIN, MAX).

Graph Databases

- A graph database stores data in a graph.
- It is capable of representing any kind of data in a highly accessible way.
- A graph database is a collection of nodes and edges
- Each node represents an entity (such as a student or business) and each edge represents a connection or relationship between two nodes.

Document Oriented databases

- A collection of documents
- Data is stored inside documents.
- A document is a key value collection where the key allows access to its value.
- Documents are flexible and easy to change.
- Documents are stored into collections in order to group different kinds of data.
- Documents can contain many different key-value pairs or even nested documents.

Companies using NoSQL.

- Google
- Facebook
- Mozilla
- Adobe
- LinkedIn

UNIT II
MODELING METHODS

Choosing and Evaluating Models:

- The goal of a data scientist is to solve business problems, identifying fraud transactions, estimating and managing the losses. Many different statistical modeling methods can be used to solve the above mentioned problems.
- To measure quality of the model two tasks are to be performed namely Model Evaluation and Model Validation. For model evaluation and validation data should be separated as training data and testing data.

Model Evaluation:

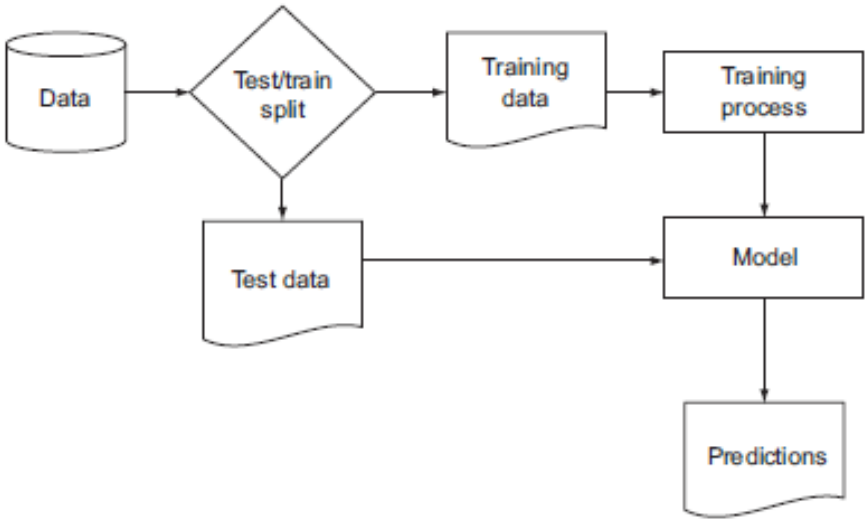
- Model evaluation is defined as checking the performance of a model.
- The model performance should be appropriate to business goals and modeling techniques.
- To evaluate the model we require training data as input.

Model Validation:

- Model validation is defined as making sure that the model will woks as estimated during evaluation. If the model works correctly during evaluation and performs poorly during validation then it is a loss to the organization.
- For validating a model testing data is given as input. Testing data checks the performance of the model which is to be as expected and then validates the model. The main cause for failure of model validation is insufficient training data.

Example:

In a banking model, training data of people who paid the loan amount regularly is not sufficient, the model also requires training data of people who did not paid the loan amount.



Mapping Problems to Machine Learning:

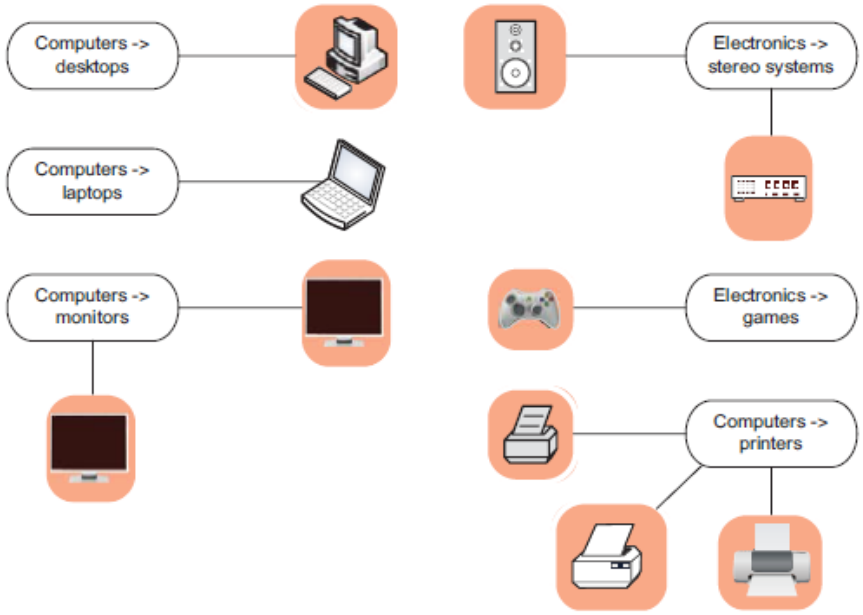
- The main task of mapping problems is to provide solutions to the issues which are raised during model evaluation and validation.
- Another main task is to identify a suitable machine learning method. This method has to take the real world situations, identify the problems and provide solutions.
- For example a data scientist at an online trading company can face problems like
  - → Identifying fraud transactions
  - → Identifying price variations of various products
  - → Providing list of items to customers who searches for items on online
  - → Grouping the customers based on similarity in purchase
  - → Organizing new products
  - → Estimating what customers might buy based on their previous transactions etc
- These problems can be classified as
  a) Classification Problems
  b) Scoring Problems
  c) Working with unknown targets

a) Solving Classification Problem:

- Classification is defined as arranging products or objects into different categories based on product properties and description. Classification is an example for "Supervised learning".
- In order to know how to classify objects we require training data which already consists of classified data sets.
- Usually classification is done manually. This can be converted into automated classification by using some of the following classification methods
  a) Naïve Bayes classification method
  b) Decision trees
  c) Logistic Regression

Example:

By using any one of the classification methods, in an electronics show room we can classify the products automatically in to computers, electronic systems. Computers can be again classified into desktops, laptops etc. Electronic systems can be classified into smart phones, video games, music systems etc.



Assigning products to product categories

b) Solving Scoring Problem:

- Scoring involves identifying different factors which are useful in increasing the sales of products, reducing fraud transactions on online.
- Following are some of the commonly used scoring methods
  a) Linear Regression
  b) Logistic Regression

Example:

By using anyone of the above methods for online shopping model we can identify different marketing methods to increase customers thereby increasing sales.

c) Working with unknown targets:

- Working with unknown targets is known as "Unsupervised Learning". In this we cannot estimate the results but we can discover similarities and relationships in the data.
- Following are some of the methods which can be used to work with unknown targets
  a) K-Means Clustering
  b) Nearest Neighbour

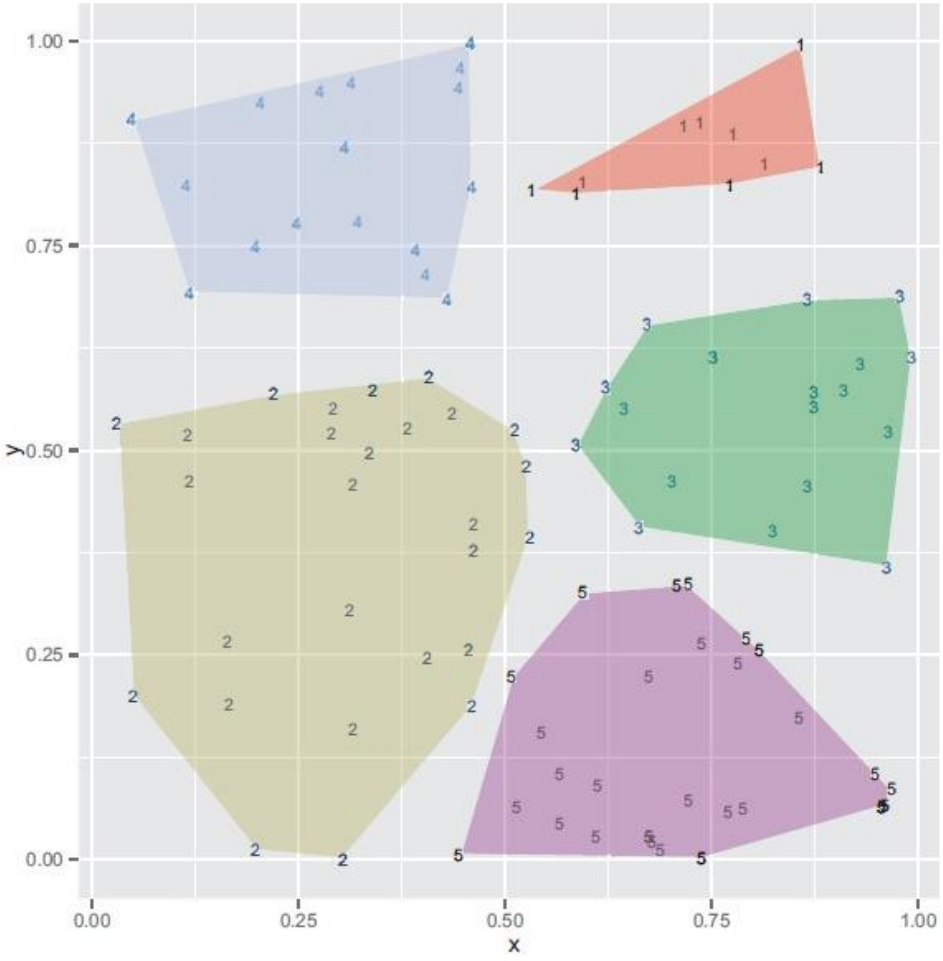
Evaluating Clustering Model:

- When building a model it is important to check whether the model works on training data or not.
- There are different models which are used to measure the model performance; Clustering model is one of them.
- Clustering models are tough to evaluate because they are unsupervised. The items present in the clusters are generated by the modeling procedures. Evaluation is checking summaries of the clusters.
- An important feature in cluster is comparing distance between two items of the same cluster to distance between two items from different clusters.
- For example to evaluate 100 random points in a plane we can form them into 5 clusters. Following R command can be used for clustering

  d <- data.frame(x=c(1:100),y=c(1:100))

  clus <- kmeans(d,centers=5)

- The above example is 2 dimensional having x and y values. It is easy to represent a two dimensional data visually



Clustering example

Validating Models:

- To check the validity of a model is a biggest challenge as the model should show similar quality on real time data also. The testing of a model on real time data is known as "Model Validation".
- Following issues should be solved during the validation of a model
  a) Identifying common model problems
  b) Checking model performance
  c) Ensuring model quality

a) Identifying common model problems: Following are some of the common model problems

1) Bias: It is a systematic error in the model such as under predicting

2) Variance: More deviation between predicted and actual values.

3) Over Fitting: An over fit model works correctly on training data but performs poorly on new data. A model error on training data is called "Training Error" and on new data is called "Generalization Error". If generalization errors are more, the model is over fit.

b) Checking model performance:

It is always important to know more about the model performance on future data and how model behavior changes when variations are there in data.

c) Ensuring Model Quality:

The quality of a model can be ensured by checking the variations in the model procedures or variations in data. Testing the model quality with the available data gives single point estimate of model performance, but the model performance should be checked on new data and deviations in data and procedures should be checked.

Cluster Analysis:

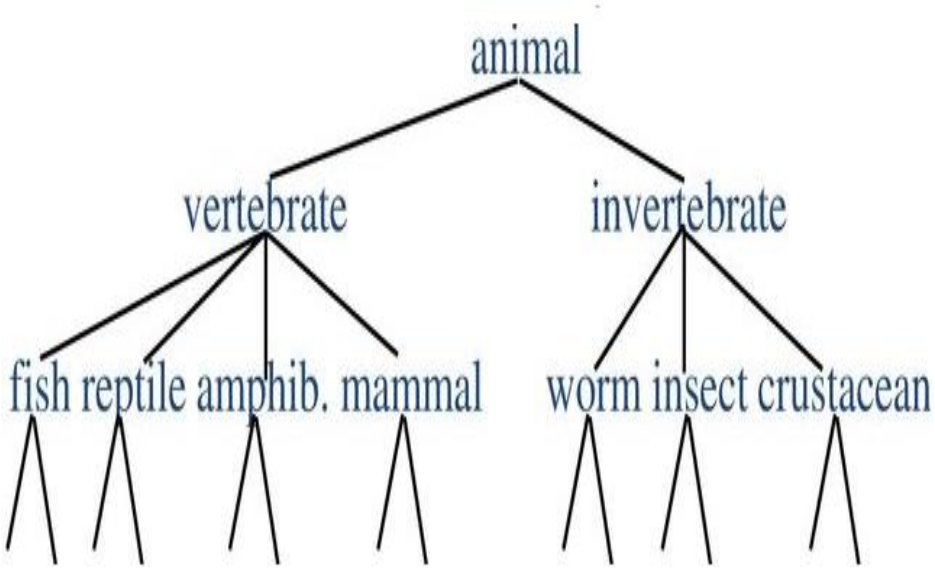
- In cluster analysis the goal is to group the observations made on data into clusters so that every data element in a cluster is more similar to other data elements in the same cluster.
- For example, a company that offers tours can cluster its clients based on behavior and tastes as follows
  Which countries they like to visit?
  Whether they prefer adventure tours, luxury tours, or educational tours;
  What kinds of activities they participate in?
  What type of sites they like to visit?
- Such information can help the company to design attractive travel packages and target the appropriate clients.
- There are two types of cluster analysis

a) Hierarchical clustering

b) K-Means clustering

a) Hierarchical Clustering:

- Hierarchical clustering is collection of group of clusters. To perform hierarchical clustering data should be prepared and loaded into R as a dataframe.
- In R for hierarchical clustering hclust( ) function can be used. This function records distance between all the points in the data.
- This function represents the cluster in a tree form. By using cutree( ) function we can extract the members of each cluster.
- For example animals can be classified based on family, genus, species



K-means clustering algorithm

- k-means is one of the simplest unsupervised learning algorithms that solve the clustering problem.
- It follows a simple and easy way to classify a given data set through a certain number of clusters.
- The main idea is to define k centers, one for each cluster.
- The next step is to take each point belonging to a given data set and associate it to the nearest center.
- After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center.

Algorithmic steps for k-means clustering

Let $X = \{x_1,x_2,x_3,\ldots\ldots..,x_n\}$ be the set of data points and $V = \{v_1,v_2,\ldots\ldots.,v_c\}$ be the set of centers.

1) Randomly select *'c'* cluster centers.

2) Calculate the distance between each data point and cluster centers.

3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.

4) Recalculate the new cluster center using:

$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i$$

where, 'c_i' represents the number of data points in i^{th} cluster.

5) Recalculate the distance between each data point and new obtained cluster centers.

6) If no data point was reassigned then stop, otherwise repeat from step 3.

UNIT III
INTRODUCTION TO R LANGUAGE

Introduction:

- R is a programming language and software environment for statistical analysis, graphical representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.
- R allows all programming language concepts such as branching, looping, modular programming using functions. It also allows to integrate functions and procedures written in other programming languages like C, C++, Java etc

Features of R:

Following are the important features of R

- R is a well-developed, simple and effective programming language which is preferred by data scientists for data analysis.
- R includes conditionals, loops, user defined functions and input and output facilities.
- R has an effective data handling and storage facility.
- R provides a set of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly on the monitor, can be stored in a file and can be printed on a paper.

R-Data types:

- In any programming language, we use various variables to store information. Variables are nothing but reserved memory locations to store values.
- We can store information of various data types like character, integer, floating point, Boolean etc.
- In R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable.
- There are many types of R-objects following are some of them:

  a) Vectors

  b) Lists

  c) Matrices

  d) Arrays

  e) Factors

  f) Data Frames

Reading and getting data into R

- For analysis we require data so getting data into R is a very important task.
- In R, we can read data from files stored outside the R environment. We can also write data into files from R. R can read and write into various file formats like csv, excel, xml etc.

Using the Combine Command c ( ) for Making Data

- The simplest way to create a sample is to use the **c ( )** command, which means combine or concatenate,

Syntax: **c (item.1, item.2, item.3, item.n)**. Everything in the parentheses is joined up to make a single item.

Entering Numerical Items as Data

- Numerical data is given as values, separated by commas, into the **c ( )** command.

Example:

>data1 = c (3, 5, 7, 5, 3, 2, 6, 8, 5, 6, 9)

Entering Text Items as Data

- If the data is in text format enter it in single or double quotes into the c ( ) command.

***Syntax:* c ("item1", "item2", 'item3')**

Example:

> day1 = c ('Mon', 'Tue', 'Wed', 'Thu')

Using read.csv ( ) function

- The csv file is a text file in which the values are separated by a comma.
- We can create this file using windows notepad. Enter the data separated by commas and save the file as .csv using the save As All files (*.*) option in notepad.
- To read data from the file use **read.csv ( )** function.

Syntax:

read.csv (Path of the File)

Example: Enter following data in a notepad file and save it as a .csv file

Id,Name,Salary,Dept
1,Rick,62000,IT
2,Dan,51500,Operations
3,Michelle,56000,IT
4,Ryan,72900,HR
5,Gary,84300,Finance
6,Nina,57800,IT
7,Simon,63200,Operations
8,Guru,72200,Finance

> employ <- read.csv ("C:/Users/Desktop/Sample.csv")

> print (employ)

Output:

id	name	salary	dept
1	Rick	62000	IT
2	Dan	51500	Operations
3	Michelle	56000	IT
4	Ryan	72900	HR
5	Gary	84300	Finance
6	Nina	57800	IT
7	Simon	63200	Operations
8	Guru	72200	Finance

Using read.xlsx ( ) function

- Microsoft Excel is the most widely used spreadsheet program which stores data in the .xls or .xlsx format.
- R can read directly from these files using some excel specific packages like - XLConnect, xlsx, gdata.

Syntax:

read.xlsx (Path of the File, sheet index)

Example: Enter following data in a excel file and save it as a .xlsx file

id	name	salary	dept
1	Rick	62000	IT
2	Dan	51500	Operations
3	Michelle	56000	IT
4	Ryan	72900	HR
5	Gary	84300	Finance
6	Nina	57800	IT
7	Simon	63200	Operations
8	Guru	72200	Finance

> employ <- read.xlsx ("C:/Users/Desktop/Sample.xlsx", sheetIndex=1)

> print (employ)

Output:

id	name	salary	dept
1	Rick	62000	IT
2	Dan	51500	Operations
3	Michelle	56000	IT
4	Ryan	72900	HR
5	Gary	84300	Finance
6	Nina	57800	IT
7	Simon	63200	Operations
8	Guru	72200	Finance

Factors in R

- Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers.
- They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.
- Factors are created using the factor () function by taking a vector as input.

Example:

```
# Create a vector as input.
>data <- c("East","West","East","North","North","East","West","West","West","East","North")
>print(data)
# Apply the factor function.
factor_data <- factor(data)
print(factor_data)
```

Output:

```
[1] "East" "West" "East" "North" "North" "East" "West" "West" "West" "East" "North"
 [1] East West East North North East West West West East North
Levels: East North West
```

- Factors in R come in two varieties: **ordered** and **unordered**
- Ordinal data is a categorical, statistical data type where the variables have natural, ordered categories
- To create an ordered factor in R, we have two options:
- We can use the factor ( ) function with the argument ordered=TRUE.

Example:

```
>status <- c ("Lo", "Hi", "Med", "Med", "Hi")
ordered.status <- factor (status, levels=c ("Lo", "Med", "Hi"), ordered=TRUE)
> ordered.status
[1] Lo Hi Med Med Hi
Levels: Lo < Med < Hi
```

To convert unordered factor into ordered we can use ordered ( ) function also

Example:

```
>credit_rating <- c ("AAA", "AA", "A", "BBB", "AA", "BBB", "A")
>credit_factor <- factor (credit_rating, levels = c ("AAA", "AA", "A", "BBB"))
>ordered (credit_factor, levels = c("AAA", "AA", "A", "BBB"))
```

Output:

```
AAA AA  A   BBB AA  BBB A
```

Levels: AAA < AA < A < BBB

Arrays in R

- Arrays are the R data objects which can store data in more than two dimensions.
  For example - If we create an array of dimension (3, 3, 2) then it creates 2 matrices each with 3 rows and 3 columns.
- An array is created using the array ( ) function. It takes vectors as input and uses the values in the dim parameter to create an array. An array can be one dimensional or two dimensional

Example: The following example creates a one dimensional array

```
> x <-  c (25, 12, 36, 7)
> a1<-  array (x)
> print (a1)
```

Example: The following example creates a two dimensional array of two 3x3 matrices each with 3 rows and 3 columns.

```
> a1 <- c (5, 9, 3)
> a2 <-c (10, 11, 12, 13, 14, 15)
> result <- array (c (a1, a2), dim = c (3, 3, 2))
> print (result)
```

Output:

```
, , 1
      [,1]  [,2]  [,3]
[1,]    5    10    13
[2,]    9    11    14
[3,]    3    12    15
, , 2
      [,1]  [,2]  [,3]
[1,]    5    10    13
[2,]    9    11    14
[3,]    3    12    15
```

Naming Columns and Rows:

- We can give names to the rows, columns and matrices in the array by using the dimnames parameter.

Example:

```
> a1 <- c (5, 9, 3)
> a2 <-c (10, 11, 12, 13, 14, 15)
> column.names <- c ("COL1","COL2","COL3")
> row.names <- c ("ROW1","ROW2","ROW3")
> array.names <- c ("Array1","Array2")
> result <- array(c(a1,a2),dim=c(3,3,2), dimnames = list(column.names, row.names, array.names))
```

> print (result)

Output:

Array1

	ROW1	ROW2	ROW3
COL1	5	10	13
COL2	9	11	14
COL3	3	12	15

Array2

	ROW1	ROW2	ROW3
COL1	5	10	13
COL2	9	11	14
COL3	3	12	15

Accessing Array Elements:

We can access array elements by giving the column index, row index

Example:

```
> a1 <- c (5, 9, 3)
> a2 <-c (10, 11, 12, 13, 14, 15)
> column.names <- c ("COL1","COL2","COL3")
> row.names <- c ("ROW1","ROW2","ROW3")
> array.names <- c ("Array1","Array2")
> result <- array(c(a1,a2),dim=c(3,3,2), dimnames = list(column.names, row.names, array.names))
#Print the third row of the second matrix of the array.
> print (result [3, , 2])
```

Output:

ROW1	ROW2	ROW3
3	12	15

```
# Print the element in the 1st row and 3rd column of the 1st Array.
> print (result [1, 3, 1])
```

Output:

[1] 13

```
# Print the 2nd Array.
> print (result [ , , 2])
```

Output:

	ROW1	ROW2	ROW3
COL1	5	10	13
COL2	9	11	14
COL3	3	12	15

Matrices in R:

- Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same type. We use matrices containing numeric elements in mathematical calculations.
- A Matrix is created using the matrix ( ) function.

Syntax: The basic syntax for creating a matrix in R is

matrix (data, nrow, ncol, byrow, dimnames) in which

data is the input vector which becomes the data elements of the matrix.

nrow is the number of rows to be created.

ncol is the number of columns to be created.

byrow is a logical value. If TRUE then the input vector elements are arranged by row.

dimname is the names assigned to the rows and columns.

Example:

```
> m1 <- matrix (c (3:11), nrow=3, ncol=3, byrow=TRUE)
> print (m1)
>m2 <- matrix (c (3:11), nrow=3, ncol=3, byrow=FALSE)
> print (m2)
```

Output:

```
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11

     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
```

Naming Columns and Rows

```
rownames = c ("row1", "row2", "row3" )
colnames = c ("col1", "col2", "col3")
>  p <- matrix(c(3:11), nrow=3, ncol=3, byrow=TRUE, dimnames=list(rownames, colnames))
> print (p)
```

Output:

```
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
```

Accessing Elements of a Matrix

- Elements of a matrix can be accessed by using the column and row index of the element.

Example:

```
rownames = c ("row1", "row2", "row3" )
colnames = c ("col1", "col2", "col3")
>  p <- matrix(c(3:11), nrow=3, ncol=3, byrow=TRUE, dimnames=list(rownames, colnames))
# Access the element at 3rd column and 1st row.
>print (p [1,3])
```

Output:

```
[1] 5
```

```
# Access the element at 2nd column and 3rd row.
>print (p [3,2])
```

Output:

```
[1] 10
```

```
# Access only the 2nd row.
>print (p [2,])
```

Output:

```
      col1  col2  col3
row2  6     7     8
```

```
# Access only the 3rd column.
>print (p [,3])
```

Output:

```
      col3
row1  5
row2  8
row3  11
```

Matrix Addition

```
# Create two 2x3 matrices.
>matrix1 <- matrix(c (3, 9, -1, 4, 2, 6), nrow=2, ncol=3)
>print (matrix1)
```

Output:

```
      [,1]  [,2]  [,3]
[1,]  3     -1    2
[2,]  9     4     6
```

```
>matrix2 <- matrix(c (5, 2, 0, 9, 3, 4), nrow=2, ncol=3)
>print (matrix2)
```

Output:

```
     [,1]  [,2]  [,3]
[1,]  5     0     3
[2,]  2     9     4
# Add the matrices.
result <-  matrix1 + matrix2
cat("Result of addition","\n")
print(result)
```

Output:

```
Result of addition
     [,1]  [,2]  [,3]
[1,]  8     -1    5
[2,]  11    13    10
```

Lists in R

- Lists are the R objects which contain elements of different types bundled together to form a single object. Items can be numbers, strings, vectors and another list also.
- List is created using list ( ) function.

Following is an example to create a list containing strings, numbers, vectors and a logical value

```
> list1 <- list ("Red", "Green", c (21, 32, 11), TRUE, 51.23)
> print (list1)
```

Output:

```
[[1]]
[1] "Red"
[[2]]
[1] "Green"
[[3]]
[1] 21 32 11
[[4]]
[1] TRUE
[[5]]
[1] 51.23
```

Naming List Elements

- The list elements can be given names and they can be accessed using these names.

```
# Create a list containing a vector, a matrix and a list.
> list_data <- list (c ("Jan","Feb","Mar"), matrix(c (3, 9, 5, 1, -2, 8), nrow=2, ncol=3), list ("green", 12.3))
# Give names to the elements in the list.
>  names (list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
# Show the list.
> print(list_data)
```

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

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"
$A_Matrix
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
$A_Inner_list
$A_Inner_list[[1]]
[1] "green"
$A_Inner_list[[2]]
[1] 12.3
```

Merging Lists

- You can merge many lists into one list by placing all the lists inside one list() function.

```
# Create two lists.
>list1 <- list (1, 2, 3)
>list2 <- list ("Sun", "Mon", "Tue")
# Merge the two lists.
merged.list <- c (list1, list2)
# Print the merged list.
print (merged.list)
```

Output:

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

[1] 2

[[3]]

[1] 3

[[4]]

[1] "Sun"

[[5]]

[1] "Mon"

[[6]]

[1] "Tue"

Data Frames in R

- A data frame is used for storing data. It is a two-dimensional object, having rows and columns.
- R treats the columns as separate variables, and the rows represent the values.
- data.frame ( ) function is used to create a data frame in r

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.

Example:

```
>emp.data <- data.frame (emp_id = c (1:5), emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),
salary = c(15000, 22200,18600,17500,11200),
start_date = as.Date(c ("2012-01-01","2013-09-23","2014-11-15","2014-05-11","2015-03-27")),)
>print (emp.data)
```

Output:

emp_id	emp_name	salary	start_date
1	Rick	15000	2012-01-01
2	Dan	22200	2013-09-23
3	Michelle	18600	2014-11-15
4	Ryan	17500	2014-05-11
5	Gary	11200	2015-03-27

Displaying Structure of the Data Frame

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

```
>emp.data <- data.frame (emp_id = c (1:5), emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),
```

```
salary = c(15000, 22200,18600,17500,11200),
start_date = as.Date(c ("2012-01-01","2013-09-23","2014-11-15","2014-05-11","2015-03-27")),)
# Get the structure of the data frame.
str(emp.data)
```

Output:

```
'data.frame': 5 obs. of 4 variables:
$ emp_id : int 1 2 3 4 5
$ emp_name : chr "Rick" "Dan" "Michelle" "Ryan" ...
$ salary : num 15000  22200 18600 17500 11200
$ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11"
```

Expanding Data Frame

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

Adding Column

```
# Add the "dept" coulmn.
```

Example1:

```
>emp.data <- data.frame (emp_id = c (1:5), emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),
salary = c(15000, 22200,18600,17500,11200),
start_date = as.Date(c ("2012-01-01","2013-09-23","2014-11-15","2014-05-11","2015-03-27")),)
emp.data$dept <- c("IT","Operations","IT","HR","Finance")
>v <- emp.data
>print (v)
```

Output:

emp_id	emp_name	salary	start_date	Dept
1	Rick	15000	2012-01-01	IT
2	Dan	22200	2013-09-23	Operations
3	Michelle	18600	2014-11-15	IT
4	Ryan	17500	2014-05-11	HR
5	Gary	11200	2015-03-27	Finance

Adding Row

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

```
>emp.data <- data.frame (emp_id = c (1:5), emp_name = c("Rick", "Dan", "Michelle", "Ryan", "Gary"),
```

salary = c(15000, 22200,18600,17500,11200), start_date = as.Date(c ("2012-01-01","2013-09-23","2014-11-15","2014-05-11","2015-03-27")),dept=c("IT","Operations","IT","HR","Finance"))

Create the second data frame

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

salary = c(11578,17220,16320), start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),

dept = c("IT","Operations","Fianance"))

Bind the two data frames.

>emp.finaldata <- rbind(emp.data,emp.newdata)

>print(emp.finaldata)

Output:

emp_id	emp_name	salary	start_date	Dept
1	Rick	15000	2012-01-01	IT
2	Dan	22200	2013-09-23	Operations
3	Michelle	18600	2014-11-15	IT
4	Ryan	17500	2014-05-11	HR
5	Gary	11200	2015-03-27	Finance
6	Rasmi	11578	2013-05-21	IT
7	Pranab	17220	2013-07-30	Operations
8	Tusar	16320	2014-06-17	Finance

UNIT IV
MAP REDUCE

Distributed File System (DFS)

- A distributed file system (DFS) is a file system with data stored on a server. The DFS makes it convenient to share information and files among users on a network in a controlled and authorized way.
- The server allows the client users to share files and store data just like they are storing the information locally. However, the servers have full control over the data and give access control to the clients.
- Sharing storage resources and information on the network is one of the key elements in both local area networks (LANs) and wide area networks (WANs). Generally, a DFS is used in a LAN, but it can be used in a WAN or over the Internet
- Different technologies have been developed for sharing resources and files on a network; a distributed file system is one of the processes used regularly.
- One process involved in implementing the DFS is giving access control and storage management controls to the client system in a centralized way, managed by the servers.
- A DFS allows efficient and well-managed data and storage sharing options on a network compared to other options.
- Another option for users in network-based computing is a shared disk file system. A shared disk file system puts the access control on the client's systems so the data is inaccessible when the client system goes offline.

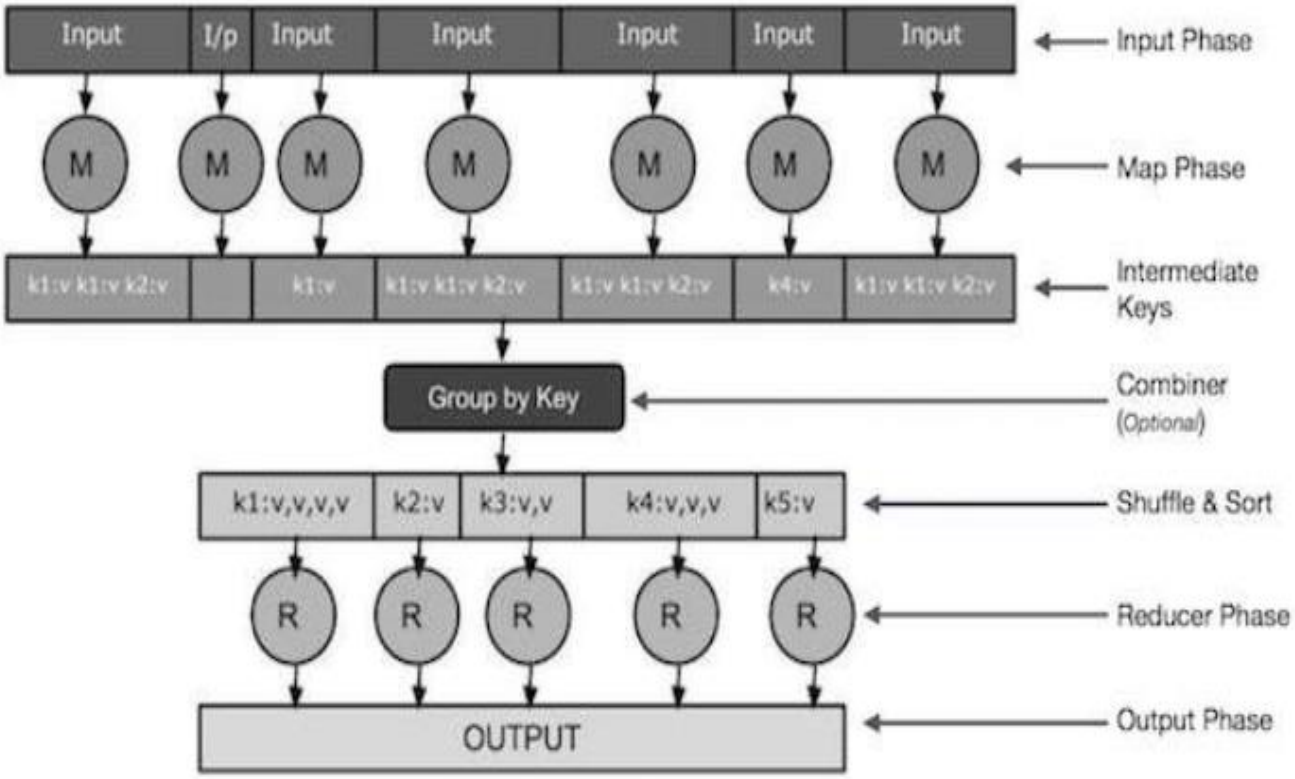
Map Reduce Introduction:

- MapReduce is a programming model for writing applications that can process Big Data.
- MapReduce provides analytical capabilities for analyzing huge volumes of complex data.
- Traditional database systems normally have a centralized server to store and process data which are not suitable to process huge volumes of data.
- The centralized system creates problems while processing multiple files simultaneously.
- Google solved these issues using an algorithm called MapReduce.
- MapReduce divides a task into small parts and assigns them to many computers. Later, the results are collected at one place and integrated to form the result dataset.

How MapReduce Works?

- The MapReduce algorithm contains two important tasks, namely Map and Reduce.
- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into key-value pairs.

- The Reduce task takes the output from the Map as an input and combines those key-value pairs into a smaller set of records. The reduce task is always performed after the map job.



Phases in Mapreduce: Following are different phases involved in MapReduce algorithm

Input Phase: Here we have a Record Reader that translates each record in an input file and sends the data to the mapper in the form of keyvalue pairs.

Map: Map is a user-defined function, which takes a series of key-value pairs and processes each one of them.

Intermediate Keys: The key-value pairs generated by the mapper are known as intermediate keys.

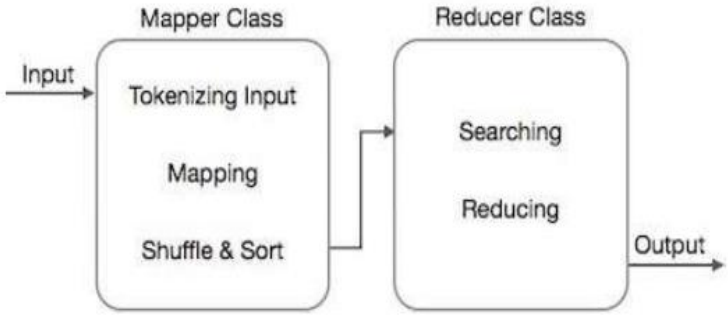
Shuffle and Sort: The Reducer task starts with the Shuffle and Sort step. The individual key-value pairs are sorted by key into a larger data list

Reducer: The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires more processing.

Output Phase: In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them to a file using a record writer.

Algorithms Using Map Reduce

- The MapReduce algorithm contains two important tasks, namely Map and Reduce.
- The map task is done by means of Mapper Class and reduce task is done by means of Reducer Class.
- Mapper class takes the input, tokenizes it, maps, and sorts it.
- The output of Mapper class is used as input by Reducer class, which in turn searches matching pairs and reduces them.



MapReduce implements various mathematical algorithms to divide a task into small parts and assign them to multiple systems. Following are different mathematical algorithms used in MapReduce

- Sorting
- Searching
- Indexing
- TF
- IDF

Sorting

- Sorting is one of the basic MapReduce algorithms to process and analyze data.
- MapReduce implements sorting algorithm to automatically sort the keyvalue pairs from the mapper.

Searching

Searching plays an important role in MapReduce algorithm. It helps in the combiner phase (optional) and in the Reducer phase.

Example: The following example shows how MapReduce use Searching algorithm to find out the details of the employee who draws the highest salary in a given employee dataset.

name, salary
satish, 26000
Krishna, 25000
Satishk, 15000
Raju, 10000

name, salary
gopal, 50000
Krishna, 25000
Satishk, 15000
Raju, 10000

name, salary
satish, 26000
kiran, 45000
Satishk, 15000
Raju, 10000

name, salary
satish, 26000
Krishna, 25000
manisha, 45000
Raju, 10000

Map phase processes each input file and provides the employee data in key-value pairs (<k, v>) namely (<emp name, salary>).

Combiner phase (searching technique) will accept the input from the Map phase as a key-value pair with employee name and salary. Using searching technique, the combiner will check all the employee salary to find the highest salaried employee in each file. Final result after searching is as follows

Reducer phase − From each file, you will find the highest salaried employee. To avoid redundancy, eliminate duplicate values if any. Final output is as follows `<gopal, 50000>`

Indexing

Indexing is used to find a particular data and its address. The indexing technique that is normally used in MapReduce is known as inverted index. Search engines like Google and Bing use inverted indexing technique.

Example: Here T[0], T[1], and t[2] are the file names with following content .

T[0] = "it is what it is" T[1] = "what is it" T[2] = "it is a banana"

After applying the Indexing algorithm, we get the following output −

"a": {2} "banana": {2} "is": {0, 1, 2} "it": {0, 1, 2} "what": {0, 1}

"a": {2} implies the term "a" appears in the T[2] file. Similarly, "is": {0, 1,2} implies the term "is" appears in the files T[0], T[1], and T[2].

Term Frequency (TF)

It measures the frequency of a particular term in a document. It is calculated by the number of times a word appears in a document divided by the total number of words in that document.

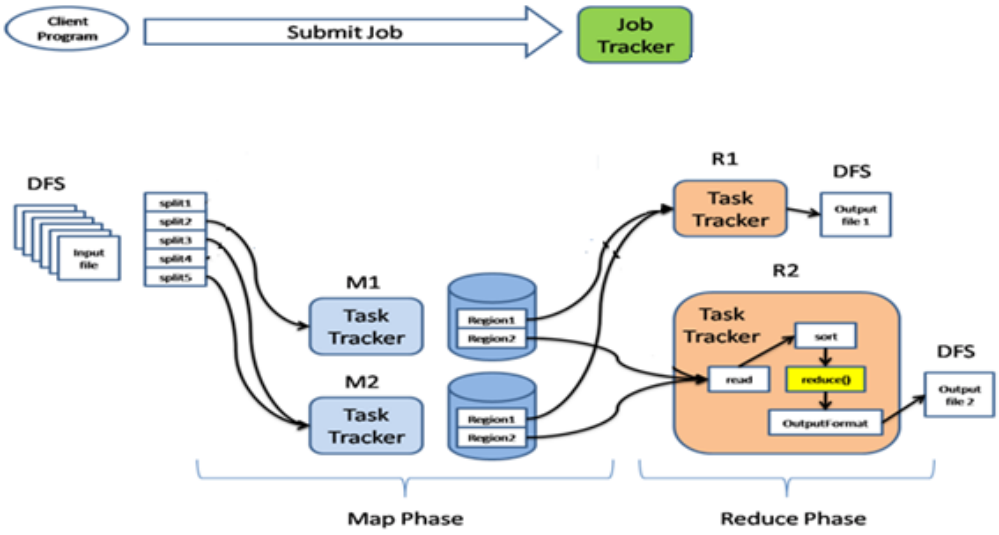
Inverse Document Frequency (IDF)

It measures the importance of a term. It is calculated by the number of documents in the database divided by the number of documents where a specific term appears.

Hadoop - Understanding the Map Reduce architecture:

- MapReduce is mainly used for parallel processing of large sets of data stored in Hadoop cluster.
- Initially, it is a hypothesis specially designed by Google to provide data distribution.
- MapReduce processes data in the form of key-value pairs.
- A key-value (KV) pair is a mapping element between two linked data items - key and its value.
- The key (K) acts as an identifier to the value.
- The MapReduce work flow undergoes different phases. Job tracker is going to take care of all MapReduce jobs that are running on various nodes present in the Hadoop cluster.
- Job tracker plays vital role in scheduling jobs and it will keep track of the entire map and reduce jobs. Actual map and reduce tasks are performed by Task tracker.

Hadoop Map Reduce architecture



- Map reduce architecture consists of mainly two processing stages.
- First one is the map stage and the second one is reduce stage.

Mapper Phase

- In Mapper Phase the input data is going to split into 2 components, Key and Value.
- Suppose, client submits input data to Hadoop system, the Job tracker assigns tasks to task tracker. The input data will be divided into several input splits.
- Record reader converts these input splits in to Key-Value (KV) pair which is the actual input data format for the further processing of data inside Task tracker.
- Combiner is also called as mini reducer. When mapper output is a huge amount of data, it is placed in combiner for better performance.

- A partition module in Hadoop plays a very important role to partition the data received from different mappers or combiners.
- Partitioner reduces the pressure that builds on reducer and gives more performance.

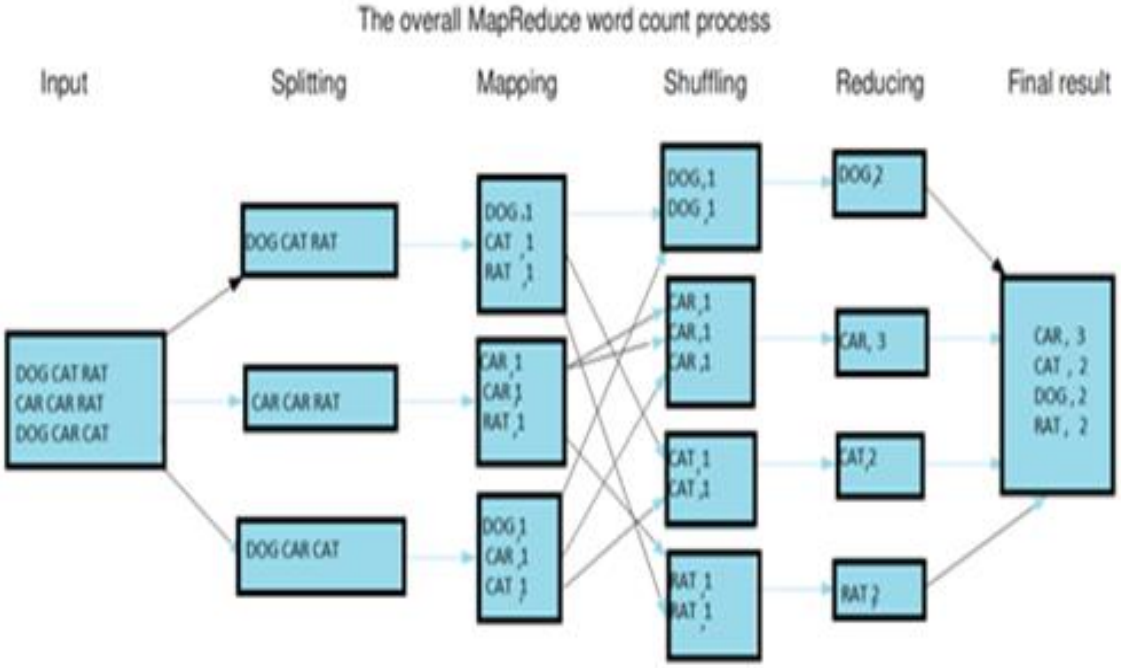
Intermediate Process

- The mapper output data undergoes shuffle and sorting in intermediate process. The intermediate data is going to get stored in local file system.

Reducer Phase

- Shuffled and sorted data is going to pass as input to the reducer.
- In this phase, all incoming data is going to combine and write into Hadoop distributed file system.
- Record writer writes data from reducer to Hadoop distributed file system.

MapReduce word count Example



UNIT V
DELIVERING RESULTS

Documentation and Deployment

Documentation:

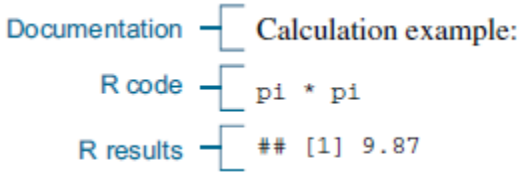
- Documentation is an important work to be completed for a successful completion of a project.
- A document represents a written work of the entire processes which were used in the project.
- The main goals of documentation are
  a) To produce effective milestone documentation
  b) Manage a complete project history

a) To Produce effective milestone documentation:

- The first documentation should be for the team members for removing errors in programme, improving features of project and the techniques used.
- This involves summary of project goals, steps taken and technical results. Milestone document is usually for the reference to project members hence it can be brief and can contain actual code.
- Milestone documentation can be easily done in R by using "KnitR" package. KnitR package produces reliable documents which show the state of a project.

  Using KnitR to produce milestone documentation:
- KnitR allows the inclusion of R-Code and results inside the document.
- KnitR supports different document types like html for online documentation.
- KnitR's main operation is called "Knit". It extracts and executes all the R-Code and builds a new result document with original document, printed code and results as contents.
- Because of KnitR we can get nicely formatted documentation which can be shared. Version control in KnitR helps to maintain details of earlier work like what was done in the project, when it was done and by whom it was done.

  Example:



b) Manage a complete project history:

- This produces a check point documentation which gives details of previous work.
- For example, check point documentation can be used to give details about how your project worked in the last year February month by giving a copy of details.

Deployment:

- Deployment includes testing the project and demonstrating to the users. A successful datascience project should include demonstration of techniques and models developed.
- Following are the different deploying plans
  a) *Deploying model as R http*:
     An easy way to demonstrate an R model which is in operation is by exposing it as an http service.
  b) *Deploying model by export:*
     Export is another method of deploying where a finished model is exported to another system.

Producing Effective Presentations

Once the building of a model is completed the work which was done should be explained to project sponsor, data scientists and end users.

Presenting results to Project Sponsor:

Project sponsor is the person who wants the project results. Hence the results of the project should be first presented to project sponsor, with following details

a) *Summarizing the Project goals:* This section of the presentation provide the goals of the project to the people who are not involved in the project. It explains the need of the project to address the business requirements.
b) *Giving the Project Results:* This section of the presentation describes the work did in the project and the results of the project.
c) *Giving results with details:* This section of presentation describes what was done in the project and how the success came. It projects all the details to understand more about the project results.
d) *Giving Recommendations and Future Work:* The presentation will be closed by suggesting some improvements to increase the model performance.

Presenting results to Data Scientists:

Data Scientists are interested in knowing the modeling process and the techniques which are used in project. Presenting results to data scientists give a chance to examine the issues which were missed in the project and also to get good suggestions or alternative methods to solve the issues. Presentation to data scientists generally has the following structure

- Introduction to the problem
- Work related discussions
- Approach related discussions
- Presenting results and findings

- Discussion about future work

Presenting results to End-Users:

It is important to note that the people who are going to use the model should be confident about the project output and willing to use the model. Hence it is better to present the results of the model to the end users. We can also provide user manuals or documents to the end-users. Presentation should make the end-users work easier than complicated. Presentation to end-users generally has the following structure

- Summarizing the idea of the project and its goals
- Showing how the model fits into users work flow and how it improves the work flow.
- Showing how to use the model.

Introduction to graphical analysis

- Graphs are a powerful way to present data and results in a simple manner.
- A graph is more understandable than words and numbers, and producing good graphs is an important skill.
- Some graphs are also useful in examining data so that we can gain some idea of patterns which can be used for correct statistical analysis.
- R has powerful and flexible graphical capabilities.
- In general R has two kinds of graphical commands: some commands generate a basic plot, and other commands are used to produce a more customized finish.
- Following are some of the basic graph types which are used in R to present data and results
  a) Scatter plots
  b) Line graphs
  c) Pie charts
  d) Bar charts

a) Scatter Plots:

- The scatter plot is used especially to show the relationship between two variables.
- One variable is chosen in the horizontal axis and another in the vertical axis.
- The simple scatter plot is created using the **plot( )** function.

Syntax: The basic syntax for creating scatter plot is

plot (x, y, main, xlab, ylab, xlim, ylim, pch=value, cex=value, col="value") in which

x is the data set whose values are the horizontal axis.

y is the data set whose values are the vertical axis.

main is the tile of the graph.

xlab is the label in the horizontal axis.

ylab is the label in the vertical axis.

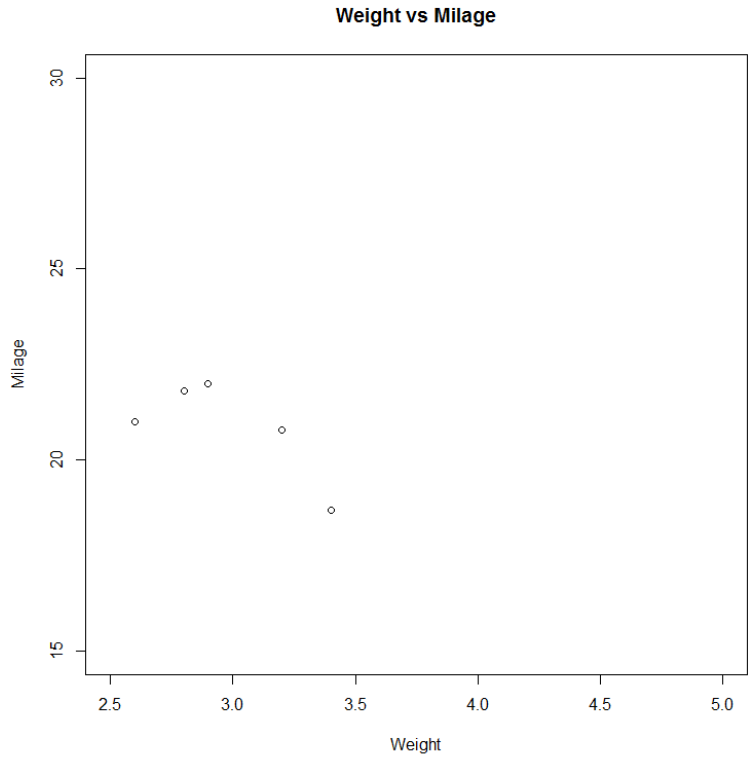
xlim is the limits of the values of x used for plotting.

ylim is the limits of the values of y used for plotting.

Example:

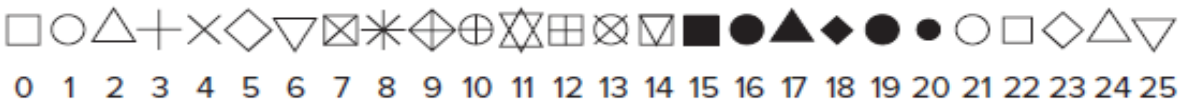
```
> x <- c (2.6, 2.8, 2.9, 3.2, 3.4)
> y <- c (21, 21.8, 22, 20.8, 18.7)
> plot (x, y, xlab = "Weight", ylab = "Mileage", xlim = c (2.5,5), ylim = c(15,30), main = "Weight vs. Mileage")
```

Output:



Plotting Symbols

- We can alter the plotting symbol by using pch (plotting character). pch values range from 0 to 25



- We can increase symbol size by using cex. By default cex=1
- We can give colour to symbol border by using col. By default col=”black”

b) Line graphs:

- A line chart is a graph that connects a series of points by drawing line between them. Line charts are usually used in identifying the trends in data.
- The plot( ) function is used to create the line graph.

Syntax: The basic syntax to create a line chart is:

plot (v, type, col, xlab, ylab, main) in which

v is a vector containing the numeric values.

xlab is the label for x axis.

ylab is the label for y axis.

main is the Title of the chart.

col is used to give colors to both the points and lines.

type is used to specify different ways to present the data on the plot area. By default type=”p”

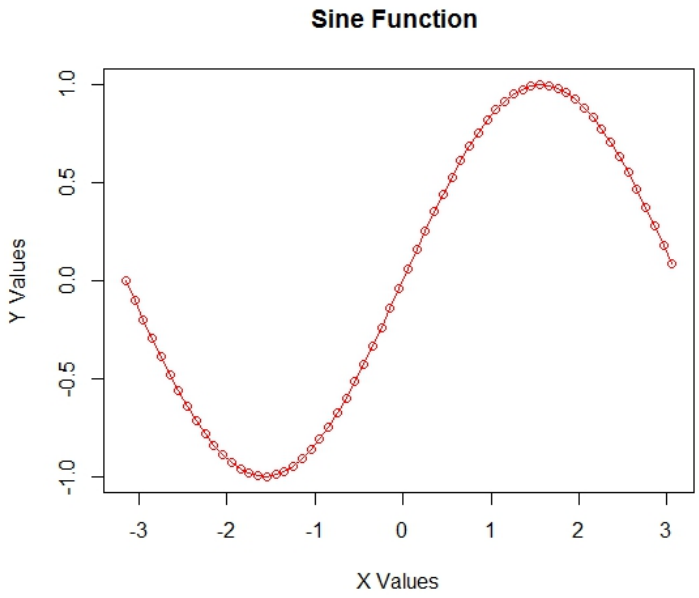
Following are different type options

type = ”p”	Points only
type = “l”	Lines
type = “b”	Points with line connection
type = “c”	Line connections without points
type = “o”	Both over plotted
type = “h”	Histogram like vertical lines
type = “s”	Stair steps
type = “S”	Stair steps, another style
type = “n”	No plotting

Example:

```
> x <- seq (-pi, pi, 0.1)
> plot (x, sin(x), type="o", col="red", xlab="X Values", ylab="Y Values", main="Sine Function ")
```

Output:



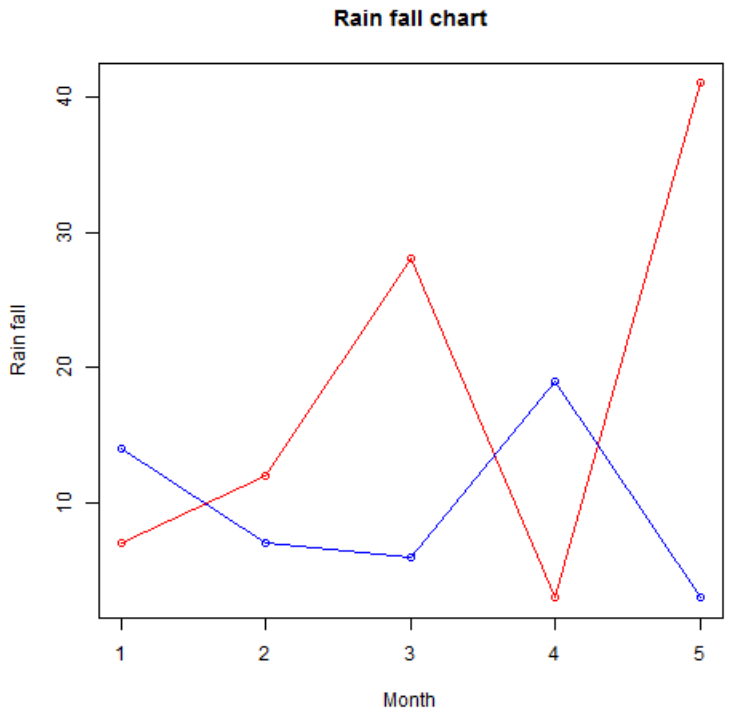
Multiple Lines in a Line Chart:

- More than one line can be drawn on the same chart by using the lines ( ) function.
- After the first line is plotted, the lines( ) function use an additional vector as input to draw the second line in the chart

Example:

```
> v <- c (7, 12, 28, 3, 41)
> t <-  c (14, 7, 6, 19, 3)
> plot (v, type="o", col="red", xlab="Month", ylab="Rain fall", main="Rain fall chart")
> lines (t, type="o", col="blue")
```

Output:



c) Pie charts

- Pie chart is a common graphic choice to illustrate data that represents division between various categories.
- You can create pie charts using the **pie ( )** command. We can use a vector of numeric values to create a pie chart

Syntax: The basic syntax for creating a pie-chart is

pie (x, labels, radius, main, col, clockwise) in which

x is a vector containing the numeric values used in the pie chart.

labels is used to give description to the slices.

radius indicates the radius of the circle of the pie chart.(value between -1 and +1).

main indicates the title of the chart.

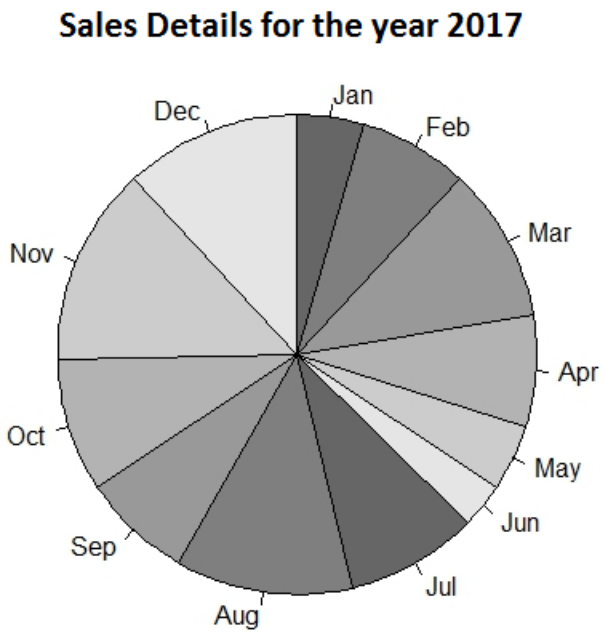
col indicates the color palette.

clockwise is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Example: Pie chart for the data that shows sales details of various items for a whole year.

```
> sales <- c (3, 5, 7, 5, 3, 2, 6, 8, 5, 6, 9, 8)
> months <-  c ("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
> pc <- c ('gray40', 'gray50', 'gray60', 'gray70', 'gray80', 'gray90')
> pie (sales, labels = months, main="Sales Details for the year 2017", col = pc, clockwise = TRUE, radius=1)
```

Output:



d) Bar charts

- The bar chart is suitable for showing data that fall into different categories. Each bar of the graph shows the number of items in a certain range of data values.
- Bar charts are widely used because they convey information in an easy way.
- We can use the **barplot ( )** command to produce bar charts.
- R can draw both vertical and horizontal bars in the bar chart.
- In bar chart each of the bars can be given different colors.

Syntax: The basic syntax to create a bar-chart in R is

barplot (v, xlab, ylab, main, names.arg, col, border) in which

v is a vector or matrix containing numeric values used in bar chart.

xlab is the label for x axis.

ylab is the label for y axis.

main is the title of the bar chart.

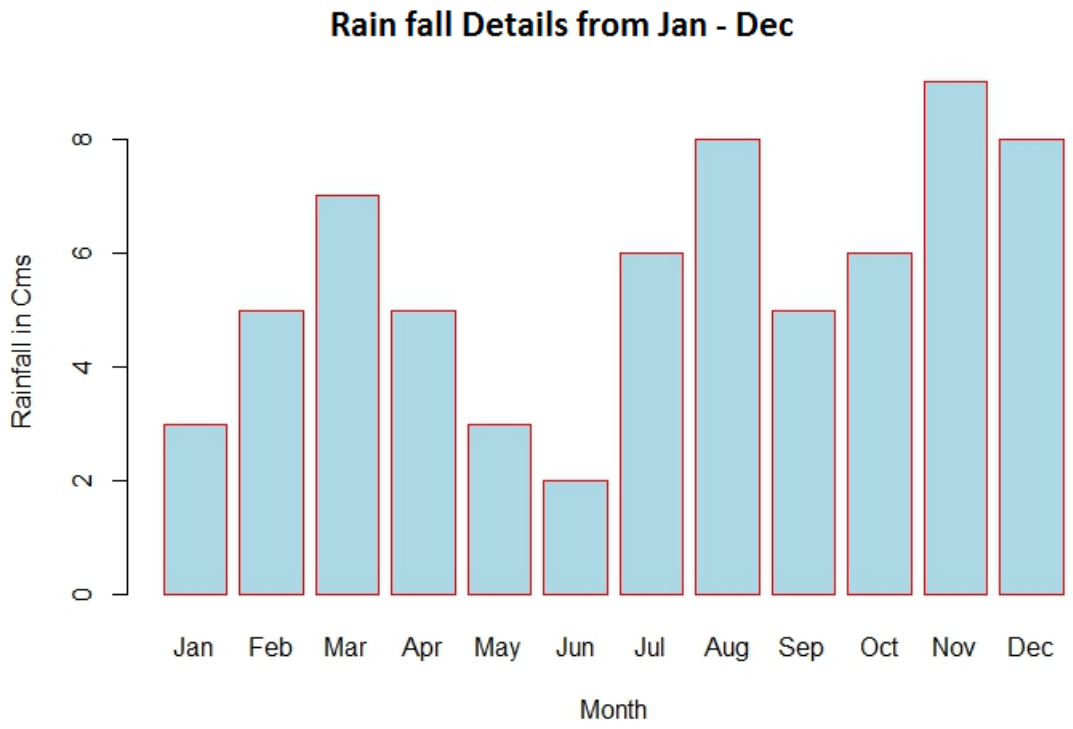
names.arg is a vector of names appearing under each bar.

col is used to give colors to the bars in the graph.

border is used to give color to the border of bar in the graph

Example:

```
> rain <- c (3, 5, 7, 5, 3, 2, 6, 8, 5, 6, 9, 8)
> month <- c ("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
> barplot (rain, xlab = "Month", ylab = "Rainfall in cms", main= " Rain fall Details from Jan-Dec",
  names.arg= month, col = "lightblue", border="red")
```



Histograms:

- A histogram represents the frequencies of values of a variable in the form of ranges.
- Histogram is similar to bar chat but the difference is it groups the values into continuous ranges.
- Each bar in histogram represents the height of the number of values present in that range.
- R creates histogram using **hist ( )** function. This function takes a vector as an input to plot histograms.

Syntax: The basic syntax for creating a histogram using R is

hist (v, main, xlab, xlim, ylim, breaks, col, border) in which

v is a vector containing numeric values used in histogram.

main indicates title of the chart.

col is used to set color of the bars.

border is used to set border color of each bar.

xlab is used to give description of x-axis.

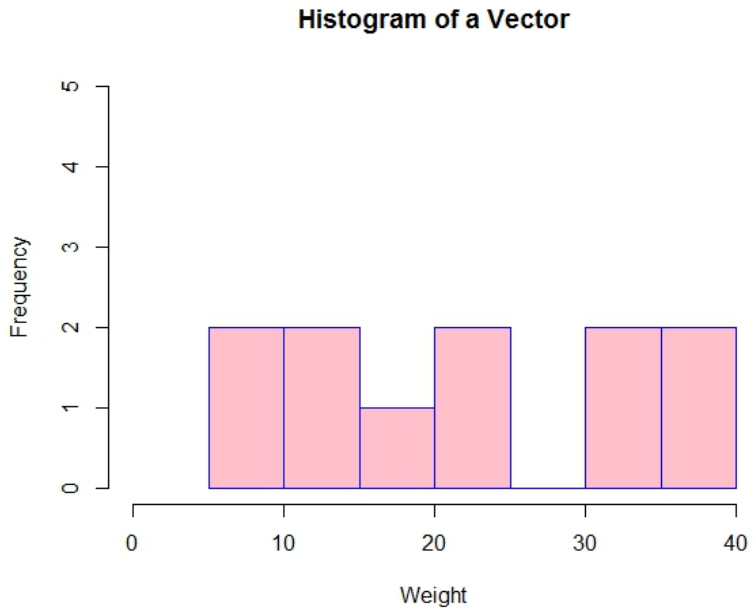
xlim is used to specify the range of values on the x-axis.

ylim is used to specify the range of values on the y-axis.

breaks is used to mention the width of each bar.

Example:

```
> v <- c(9,13,21,8,36,22,12,41,31,33,19)
> hist(v, xlab="Weight", col="green", border="red", xlim = c(0,40), ylim = c(0,5), breaks = 5,
      main= “Histogram of a Vector” )
```



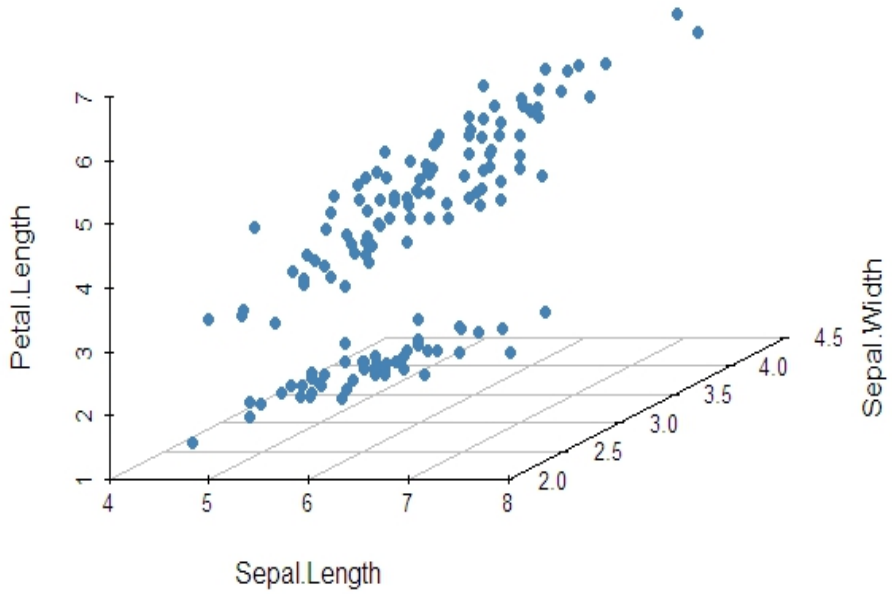
Displaying multivariate data

- Data depending on 2 variables can be easily visualized and the relationship between the two variables can be displayed easily by plotting a simple scatter plot.
- For a data set containing three continuous variables, you can create a **3d scatter plot**.
- For a small data set with more than three variables, it’s possible to visualize the relationship between each pairs of variables by creating a **scatter plot matrix**
- For a large multivariate data set, it is more difficult to visualize their relationships. These data requires specific statistical techniques. **Multivariate analysis** (MVA) refers to a set of approaches used for analysing a data set containing multiple variables. Among these techniques, there are:
- Cluster analysis for identifying groups of observations with similar profile according to specific criteria.
- Principal component methods, which consist of summarizing and visualizing the most important information contained in a multivariate data set.

3d scatter plot for data set containing 3 variables

- We can create a 3d scatter plot using the R package scatterplot3d.
- Install: install.packages("scatterplot3d")
- Create a basic 3d scatter plot:
- library(scatterplot3d)
- scatterplot3d(iris[,1:3], pch = 19, color = "steelblue", grid = TRUE, box = FALSE)

Output:



Cluster Analysis:

- Cluster analysis is one of the important methods for discovering knowledge in multidimensional data.
- The goal of clustering is to identify pattern or groups of similar objects within a data set.
- In R hclust ( ) function is used for multivariate analysis. It takes a matrix as an input.

Principal component analysis

- Principal component analysis (PCA) is a multivariate data analysis approach that allows us to summarize and visualize the most important information contained in a multivariate data set.
- PCA reduces the data into few new dimensions (or axes), which are a linear combination of the original variables.
- You can visualize a multivariate data by drawing a scatter plot of the first two dimensions, which contain the most important information in the data.
- In R PCA is done by using prcomp ( ) function

*****THE END*****