Vidya Vikas Education Trust's
Universal College of Engineering
(Permanently unaided | Approved by AICTE, DTE & Affiliated to University of Mumbai)

Department of Computer Science & Engineering

DATA WAREHOUSE MANAGEMENT LAB

(CSL 503)

Lab Manual

Name: **VIVEK.SHIVAKUMAR. HOTTI**

Roll No.: **31**

Class: **T. E-Computer Engineering**

Division: **A**

Semester: **5**

Professor: **Professor Ancy Gonsalves**

2020 – 2021

Sem 5

Vision:
To be recognized globally as a department provides quality technical education that eventually caters to helping and serving the community.

Mission:
To develop human resources with sound knowledge in theory and practice of computer science and engineering. To motivate the students to solve real-world problems to help the society grow. To provide a learning ambience to enhance innovations, team spirit and leadership qualities for students.

Lab Code	Lab Name	Credits
CSL503	**Data Warehouse Management Lab**	**1**

Practical & Oral:
Examination is to be conducted based on respective departmental level courses by pair of internal and external examiners appointed by the University of Mumbai.

Vidya Vikas Education Trust's
Universal College of Engineering
(Permanently unaided | Approved by AICTE, DTE & Affiliated to University of Mumbai)

INDEX

Name: Vivek. Hotti **Roll:** 31
Class: TE- Computers **Division:** A
Semester: 5

DWM Lab Manual

EXPERIMENT – 01

AIM:

Build Data Warehouse / Data Mart for a given problem statement:

i) Identifying the source tables and populating sample data.
ii) Design dimensional data model i.e., Star schema, Snowflake schema and Fact Constellation schema (if applicable)

THEORY:

Schema:

- Schema is such that describes the structural read of an information that confirms the tables that will be concerned in making an information, the table's attributes, and their association.
- Schema may be a structural read of an info or database.
- Schema once declared mustn't be changed often.
- In schema, Tables name, fields name, its sorts as well as constraints are included.
- For info, Schema is specified by DDL.

Dimension Table:

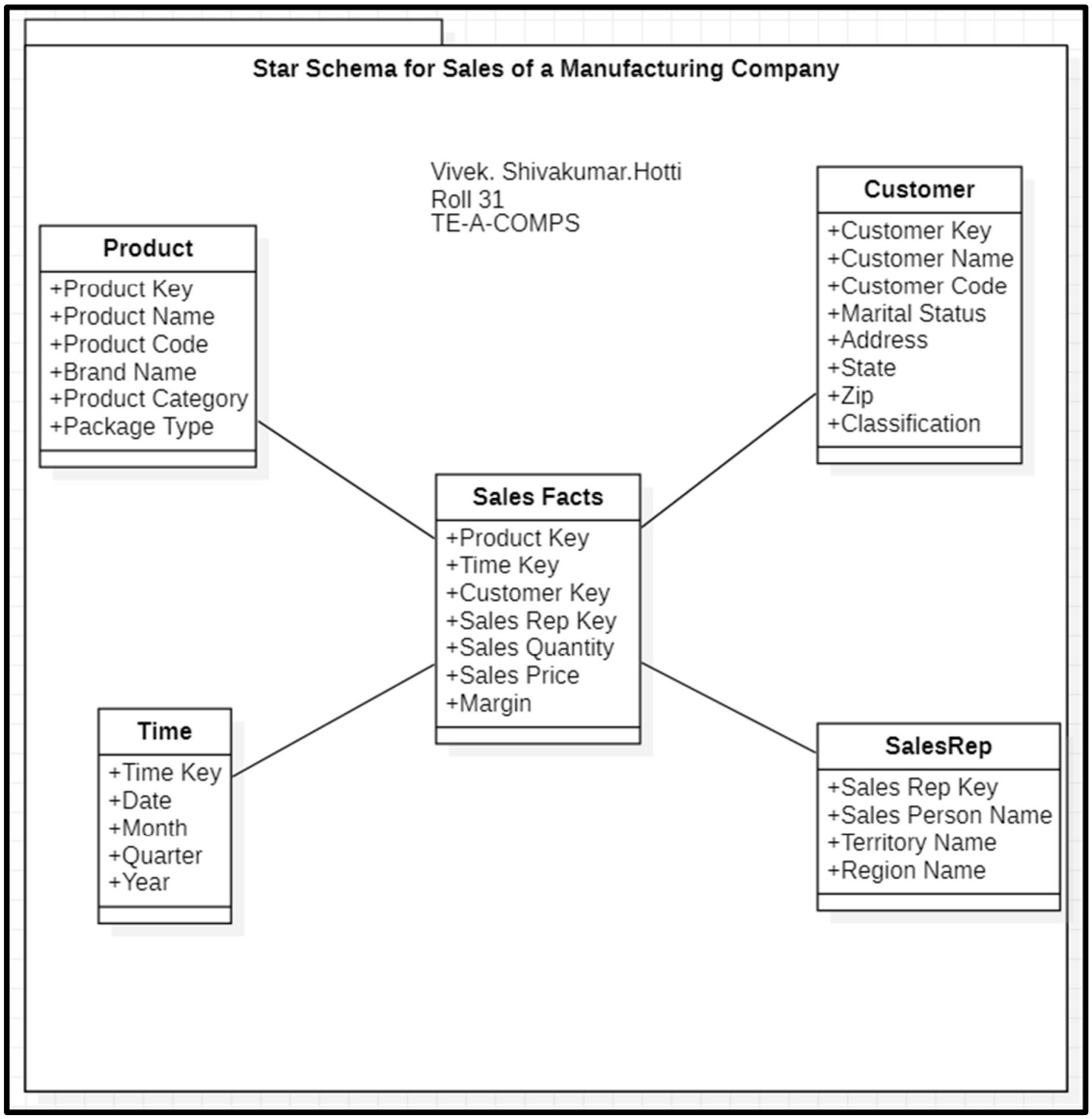
- Dimension Table is a table in a star schema of a data warehouse. Data warehouses are built using dimensional data models which consist of fact and dimension tables. Dimension tables are used to describe dimensions; they contain dimension keys, values, and attributes.
- For example, the time dimension would contain every hour, day, week, month, quarter, and year that has occurred since you started your business operations. Product dimension could contain a name and description of products you sell, their unit price, colour, weight and other attributes as applicable. Attributes would be a customer's first and last name, age, gender etc.
- Dimension tables are typically small, ranging from a few to several thousand rows. Occasionally dimensions can grow large, however. For example, a large credit card company could have a customer dimension with millions of rows. Dividing a data warehouse project into dimensions, provides structured information for reporting purpose.

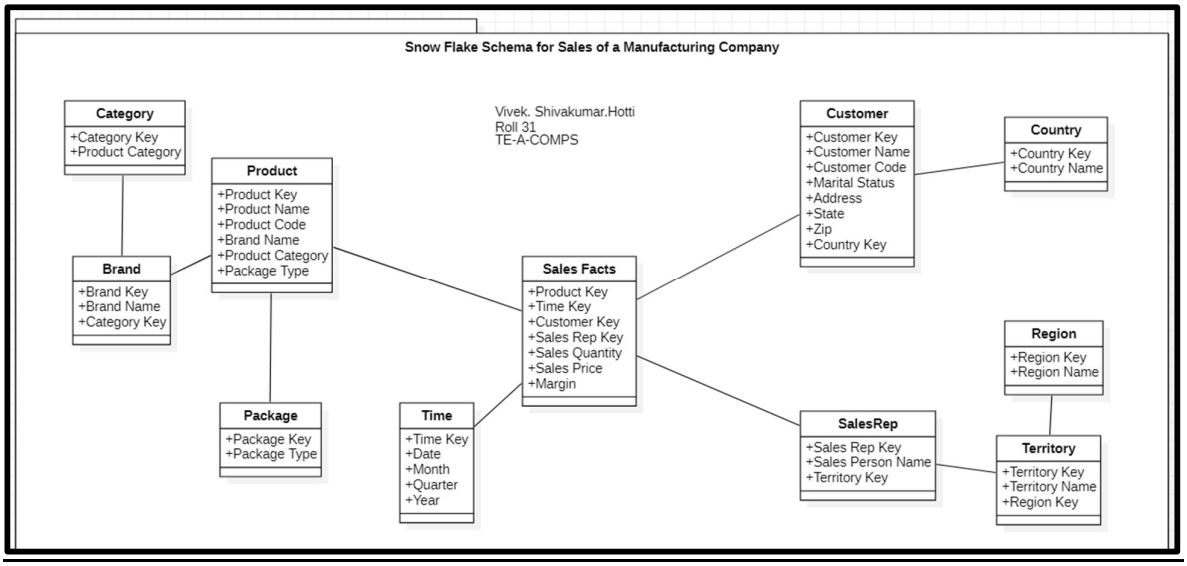
Fact Table:

- A fact table or a fact entity is a table or entity in a star or snowflake schema that stores measures that measure the business, such as sales, cost of goods, or profit. ... Fact tables and entities use primary keys that are composite keys. A composite key is made up of a subset of other keys.

PROBLEM STATEMENT:

Building a Datawarehouse different schemas for Sales of a Manufacturing Company Use Case.





CONCLUSION:

Hence, we have successfully made a Data Warehouse for a use case along with different Fact & Dimension Tables.

x-x-x

Experiment 01 Over

EXPERIMENT – 02

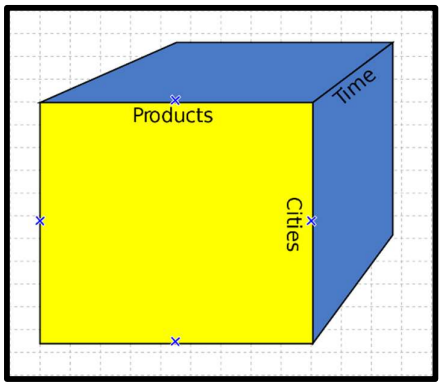
AIM:

To perform various OLAP operations such as slice, dice, drilldown, rollup, pivot.

THEORY:

OLAP Cube:

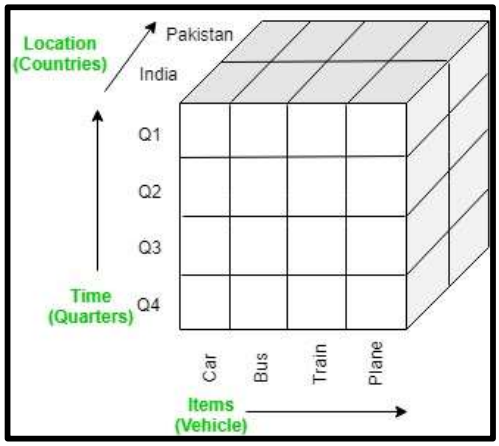
At the core of the OLAP concept, is an OLAP Cube. The OLAP cube is a data structure optimized for very quick data analysis.
The OLAP Cube consists of numeric facts called measures which are categorized by dimensions. OLAP Cube is also called the hypercube.
Usually, data operations and analysis are performed using the simple spreadsheet, where data values are arranged in row and column format. This is ideal for two-dimensional data. However, OLAP contains multidimensional data, with data usually obtained from a different and unrelated source. Using a spreadsheet is not an optimal option. The cube can store and analyse multidimensional data in a logical and orderly manner.



OLAP Operations:

1) Roll-up:

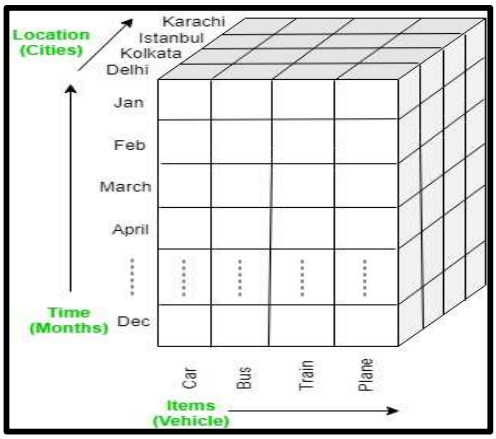
- The roll-up operation (also known as drill-up or aggregation operation) performs aggregation on a data cube, by climbing down concept hierarchies, i.e., dimension reduction. Roll-up is like zooming out on the data cubes.



(Roll-up Operation on a OLAP Cube)

2) Drill Down:

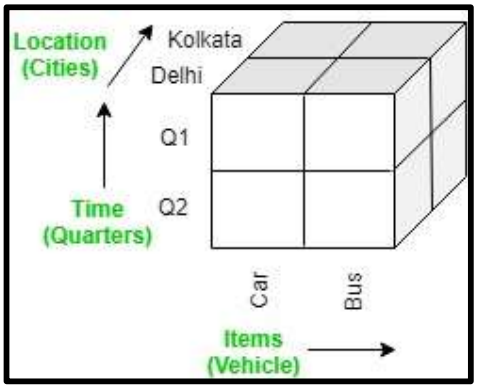
- In drill-down operation, the less detailed data is converted into highly detailed data. It can be done by:
  - Moving down in the concept hierarchy
  - Adding a new dimension.



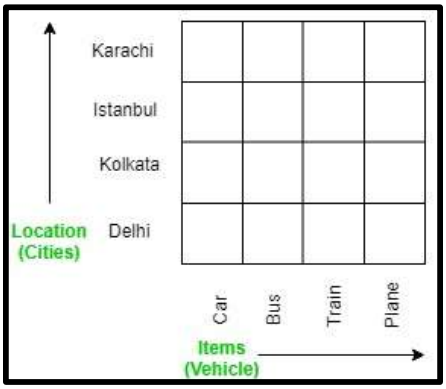
(Drill-down Operation on a OLAP Cube)

3) Slice & Dice:

- Dice selects a sub-cube from the OLAP cube by selecting two or more dimensions.
- Slice selects a single dimension from the OLAP cube which results in a new sub-cube creation.



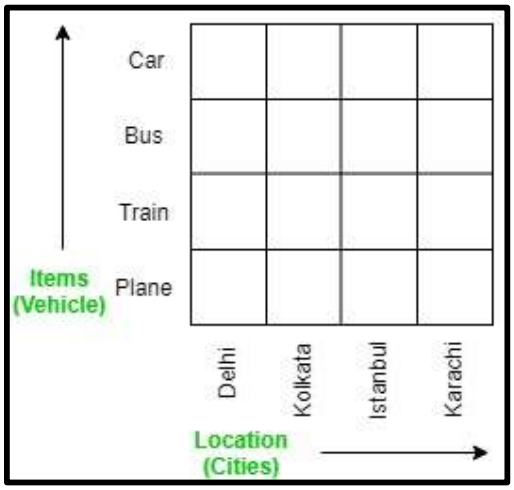
(Dice Operation on a OLAP Cube)



(Slice Operation on a OLAP Cube)

4) Pivot (Rotate):

- It is also known as rotation operation as it rotates the current view to get a new view of the representation.



(Pivot Operation on a OLAP Cube)

IMPLEMENTATION:

Step 1) Creating the Database & Entering the required values:

```
C:\Users\Vivek hotti\Downloads\sqlite-tools-win32-x86-3360000\sqlite-tools-win32-x86-3360000\sqlite3.exe
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> CREATE TABLE sales (continent varchar (20), country varchar (20), city varchar (20), units_sold integer);
sqlite> INSERT INTO sales VALUES ('North America', 'Canada', 'Toronto', 10000);
sqlite> INSERT INTO sales VALUES ('North America', 'Canada', 'Montreal', 1000);
sqlite> INSERT INTO sales VALUES ('North America', 'Canada', 'Vancouver', 15000);
sqlite> INSERT INTO sales VALUES ('Asia', 'India', 'Mumbai', 20000);
sqlite> INSERT INTO sales VALUES ('Asia', 'India', 'Hyderabad', 18000);
sqlite> INSERT INTO sales VALUES ('Asia', 'India', 'Pune', 30000);
sqlite> INSERT INTO sales VALUES ('Asia', 'India', 'Jaipur', 2000);
sqlite> INSERT INTO sales VALUES ('Asia', 'Japan', 'Toliyo', 40000);
sqlite> INSERT INTO sales VALUES ('Europe', 'UK', 'London', 22000);
sqlite> INSERT INTO sales VALUES ('Europe', 'UK', 'Manchester', 30000);
sqlite> INSERT INTO sales VALUES ('Europe', 'France', 'Paris', 21000);
```

Step 2) Rollup Operation and its output:

```
sqlite> SELECT continent, sum(units_sold) FROM sales
   ...> GROUP BY continent;
Asia|110000
Europe|73000
North America|26000
```

Step 3) Drill Down Operation and its output:

```
sqlite> SELECT city, sum(units_sold) FROM sales
   ...> GROUP BY city;
Hyderabad|18000
Jaipur|2000
London|22000
Manchester|30000
Montreal|1000
Mumbai|20000
Paris|21000
Pune|30000
Toliyo|40000
Toronto|10000
Vancouver|15000
```

Step 4) Slice Operation and its output:

```
sqlite> SELECT * FROM sales WHERE continent='Asia';
Asia|India|Mumbai|20000
Asia|India|Hyderabad|18000
Asia|India|Pune|30000
Asia|India|Jaipur|2000
Asia|Japan|Toliyo|40000
```

Step 5) Dice Operation and its output:

```
sqlite> SELECT * FROM sales WHERE continent='Asia' AND units_sold>20000;
Asia|India|Pune|30000
Asia|Japan|Toliyo|40000
```

Step 6) Rotate Operation & its output:

```
sqlite> SELECT units_sold,city,country,continent FROM sales;
10000|Toronto|Canada|North America
1000|Montreal|Canada|North America
15000|Vancouver|Canada|North America
20000|Mumbai|India|Asia
18000|Hyderabad|India|Asia
30000|Pune|India|Asia
2000|Jaipur|India|Asia
40000|Toliyo|Japan|Asia
22000|London|UK|Europe
30000|Manchester|UK|Europe
21000|Paris|France|Europe
```

CONCLUSION:

Hence, we have successfully performed OLAP operations.

x-x-x

Experiment 02 Over

EXPERIMENT – 03

AIM:

Implementation of Bayesian Classification Algorithm.

THEORY:

Bayes' Theorem:

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where A and B are events and P(B)?0.

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior probability, i.e., Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event B).
- P(A|B) is a posteriori probability of B, i.e., probability of event after evidence is seen.

Now, with regards to our dataset, we can apply Bayes' theorem in following way:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

where, y is class variable and X is a dependent feature vector (of size *n*) where:

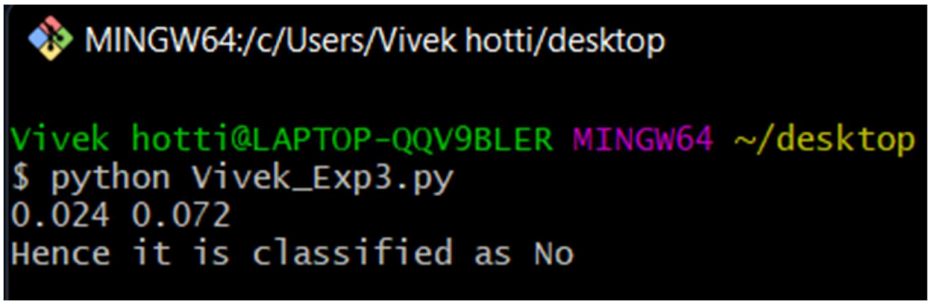
$$X = (x_1, x_2, x_3, ..., x_n)$$

IMPLEMENTATION:

CODE (INPUT):

```
colour =
['Red','Red','Red','Yellow','Yellow','Yellow','Yellow','Yellow','Red','
Red']
type_car =
['sports','sports','sports','sports','sports','SUV','SUV','SUV','SUV','
sports']
origin =
['domestic','domestic','domestic','domestic','imported','imported','imp
orted','domestic','imported','imported']
stolen = ['yes','no','yes','no','yes','no','yes','no','no','yes']

p_y = stolen.count('yes')/10
p_n = stolen.count('no')/10
p_yes = stolen.count('yes')
p_no = stolen.count('no')
p_r_yes = 0
p_r_no = 0
for i in range(10):
    if colour[i] == 'Red' and stolen[i] =='yes':
        p_r_yes+=1
    elif colour[i] =='Red' and stolen[i]=='no':
        p_r_no+=1
        p_s_yes =0
        p_s_no=0
for i in range(10):
    if type_car[i] == 'SUV' and stolen[i] =='yes':
        p_s_yes+=1
    elif type_car[i] =='SUV' and stolen[i]=='no':
        p_s_no+=1
        p_d_yes=0
        p_d_no=0
for i in range(10):
    if origin[i] == 'domestic' and stolen[i] =='yes':
        p_d_yes+=1
    elif origin[i] =='domestic' and stolen[i]=='no':
        p_d_no+=1

prob_yes = (p_r_yes/p_yes) * (p_d_yes/p_yes) * (p_s_yes/p_yes) * p_y
prob_no = (p_r_no/p_no) * (p_d_no/p_no) * (p_s_no/p_no) * p_n

print(prob_yes,prob_no)

if prob_yes > prob_no:
    print("Hence it is classified as Yes")
else:
    print("Hence it is classified as No")
```

CODE (OUTPUT):



CONCLUSION:

Hence, we have successfully implemented Bayesian Classification Algorithm.

x-x-x

Experiment 03 Over

EXPERIMENT – 04

AIM:

Implementation of K-Means Classification Algorithm.

THEORY:

K-Means Classification Algorithm:

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

The k-means clustering algorithm mainly performs two tasks:

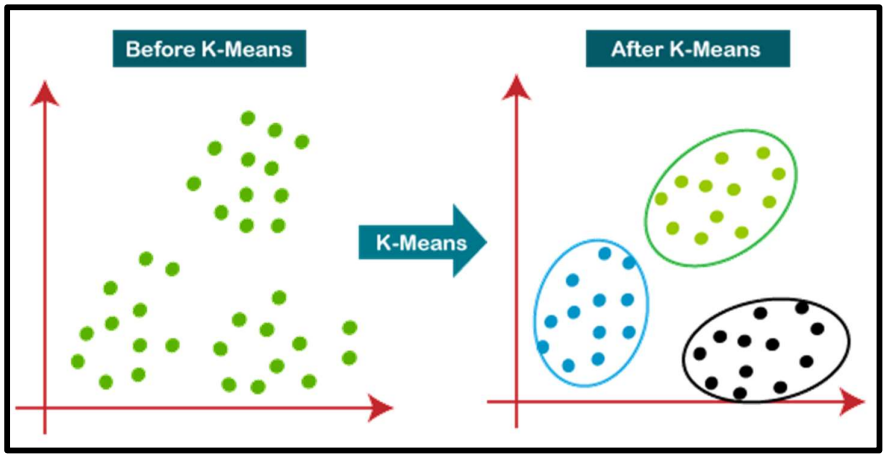
- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

kmeans algorithm is very popular and used in a variety of applications such as market segmentation, document clustering, image segmentation and image compression, etc.

The below diagram explains the working of the K-means Clustering Algorithm:

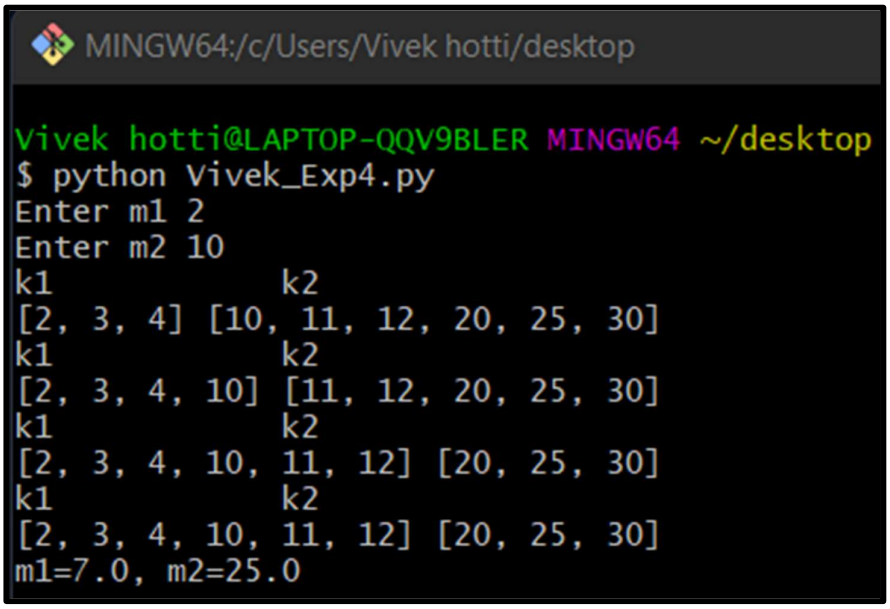


IMPLEMENTATION:

CODE (INPUT):

```
dataset = [2,3,4,10,11,12,20,25,30]
m1 = int(input("Enter m1 "))
m2 = int(input("Enter m2 "))
avg1=0
avg2=0
p1=0
p2=0

while m1!=p1 and m2!=p2:
        j=1
        k1=[]
        k2=[]
        for i in dataset:
                if abs(i-m1) < abs(i-m2):
                        k1.append(i)
                else:
                        k2.append(i)

        avg1 = sum(k1)/len(k1)
        avg2 = sum(k2)/len(k2)
        print("k{}                k{}".format(j,j+1))
        print(k1,k2)
        p1=m1
        p2=m2
        m1=avg1
        m2=avg2

print("m1={}, m2={}".format(m1,m2))
```

CODE (OUTPUT):



CONCLUSION:

Hence, we have successfully implemented K-Means Classification Algorithm.

x-x-x

Experiment 04 Over

EXPERIMENT – 05

AIM:

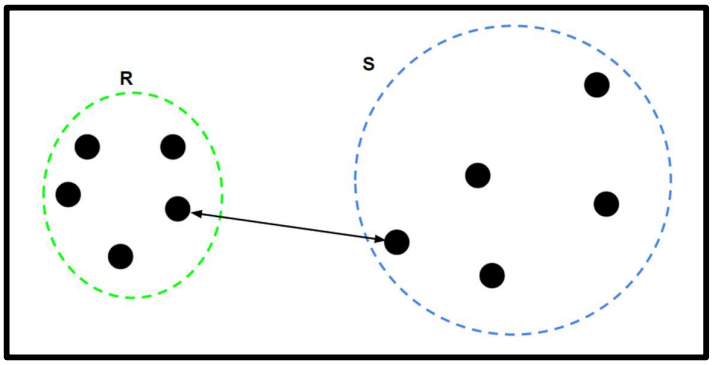
Implementation of Agglomerative Single Link Algorithm.

THEORY:

The process of Hierarchical Clustering involves either clustering sub-clusters (data points in the first iteration) into larger clusters in a bottom-up manner or dividing a larger cluster into smaller sub-clusters in a top-down manner. During both the types of hierarchical clustering, the distance between two sub-clusters needs to be computed. The different types of linkages describe the different approaches to measure the distance between two sub-clusters of data points.

Single Linkage Algorithm: For two clusters R and S, the single linkage returns the minimum distance between two points i and j such that i belongs to R and j belongs to S.

$$L(R,S) = min(D(i,j)), i \epsilon R, j \epsilon S$$



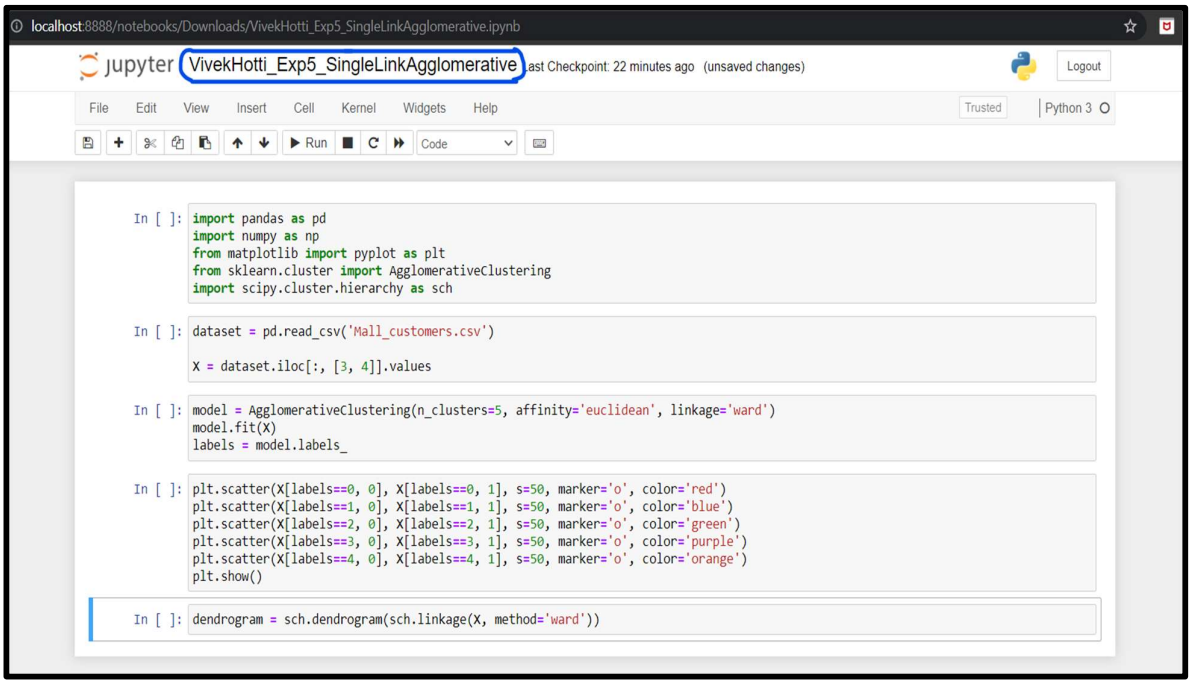
Single link algorithm is an example of agglomerative hierarchical clustering method. We recall that is a bottom-up strategy: compare each point with each point. Each object is placed in a separate cluster, and at each step we merge the closest pair of clusters, until certain termination conditions are satisfied. This requires defining a notion of cluster proximity.

IMPLEMENTATION:

CODE (INPUT):

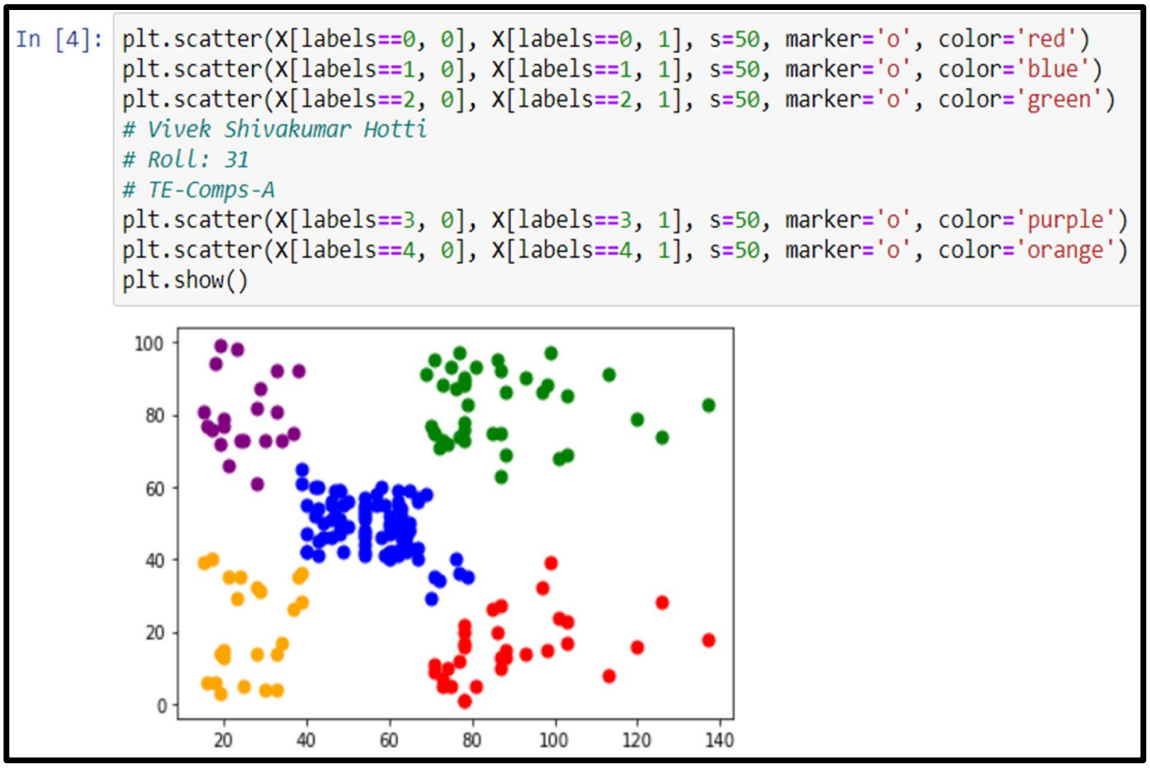
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch

dataset = pd.read_csv('Mall_customers.csv')
X = dataset.iloc[:, [3, 4]].values

model = AgglomerativeClustering(n_clusters=5,
affinity='euclidean', linkage='ward')
model.fit(X)
labels = model.labels_

plt.scatter(X[labels==0, 0], X[labels==0, 1], s=50,
marker='o', color='red')
plt.scatter(X[labels==1, 0], X[labels==1, 1], s=50,
marker='o', color='blue')
plt.scatter(X[labels==2, 0], X[labels==2, 1], s=50,
marker='o', color='green')
plt.scatter(X[labels==3, 0], X[labels==3, 1], s=50,
marker='o', color='purple')
plt.scatter(X[labels==4, 0], X[labels==4, 1], s=50,
marker='o', color='orange')
plt.show()

dendrogram = sch.dendrogram(sch.linkage(X,
method='ward'))
```

Implementing the above code in Jupyter NoteBook:



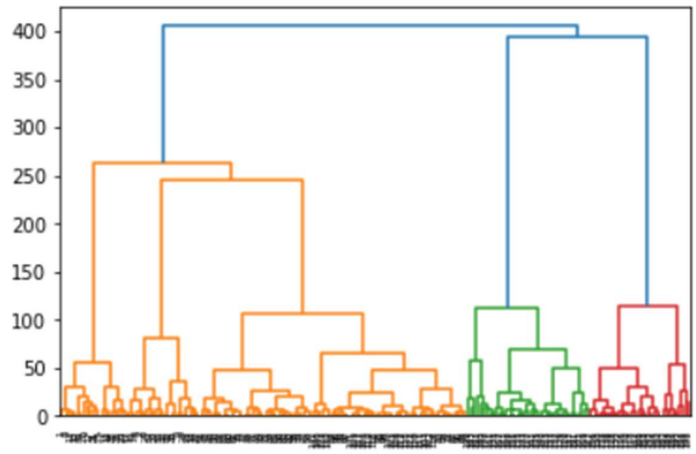
OUTPUT:

Firstly, creating a Scatter Graph for clustering:



Finally, plotting the Dendrogram:

```
In [5]: dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
        # Vivek Shivakumar Hotti
        # Roll: 31
        # TE-Comps-A
```



CONCLUSION:

Hence, we have successfully implemented Agglomerative Single Link Algorithm & created a Dendrogram.

x-x-x

Experiment 05 Over

EXPERIMENT – 06

AIM:

Implementation of Page Rank Algorithm.

THEORY:

PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages.

According to Google:

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

IMPLEMENTATION:

CODE (INPUT):

```
import java.util.*;
import java.io.*;
public class PageRank {

 public int path[][] = new int[10][10];
 public double pagerank[] = new double[10];

 public void calc(double totalNodes) {

  double InitialPageRank;
  double OutgoingLinks = 0;
  double DampingFactor = 0.85;
  double TempPageRank[] = new double[10];
  int ExternalNodeNumber;
  int InternalNodeNumber;
  int k = 1; // For Traversing
  int ITERATION_STEP = 1;
```

```
  InitialPageRank = 1 / totalNodes;
  System.out.printf(" Total Number of Nodes :" + totalNodes +
"\t Initial PageRank  of All Nodes :" + InitialPageRank +
"\n");

  // 0th ITERATION  _ OR _ INITIALIZATION PHASE //

  for (k = 1; k <= totalNodes; k++) {
   this.pagerank[k] = InitialPageRank;
  }

  System.out.printf("\n Initial PageRank Values , 0th Step
\n");
  for (k = 1; k <= totalNodes; k++) {
   System.out.printf(" Page Rank of " + k + " is :\t" +
this.pagerank[k] + "\n");
  }

  while (ITERATION_STEP <= 2) // Iterations
  {
   // Store the PageRank for All Nodes in Temporary Array
   for (k = 1; k <= totalNodes; k++) {
    TempPageRank[k] = this.pagerank[k];
    this.pagerank[k] = 0;
   }

   for (InternalNodeNumber = 1; InternalNodeNumber <=
totalNodes; InternalNodeNumber++) {
    for (ExternalNodeNumber = 1; ExternalNodeNumber <=
totalNodes; ExternalNodeNumber++) {
     if (this.path[ExternalNodeNumber][InternalNodeNumber] ==
1) {
      k = 1;
      OutgoingLinks = 0; // Count the Number of Outgoing Links
for each ExternalNodeNumber
      while (k <= totalNodes) {
       if (this.path[ExternalNodeNumber][k] == 1) {
        OutgoingLinks = OutgoingLinks + 1; // Counter for
Outgoing Links
       }
       k = k + 1;
      }
      // Calculate PageRank
      this.pagerank[InternalNodeNumber] +=
TempPageRank[ExternalNodeNumber] * (1 / OutgoingLinks);
     }
    }
   }
```

```
   System.out.printf("\n After " + ITERATION_STEP + "th Step
\n");

   for (k = 1; k <= totalNodes; k++)
    System.out.printf(" Page Rank of " + k + " is :\t" +
this.pagerank[k] + "\n");

   ITERATION_STEP = ITERATION_STEP + 1;
  }
  // Add the Damping Factor to PageRank
  for (k = 1; k <= totalNodes; k++) {
   this.pagerank[k] = (1 - DampingFactor) + DampingFactor *
this.pagerank[k];
  }

  // Display PageRank
  System.out.printf("\n Final Page Rank : \n");
  for (k = 1; k <= totalNodes; k++) {
   System.out.printf(" Page Rank of " + k + " is :\t" +
this.pagerank[k] + "\n");
  }

 }

 public static void main(String args[]) {
  int nodes, i, j, cost;
  Scanner in = new Scanner(System.in);
  System.out.println("Enter the Number of WebPages \n");
  nodes = in .nextInt();
  PageRank p = new PageRank();
  System.out.println("Enter the Adjacency Matrix with 1->PATH
& 0->NO PATH Between two WebPages: \n");
  for (i = 1; i <= nodes; i++)
   for (j = 1; j <= nodes; j++) {
    p.path[i][j] = in .nextInt();
    if (j == i)
     p.path[i][j] = 0;
   }
  p.calc(nodes);

 }
}
```

OUTPUT:

```
MINGW64:/c/Users/Vivek hotti/desktop
Vivek hotti@LAPTOP-QQV9BLER MINGW64 ~/desktop
$ java PageRank
Enter the Number of WebPages

5
Enter the Adjacency Matrix with 1->PATH & 0->NO PATH Between two WebPages:

0 1 0 0 0
0 0 0 0 1
1 1 0 1 1
0 0 1 0 1
0 0 0 1 0
 Total Number of Nodes :5.0      Initial PageRank  of All Nodes :0.2

 Initial PageRank Values , 0th Step
 Page Rank of 1 is :    0.2
 Page Rank of 2 is :    0.2
 Page Rank of 3 is :    0.2
 Page Rank of 4 is :    0.2
 Page Rank of 5 is :    0.2

 After 1th Step
 Page Rank of 1 is :    0.05
 Page Rank of 2 is :    0.25
 Page Rank of 3 is :    0.1
 Page Rank of 4 is :    0.25
 Page Rank of 5 is :    0.35

 After 2th Step
 Page Rank of 1 is :    0.025
 Page Rank of 2 is :    0.07500000000000001
 Page Rank of 3 is :    0.125
 Page Rank of 4 is :    0.375
 Page Rank of 5 is :    0.4

 Final Page Rank :
 Page Rank of 1 is :    0.17125
 Page Rank of 2 is :    0.21375000000000002
 Page Rank of 3 is :    0.25625000000000003
 Page Rank of 4 is :    0.46875
 Page Rank of 5 is :    0.49000000000000005
```

CONCLUSION:

Hence, we have successfully implemented Page Rank Algorithm.

x-x-x

Experiment 06 Over

EXPERIMENT – 07

AIM:

Perform data Pre-processing task and Demonstrate performing Classification, Clustering, Association algorithm on data sets using WEKA data mining.

THEORY:

Weka:
Weka is data mining software that uses a collection of machine learning algorithms. These algorithms can be applied directly to the data or called from the Java code.
Weka is a collection of tools for:

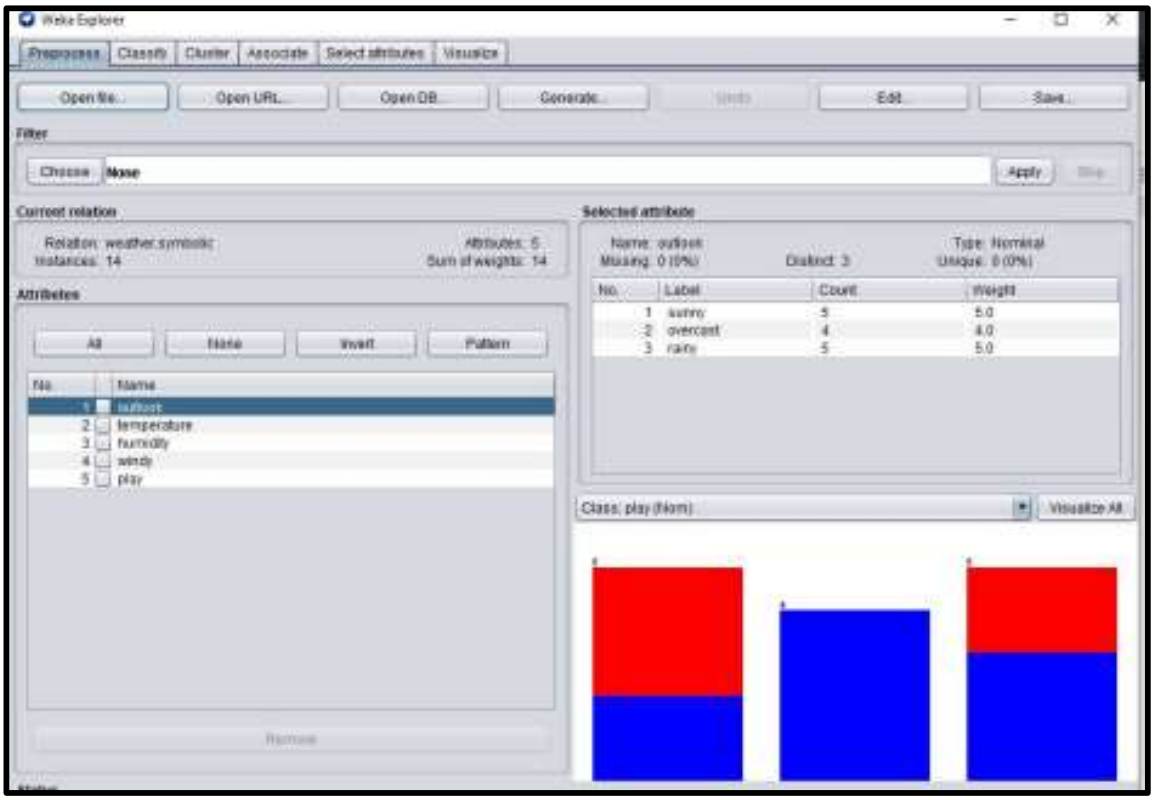
- Clustering
- Association
- Data pre-processing
- Classification
- Visualisation

Weka Explorer:
The Weka Explorer consists of the following:

1) Pre-process: This allows us to choose the data file. There are three ways to inject the data for pre-processing:

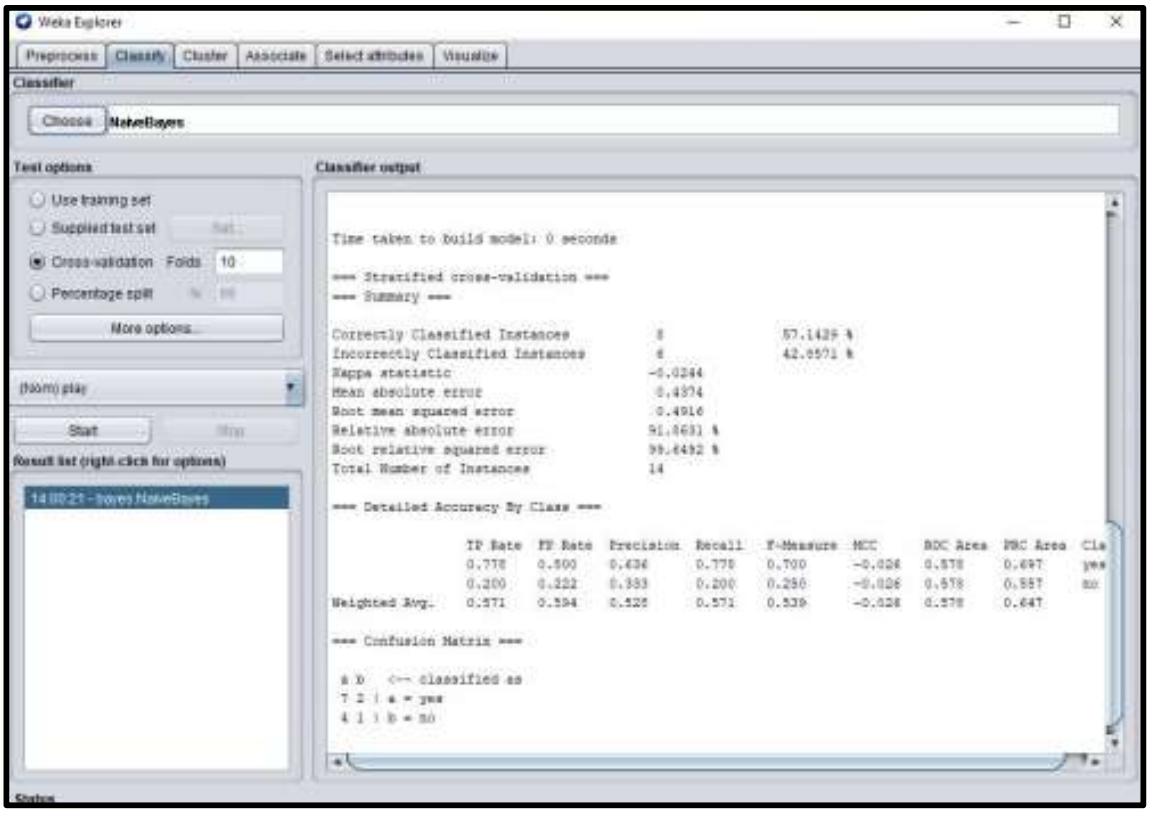
- ***Open File*** – enables the user to select the file from the local machine
- ***Open URL*** – enables the user to select the data file from different locations
- ***Open Database*** – enables users to retrieve a data file from a database source.

After loading the data in Explorer, we can refine the data by selecting different options. We can also select or remove the attributes as per our need and even apply filters on data to refine the result.



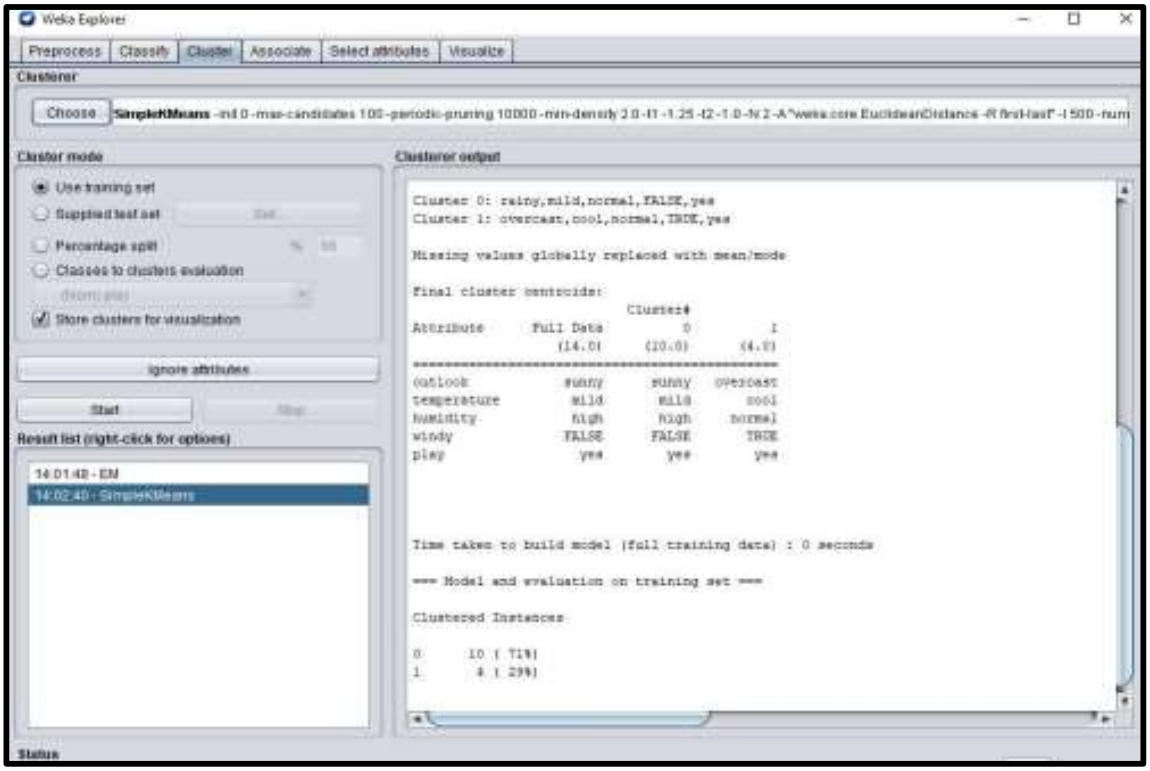
(1.Pre-Process)

2) Classify: This allows us to apply and experiment with different algorithms on pre-processed data files. To predict nominal or numeric quantities, we have classifiers in Weka. Once the data has been loaded, all the tabs are enabled. Based on the requirements and by trial and error, we can find out the most suitable algorithm to produce an easily understandable representation of data.



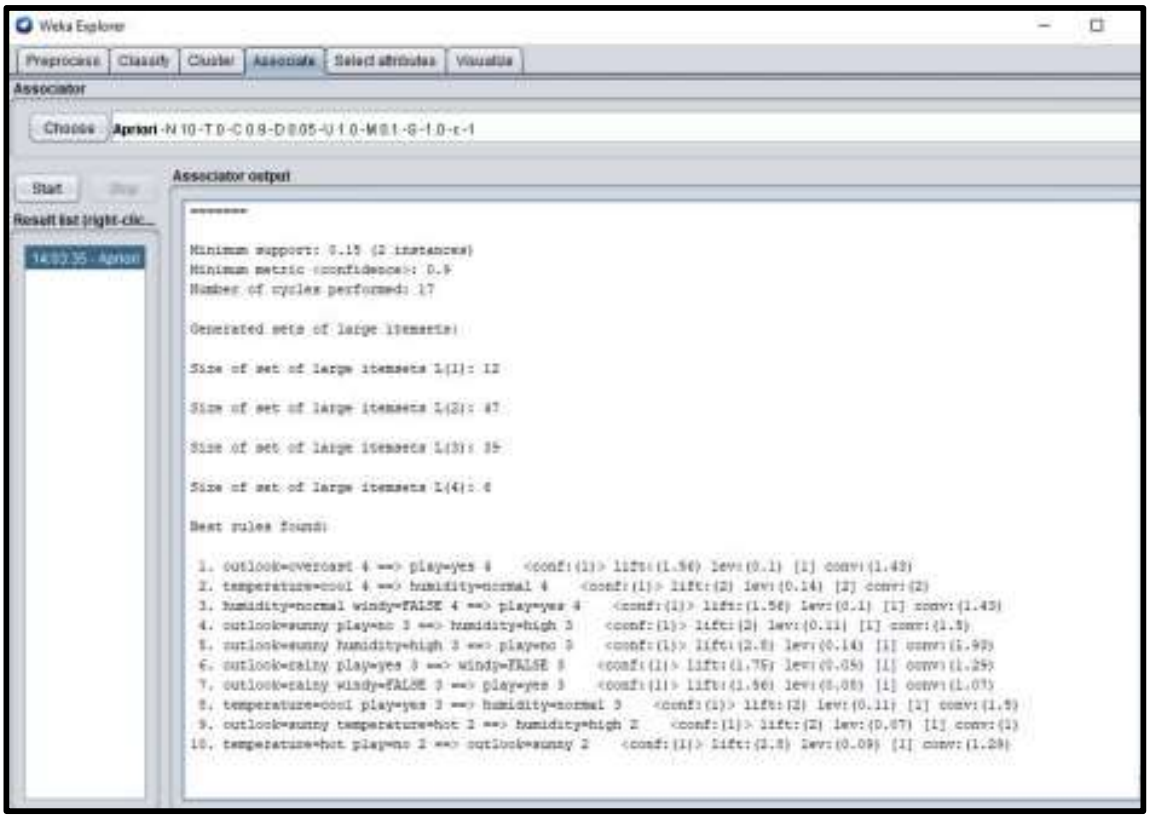
(2. Classify)

3) Cluster: This allows us to apply different clustering tools, which identify clusters within the data file. Clustering can provide data for the user to analyse. The training set, percentage split, supplied test set and classes are used for clustering, for which the user can ignore some attributes from the data set, based on the requirements.



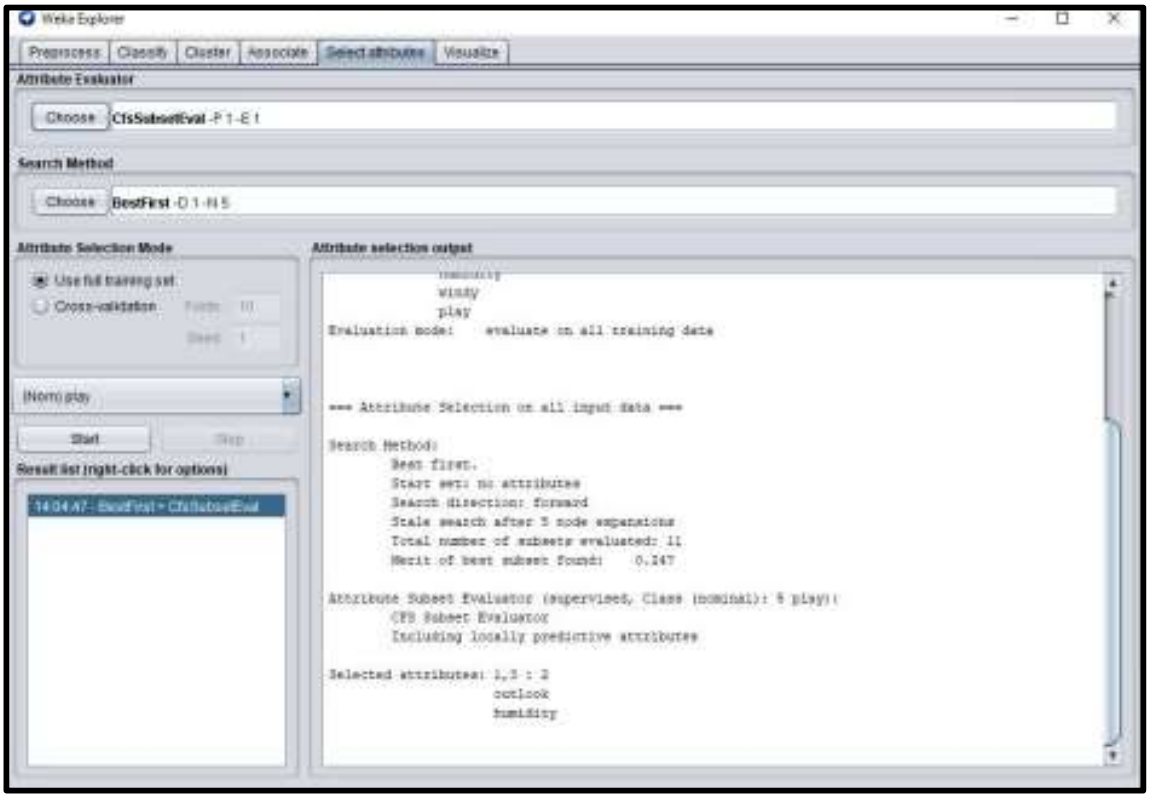
(3.Cluster)

4) Association: This allows us to apply association rules, which identify the association within the data. It identifies statistical dependencies between clusters of attributes, and only works with discrete data.



(4.Association)

5) Select attributes: These allow us to see the changes on the inclusion and exclusion of attributes from the experiment.



(5.Select Attributes)

6) Visualize: This allows us to see the possible visualisation produced on the data set in a 2D format, in scatter plot and bar graph output.

The user cannot move between the different tabs until the initial pre-processing of the data set has been completed.

CONCLUSION:

Thus, we have studied implementation of Classification, Clustering, Association algorithm on data sets using WEKA data mining.

x-x-x

Experiment 07 Over

EXPERIMENT – 08

AIM:

Implementation of HITS Algorithm.

THEORY:

Hyperlink Induced Topic Search (HITS) Algorithm is a Link Analysis Algorithm that rates webpages, developed by Jon Kleinberg. This algorithm is used to the web link-structures to discover and rank the webpages relevant for a particular search.
HITS uses hubs and authorities to define a recursive relationship between webpages. Before understanding the HITS Algorithm, we first need to know about Hubs and Authorities

Given a query to a Search Engine, the set of highly relevant web pages are called Roots. They are potential Authorities.
Pages that are not very relevant but point to pages in the Root are called Hubs. Thus, an Authority is a page that many hubs link to whereas a Hub is a page that links to many authorities.

The idea of this algorithm originated from the fact that an ideal website should link to other relevant sites and also being linked by other important sites.

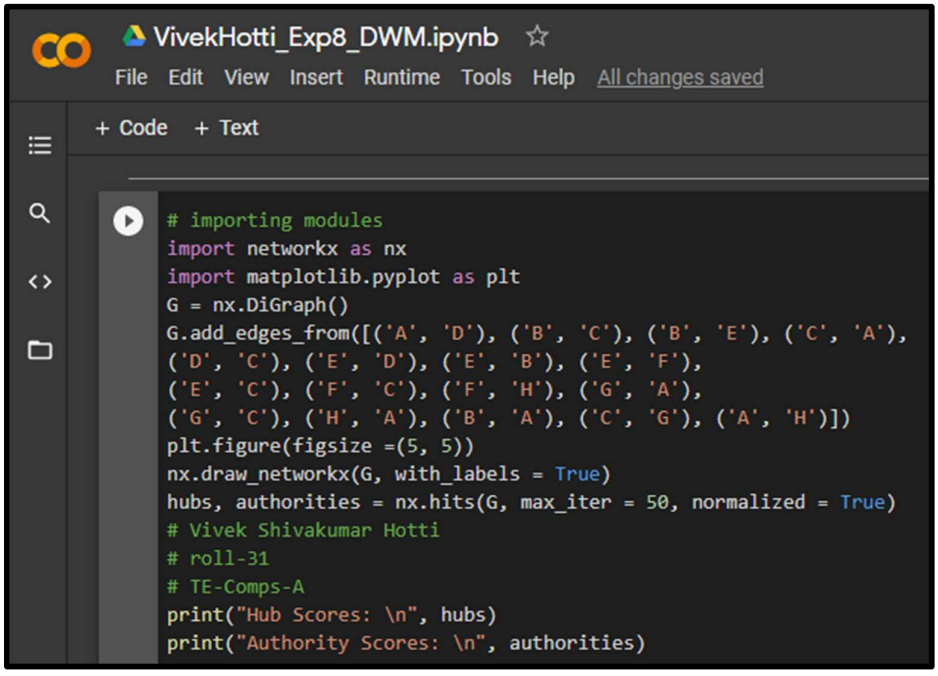
Overall, Hyperlink Induced Topic Search (HITS) is an algorithm used in link analysis. It could discover and rank the webpages relevant for a particular search. The idea of this algorithm originated from the fact that an ideal website should link to other relevant sites and also being linked by other important sites.

IMPLEMENTATION:

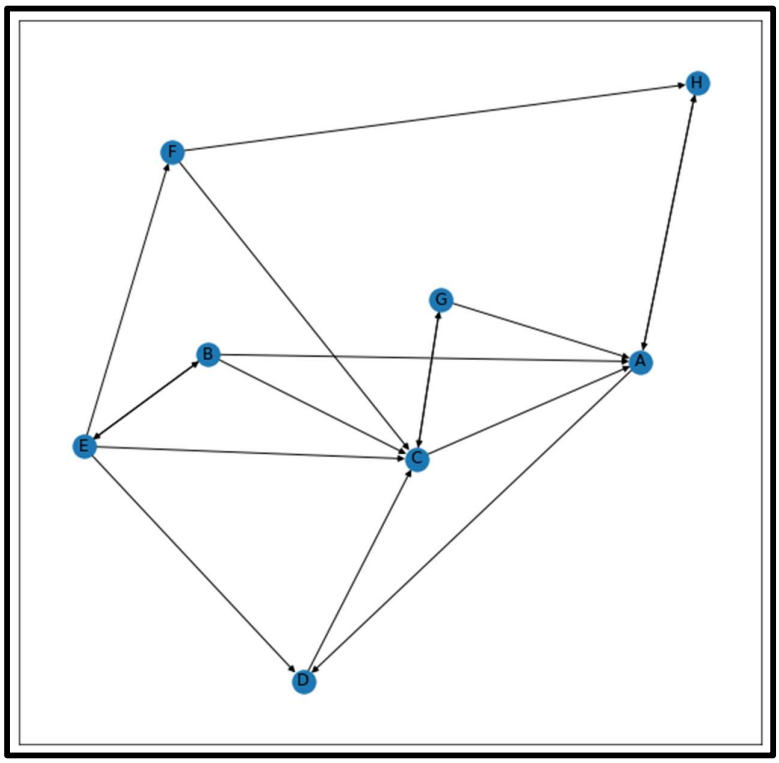
CODE (INPUT):

```
# importing modules
import networkx as nx
import matplotlib.pyplot as plt
G = nx.DiGraph()
G.add_edges_from([('A', 'D'), ('B', 'C'), ('B', 'E'), ('C',
'A'),('D', 'C'), ('E', 'D'), ('E', 'B'), ('E', 'F'),
('E', 'C'), ('F', 'C'), ('F', 'H'), ('G', 'A'),
('G', 'C'), ('H', 'A'), ('B', 'A'), ('C', 'G'), ('A', 'H')])
plt.figure(figsize =(5, 5))
nx.draw_networkx(G, with_labels = True)
hubs, authorities = nx.hits(G, max_iter = 50, normalized =
True)
# The in-built hits function returns two dictionaries keyed by
nodes
# containing hub scores and authority scores respectively.
print("Hub Scores: \n", hubs)
print("Authority Scores: \n", authorities)
```

Implementing the above code in Google Collab Notebook.



OUTPUT:




```
Hub Scores:
 {'A': 0.05660059219529647, 'D': 0.10492390650243162, 'B': 0.20040264500110516, 'C': 0.07952198102398703, 'E': 0.1860890363503576, 'F': 0.12948844415506114, 'H': 0.0690247441346647, 'G': 0.17394865063709633}
Authority Scores:
 {'A': 0.21799842834451, 'D': 0.10117834739141149, 'B': 0.0775813176624792S, 'C': 0.3313774936806945, 'E': 0.08354872252097575, 'F': 0.0775813176624792S, 'H': 0.0775813176624792G, 'G': 0.03315305507497045}
```

Hub Scores:

{'A': 0.05660059219529647,
'D': 0.10492390650243162,
'B': 0.20040264500110516,
'C': 0.07952198102398703,
'E': 0.1860890363503576,
'F': 0.12948844415506114,
'H': 0.0690247441346647,
'G': 0.17394865063709633}

Authority Scores:

{'A': 0.21799842834451,
'D': 0.10117834739141149,
'B': 0.07758131766247925,
'C': 0.3313774936806945,
'E': 0.08354872252097575,
'F': 0.07758131766247925,
'H': 0.07758131766247926,
'G': 0.03315305507497045 4}

CONCLUSION:

Thus, we have studied & implemented HITS (Hyperlink Induced Topic Search) algorithm.

x-x-x

Experiment 08 Over

((((( ))))))

(You have reached the end of the D.W.M Lab Manual)