Experiment-7

Aim: Demonstrate zeroR technique on iris dataset.

Objectives:

IRis is an open access flower-based dataset is normally available on UCI dataset. The major objective of this research work is to examine the IRIS data using data mining techniques available supported in weka. In this work, four different classifier, viz Bayes, Network classifier, J48, Random forest and oneR has been successfully used to classify attributes the dataset.

ZeroR:

→ ZeroR is the simplest classification which relies on the largest and ignores all predictors.

→ ZeroR classifier simply predicts the majority category.

→ Although there is no predictability power in zeroR it is useful for determining a baseline performance as a benchmark for other classification methods.

Iris Dataset:

The Iris flower dataset is a famous dataset from statistics and is heavily borrowed by researches in machine learning. It contains 150 instances and 4 attributes and a class attribute for the species of iris flower.

WEKA:

WEKA (Waikola environment for knowledge Analysis is a popular suite of machine learning software written in java, developed at the university of waikota, New zealand weka is a free software available under the GNO, General Public license.

Preprocess tab:

It is first step in machine learning is to preprocess the data. It is used to select the data files, preprocess it and make it fit for applying the various machine learning algorithms.

Loading Data:- The first four buttons at the top of the preprocess section enable you to load data into WEKA.

→ open file:- Brings up a dialog box allowing you to browse for the data file on the local file system.

→ open URL... Asks for a uniform Resource locater address for where the data is stored.

→ open DB... Reads data from a database.

→ Generate.. Enables you to generate artificial data from variety of data generators. Using the open files... button you can read files in a variety of formats - WEKA's ARFF format, CSV format, C4.5 format.

classify:- The classify tab provide you several machine learning algorithms for the classification of your data - such as linear regression, logistic regression.

Test options:-

Before you run the classification algorithm, you need to set test options. Set test options in the Test options box. The test options that available are:

1. Use training set:- Evaluates the classifier on how well it predicts the class at the instances it was trained on.

2. Supplied test set:- Evaluates the classifier on how well it predicts the class of a set of instances loaded from a file. Clicking on the set... button brings up a dialog allowing you to choose the file to test on.

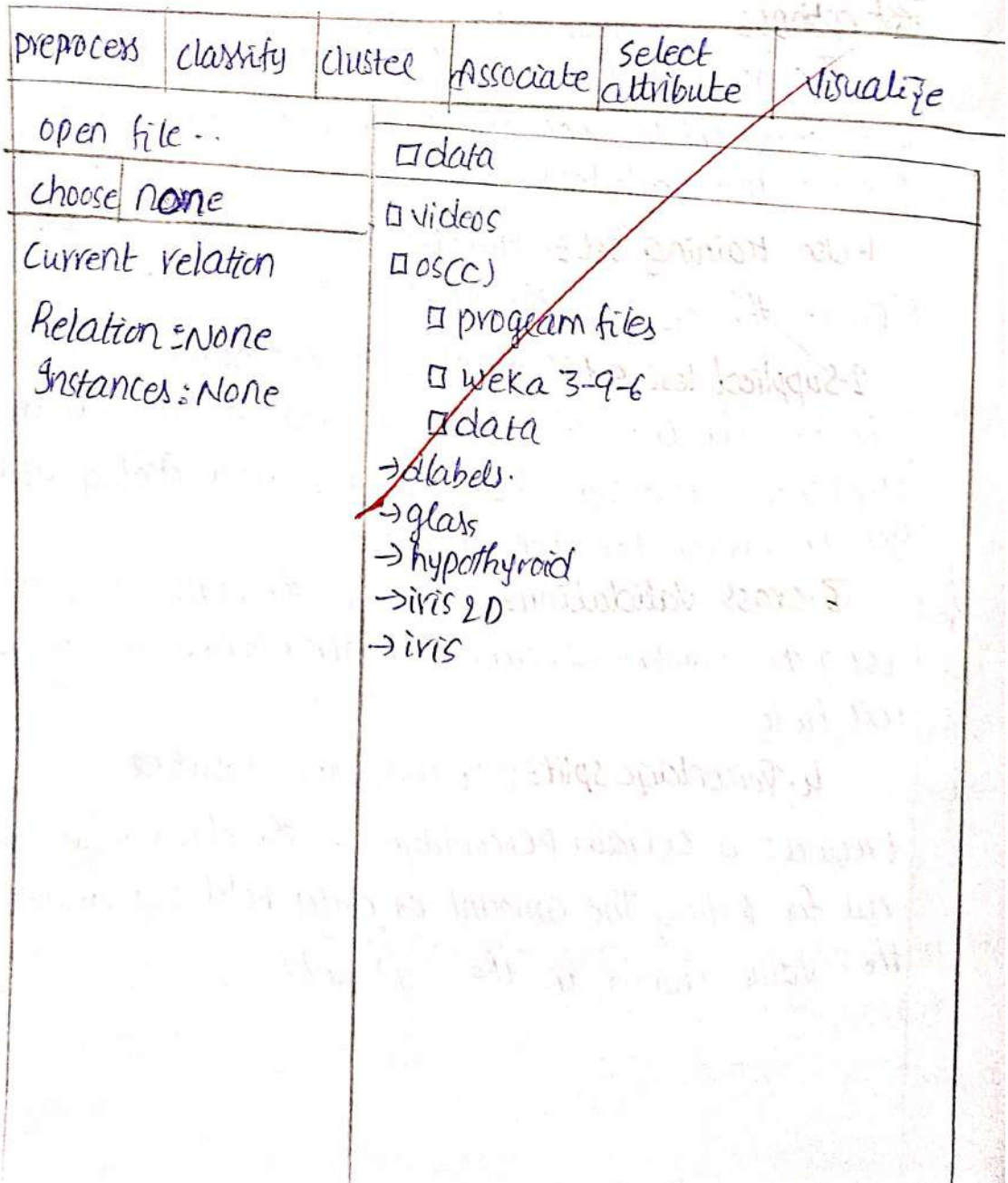
3. Cross validation:- Evaluates the classifier by cross-validation using the number of folds that are entered in the 'Folds' text field.

4. Percentage split:- Evaluates the classifier on how well it predicts a certain percentage of the data, which is held out for testing. The amount of data held out depends on the value entered in the '%' field.

Steps Required:

1. Open WEKA tool.
2. Click on WEKA explorel.
3. Click on the preprocess, and click on open file.
4. Open cdrive, then click on open file, then select WEKA 3.9.6
5. Select data file.
6. Select iris dataset.
7. You can see the Jaciance of the graph.
8. Click on the classifiel, then zeroR
9. And set cross Jalidation fold as 10.
10. Click on submit.

Output:



preprocess | classify | cluster | Associate | select Attribute | visualize

classifier

choose | ZeroR

Test options.
- Use training set
- Supplied test set
- Cross-validation Folds 10
- percentage splits % 60

More options...

Start | Stop

classifier output:

21/10/22

Experiment-10

Aim: write a program to calculate chi-square value using Python. Report your observation.

Objective:

Chi-square Test

The chi-square test is a statistical procedure for determining the difference between observed and expected data. This test can also be used to determine whether it correlates to the categorical variables in our data. It helps to find out whether a difference between two categorical variables is due to chance or a relationship between them.

chi2: The test statistic

P: The p-value of the test

dof: Degrees of freedom

expected: The expected frequencies, based on the marginal sums of the table.

Rolling 36 times

Table:

2 rows * 6 columns	1	2	3	4	5
Expected (E)	6	6	6	6	6
Observed (O)	2	4	8	9	3

$$\frac{(O-E)^2}{E} + \frac{(O-E)^2}{E} + \frac{(O-E)^2}{E} + \frac{(O-E)^2}{E} + \frac{(O-E)^2}{E} + \frac{(O-E)^2}{E}$$

$$\frac{(2-6)^2}{6} + \frac{(4-6)^2}{6} + \frac{(8-6)^2}{6} + \frac{(9-6)^2}{6} + \frac{(3-6)^2}{6} + \frac{(10-6)^2}{6}$$

① Chi-Squared - 9.6

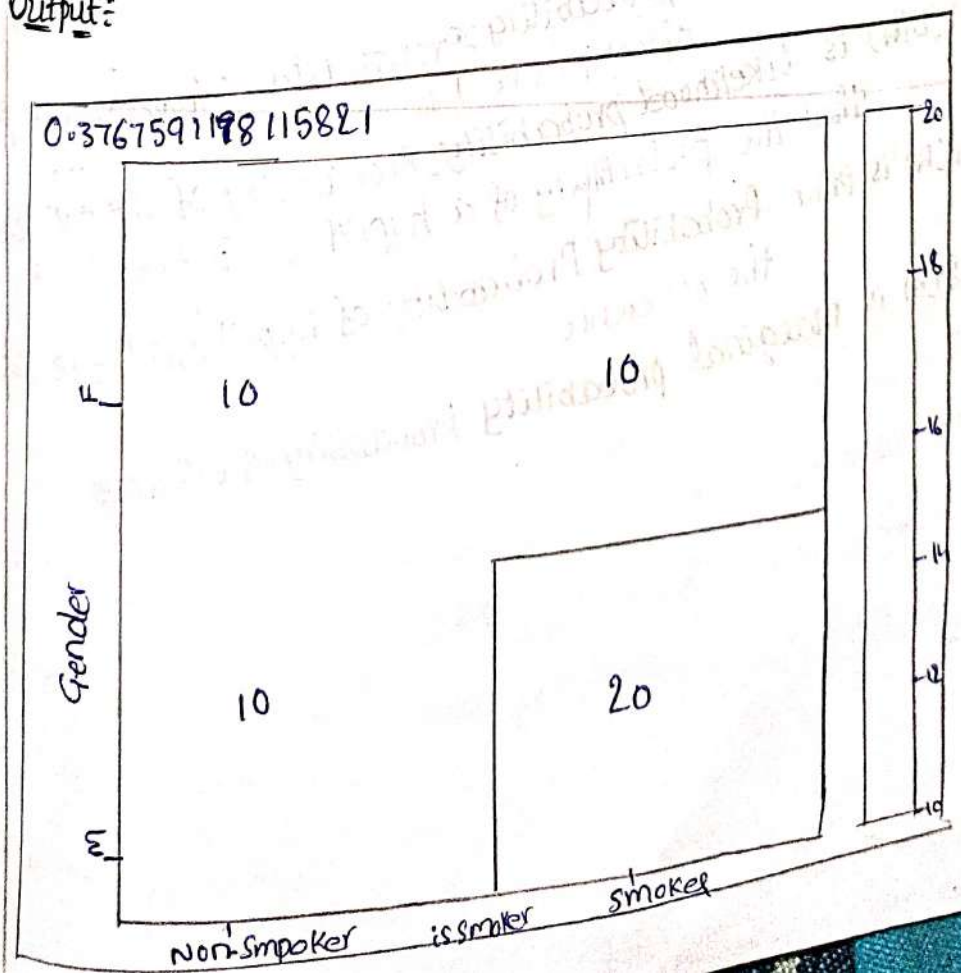
② Degree of freedom

(#rows - 1) * (columns - 1) = (2-1) * (6-1) = 5

program:

```
import pandas as pd
import numpy as np
from scipy.stats import chi2_contingency
import seaborn as sns
import matplotlib.pyplot as plt
%.matplotlib inline
df=pd.DataFrame({'Gender':['M','M','M','F','F']*10,
'isSmoker':['Smoker','Smoker','Non-Smpoker','Non-Smpoker','Smoker']*10})
df.head()
Contigency=pd.crosstab(df['Gender'],df['isSmoker'])
Contigency
Contigency_pct=pd.crosstab(df['Gender'],df['isSmoker'],normalize='index')
Contigency_pct
plt.figure(figsize=(12,8))
sns.heatmap(Contigency,annot=True,cmap="YlGnBu")
C,P,dof,expected=chi2_contigency(Contigency)
Print(P)
```

Output:



Experiment-11

Aim: Write a program of Naive Bayesian classification using Python programming language.

Objectives:

The Naive Bayes algorithm is comprised of two words Naive and Bayes, which can be described as:

- **Naive:** It is called Naive because it assumes that the occurences of a certain feature is independent of the occurence of other features.
- **Bayes:** It is called Bayes because it depends on the principle of Baye's theorem.

Baye's theorem:

- Baye's theorem is also known as Baye's Rule or Baye's law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability
- The formula for Baye's theorem is given as:

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

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability Probability of hypothesis before observing the evidence

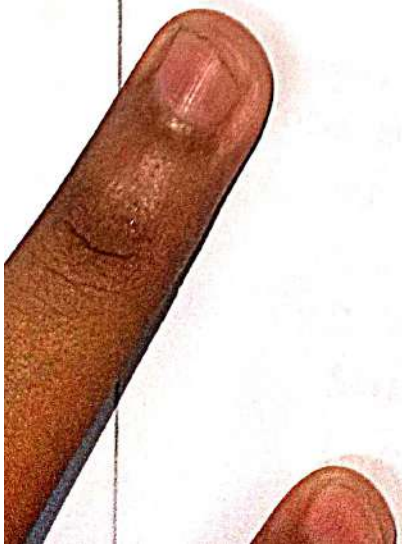
P(B) is marginal probability Probability of evidence

Program:-

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
Y = iris.target
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4,
        random_state=1)
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, Y_train)
Y_pred = gnb.predict(X_test)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy (in %): ", metrics.accuracy_
        score(Y_test, Y_pred)*100)
```

Output:-

Gaussian Naive Bayes model accuracy (in %): 95.0



Experiment-15

Aim: Write a program to compute/display dissimilarity matrix (for your own dataset containing at least four instances with two attributes) using Python.

Objectives:

The dissimilarity matrix describes pairwise distinction between M objects. It is a square symmetrical MxM matrix with the (i,j)th element equal to the value chosen measure of distinction b/w the (i)th and the (j)th object.

Program:

```
from sklearn.manifold import MDS
from matplotlib import pyplot as plt
import sklearn.datasets as dt
import seaborn as sns
import numpy as np
from sklearn.metrics.pairwise import manhattan_distances,
euclidean_distances
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
X=np.array([[0,0,0],[0, 0, 1], [1,1,1], [0,1,0], [0,1,1]])
mds= MDS(random_state=0)
X_transform =mds.fit_transform(X)
print(X_transform)
```

Output:

```
[[ 0.72521687   0.52943352]
 [ 0.61640884  -0.48411805]
 [-0.9113603   -0.47905115]
 [-0.2190564    0.71505714]
 [-0.21120901  -0.28132146]]
```

Experiment-12

Aim: Implement a java program to peeform apriori algorithm.

Objectives:

Apriori algorithm refees to an algorithm that is used in mining frequent product sets and relevant association rules. Generally, the apriori algorithm operates on a database containing a huge number of transactions.

Components of apriori algorithm

* The given three components comprise the apriori algorithm.
  1. Support
  2. Confidence
  3. Lift

Program:

```
import java.util.*;
Public class main{
    Public static void main(String args[]){
        Scanner terminal=new Scanner(System.in);
        System.out.print("Numbel of transactions: ");
        int numberOfTransactions=Integer.parseInt(terminal.nextLine());
        System.out.println("Enter transactions seperated by new line
        and items seperated by spaces:");
        ArrayList<ArrayList<String>>transactions=new ArrayList<Array
          List<String>>();
        ArrayList <ArrayList<String>>_transactions=new ArrayList<
        ArrayList <String>>();
        ArrayList<ArrayList<String>>prevItemSetsWithMinSupportCoun
        =new ArrayList <ArrayList <String>>();
        for(int i=0; i<numberOfTransactions;i++){
            ArrayList<String>transaction=new ArrayList<String>();
            String str=terminal.nextLine();
            String arr[]=str.split(" ");
            for(int j=0;j<arr.length;j++)transaction.add(arr[j]);
            transactions.add(transaction);
```

```
transactions.add(transaction);
}
System.out.print("Minimum support count: ");
int minSupportCount=Integer.parseInt(terminal.nextLine());
ArrayList<String> items=getUniqueItems(transactions);
int x=0;
while(true){
    x++;
    ArrayList<Integer> supportCountList=new ArrayList<Integer>
    ArrayList<ArrayList<String>>itemSets=getItemSets(items,x);
    for(ArrayList<String> itemset:itemSets){
        int count=0;
        for(ArrayList<String>transaction:transactions){
            if(exists InTransaction(itemset,transaction)){count++;
            }
            supportCountList.add(count);
    }
    ArrayList<ArrayList<String>>itemsetsWithMinSupportCount=
    getItemSetsWithMinSupportCount(itemSets,supportCountList,
    minSupportCount);
    if(itemsetsWithMinSupportCount.size()==0){
        System.out.print("The itemset(s) that are the most
        frequent itemset(s):");
        System.out.println(prevItemSetsWithMinSupportCount
        break;
    }
    items=getUniqueItems(itemSetsWithMinSupportCount);
    PrevItemSetsWithMinSupportCount=itemSetsWithMinSuppo
            Count;
    }
}
private static ArrayList<String>getUniqueItems(ArrayList<Arra
        List<String>>data){
```

```
ArrayList<String> toReturn = new ArrayList<String>();
for(ArrayList<String> transaction : data){
    for(String item: transaction){
        if(!toReturn.contains(item))toReturn.add(item);
    }
}
Collections.sort(toReturn);
return toReturn;
}
Private static ArrayList<ArrayList<String>>getItemSets(ArrayList
<String>items, int number)
{
  if(number==1)
  {
    ArrayList<ArrayList<String>> toReturn=new ArrayList<
        ArrayList<String>>();
    for(String item :items){
        Arraylist<String>alist=new ArrayList<String>();
        aList.add(item);
    }       toReturn.add(aList);
    return toReturn;
  }
  else
   {
    int size=items.size();
     ArrayList<ArrayList<String>> toRetun=new ArrayList
                <ArrayList<String>>();
    for(int i=0;i< size;i++){
       Arraylist<String>-items=new ArrayList<String>();
       for(String item: items){
              -items.add(item);
       }
        String thisItem=items.get(i);
       for(int j=0;j<=i;j++)
              -items.remove(0);
    }
```

```
        ArrayList<ArrayList<String>> permutationsBelow =
            getItemsetsC_items, number-1);
        for(ArrayList<String>aList:permutationsBelow){
            aList.add(thisItem);
            Collections.sort(aList);
            toReturn.add(aList);
        }
    }
    return toReturn;
  }
}
Private static boolean existsInTransaction(ArrayList<String>items,
    ArrayList<String> transaction){
    for(String item: items){
        if(!transaction.contains(item)) return false;
    }
    return true;
}
Private static ArrayList<ArrayList<String>>getItemSetsWith
  MinSupportCount(ArrayList<ArrayList<String>>itemSets,
  ArrayList<Integer>Count,int minSupportCount){
    ArrayList<ArrayList<String>> toReturn=new ArrayList<
        ArrayList<String>>();
    for(int i=0;i<Count.size();i++)
    {
        int c= Count.get(i);
        if(c>=minSupportCount){
            toReturn.add(itemSets.get(i));
        }
    }
    return toReturn;
}
}
```

Output:

Number of transactions: 4

Enter transactions seperated by new line and items seperated by spac

Pizza burger dosa

pizza burger

pizza onion ~~tea~~ tea

burger onion tea

Minimum support count: 2

The itemset(s) that are the most frequent itemset(s): [[burger, pizza], [onion, tea]]

Experiment-16

Aim: Visualize the datasets using matplotlib in python. (Histogram, Box plot, Bar

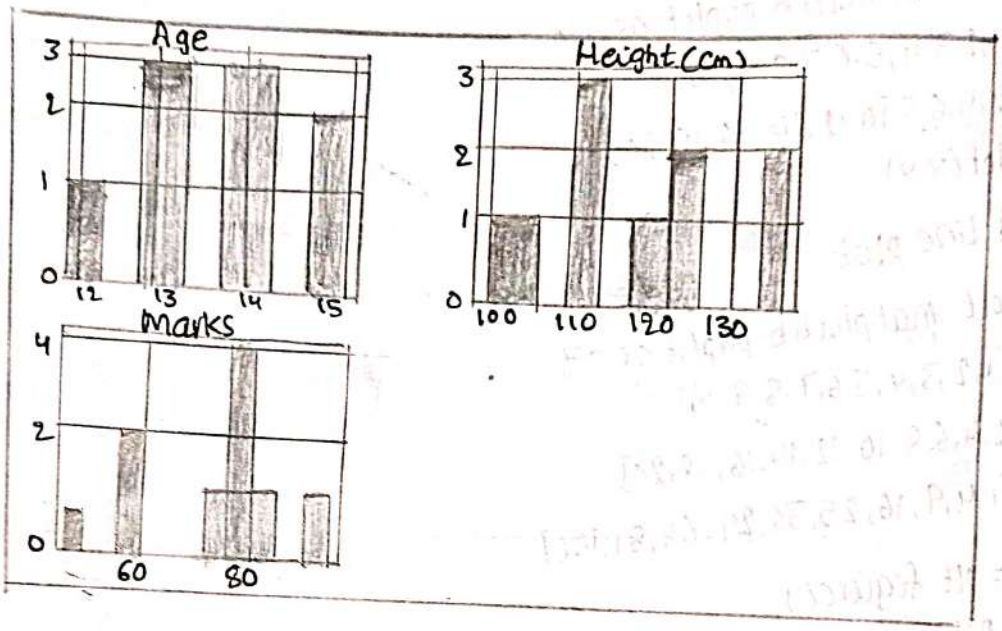
Program:

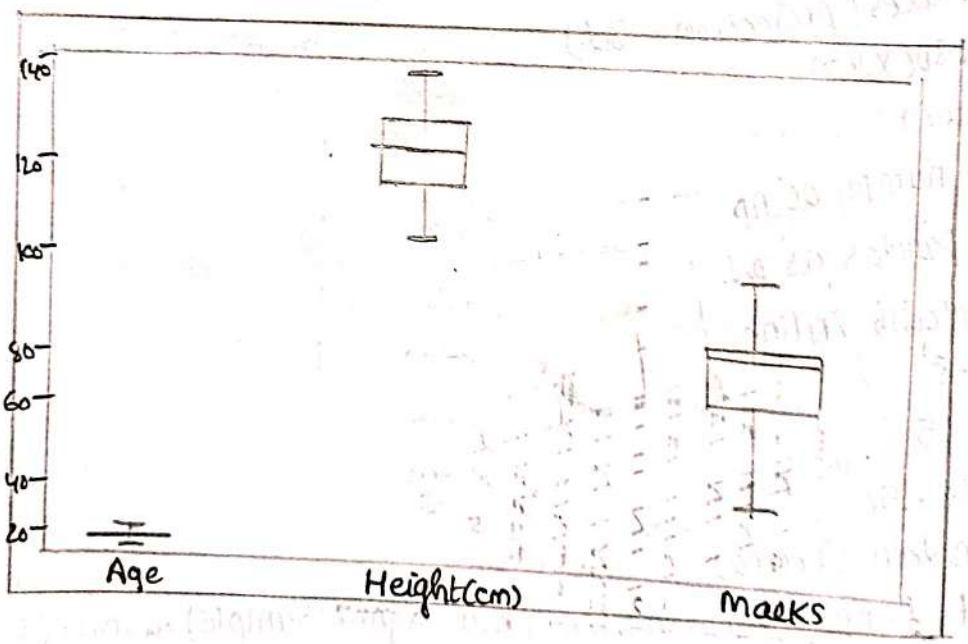
```
#Histogram
import pandas as pd
import matplotlib.pyplot as plt
#create 2D array of student Details
stdData=[['S1','M',13,123,46],
         ['S2','M',12,134,82],
         ['S3','F',14,114,77],
         ['S4','M',13,136,73],
         ['S5','F',13,107,56],
         ['S6','F',12,121,80],
         ['S7','M',14,113,76],
         ['S8','F',15,123,95],
         ['S9','F',14,112,78],
         ['S10','M',15,100,60]]
#creating the dataframe from the above data
df=pd.DataFrame(stdData,columns=['ID','Gender','Age','Height(cm
        'marks'])
df.hist()
plt.show()
#Box plot
df.plot.box()
plt.show()
#Scatter plot
import numpy as np
import matplotlib.pyplot as plt
```

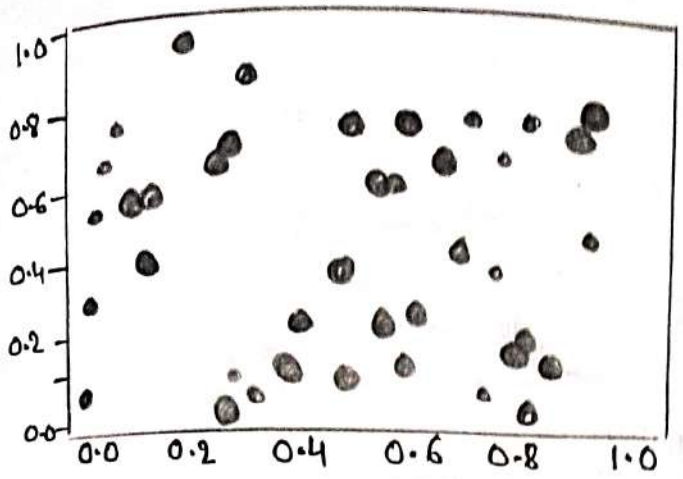
```
x=np.random.rand(50)
y=np.random.rand(50)
sizes=np.random.rand(50)
plt.scatter(x,y,s=sizes*500)
plt.show()

#2nd line plot
import matplotlib.pyplot as plt
x=[1,2,3,4,5,6,7,8,9,10]
y=[2,4,6,8,10,12,14,16,18,20]
plt.plot(x,y)

#3d line plot
import matplotlib.pyplot as plt
x=[1,2,3,4,5,6,7,8,9,10]
y=[2,4,6,8,10,12,14,16,18,20]
z=[1,4,9,16,25,36,29,64,81,100]
fig=plt.figure()
ax=plt.axes(projection='3d')
ax.plot3D(x,y,z)
plt.show()
import numpy as np
import pandas as pd
%matplotlib inline
mu=168
sigma=5
sample=250
np.random.seed(0)
height_f=np.random.normal(mu,sigma,sample).astype(int)
mu=176
sigma=6
sample=250
np.random.seed(1)
height_m=np.random.normal(mu,sigma,sample).astype(int)
gym=pd.DataFrame({'height_f':height_f,'height_m':height_m})
```

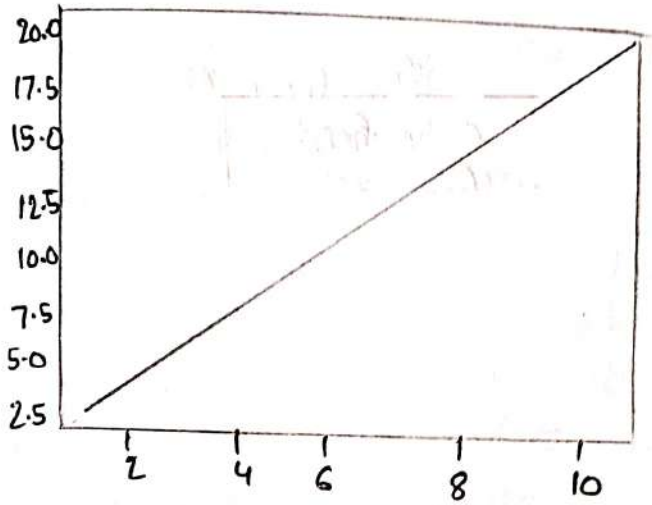
```
gym
gym.plot()
gym.groupby('height_m').count()
gym.groupby('height_m').count().plot()
gym.groupby('height_m').count().plot.bar()
gym.groupby('height_m').count().plot(kind='bar')
```

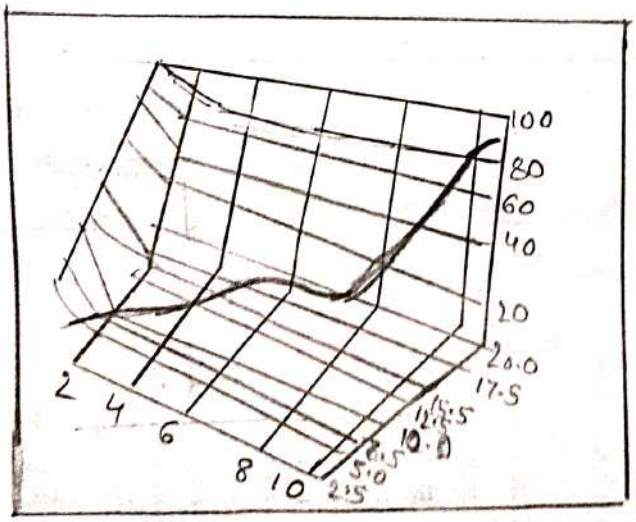
Output:

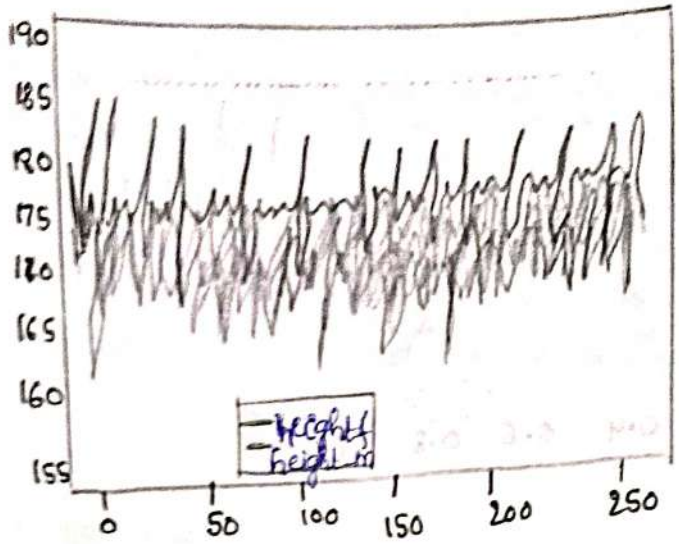


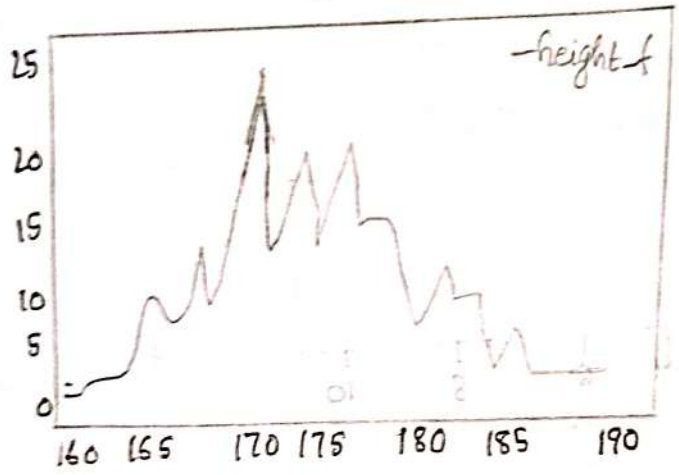


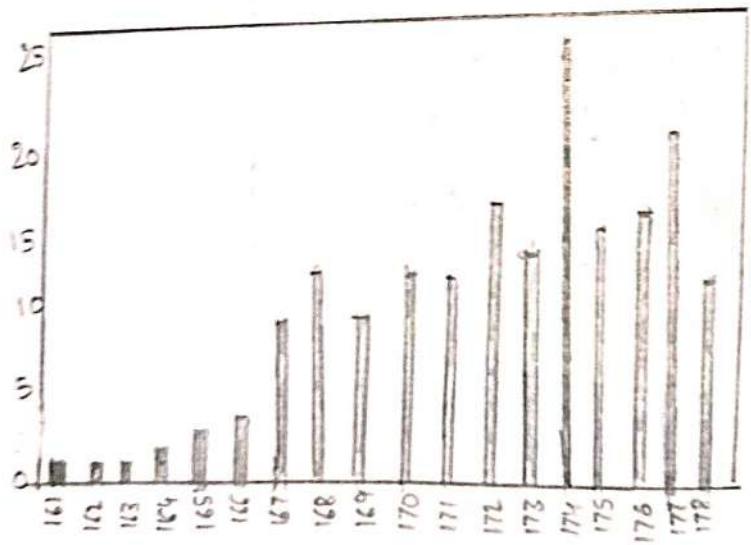

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
2
4
6
8
10


100
80
60
40
20
20.0
17.5
2
4
6
8
10


190
185
180
175
170
165
160
155
0
50
100
150
200
250
weight
height_m


25
20
15
10
5
0
160
165
170
175
180
185
190
height_f


25
20
15
10
5
0
161
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178

Experiment-1

Aim: Creation of a Data warehouse.

→ Build Data warehouse / Data mart.
→ Design multi-dimensional data models namely star, snowflake, and fact constellation schemas for any one enterprise.
→ write ETL scripts and implement using data warehouse tools.
→ perform various OLAP operations such slice, dice, roll up, drill up and pilot.

Input: Multi-dimensional data models.

Analysis: star schema, snowflake schema and fact constellation schema, OLAP operations.

Objective:

The goal of a data warehouse is to create a store of historical data that can be retrieved and analyzed to provide useful insight into the organizations operations. A data warehouse is a vital component of business intelligence.

Multi dimensional data models:

A multi dimensional model views data in the form of a data cube. A data cube enables data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.

Star schemas:

Star schema is the fundamental schema among the data mart schema and it is simplest. This schema is widely used to develop a build a data warehouse and dimensional data marts.

Snowflake schema:

The snow flake schema is a variant of the star schema. Here the centralized fact table is connected to multiple dimensions. In the snowflake schema, dimensions are present in a normalized form in multiple related tables.

Fact Constellation schema:

Fact constellation is a schema for representing multidimensional model. It is a collection of multiple fact tables having some common dimension tables.

OLAP:

OLAP stands for online analytical processing, OLAP is a class of software technology which authorizes analysts, managers and executives to gain insight into information through fast, consistent, interactive access in a wide variety of possible views of data.

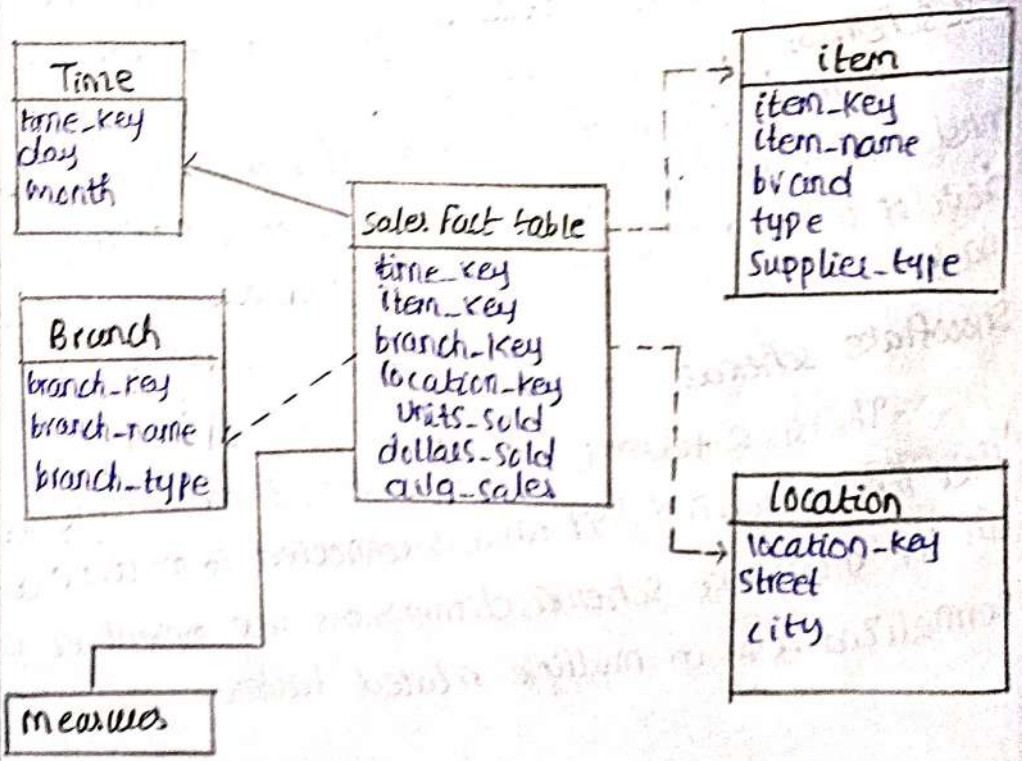
Slice: It selects a single dimension from the OLAP cube which results in a new sub-cube creation.

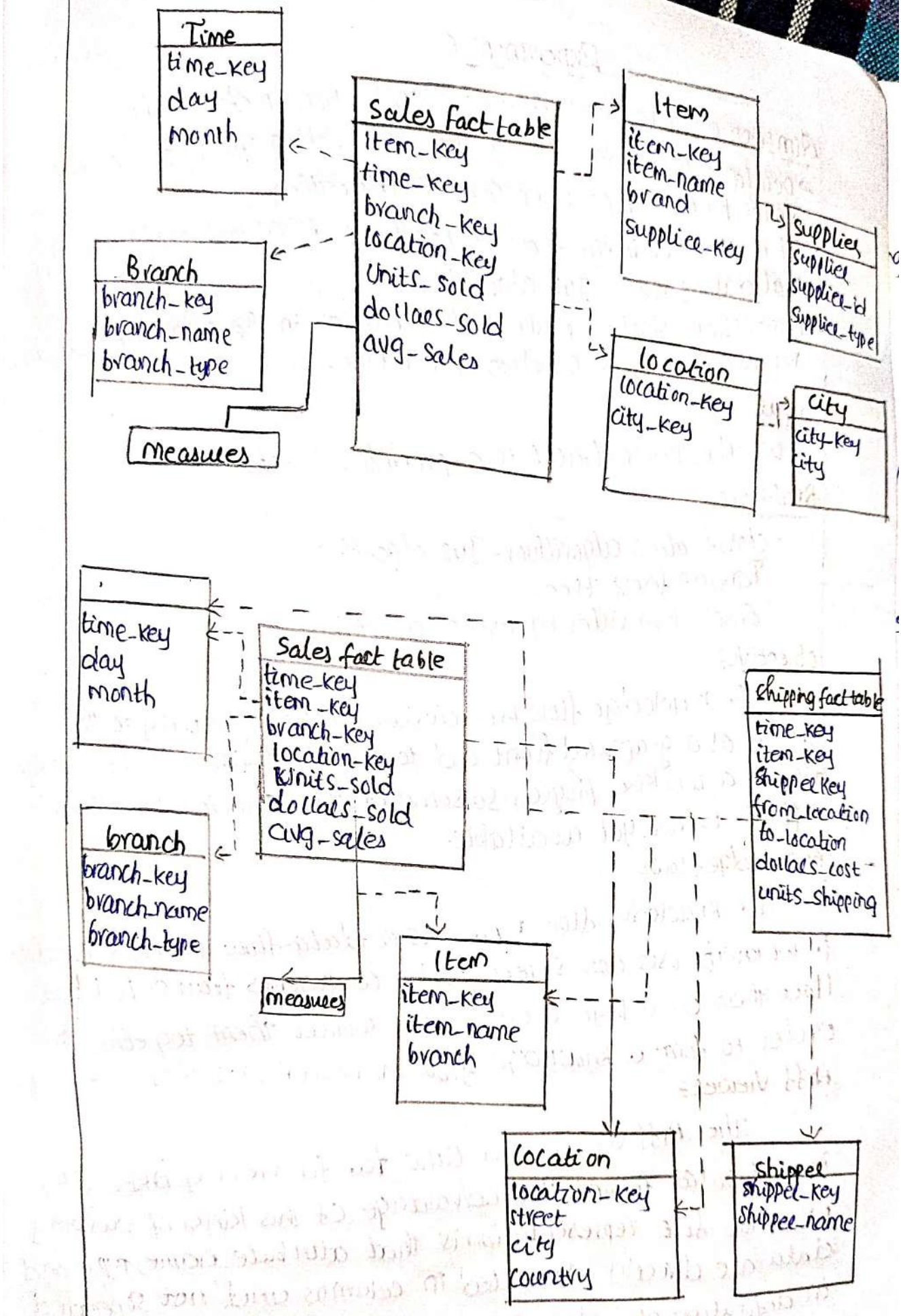
Dice: It selects a sub-cube from the OLAP cube by selecting two or more dimensions.

Roll up: It is just opposite of the drill down operation. It performs aggregation on the OLAP cube.

Drill down: In drill down operation, the less detailed data is converted into highly detailed data.

Pivot: It is also known as rotation operation as it rotates the current view to get a new view of the representation. In the sub-cube obtained at the slice operation.




Time
time_key
day
month
Sales fact table
item_key
time_key
branch_key
location_key
units_sold
dollars_sold
avg_sales
Item
item_key
item_name
brand
supplier_key
Supplier
supplier
supplier_id
supplier_type
Branch
branch_key
branch_name
branch_type
location
location_key
city_key
city
city_key
city
measures
time_key
day
month
Sales fact table
time_key
item_key
branch_key
location_key
units_sold
dollars_sold
avg_sales
Shipping fact table
time_key
item_key
shipper key
from_location
to_location
dollars_cost
units_shipping
branch
branch_key
branch_name
branch_type
measures
Item
item_key
item_name
branch
location
location_key
street
city
country
Shipper
shipper_key
shipper_name

Experiment-6

Aim: Demonstrate Knowledge flow application on data sets.

→ Develop a Knowledge flow layout for finding strong association rules by using Apriori, FP Growth algorithms.

→ Set up the Knowledge flow to load an ARFF and perform a cross Validating using J48 algorithm.

→ Demonstrate plotting multiple ROC curves in the same plot windo by using j48 and Random forest tree.

Input:

Weather.nominal.arff is a predefined data set.

Analysis:

Classification algorithms - J48 algorithm,
Random forest tree.
Association rules by using apriori

Objective:

The knowledge flow provides an alternative way to the explorer as a graphical front end to WEKA's algorithm knowledg flow is a working progress. So, some of the functionality from explorer is not yet available.

Knowledge flow:

The knowledge flow represents a data-flow inspired interfa to WEKA. The user can select WEKA components from a tool bar Place them on a layout canvas and connect them together in order to form a knowledge flow for processing and analyzing d

Arff Viewer:

The Arff Viewer is a little tool for viewing ARFF files in a tabular format. The advantage of this kind of display over the file representation is that attribute name, type and data are directly associated in columns and not seperated in definition and data part.

Association Rule: Association rules are mined out after frequent itemsets in a big dataset are found. These datasets are found out using mining algorithms such Apriori and FP Growth.

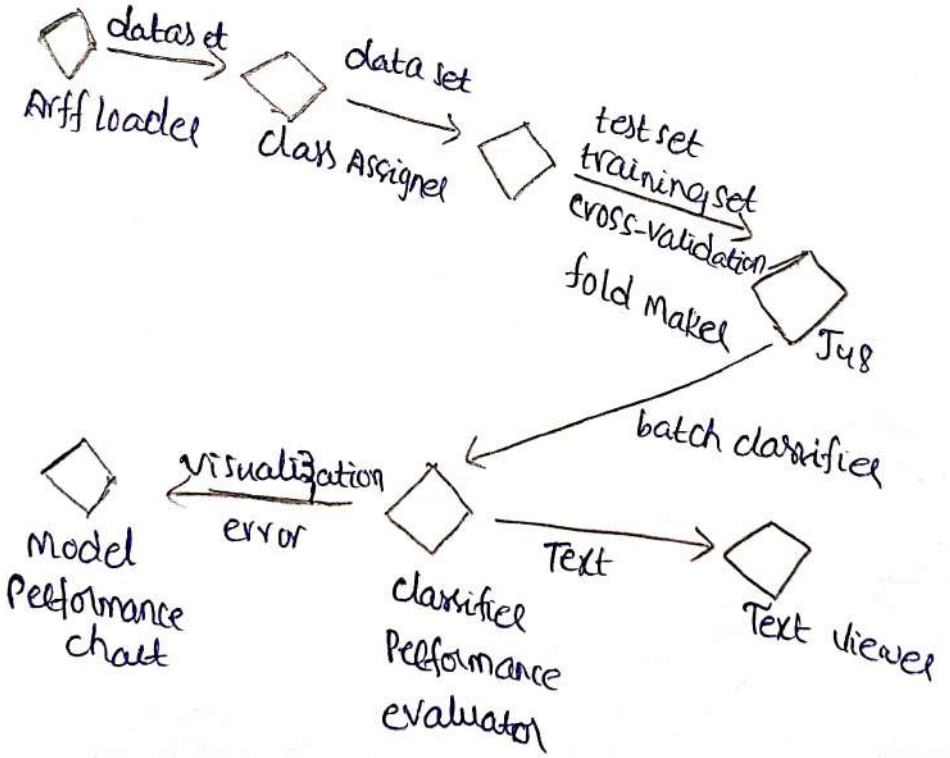
Apriori algorithm: Apriori is simple algorithm, which applied for mining of repeated the patterns from the transaction data set to find frequent itemsets and association between various item sets.

FP Growth algorithm: FP Growth is an algorithm for finding patterns in data and its much more efficient than its preprocessor. WEKA implementation of FP Growth requires data be supplied in binary format.

Roc curve: The WEKA explorer enables you to plot the Roc curve for a certain class label of dataset, run a classifier on a dataset, right-click in the result list on the result you want to display the curve for select visualize threshold curve and choose.

Cross-Validation: WEKA does stratified cross-validation by default and with 10-fold cross validation. WEKA invokes the learning algorithm.

Output:



Exercise-8

Aim:- Write a java progeam to prepare a simulated dataset with unique instances.

Program:-

```
import java.sql.*;
class DataBase{
Public static void main (String args[]){
try{
        Class.forName("Oracle.jdbc.driver.OracleDriver");
        Connection con=DriverManager.getConnection ("jdbc:oracle:
        thin:@localhost:1521:xe", "system", "8639");
        Statement stmt=con.createStatement();
        ResultSet rs=stmt.executeQuery("select * from employee
        while(rs.next())
        {
          System.out.println(rs.getInt(1)+" "+rs.getString(2)+"
         rs.getInt(3)+" "+rs.getInt(4));
          con.close();
        }
        Catch(Exception e){
              System.out.println(e);
    }
}
```

Output:-

1	Venkat	672	0
2	Nirmala	671	1
3	Pradeep	669	1
4	Srinivas	669	1
5	Krishna	668	2
6	Deepa	668	3
7	Keerthi	668	4
8	Aravind	671	1
9	Srikanth	668	8
10	Suresh	667	3
11	Rahul	667	9
12	Kunal	667	4

Experiment-13

Aim: Write a program to cluster your choice of data using simple K-means algorithm using jdk.

Objective:

K-means clustering is an unsupervised learning algorithm, which groups the unlabeled dataset as input divides the dataset into K-number of clusters, and repeats the process until it does not find the best clusters. The value of K should be predetermined in this algorithm.

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.

Program:

```
import java.util.*;
class main{
    Public static void main(String args[]){
      int dataset[][]={ {2,1},{5,2},{2,2},{4,1},{4,3},
                        {7,5},{3,6},{5,7},{1,4},{4,1}};
      int i,j,K=2
      int part1[][]= new int [10][2];
      int part2[][]=new int [10][2];
      float mean1[][]=new float[1][2];
      float mean2[][]=new float[1][2];
      float temp1[][]=new float[1][2];
      float temp2[][] = new float[1][2];
      int sum11=0;
      int sum12=0, sum21=0, sum22=0;
```

```
double dist1,dist2;
int i1=0,i2=0,itr=0;
System.out.println("Dataset:");
for(i=0;i<10;i++)
{
    System.out.println(dataset[i][0]+" "+dataset[i][1]);
}
System.out.println("\nNumber of partitions:"+k);
mean1[0][0]=2;
mean1[0][1]=2;
mean2[0][0]=5;
mean2[0][1]=7;
while(!Arrays.deepEquals(mean1,temp1)||!Arrays.deepEquals
    (mean2,temp2))
{
    for(i=0;i<10;i++)
    {
        Part1[i][0]=0;
        Part1[i][1]=0;
        Part2[i][0]=0;
        Part2[i][1]=0;
    }
    i1=0;i2=0;
    for(i=0;i<10;i++)
    {
        dist1=Math.sqrt(Math.pow(dataset[i][0]-mean1[0][0],
        2)+Math.pow(dataset[i][1]-mean1[0][1],2);
        dist1=Math.sqrt(Math.pow(dataset[i][0]-mean2[0][0],
        +Math.pow(dataset[i][1]-mean2[0][1],2));
        Part1[i1][0]=dataset[i][0];
        Part1[i1][1]=dataset[i][1];
        i1++;
    }
    else
    {
```

```
        Part2[i2][0]=dataset[i][0];
        Part2[i2][i]=dataset[i][i];
        i2++;
      }
    }
    temp1[0][0]=mean1[0][0];
    temp1[0][i]=mean2[0][i];
    temp2[0][0]=mean2[0][0];
    temp2[0][i]=mean2[0][i];
    Sum11=0;sum12=0; Sum21=0;Sum22=0;
    for(i=0;i<i1;i++)
    {
        Sum11+=part1[i][0];
        Sum12+=part1[i][i];
    }
    for(i=0;i<i2;i++)
    {
        Sum21+=part2[i][0];
        Sum22+=part2[i][i];
    }
    mean1[0][0]=(float)sum11/i1;
    mean1[0][i]=(float)sum12/i1;
    mean2[0][0]=(float)sum21/i2;
    mean2[0][i]=(float)sum22/i2;
    itr++;
}
System.out.println("\nFinal partition:");
System.out.println("part1:");
for(i=0;i<i1;i++)
{
  System.out.println(part1[i][0]+" "+part1[i][i]);
}
System.out.println("\nPart2:");
for(i=0;i<i2;i++)
{
```

```
        System.out.println(part1[i][0]+" "+part2[i][j]);
    }
    System.out.println("\n Final mean:");
    System.out.println("mean1:"+mean1[0][0]+" "+mean1[0][1]);
    System.out.println("mean2:"+mean2[0][0]+" "+mean2[0][1]);
    System.out.println("\n Total Iteration:"+itr);
}
```

Output:

```
Dataset:
2 1
5 2
2 2
4 1
4 3
7 5
3 6
5 7
1 4
4 1
Number of partitions:2
Final Partition: Part-1:2 1 5 2 2 2 4 4 3 1 4 1 | Part-2:7 5 3 6 5 7 1 4 4 1
Final mean:
Mean1: 3.142857 2.0
Mean2: 5.0 6.0
Total Iteration: 2
```