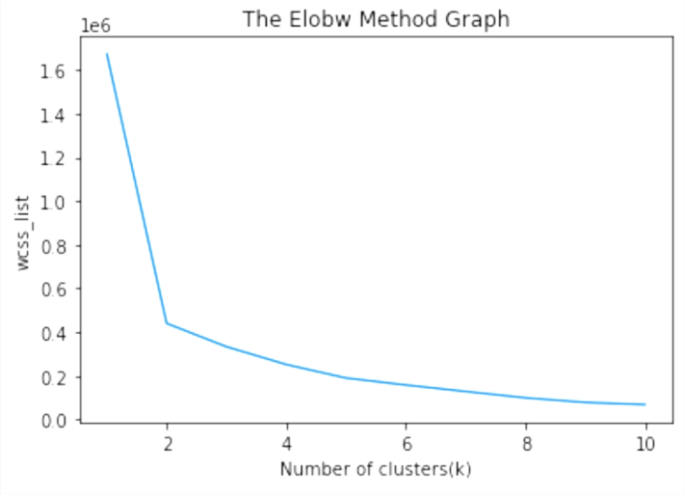
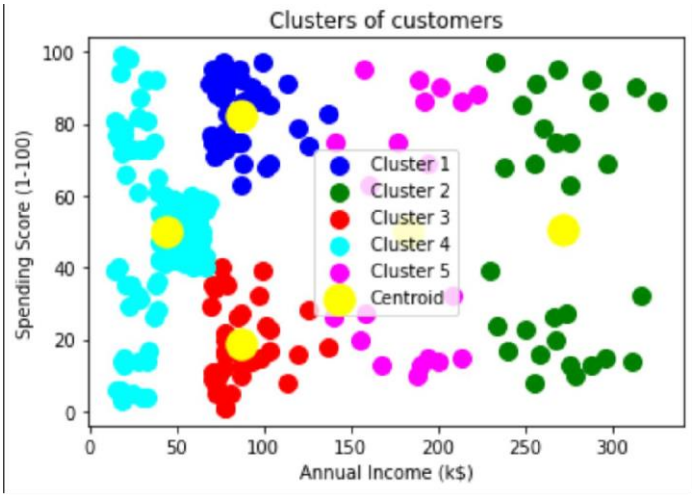
OUTPUT:-


The Elobw Method Graph
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
wcss_list
2
4
6
8
10
Number of clusters(k)


Clusters of customers
100
80
60
40
20
0
Spending Score (1-100)
0
50
100
150
200
250
300
Annual Income (k$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroid

EXPERIMENT-10

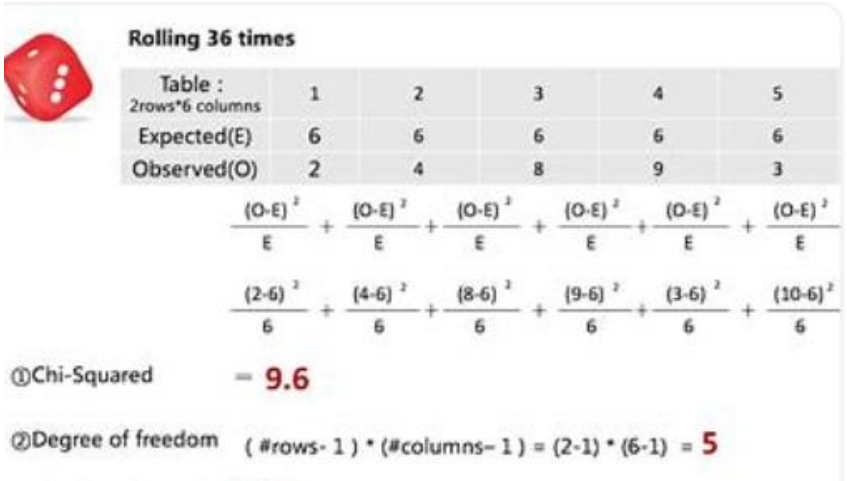
AIM:- Write a program to calculate chi-square value using Python. Report your observation.

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

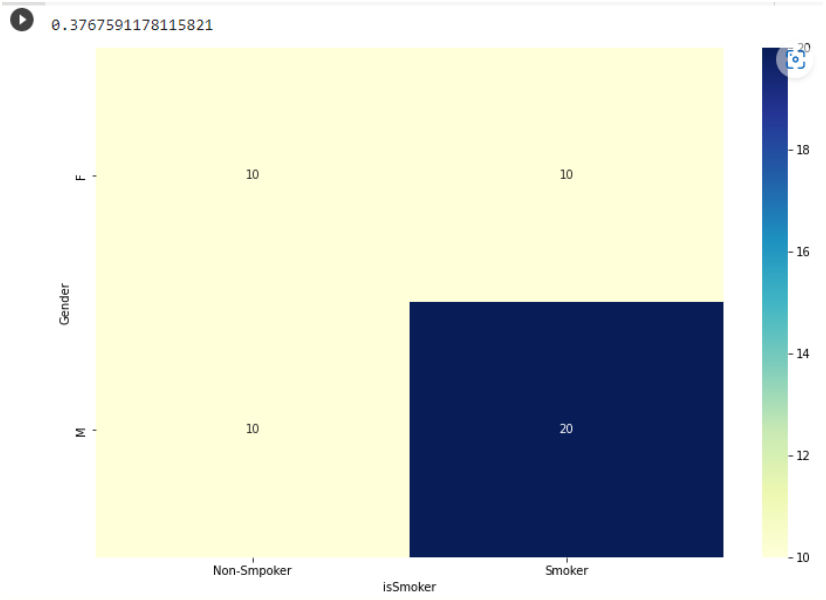


PROGRAM:

```
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")
# Chi-square test of independence.
c, p, dof, expected = chi2_contingency(contigency)
# Print the p-value
print(p)
```

OUTPUT:



EXPERIMENT-11

AIM: write a program of naïve Bayesian classification using python programming language.

OBJECTIVES:

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes theorem

Bayes' Theorem:

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

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

Where,

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:

```
# load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# splitting X and y into training and testing sets
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)

# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
y_pred = gnb.predict(X_test)

# comparing actual response values (y_test) with predicted response val
ues (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_sc
ore(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:

Dissimilarity matrix: The dissimilarity matrix (also called **distance matrix**) describes pairwise distinction between M objects. It is a square symmetrical MxM matrix with the (ij)th element equal to the value of a chosen measure of distinction between the (i)th and the (j)th object.

Program:

```
//Dissimilarity matrix

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_dis
tances
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 perform apriori algorithm.

Objectives:

Apriori algorithm refers to an algorithm that is used in mining frequent products sets and relevant association rules. Generally, the apriori algorithm operates on a database containing a huge number of transactions. For example, the items customers but at a Big Bazar.

Apriori algorithm helps the customers to buy their products with ease and increases the sales performance of the particular store.

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("Number of transactions: ");

                int numberOfTransactions = Integer.parseInt(terminal.nextLine());

                System.out.println("Enter transactions separated by new line and items separated by spaces:");

                ArrayList<ArrayList<String>> transactions = new ArrayList<ArrayList<String>>();

                ArrayList<ArrayList<String>> _transactions = new ArrayList<ArrayList<String>>();

                ArrayList<ArrayList<String>> prevItemSetsWithMinSupportCount = 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("Minumum support count: ");
        int minSupportCount = Integer.parseInt(terminal.nextLine());
        // Get all items
        ArrayList<String> items = getUniqueItems(transactions);
        int x = 0; // x is the number of elements in the item-sets to consider
        while (true) {
                // Consider one more item than the last iteration
                x++;
                // List of support count of each itemset
                ArrayList<Integer> supportCountList = new ArrayList<Integer>();
               // Get permuted itemsets with items. There will be x elements in each itemset.
                ArrayList<ArrayList<String>> itemSets = getItemSets(items, x);
                // Calculate each itemset's support count
                for (ArrayList<String> itemSet : itemSets) {
                int count = 0;
                        for (ArrayList<String> transaction : transactions) {
                                if (existsInTransaction(itemSet, transaction)) count++;
                        }
                        supportCountList.add(count);
                }
```

```
        ArrayList<ArrayList<String>>itemSetsWithMinSupportCount=getItemSetsWithMinSupportCount(itemSets,
supportCountList, minSupportCount);

                        // No itemSetsWithMinSupportCount exist

                        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 = itemSetsWithMinSupportCount;

                }

        }

        // Returns the list of unqiue items from a list of transactions

        private static ArrayList<String> getUniqueItems (ArrayList<ArrayList<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;

        }

        // Returns a list of itemsets, where each itemset has x number of items

        private static ArrayList<ArrayList<String>> getItemSets (ArrayList<String> items, int number) {

                if (number == 1) {

                        // Return ArrayList of (ArrayList with one item)
```

```
                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>> toReturn = new ArrayList<ArrayList<String>>();
                for (int i = 0; i < size; i++) {
                        // Copy items to _items
                        ArrayList<String> _items = new ArrayList<String>();
                        for (String item : items) {
                                _items.add(item);
                        }
                        // Get item at i-th position
                        String thisItem = items.get(i);
                        // Remove items upto i, inclusive
                        for (int j = 0; j <= i; j++) {
                                _items.remove(0);
                        }
                        // Get permutations of the remaining items
                        ArrayList<ArrayList<String>> permutationsBelow = getItemSets(_items, number - 1);

                        // Add thisItem to each permutation and add the permutation to toReturn
                        for (ArrayList<String> aList : permutationsBelow) {
```

```
                                        aList.add(thisItem);
                                        Collections.sort(aList);
                                        toReturn.add(aList);
                                    }
                                }
                                return toReturn;
                            }
                        }
                        // Check if all items exist in a transaction
                        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>> getItemSetsWithMinSupportCount (
                            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 separated by new line and items separated by spaces:
pizza burgar dosa
pizza burgar
pizza onion tea
burgar onion tea
Minumum support count: 2
The itemset(s) that are the most frequent itemset(s): [[burgar, pizza], [onion, tea]]
```

EXPERIMENT-16

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

Program:

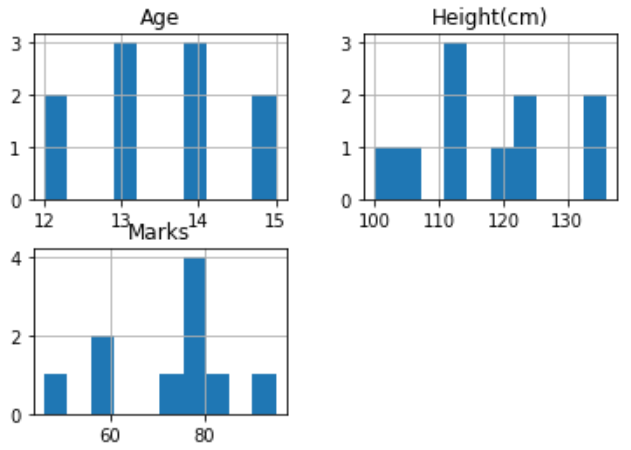
```
# Histogram

# import required modules

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()# create histogram for the numeric data(Age, Height,Marks)

plt.show() #displaying the plot

#Box plot
```

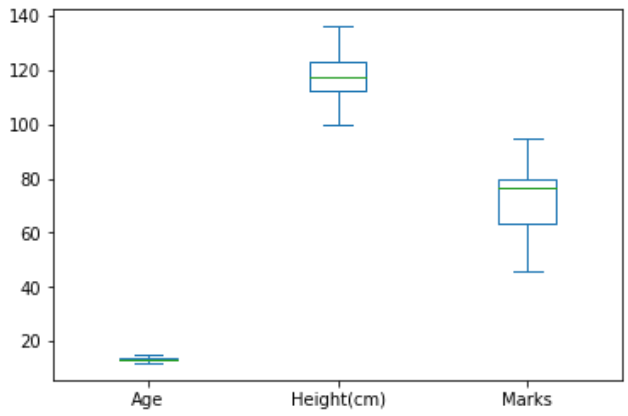
```
df.plot.box() # Plotting box plot for each numeric column of the student dataframe df

plt.show()

#Box plot

#Scatter plot

import numpy as np

import matplotlib.pyplot as plt

#creating 3 variables holding an array of 50 random values in the range 0 to 1

x=np.random.rand(50)  #represents the x axes coordinates

y=np.random.rand(50) #represents the y axes coordinates

sizes=np.random.rand(50) #represents the values and thus the size of the bubbles

plt.scatter(x,y,s=sizes*500)

plt.show()


#2d line plot

import matplotlib.pyplot as plt

#creating the three values of the three axes

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

#creating the three values of the three axes

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

y = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

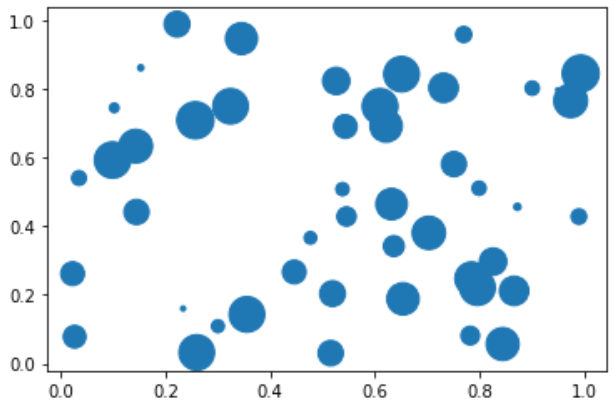
```
z = [1, 4, 9, 16, 25, 36, 29, 64, 81, 100]

# Creating the figure object

fig = plt.figure()

#creating the 3D coordinate axes

ax = plt.axes(projection = '3d')

ax.plot3D(x,y,z)

plt.show()

import numpy as np

import pandas as pd

%matplotlib inline

mu = 168 #mean

sigma = 5 #stddev

sample = 250

np.random.seed(0)

height_f = np.random.normal(mu, sigma, sample).astype(int)

mu = 176 #mean

sigma = 6 #stddev

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()
```

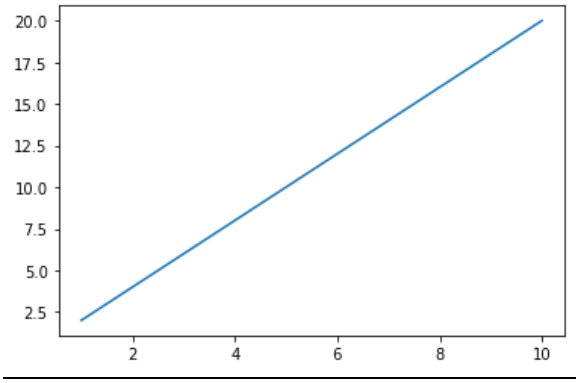
```
gym.groupby('height_m').count().plot()

gym.groupby('height_m').count().plot.bar()

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

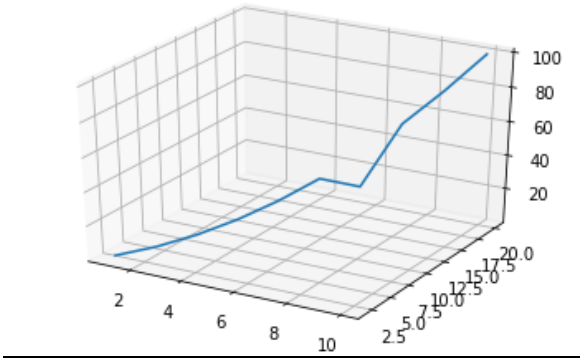
Output:-

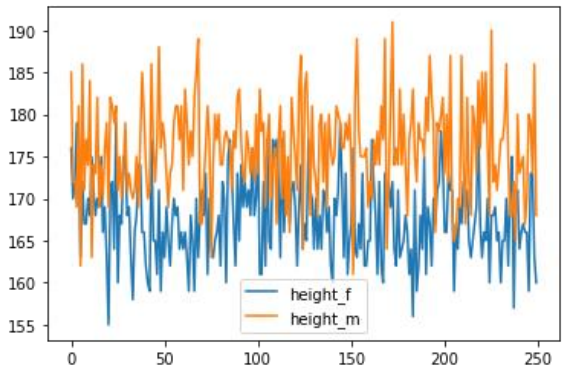


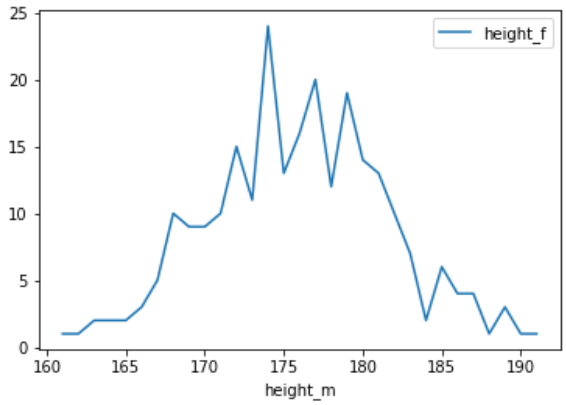


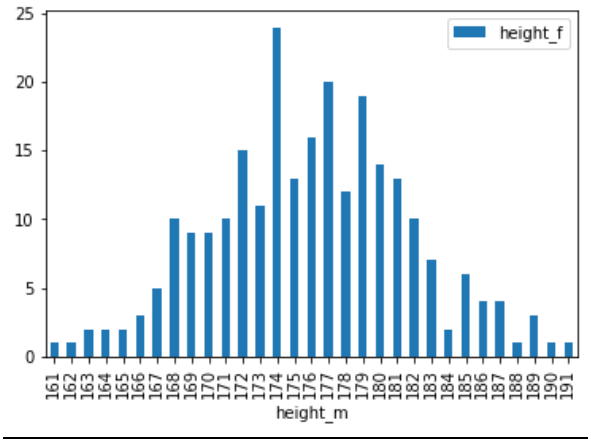


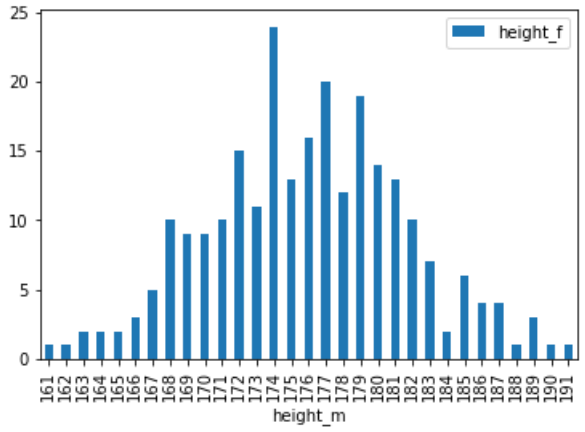

20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
2
4
6
8
10


100
80
60
40
20
2
4
6
8
10
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


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


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


height_f
height_m


height_f
height_m

EXPERIMENT-8

Aim: Write a java program to prepare a simulated data set with unique instances.

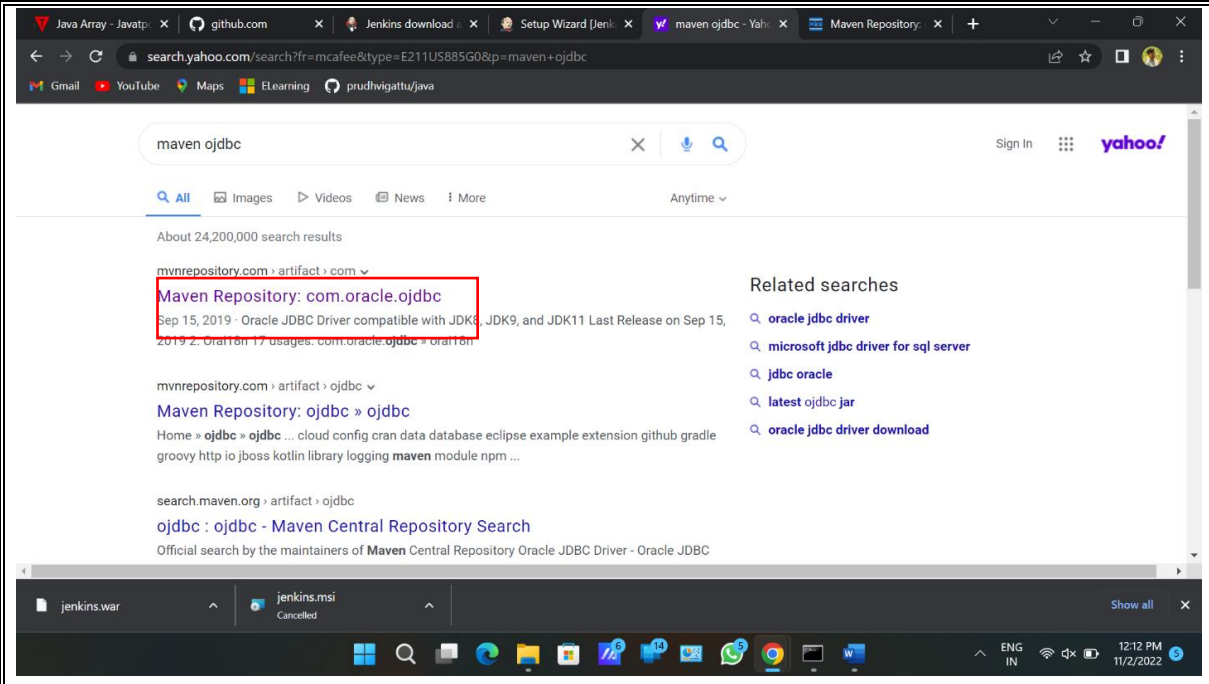
Program:

```
import java.sql.*;
class DataBase{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","8639");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
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));

//step5 close the connection object
con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

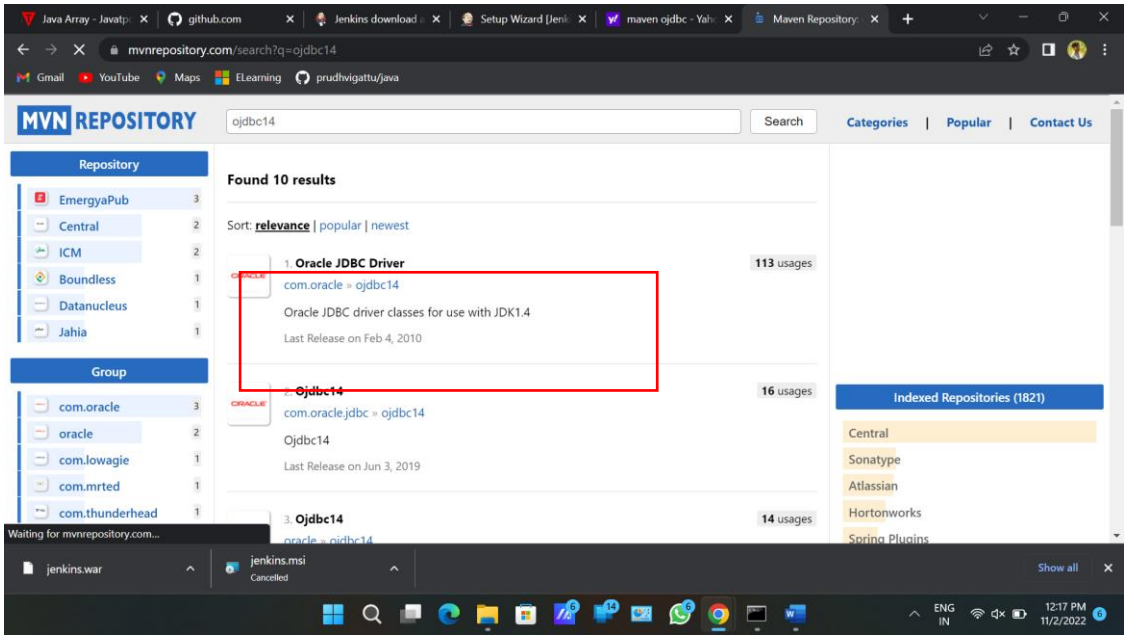
STEPS:

1)install the eclipse ide

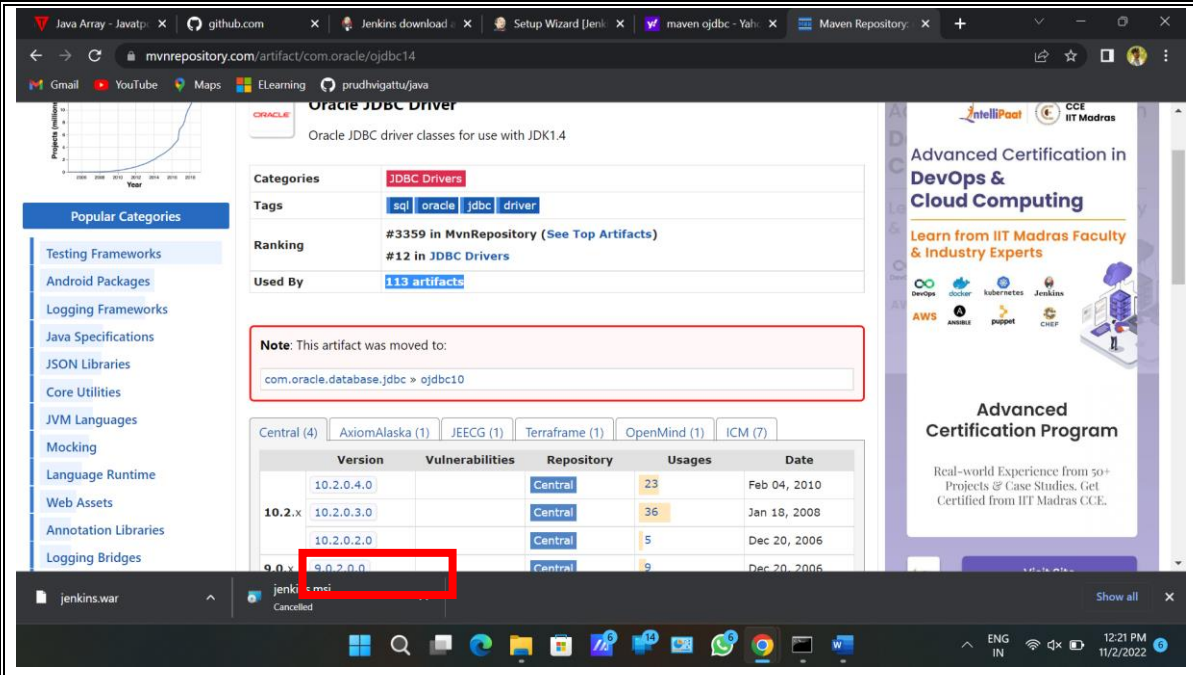
2)go to chrome and search for maven repository.



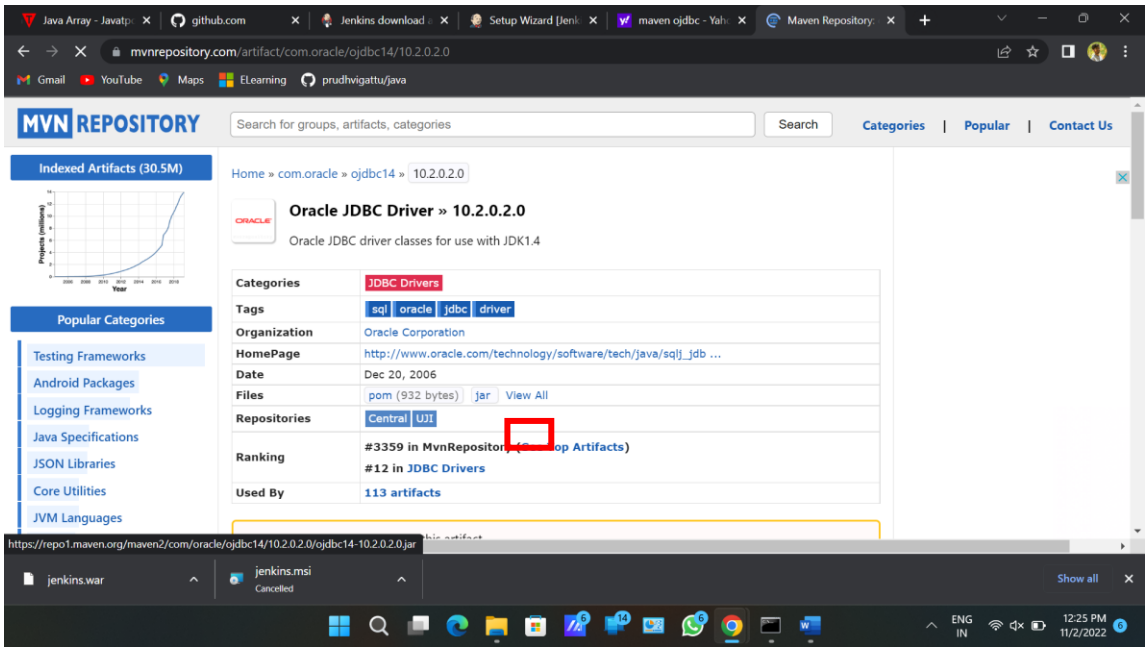
3)In the search bar,search for ojdbc14 and click the below selected oracle jdbc driver.



4)click on the version 10.2.0.2.0 as shown below.

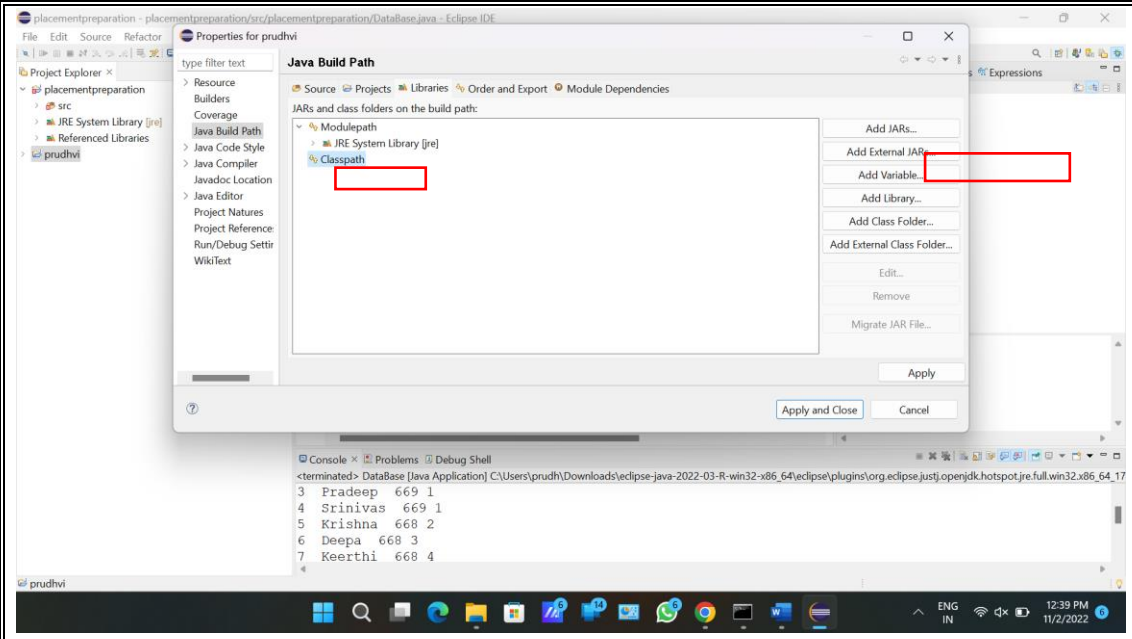


5)In the Files section download the Jar file.



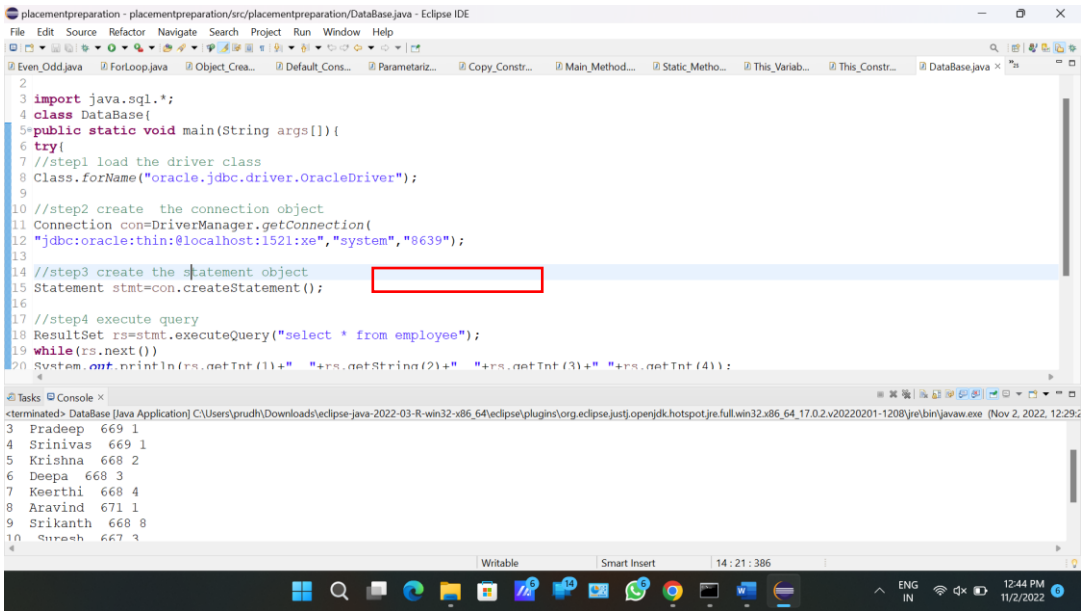
6)open the eclipse IDE and click on project explorer and select any project in which you want to do database connectivity.

Right click on the selected project and click on build path and select configure build path. And it will navigate to the following slide. Click on class path and select the add external jars.



Load the downloaded ojdbc driver and click on apply.

7)Now connect your oracle database with the eclipse ide with providing the valid credentials of your oracle database.



Replace “system” with your oracle database username and password with your oracle password.

And run the program you will get the below output.

OUTPUT:


<terminated> DataBase [Java Application] C:\Users\prudh\Downloads\eclipse-java-2022-03-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Nov 2, 2022, 10:5
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 Kumar 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 into different clusters. The algorithm takes 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], temp2[][] = new float[1][2];
                int sum11 = 0, sum12 = 0, sum21 = 0, sum22 = 0;
                double dist1, dist2;
                int i1 = 0, i2 = 0, itr = 0;

                // Printing the dataset
                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);

                // Assuming (2,2) and (5,7) are random means
                mean1[0][0] = 2;
                mean1[0][1] = 2;
                mean2[0][0] = 5;
                mean2[0][1] = 7;

                // Loop till the new mean and previous mean are same
        while(!Arrays.deepEquals(mean1, temp1) || !Arrays.deepEquals(mean2, temp2))
{

                        //Empting the partitions
                        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;

//Finding distance between mean and data point and store the data point in the corresponding
//partition
                        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));
dist2 = Math.sqrt(Math.pow(dataset[i][0] - mean2[0][0],2) + Math.pow(dataset[i][1] -
mean2[0][1],2));

                                if(dist1 < dist2) {
```

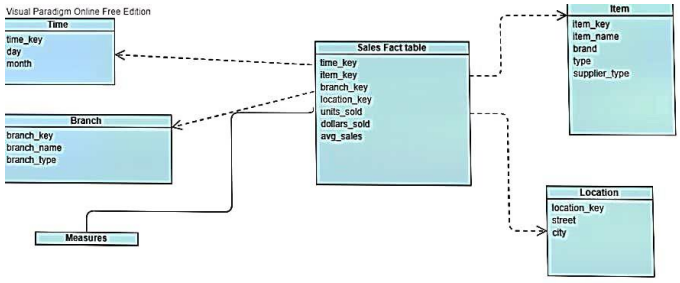
```
                        part1[i1][0] = dataset[i][0];
                        part1[i1][1] = dataset[i][1];

                        i1++;
                }
                else {
                        part2[i2][0] = dataset[i][0];
                        part2[i2][1] = dataset[i][1];

                        i2++;
                }
        }

        //Storing the previous mean
        temp1[0][0] = mean1[0][0];
        temp1[0][1] = mean1[0][1];
        temp2[0][0] = mean2[0][0];
        temp2[0][1] = mean2[0][1];

        //Finding new mean for new partitions
        sum11 = 0; sum12 = 0; sum21 = 0; sum22 = 0;

        for(i=0;i<i1;i++) {
                sum11 += part1[i][0];
                sum12 += part1[i][1];
        }
        for(i=0;i<i2;i++) {
                sum21 += part2[i][0];
                sum22 += part2[i][1];
        }
        mean1[0][0] = (float)sum11/i1;
        mean1[0][1] = (float)sum12/i1;
        mean2[0][0] = (float)sum21/i2;
        mean2[0][1] = (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][1]);
```

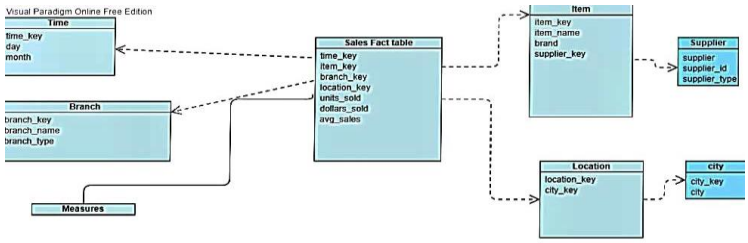
```
                }
                System.out.println("\nPart2:");
                for(i=0;i<i2;i++) {
                        System.out.println(part2[i][0]+" "+part2[i][1]);
                }
                System.out.println("\nFinal Mean: ");
                System.out.println("Mean1 : "+mean1[0][0]+" "+mean1[0][1]);
                System.out.println("Mean2 : "+mean2[0][0]+" "+mean2[0][1]);
                System.out.println("\nTotal Iteration: "+itr);
        }
}
```

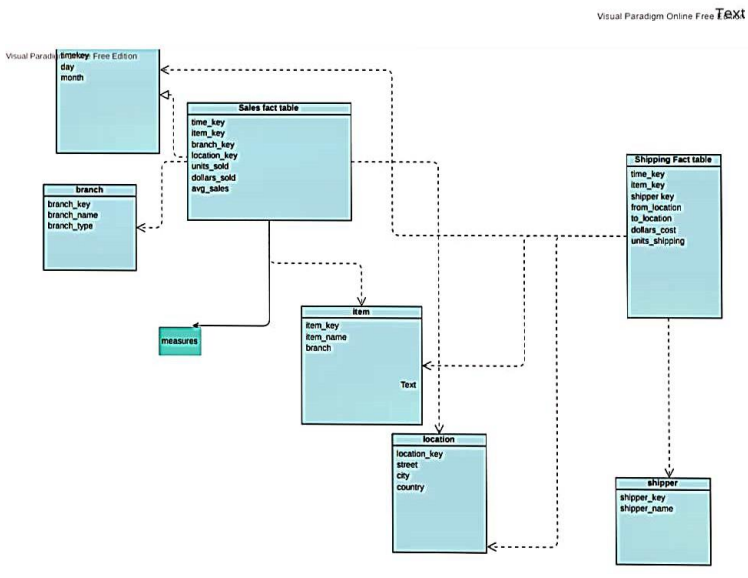
OUTPUT:

```
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:
Part1:
2 1
5 2
2 2
4 1
4 3
1 4
4 1

Part2:
7 5
3 6
5 7

Final Mean:
Mean1 : 3.142857 2.0
Mean2 : 5.0 6.0

Total Iteration: 2
```

OUTPUT:

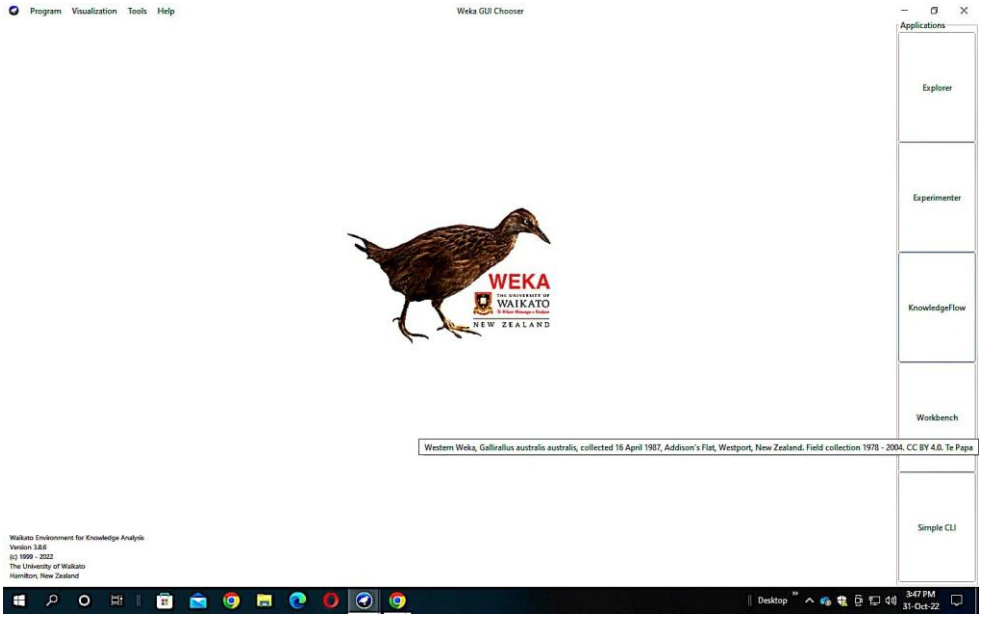

Visual Paradigm Online Free Edition
Time
time_key
day
month
Sales Fact table
time_key
item_key
branch_key
location_key
units_sold
dollars_sold
avg_sales
Item
item_key
item_name
brand
type
supplier_type
Branch
branch_key
branch_name
branch_type
Location
location_key
street
city
Measures


Visual Paradigm Online Free Edition
Time
time_key
day
month
Sales Fact table
time_key
item_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

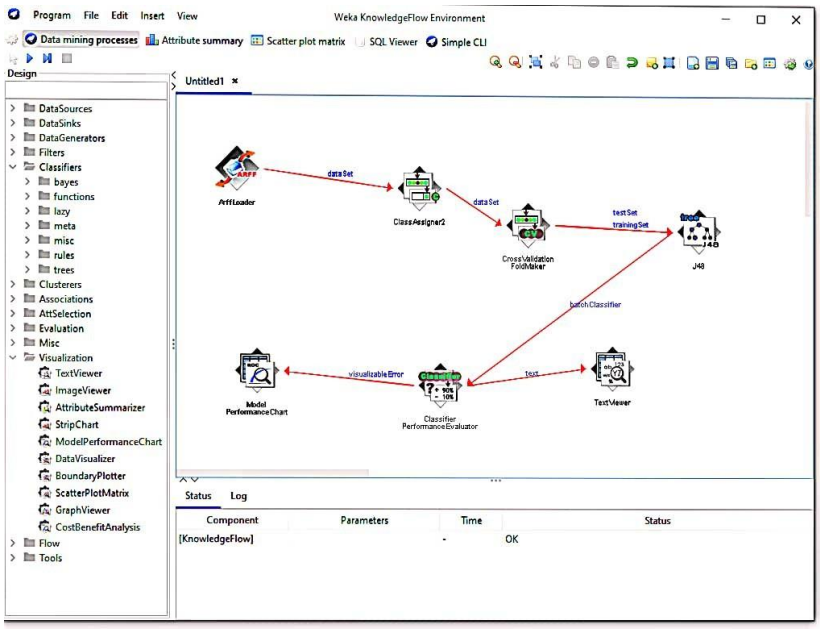

Visual Paradigm Online Free Edition
time_key
day
month
Sales fact table
time_key
item_key
branch_key
location_key
units_sold
dollars_sold
avg_sales
Shipping Fact table
time_key
item_key
shipper key
from_location
to_location
dollars_cost
units_shipping
branch
branch_key
branch_name
branch_type
measures
item
item_key
item_name
branch
location
location_key
street
city
country
shipper
shipper_key
shipper_name

OUTPUT:

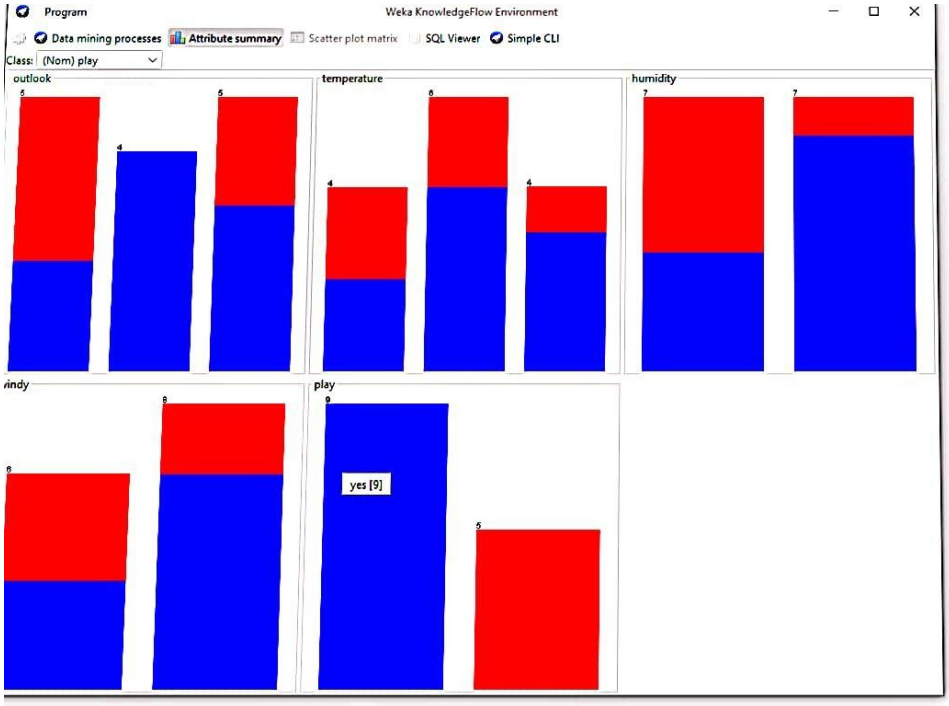
1)Open WEKA,click on knowledge flow



2)select arff loader,class assigner,cross validationfold,j48,text viewer,classifier performance evaluator,model performance chart and make the connections as follow.



3)Click on attribute summary to visualize the bar graph.



4)click on scatter plot matrix to visualize scatter plot.