Experiment-8:
8.Write a java program to prepare a simulated dataset with unique instances.

```
import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.Random;


class Car {

   private String carName;

   private String carDescription;

   private int carNo;


   // Constructor

   public Car(String carName, String carDescription, int carNo) {

      this.carName = carName;

      this.carDescription = carDescription;

      this.carNo = carNo;

   }


   @Override

   public String toString() {

      return "Car{" +

            "carName='" + carName + '\'' +
```

```
            ", carDescription='" + carDescription + '\'' +
            ", carNo=" + carNo +
            '}';
    }

    // Getters for uniqueness
    public String getCarName() {
        return carName;
    }

    public int getCarNo() {
        return carNo;
    }
}

public class SimulatedDataset {
    public static void main(String[] args) {
        Set<String> uniqueCarNames = new HashSet<>();
        List<Car> carList = new ArrayList<>();
        Random random = new Random();

        while (carList.size() < 10) { // Generate 10 unique cars
            String carName = "Car" + random.nextInt(100); // Random name
            String carDescription = "Description of " + carName; // Random description
```

```
        int carNo = random.nextInt(1000); // Random car number

        if (uniqueCarNames.add(carName)) { // Ensures uniqueness
            carList.add(new Car(carName, carDescription, carNo));
        }
    }

    // Print the dataset
    for (Car car : carList) {
        System.out.println(car);
    }
  }
}
```

OutPut:

```
java -cp /tmp/LeFwAKnD6m/SimulatedDataset
Car{carName='Car47', carDescription='Description of Car47', carNo=113}
Car{carName='Car65', carDescription='Description of Car65', carNo=73}
Car{carName='Car52', carDescription='Description of Car52', carNo=264}
Car{carName='Car34', carDescription='Description of Car34', carNo=831}
Car{carName='Car87', carDescription='Description of Car87', carNo=639}
Car{carName='Car43', carDescription='Description of Car43', carNo=138}
Car{carName='Car2', carDescription='Description of Car2', carNo=630}
Car{carName='Car62', carDescription='Description of Car62', carNo=473}
```

Car{carName='Car66', carDescription='Description of Car66', carNo=44}

Car{carName='Car67', carDescription='Description of Car67', carNo=530}

Experiment-9:

9. Write a program to generate frequent item set/association rule using apriori algorithm.

```
import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules


# Sample transaction data

dataset = [['Milk', 'Bread', 'Butter'],

        ['Bread', 'Diaper', 'Beer', 'Eggs'],

        ['Milk', 'Bread', 'Diaper', 'Beer'],

        ['Milk', 'Bread'],

        ['Bread', 'Butter', 'Diaper']]


# Convert the dataset into a DataFrame suitable for the apriori function

from mlxtend.preprocessing import TransactionEncoder


encoder = TransactionEncoder()

onehot = encoder.fit(dataset).transform(dataset)

df = pd.DataFrame(onehot, columns=encoder.columns_)


# Generate frequent itemsets with a minimum support of 0.4

frequent_itemsets = apriori(df, min_support=0.4, use_colnames=True)

print("Frequent Itemsets:")

print(frequent_itemsets)
```

```
# Generate association rules with a minimum confidence of 0.6
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)
print("\nAssociation Rules:")
print(rules)
```

Output:

```
Frequent Itemsets:
    support            itemsets
0       0.4              (Beer)
1       1.0             (Bread)
2       0.4            (Butter)
3       0.6            (Diaper)
4       0.6              (Milk)
5       0.4       (Beer, Bread)
6       0.4      (Diaper, Beer)
7       0.4    (Butter, Bread)
8       0.6    (Diaper, Bread)
9       0.6      (Milk, Bread)
10      0.4  (Diaper, Beer, Bread)

Association Rules:
      antecedents     consequents  antecedent support  consequent support  \
0          (Beer)         (Bread)                 0.4                 1.0
1        (Diaper)          (Beer)                 0.6                 0.4
```

2	(Beer)	(Diaper)	0.4	0.6
3	(Butter)	(Bread)	0.4	1.0
4	(Diaper)	(Bread)	0.6	1.0
5	(Bread)	(Diaper)	1.0	0.6
6	(Milk)	(Bread)	0.6	1.0
7	(Bread)	(Milk)	1.0	0.6
8	(Diaper, Beer)	(Bread)	0.4	1.0
9	(Diaper, Bread)	(Beer)	0.6	0.4
10	(Beer, Bread)	(Diaper)	0.4	0.6
11	(Diaper)	(Beer, Bread)	0.6	0.4
12	(Beer)	(Diaper, Bread)	0.4	0.6

	support	confidence	lift	leverage	conviction	zhangs_metric
0	0.4	1.000000	1.000000	0.00	inf	0.000000
1	0.4	0.666667	1.666667	0.16	1.8	1.000000
2	0.4	1.000000	1.666667	0.16	inf	0.666667
3	0.4	1.000000	1.000000	0.00	inf	0.000000
4	0.6	1.000000	1.000000	0.00	inf	0.000000
5	0.6	0.600000	1.000000	0.00	1.0	0.000000
6	0.6	1.000000	1.000000	0.00	inf	0.000000
7	0.6	0.600000	1.000000	0.00	1.0	0.000000
8	0.4	1.000000	1.000000	0.00	inf	0.000000
9	0.4	0.666667	1.666667	0.16	1.8	1.000000
10	0.4	1.000000	1.666667	0.16	inf	0.666667
11	0.4	0.666667	1.666667	0.16	1.8	1.000000

12	0.4	1.000000	1.666667	0.16	inf	0.666667

Experiment-10:

10: write a python program to calculate chi square value. Report u r observation.

```
pip install scipy

import numpy as np

import pandas as pd

from scipy.stats import chi2_contingency


# Sample data: A contingency table

data = np.array([[10, 20, 30],

                [6,  12, 18],

                [5,  10, 15]])


# Creating a DataFrame for better visualization

df = pd.DataFrame(data, columns=['Category 1', 'Category 2', 'Category 3'], index=['Group 1',
'Group 2', 'Group 3'])

print("Contingency Table:")

print(df)


# Performing the Chi-Square test

chi2_stat, p_value, dof, expected = chi2_contingency(data)


# Output the results

print("\nChi-Square Statistic:", chi2_stat)
```

```
print("P-Value:", p_value)

print("Degrees of Freedom:", dof)

print("Expected Frequencies:")

print(expected)


# Interpretation

alpha = 0.05

if p_value < alpha:

    print("\nReject the null hypothesis: There is a significant association between the variables.")

else:

    print("\nFail to reject the null hypothesis: There is no significant association between the variables.")
```

Output:

Contingency Table:

	Category 1	Category 2	Category 3
Group 1	10	20	30
Group 2	6	12	18
Group 3	5	10	15

Chi-Square Statistic: 0.0

P-Value: 1.0

Degrees of Freedom: 4

Expected Frequencies:

```
[[10. 20. 30.]
 [ 6. 12. 18.]
 [ 5. 10. 15.]]
```

Fail to reject the null hypothesis: There is no significant association between the variables.

Experiment-11:

11. Write a program of Naïve Bayesian classification using python programming language.

```
pip install numpy pandas scikit-learn

import numpy as np

import pandas as pd

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Load the Iris dataset

iris = datasets.load_iris()

X = iris.data  # Features

y = iris.target  # Target variable (species)


# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Initialize the Gaussian Naïve Bayes classifier

model = GaussianNB()


# Fit the model on the training data

model.fit(X_train, y_train)
```

```
# Make predictions on the test data

y_pred = model.predict(X_test)


# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

cm = confusion_matrix(y_test, y_pred)

report = classification_report(y_test, y_pred)


# Output the results

print("Accuracy:", accuracy)

print("\nConfusion Matrix:\n", cm)

print("\nClassification Report:\n", report)
```

output:

```
Accuracy: 1.0

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Experiment-12:

12:Implement a Java Program for Apriori Algorithm.

```
import java.util.*;

public class Apriori {

    public static void main(String[] args) {
        List<Set<String>> transactions = new ArrayList<>();

        // Sample transactions
        transactions.add(new HashSet<>(Arrays.asList("Milk", "Bread", "Butter")));
        transactions.add(new HashSet<>(Arrays.asList("Bread", "Diaper", "Beer", "Eggs")));
        transactions.add(new HashSet<>(Arrays.asList("Milk", "Bread", "Diaper", "Beer")));
        transactions.add(new HashSet<>(Arrays.asList("Milk", "Bread")));
        transactions.add(new HashSet<>(Arrays.asList("Bread", "Butter", "Diaper")));

        double minSupport = 0.4;

        Set<Set<String>> frequentItemsets = apriori(transactions, minSupport);

        System.out.println("Frequent Itemsets:");
        for (Set<String> itemset : frequentItemsets) {
            System.out.println(itemset);
```

```
    }
}

public static Set<Set<String>> apriori(List<Set<String>> transactions, double minSupport) {
    Map<Set<String>, Integer> itemCounts = new HashMap<>();
    int transactionCount = transactions.size();
    Set<Set<String>> frequentItemsets = new HashSet<>();

    // Count individual item occurrences
    for (Set<String> transaction : transactions) {
        for (String item : transaction) {
            Set<String> singleItemSet = new HashSet<>();
            singleItemSet.add(item);
            itemCounts.put(singleItemSet, itemCounts.getOrDefault(singleItemSet, 0) + 1);
        }
    }

    // Generate frequent 1-itemsets
    for (Map.Entry<Set<String>, Integer> entry : itemCounts.entrySet()) {
        if (entry.getValue() / (double) transactionCount >= minSupport) {
            frequentItemsets.add(entry.getKey());
        }
    }

    // Generate frequent k-itemsets
```

```
Set<Set<String>> currentItemsets = new HashSet<>(frequentItemsets);
int k = 2;

while (!currentItemsets.isEmpty()) {
    Set<Set<String>> candidateItemsets = generateCandidateItemsets(currentItemsets, k);
    itemCounts.clear();

    // Count occurrences of candidate itemsets
    for (Set<String> transaction : transactions) {
        for (Set<String> candidate : candidateItemsets) {
            if (transaction.containsAll(candidate)) {
                itemCounts.put(candidate, itemCounts.getOrDefault(candidate, 0) + 1);
            }
        }
    }

    currentItemsets.clear();
    for (Map.Entry<Set<String>, Integer> entry : itemCounts.entrySet()) {
        if (entry.getValue() / (double) transactionCount >= minSupport) {
            currentItemsets.add(entry.getKey());
            frequentItemsets.add(entry.getKey());
        }
    }

    k++;
```

```
        }

        return frequentItemsets;
    }

    private static Set<Set<String>> generateCandidateItemsets(Set<Set<String>> itemsets, int k)
{
        Set<Set<String>> candidates = new HashSet<>();

        List<Set<String>> itemsetsList = new ArrayList<>(itemsets);
        for (int i = 0; i < itemsetsList.size(); i++) {
            for (int j = i + 1; j < itemsetsList.size(); j++) {
                Set<String> first = itemsetsList.get(i);
                Set<String> second = itemsetsList.get(j);

                // Join step: combine two itemsets if they have (k-2) items in common
                Set<String> candidate = new HashSet<>(first);
                candidate.addAll(second);

                if (candidate.size() == k) {
                    candidates.add(candidate);
                }
            }
        }
```

```
        return candidates;
    }
}
```

Output:

java -cp /tmp/dBDNzOnm9m/Apriori

Frequent Itemsets:

[Butter, Bread]

[Butter]

[Bread, Beer, Diaper]

[Bread, Beer]

[Milk]

[Bread]

[Diaper]

[Beer]

[Milk, Bread]

[Bread, Diaper]

[Diaper, Beer]

=== Code Execution Successful ===

Experiment-13:

13. Write a program to cluster your choice of data using simple k-means algorithm using JDK

```
import java.io.*;

import java.lang.*;

class Kmean

{

public static void main(String args[])

{

int N=9;

int arr[]={2,4,10,12,3,20,30,11,25};    // initial data

int i,m1,m2,a,b,n=0;

boolean flag=true;

float sum1=0,sum2=0;

a=arr[0];b=arr[1];

m1=a; m2=b;

int cluster1[]=new int[9],cluster2[]=new int[9];

for(i=0;i<9;i++)

   System.out.print(arr[i]+ "\t");

System.out.println();


do

{

 n++;

 int k=0,j=0;
```

```
for(i=0;i<9;i++)
{
  if(Math.abs(arr[i]-m1)<=Math.abs(arr[i]-m2))
  {   cluster1[k]=arr[i];
      k++;
  }
  else
  {   cluster2[j]=arr[i];
      j++;
  }
}
  System.out.println();
  for(i=0;i<9;i++)
      sum1=sum1+cluster1[i];
  for(i=0;i<9;i++)
      sum2=sum1+cluster2[i];
  a=m1;
  b=m2;
  m1=Math.round(sum1/k);
  m2=Math.round(sum2/j);
  if(m1==a && m2==b)
      flag=false;
  else
      flag=true;
```

```
    System.out.println("After iteration "+ n +" , cluster 1 :\n");    //printing the clusters of each iteration

    for(i=0;i<9;i++)

        System.out.print(cluster1[i]+ "\t");


    System.out.println("\n");

    System.out.println("After iteration "+ n +" , cluster 2 :\n");

    for(i=0;i<9;i++)

        System.out.print(cluster2[i]+ "\t");


}while(flag);


    System.out.println("Final cluster 1 :\n");            // final clusters

    for(i=0;i<9;i++)

        System.out.print(cluster1[i]+ "\t");


    System.out.println();

    System.out.println("Final cluster 2 :\n");

    for(i=0;i<9;i++)

        System.out.print(cluster2[i]+ "\t");

 }

}
```

output:

```
java -cp /tmp/rba3NPn0JS/Kmean

2	4	10	12	3	20	30	11	25
```

After iteration 1 , cluster 1 :

2 3 0 0 0 0 0 0 0

After iteration 1 , cluster 2 :

4 10 12 20 30 11 25 0 0

After iteration 2 , cluster 1 :

2 4 10 12 3 20 30 11 25

After iteration 2 , cluster 2 :

4 10 12 20 30 11 25 0 0

After iteration 3 , cluster 1 :

2 4 10 12 3 20 30 11 25

After iteration 3 , cluster 2 :

4 10 12 20 30 11 25 0 0

After iteration 4 , cluster 1 :

2 4 10 12 3 20 30 11 25

After iteration 4 , cluster 2 :

4 10 12 20 30 11 25 0 0

After iteration 5 , cluster 1 :

2 4 10 12 3 20 30 11 25

After iteration 5 , cluster 2 :

4 10 12 20 30 11 25 0 0

After iteration 6 , cluster 1 :

2 4 10 12 3 20 30 11 25

After iteration 6 , cluster 2 :

4 10 12 20 30 11 25 0 0

After iteration 7 , cluster 1 :

2 4 10 12 3 20 30 11 25

After iteration 7 , cluster 2 :

4 10 12 20 30 11 25 0 0

After iteration 8 , cluster 1 :

2

Experiment-14:

14. Write a program of cluster analysis using simple k-means algorithm Python Programming language.

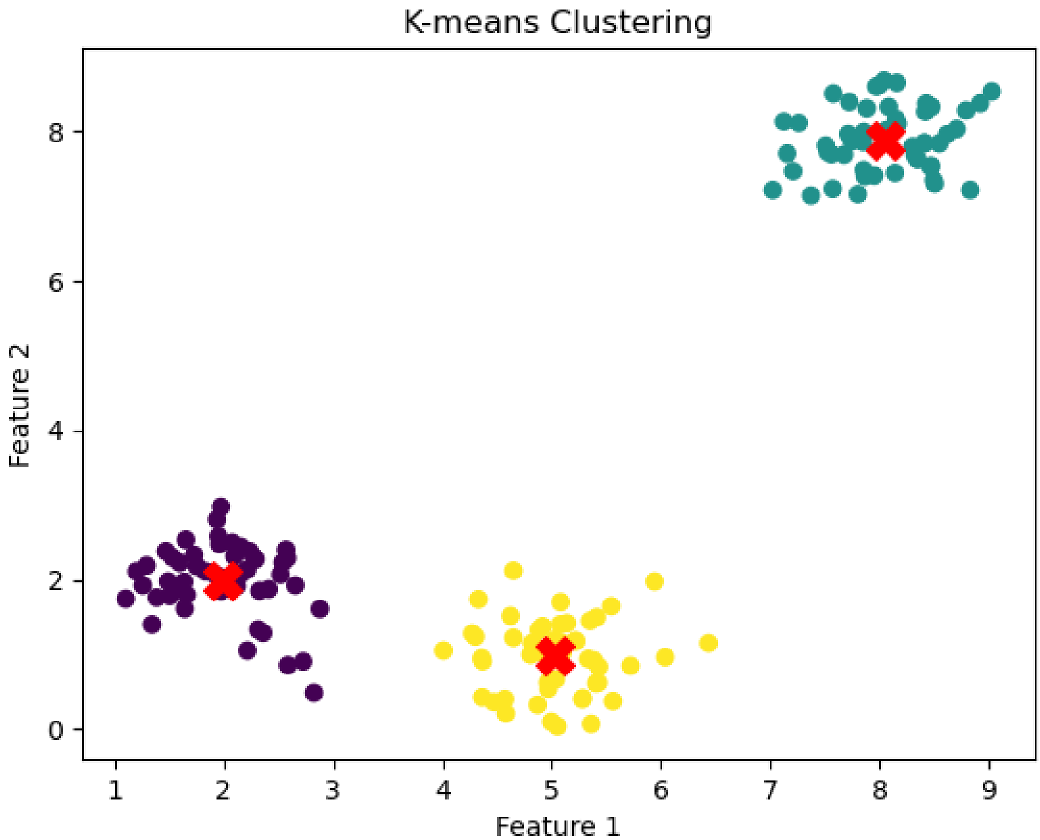
```
import numpy as np

import matplotlib.pyplot as plt


# Generate synthetic data

def generate_data(num_points, centers, spread):

    data = []

    for center in centers:

        points = np.random.randn(num_points, 2) * spread + center

        data.append(points)

    return np.vstack(data)


# K-means algorithm

def k_means(data, k, max_iterations=100):

    # Randomly initialize centroids

    random_indices = np.random.choice(data.shape[0], k, replace=False)

    centroids = data[random_indices]


    for _ in range(max_iterations):

        # Assign clusters

        distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)

        labels = np.argmin(distances, axis=1)
```

```
        # Update centroids
        new_centroids = np.array([data[labels == i].mean(axis=0) for i in range(k)])

        # Check for convergence
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids

    return centroids, labels

import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data
def generate_data(num_points, centers, spread):
    data = []
    for center in centers:
        points = np.random.randn(num_points, 2) * spread + center
        data.append(points)
    return np.vstack(data)

# K-means algorithm
def k_means(data, k, max_iterations=100):
    # Randomly initialize centroids
    random_indices = np.random.choice(data.shape[0], k, replace=False)
```

```
    centroids = data[random_indices]

    for _ in range(max_iterations):
        # Assign clusters
        distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)

        # Update centroids
        new_centroids = np.array([data[labels == i].mean(axis=0) for i in range(k)])

        # Check for convergence
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids

    return centroids, labels

# Visualization function
def plot_clusters(data, centroids, labels):
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', marker='o')
    plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200)
    plt.title('K-means Clustering')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()
```

```
# Parameters
num_points_per_cluster = 50
centers = [(2, 2), (8, 8), (5, 1)]
spread = 0.5
k = len(centers)

# Generate data
data = generate_data(num_points_per_cluster, centers, spread)

# Run K-means
centroids, labels = k_means(data, k)

# Plot results
plot_clusters(data, centroids, labels)
```

Output:


K-means Clustering
Feature 2
Feature 1
0
2
4
6
8
1
2
3
4
5
6
7
8
9

Experiment-15:

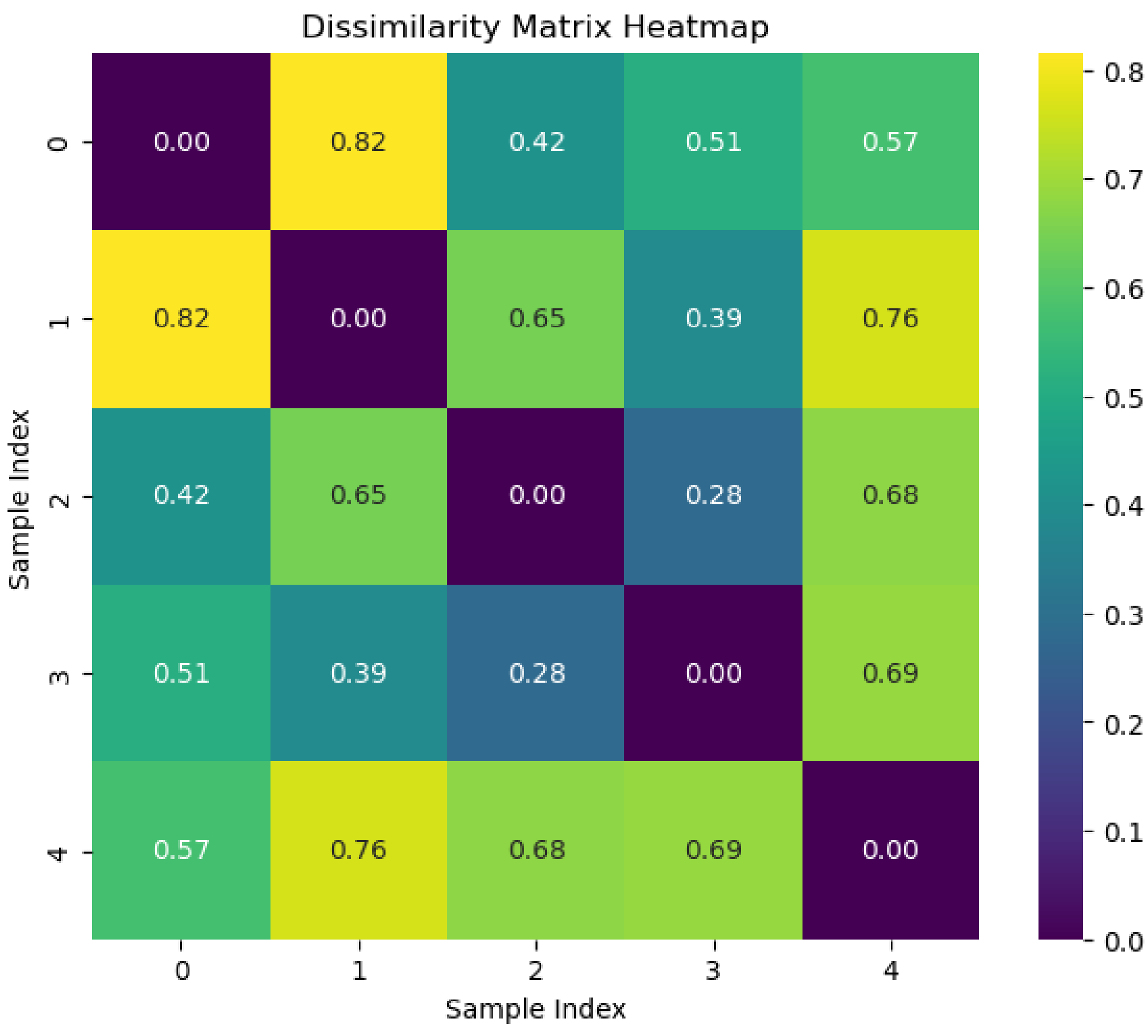
15. Write a program to compute/ display dissimilarity matrix using python.

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


# Function to compute dissimilarity matrix

def compute_dissimilarity_matrix(data):

    num_samples = data.shape[0]

    dissimilarity_matrix = np.zeros((num_samples, num_samples))


    for i in range(num_samples):

        for j in range(num_samples):

            dissimilarity_matrix[i, j] = np.linalg.norm(data[i] - data[j])


    return dissimilarity_matrix


# Generate synthetic data

def generate_data(num_points, num_features):

    return np.random.rand(num_points, num_features)


# Parameters

num_points = 5

num_features = 3
```

```
# Generate data

data = generate_data(num_points, num_features)


# Compute dissimilarity matrix

dissimilarity_matrix = compute_dissimilarity_matrix(data)


# Display the dissimilarity matrix

print("Dissimilarity Matrix:")

print(dissimilarity_matrix)


# Visualize the dissimilarity matrix using a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(dissimilarity_matrix, annot=True, fmt=".2f", cmap='viridis', square=True)

plt.title('Dissimilarity Matrix Heatmap')

plt.xlabel('Sample Index')

plt.ylabel('Sample Index')

plt.show()
```

OUTPUT:

```
Dissimilarity Matrix:
[[0.         0.8157134  0.41705939 0.51141451 0.57212754]
 [0.8157134  0.         0.64619235 0.3866796  0.76280542]
 [0.41705939 0.64619235 0.         0.27939281 0.67770535]
 [0.51141451 0.3866796  0.27939281 0.         0.68512593]
 [0.57212754 0.76280542 0.67770535 0.68512593 0.        ]]
```


Dissimilarity Matrix Heatmap
0.00
0.82
0.42
0.51
0.57
0.82
0.00
0.65
0.39
0.76
0.42
0.65
0.00
0.28
0.68
0.51
0.39
0.28
0.00
0.69
0.57
0.76
0.68
0.69
0.00
Sample Index
Sample Index
0
1
2
3
4
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

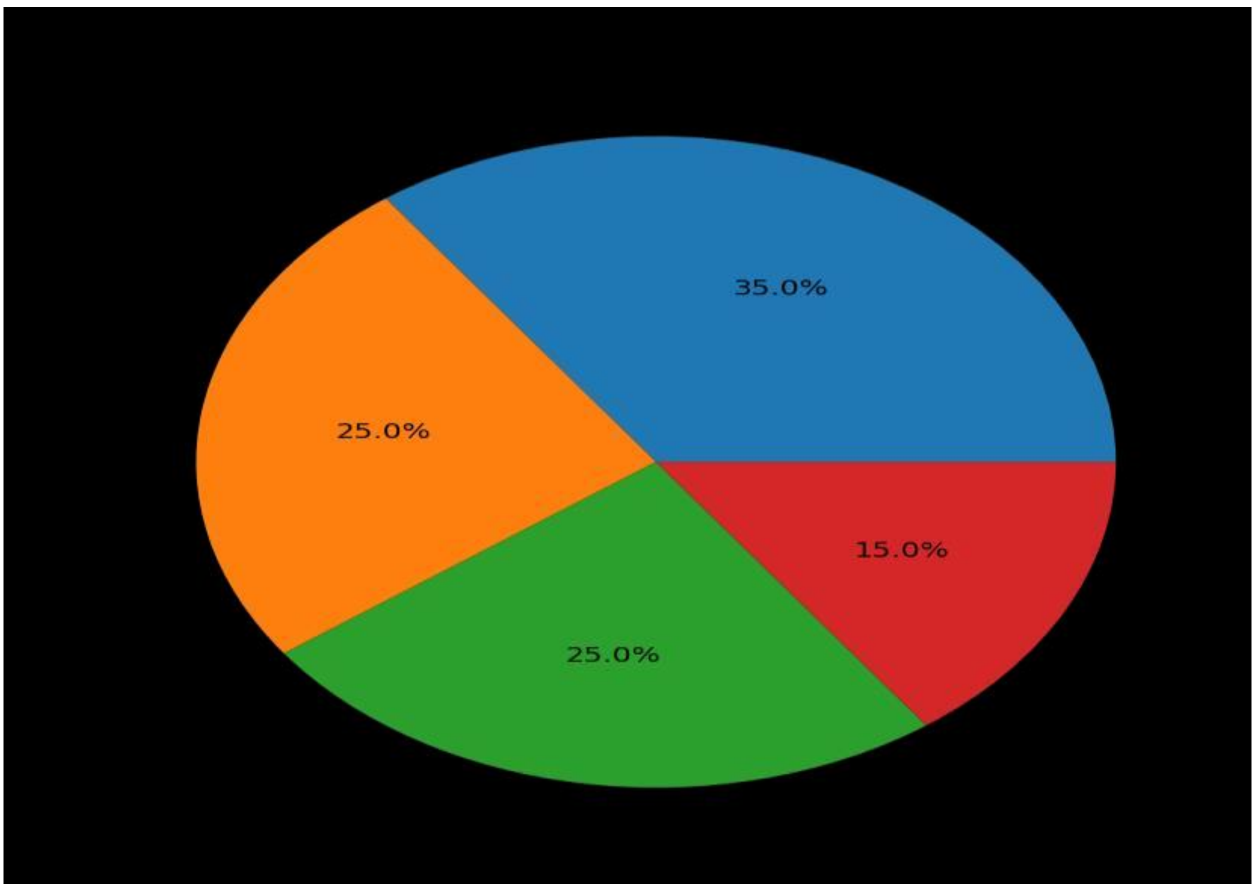
Experiment-16:

16. Visualize the datasets using matplot lib in python. (Histogram, Box Plot, Bar chart, Pie Chart).

program for piechart:

```
import matplotlib.pyplot as plt
# Sample data
labels = ['Apples', 'Bananas', 'Cherries', 'Dates']
sizes = [35, 25, 25, 15]
# Create the pie chart
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title('Fruit Distribution')
plt.show()
```

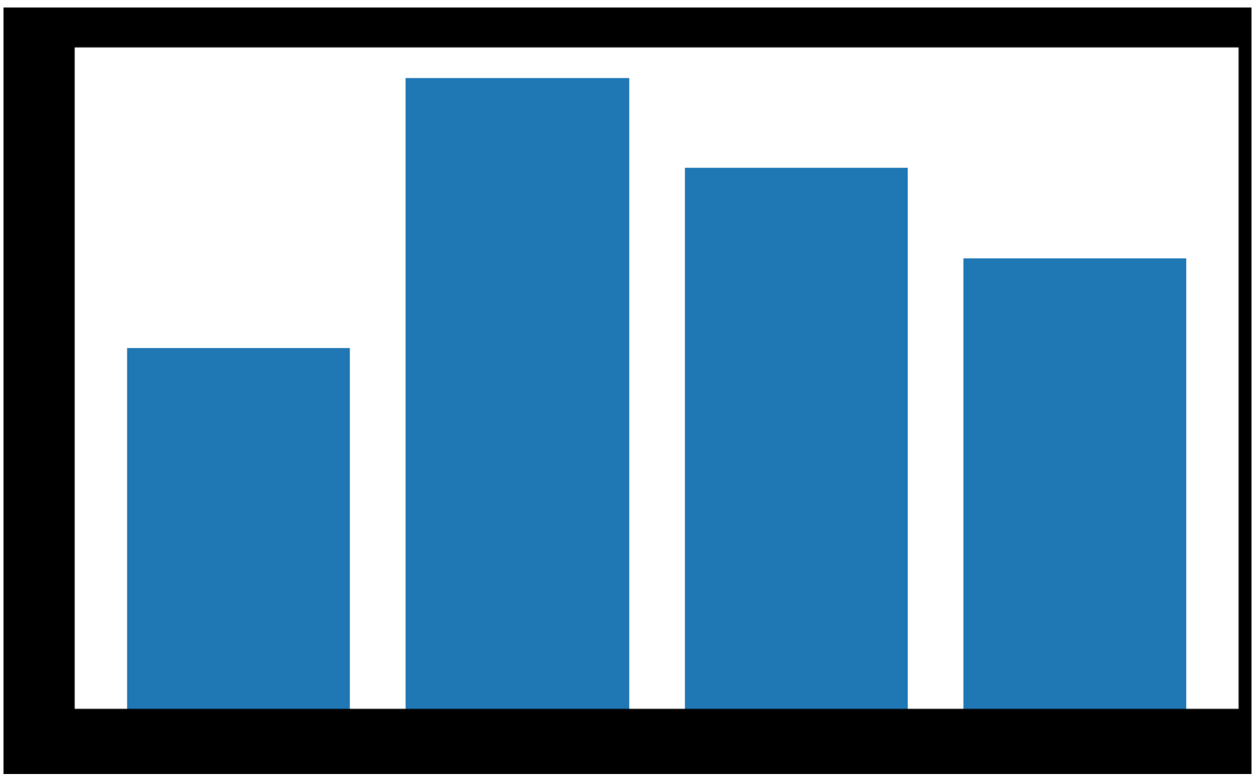
Output:



program for Bargraph:

```
import matplotlib.pyplot as plt
import pandas as pd
# Sample data
data = {'Category': ['A', 'B', 'C', 'D'], 'Values': [20, 35, 30, 25]}df = pd.DataFrame(data)
# Create a bar graph
plt.bar(df['Category'], df['Values'])
plt.xlabel('Category')
plt.ylabel('Values')
plt.title('Bar Graph Example')
plt.show()
```

Output:



Program for Barplot:

```
import matplotlib.pyplot as plt
import numpy as np
# Creating dataset
np.random.seed(10)
data = np.random.normal(100, 20, 200)
fig = plt.figure(figsize =(10, 7))
# Creating plot
plt.boxplot(data)
# show plot
plt.show()
```

Output: