Experiment No: 02

- *Create database and table:*

```
create database Pharmacy;
show  databases;
use Pharmacy;
create table Pharma_data (cutomers_id int,cutomers_name varchar(20),cutomers_address varchar(20), orders int, products varchar(20),payment int);
describe Pharma_data;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
cutomers_id	int	YES		NULL	
cutomers_name	varchar(20)	YES		NULL	
cutomers_address	varchar(20)	YES		NULL	
orders	int	YES		NULL	
products	varchar(20)	YES		NULL	
payment	int	YES		NULL	

- *Insert Values :*

```
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(01,"Priyush","Mumbai", 180, "Paracetamol",3000);
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(02,"Hardik","Satara",100, "Albendazole ",6000);
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(03,"Aniket","Pune",120, "Cloxacillin",1000);
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(04,"Rushi","Nagpur",320, "Ranitidine ",1300);
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(05,"Shubham","Panvel",220, "Azee",2300);
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(06,"Sachin","Amravati",220, "betnsoil",4300);
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(07,"Amol","Buldhana",720, "cefix",2200);
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(08,"RAvi","Ahmednagar",820, "cipox",1200);
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(09,"Ritesh","Ahmednagar",120, "Dolo",1270);
insert into Pharma_data (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(10,"Rupesh","Aurangabad",920, "Pantop",8270);
select * from Pharma_data;
```

```
select products,sum(payment) from Pharma_data group by products;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

cutomers_id	cutomers_name	cutomers_address	orders	products	payment
1	Priyush	Mumbai	180	Paracetamol	3000
1	Priyush	Mumbai	180	Paracetamol	3000
10	Rupesh	Aurangabad	920	Pantop	8270
2	Hardik	Satara	100	Albendazole	6000
3	Aniket	Pune	120	Cloxacillin	1000
4	Rushi	Nagpur	320	Ranitidine	1300
5	Shubham	Panvel	220	Azee	2300
6	Sachin	Amravati	220	betnsoil	4300
7	Amol	Buldhana	720	cefix	2200
8	RAvi	Ahmednagar	820	cipox	1200
9	Ritesh	Ahmednagar	120	Dolo	1270
10	Rupesh	Aurangabad	920	Pantop	8270

- **Roll-Up:**

```
SELECT cutomers_name, SUM(payment) AS "Total Payment"
FROM Pharma_data
GROUP BY cutomers_name WITH ROLLUP;
```

```
insert into Pharma_data01 (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(01,"Krish","Mumbai", 180, "Paracetamol",3000);
insert into Pharma_data01 (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(02,"Om","Satara",100, "Albendazole ",6000);
insert into Pharma_data01 (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(03,"King","Pune",120, "Cloxacillin",1000);
insert into Pharma_data01 (cutomers_id,cutomers_name,cutomers_address,orders,products,payment)values(04,"Raj","Nagpur",320, "Ranitidine ",1300);

SELECT Pharma_data.cutomers_name, Pharma_data.orders, Pharma_data.products,  Pharma_data01.payment
FROM Pharma_data
INNER JOIN Pharma_data01
ON  Pharma_data01.cutomers_id = Pharma_data01.cutomers_id;
```

Priyush	180	Paracetamol	1000
Priyush	180	Paracetamol	6000
Priyush	180	Paracetamol	3000
Rupesh	920	Pantop	1300
Rupesh	920	Pantop	1000
Rupesh	920	Pantop	6000
Rupesh	920	Pantop	3000
Hardik	100	Albendazole	1300
Hardik	100	Albendazole	1000
Hardik	100	Albendazole	6000
Hardik	100	Albendazole	3000
Aniket	120	Cloxacillin	1300
Aniket	120	Cloxacillin	1000
Aniket	120	Cloxacillin	6000
Aniket	120	Cloxacillin	3000
Rushi	320	Ranitidine	1300
Rushi	320	Ranitidine	1000
Rushi	320	Ranitidine	6000

sult 24

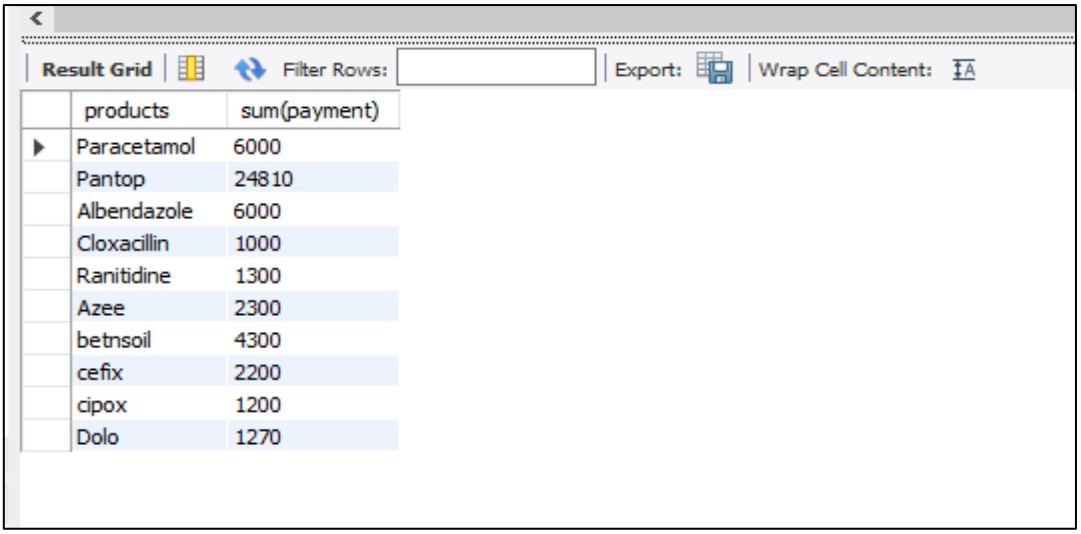
- **Slice:**

Result Grid | Filter Rows:

cutomers_name	Total Payment
Amol	2200
Aniket	1000
Hardik	6000
Priyush	6000
RAvi	1200
Ritesh	1270
Rupesh	24810
Rushi	1300
Sachin	4300
Shubham	2300
NULL	50380

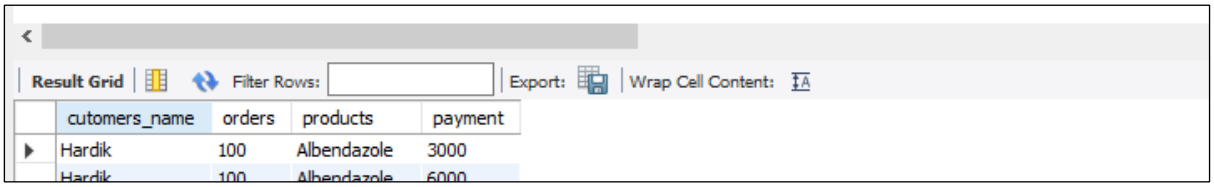
- **Dice:**

```
SELECT Pharma_data.cutomers_name, Pharma_data.orders, Pharma_data.products,  Pharma_data01.payment
FROM Pharma_data
INNER JOIN Pharma_data01
where Pharma_data.cutomers_name = "Hardik"
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

products	sum(payment)
Paracetamol	6000
Pantop	24810
Albendazole	6000
Cloxacillin	1000
Ranitidine	1300
Azee	2300
betnsoil	4300
cefix	2200
cipox	1200
Dolo	1270



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

cutomers_name	orders	products	payment
Hardik	100	Albendazole	3000
Hardik	100	Albendazole	6000

- **Drill-Down:**

```
20
21 •  SELECT cutomers_name,products, SUM(payment) AS "Total Payment"
22    FROM Pharma_data
23    GROUP BY cutomers_name WITH ROLLUP;
24
```

Experiment No: 03

```
import mean as mean
import numpy as np
import math
from sklearn.datasets import load_iris
from sklearn import datasets, linear_model, metrics
dataset = load_iris()
a = dataset.data
b = np.zeros(150)
# take 1st column among 4 column of data set
for i in range (150):
     b[i]=a[i,1]
b=np.sort(b) #sort the array
# create bins
bin1=np.zeros((30,5))
bin2=np.zeros((30,5))
bin3=np.zeros((30,5))
# for mean
for i in range (0,150,5):
    k=int(i/5)
    mean=(b[i] + b[i+1] + b[i+2] + b[i+3] + b[i+4])/5
    for j in range(5):
        bin1[k,j]=mean
    print("Bin Mean: \n",bin1)
#for boundaries
for i in range (0,150,5):
    k=int(i/5)
    for j in range (5):
     if (b[i+j]-b[i]) < (b[i+4]-b[i+j]):
        bin2[k,j]=b[i]
    else:
        bin2[k,j]=b[i+4]
print("Bin Boundaries: \n",bin2)
#for median
for i in range (0,150,5):
    k=int(i/5)
    for j in range (5):
      bin3[k,j]=b[i+2]
print("Bin Median: \n",bin3)
```

Experiment No: 04

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split


class NaiveBayes(object):

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # mean, variance, priors
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

        # extracting mean, variance and priors for each class
        # useful in calculating pdf during prediction
        for c in self._classes:
            X_c = X[y == c]
            self._mean[c, :] = X_c.mean(axis=0)
            self._var[c, :] = X_c.var(axis=0)
            self._priors[c] = X_c.shape[0] / float(n_samples)

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        posteriors = []

        # calculate posterior probability for each class
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            class_conditional = np.sum(np.log(self.gaussian_pdf(idx, x)))
            posterior = prior + class_conditional
            posteriors.append(posterior)

        # return class with highest posterior probability
        return self._classes[np.argmax(posteriors)]

    def gaussian_pdf(self, class_idx, x):
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(- (x-mean)**2 / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator


def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy
```

```
iris_data = load_iris()

X_train, X_test, y_train, y_test = train_test_split(
    iris_data.data, iris_data.target,  test_size=0.2, random_state=47)

print("Training set shape: ", X_train.shape, y_train.shape)
print("Testing set shape: ", X_test.shape, y_test.shape)

nb = NaiveBayes()
nb.fit(X_train, y_train.ravel())
y_pred = nb.predict(X_test)


print(f"Naive Bayes accuracy: {accuracy(y_test, y_pred)}")
```

OUTPUT:

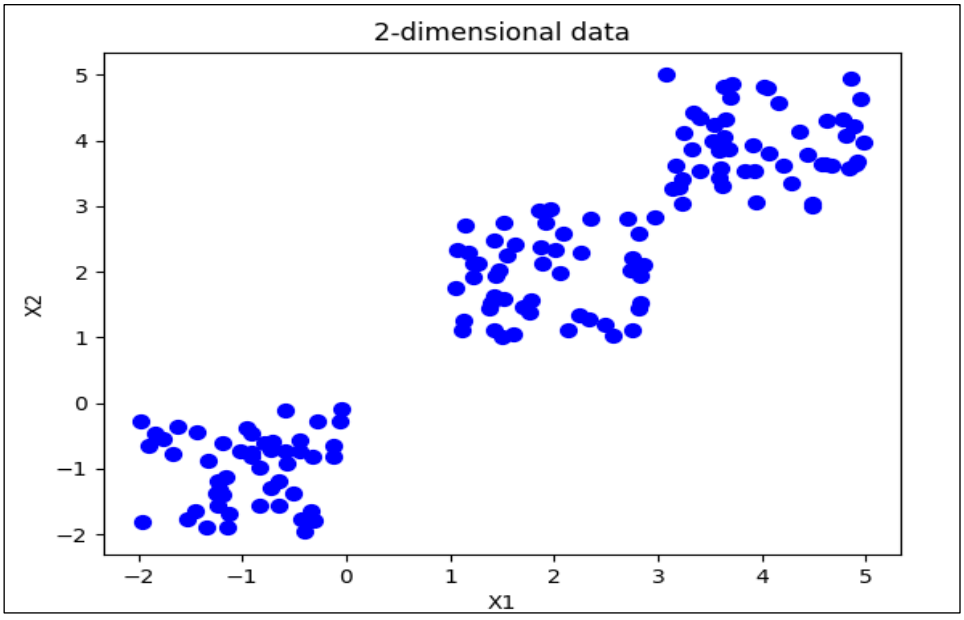
Experiment No: 05

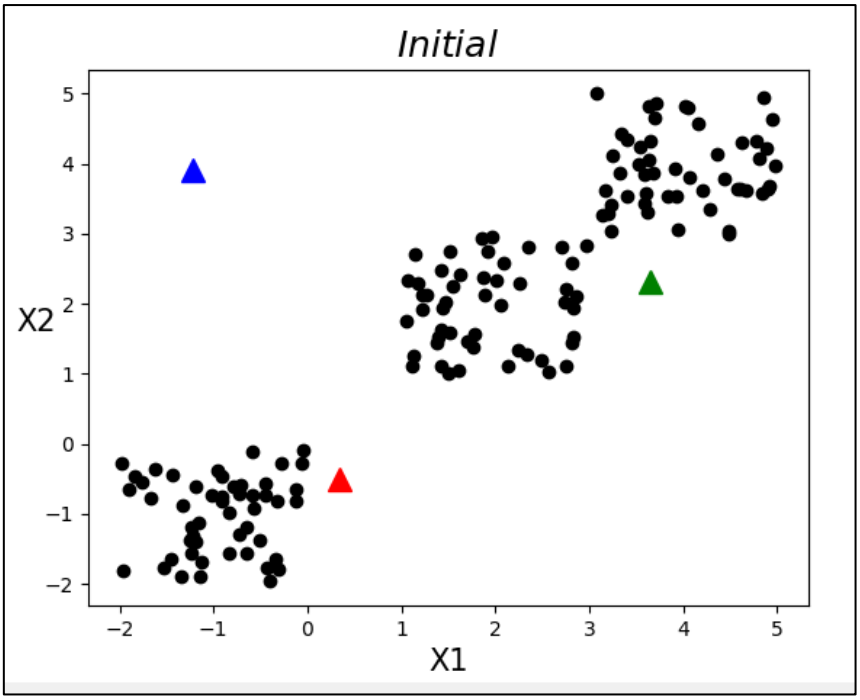
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans


def euclideanDistance(centroid, datapoint):
    '''
    Description:{
        Calculates the Euclidean distance between a centroid and
        a data point
    }

    inputs: {
        centroid: centroid position
        datapoint: a single datapoint whose distance from
            centroid is to be calculated
    }
    return: {
        [scalar] Distance between centroid and data
    }
    '''

    distance = np.sum(np.power(centroid - datapoint, 2))
    return np.sqrt(distance)


def calculateDistances(data, centroid):
    '''
    Description: {
        This function calculates distances between a
        centroid and the dataset.
        Uses the euclideanDistance function
    }

    inputs: {
        centroid: centroid position
        data: dataset whose distance from a
            centroid is to be calculated
    }

    return: {
        [list] distance of each datapoint from the centroid
    }
    '''

    dist = []
    for i in range(len(data)):
        point_dist_to_centroid = euclideanDistance(centroid, data[i])
        dist.append(point_dist_to_centroid)
    return dist


def distanceToCentroids(data, centroids):
    '''
    Description: {
        This function calculates distance of between all datapoints
```

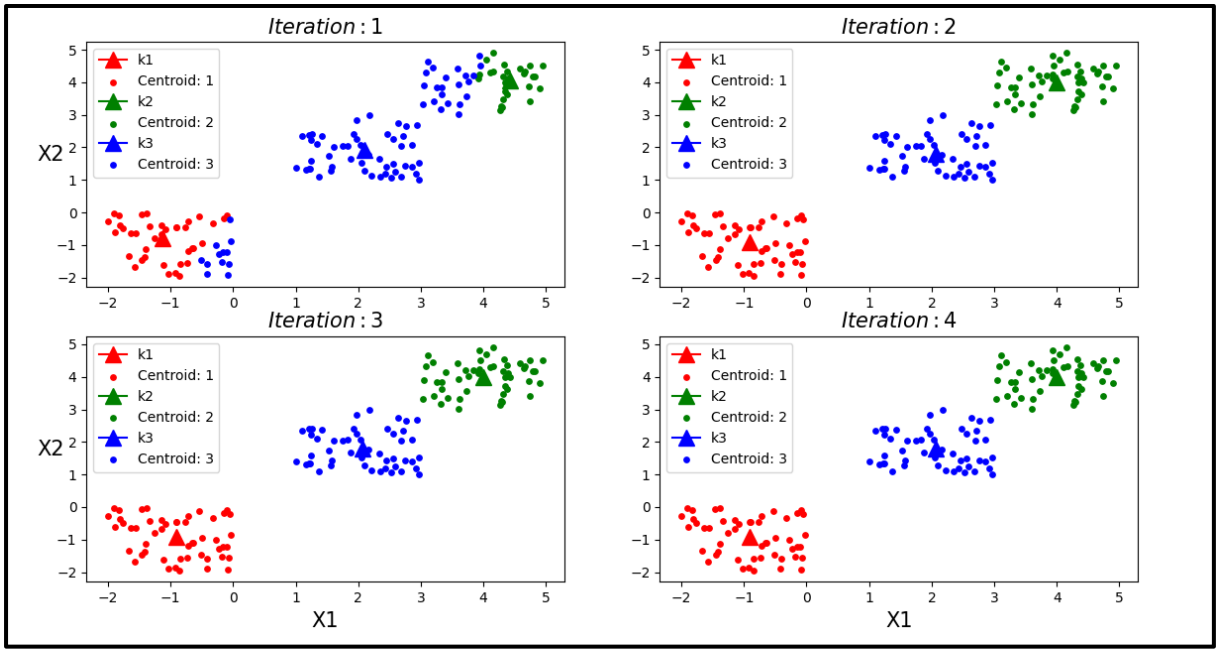
```
        and the specified number of centroids.
        Uses calculateDistances function
    }

    inputs: {
        data: dataset
        centroids: list of list of initial or intermediary
            centroids positions.
    }

    return: {
        [list of list] each row contains distance between the
            datapoint and all centroids
    }
    '''

    data_dist_from_centroid = [[] for i in range(len(data))]

    for k in centroids:
        dist = calculateDistances(data, k)
        for x, y in enumerate(dist):
            data_dist_from_centroid[x].append(y)

    return data_dist_from_centroid


def getNewCentroid(data, distances, numOfCentroids,
assigned_centroids=None):
    '''
    Description: {
        This functions assigns datapoints to each of
            the centroids and calculates new position for
            all the centroids
    }

    inputs: {
        data: dataset
        distances: calculated distances between datapoints and centroids
        numOfCentroids: number of centroids initialized
    }

    return: {
        [list of list] new poistions of each centroid
    }
    '''

    assigned_centroids = [[] for _ in range(numOfCentroids)]

    # assignnig data point to centroids
    for i in range(data.shape[0]):
        min_index = np.argmin(distances[i])
        assigned_centroids[min_index].append(data[i])

    new_k = [np.array(i).mean(axis=0) for i in assigned_centroids]

    return new_k, assigned_centroids


# generating dataset

X = -2 * np.random.rand(150, 2)
```

```
X1 = 1 + 2 * np.random.rand(50, 2)
X2 = 3 + 2 * np.random.rand(50, 2)

X[50:100, :] = X1
X[100:150, :] = X2
print(X.shape)

plt.scatter(X[:, 0], X[:, 1], s=50, c="b")
plt.title("2-dimensional data")
plt.xlabel("X1")
plt.ylabel("X2")
plt.savefig("./K-means_dataset.png")
plt.show()


# num_centroids = int(input("Number of centroids: "))
num_centroids = 3
centroids = []

# Initial centroids
num_dim = X.shape[1]

# dynamic
for i in range(num_centroids):
    k = [np.random.choice(X[:, z] + 1 * 2 * np.random.rand(X.shape[0]))
         for z in range(num_dim)]
    centroids.append(k)

print(centroids)

num_iterations = 4

# centroids = np.array([[-2, 3],
#                       [3, -2]])

colors = ["red", "green", "blue"]

for idx, centroid in enumerate(centroids, 1):
    plt.plot(centroid[0], centroid[1], marker="^",
             c=colors[idx - 1], markersize=12, label=f'k{idx}')


plt.scatter(X[:, 0], X[:, 1], c="k")

plt.title("$Initial$", fontsize=18)
plt.xlabel("X1", fontsize=15)
plt.ylabel("X2", fontsize=15, rotation=0)
plt.savefig("./K-means_Initial.png")
plt.show()


distances = distanceToCentroids(X, centroids)
centroids, centroid_points = getNewCentroid(X, distances, num_centroids)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 9))

for i in range(num_iterations):
    plt.sca(axes[i // 2, i % 2])

    for idx, centroid in enumerate(centroids, 1):
        x1, x2 = [], []
```

```
            for a, b in centroid_points[idx - 1]:
                x1.append(a)
                x2.append(b)

            plt.plot(centroid[0], centroid[1], marker="^",
                     c=colors[idx - 1], markersize=12, label=f'k{idx}')
            plt.scatter(x1, x2, s=15, c=colors[idx - 1], label=f'Centroid:
{idx}')

            plt.title(f"$Iteration: {i + 1}$", fontsize=15)

            if i in (0, 1):
                plt.xlabel("")
            else:
                plt.xlabel("X1", fontsize=15)

            if i in (1, 3):
                plt.ylabel("")
            else:
                plt.ylabel("X2", fontsize=15, rotation=0)
        plt.legend()

        distances = distanceToCentroids(X, centroids)
        centroids, centroid_points = getNewCentroid(X, distances,
num_centroids)
plt.savefig("./K-means_Final.png")
plt.show()
```

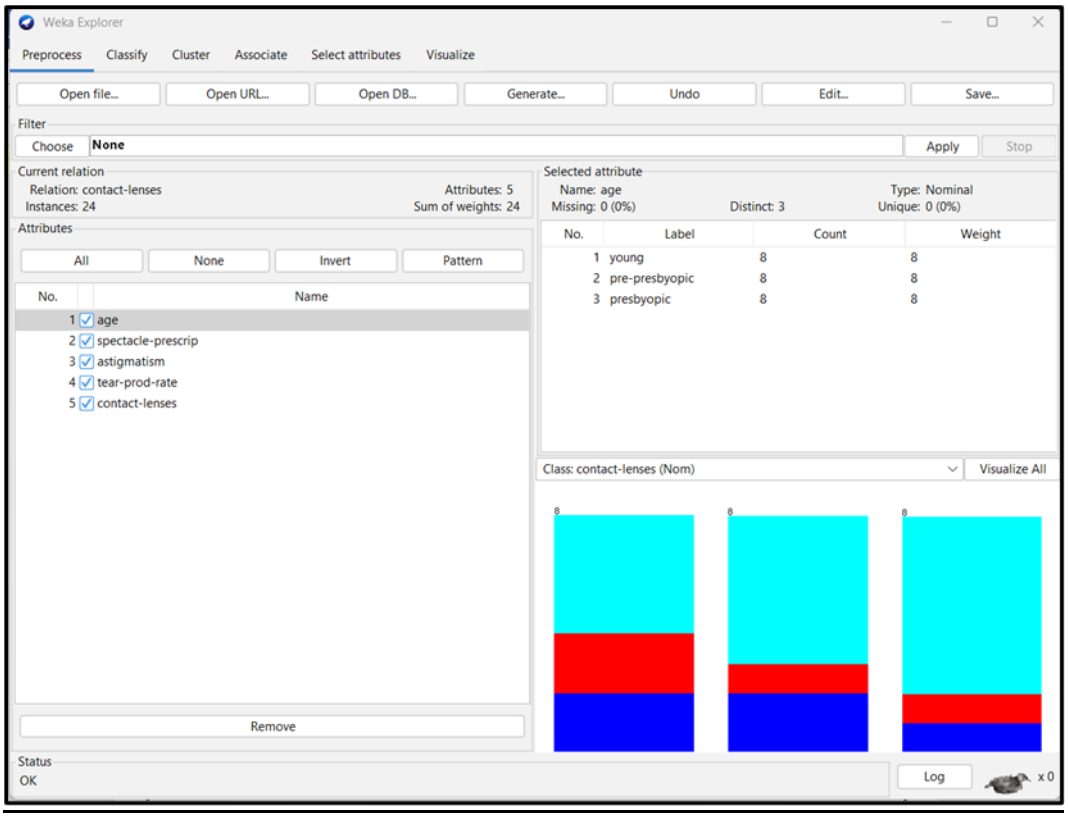


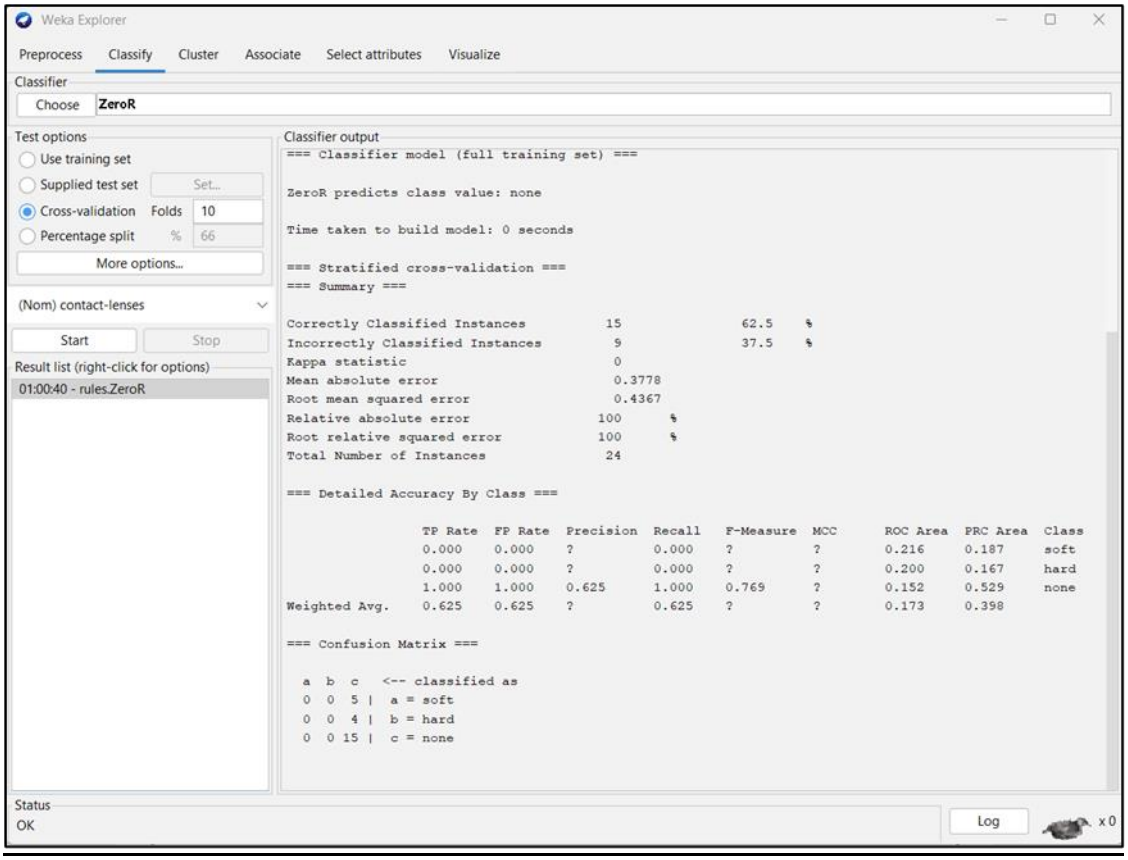


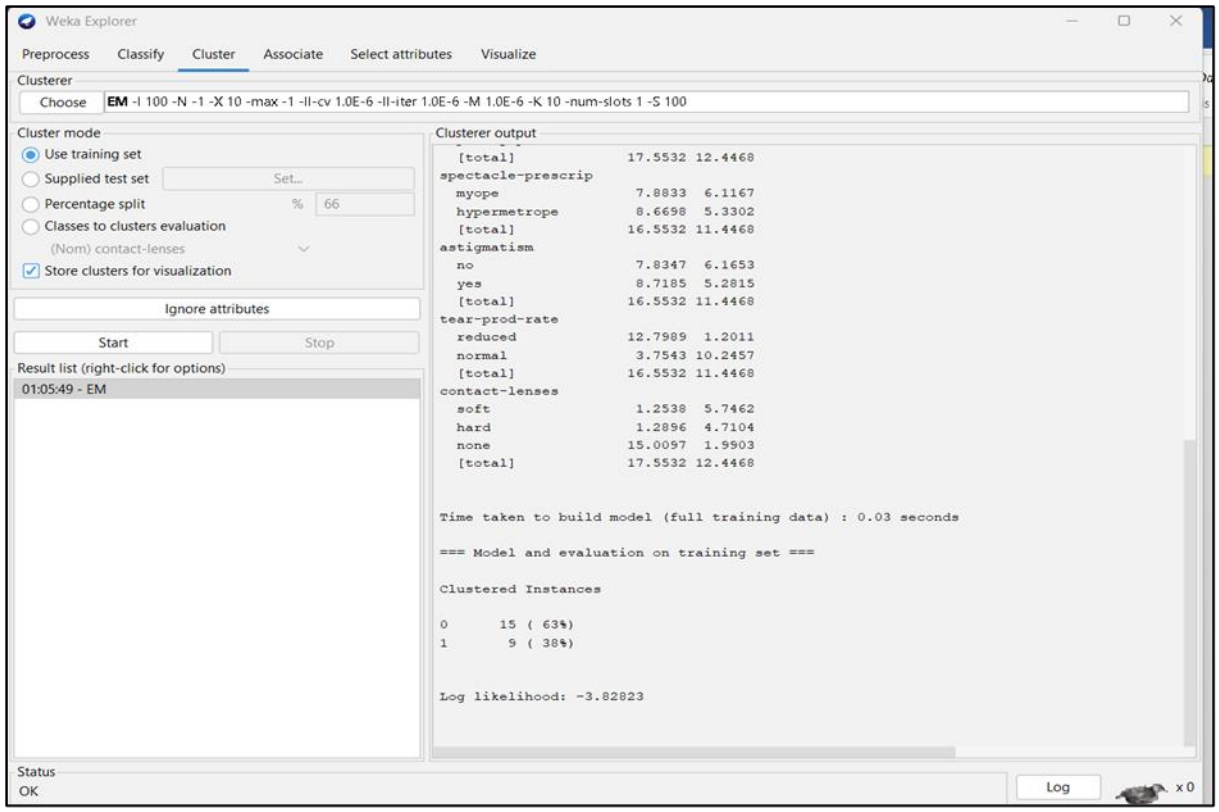


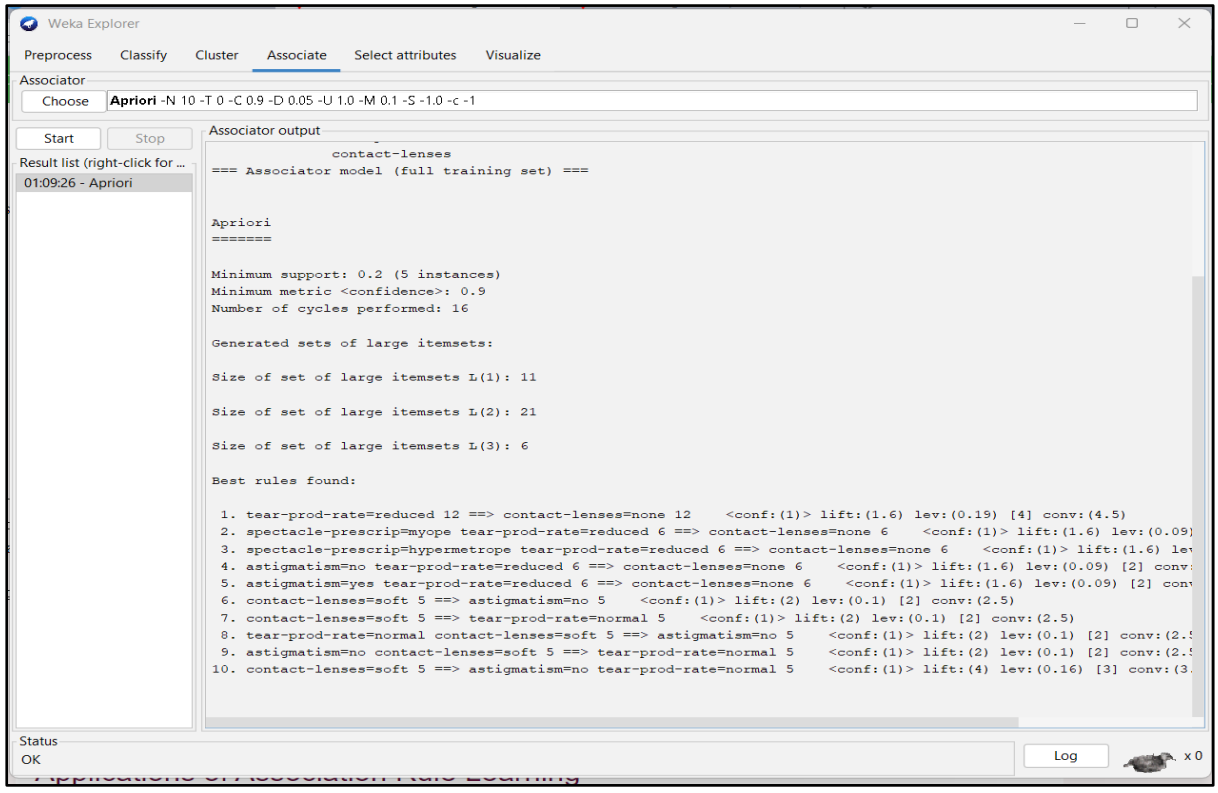

```
Run:  k
C:\Users\priyush\PycharmProjects\K-mean\venv\Scripts\python.exe C:\Users\priyush\PycharmProjects\K-mean\k.py
(150, 2)
[[0.7056955362503072, 3.0477398367739355], [5.0690676811166, 1.828117443465824], [1.8014589093032696, 2.5704369066105164]]

Process finished with exit code 0
```

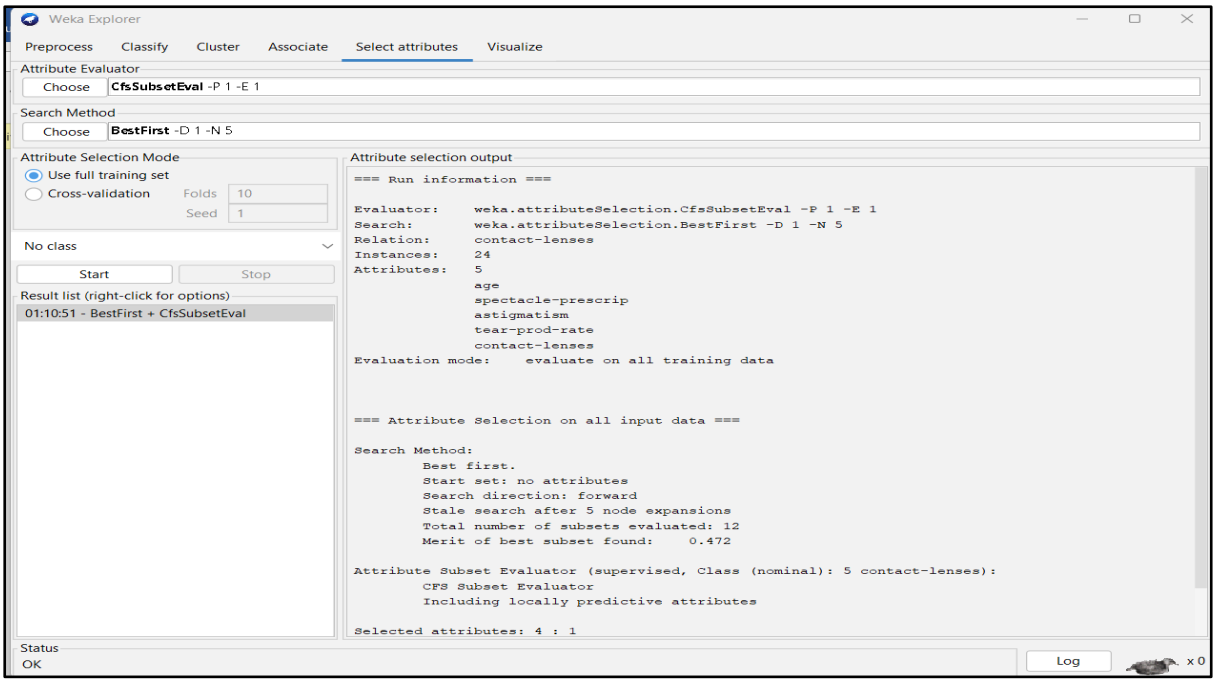
Experiment No: 06

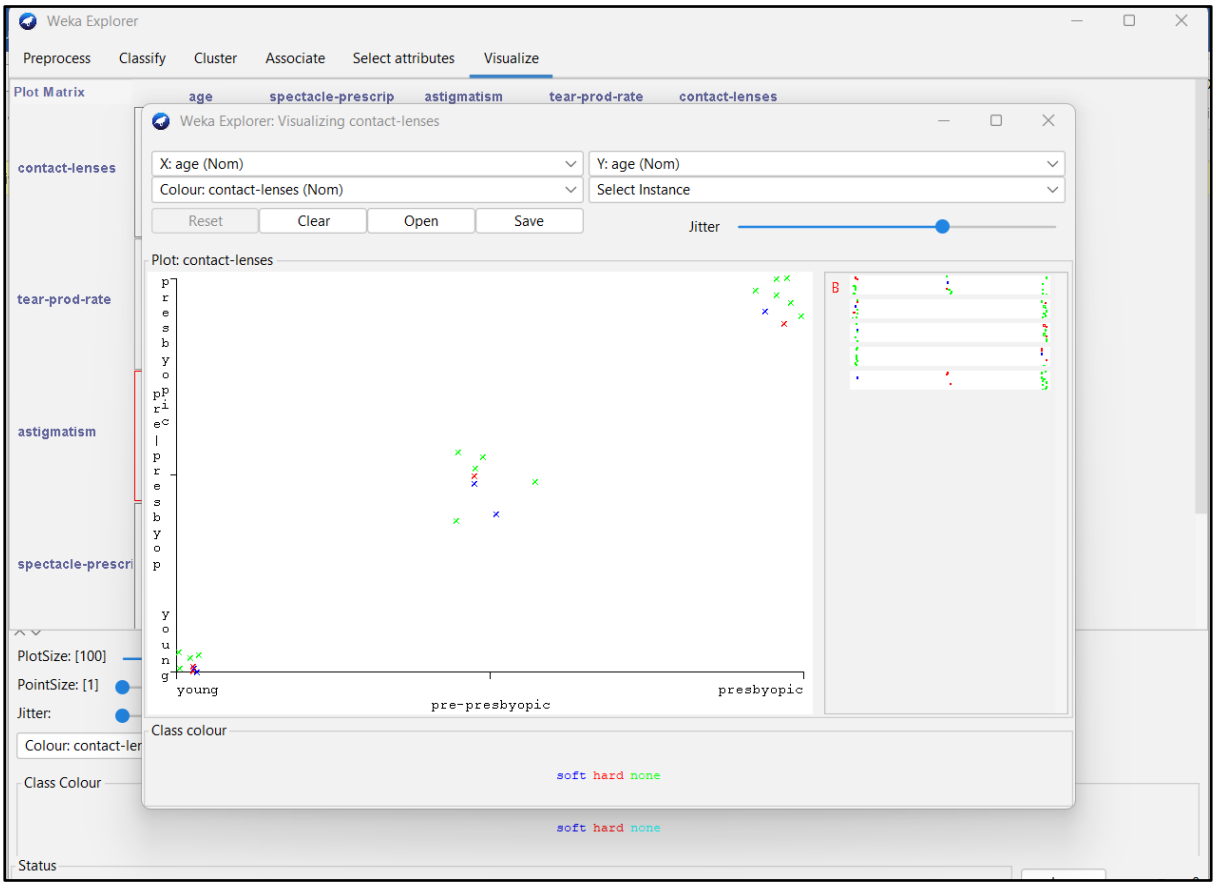





Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
EM -I 100 -N -1 -X 10 -max -1 -ll-cv 1.0E-6 -ll-iter 1.0E-6 -M 1.0E-6 -K 10 -num-slots 1 -S 100
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
%
66
Classes to clusters evaluation
(Nom) contact-lenses
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options)
01:05:49 - EM
Clusterer output
[total] 17.5532 12.4468
spectacle-prescrip
myope 7.8833 6.1167
hypermetrope 8.6698 5.3302
[total] 16.5532 11.4468
astigmatism
no 7.8347 6.1653
yes 8.7185 5.2815
[total] 16.5532 11.4468
tear-prod-rate
reduced 12.7989 1.2011
normal 3.7543 10.2457
[total] 16.5532 11.4468
contact-lenses
soft 1.2538 5.7462
hard 1.2896 4.7104
none 15.0097 1.9903
[total] 17.5532 12.4468
Time taken to build model (full training data) : 0.03 seconds
=== Model and evaluation on training set ===
Clustered Instances
0 15 (63%)
1 9 (38%)
Log likelihood: -3.82823
Status
OK
Log
x 0


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Associator
Choose
Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Start
Stop
Result list (right-click for ...
01:09:26 - Apriori
Associator output
contact-lenses
=== Associator model (full training set) ===
Apriori
=======
Minimum support: 0.2 (5 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 16
Generated sets of large itemsets:
Size of set of large itemsets L(1): 11
Size of set of large itemsets L(2): 21
Size of set of large itemsets L(3): 6
Best rules found:
1. tear-prod-rate=reduced 12 ==> contact-lenses=none 12 <conf:(1)> lift:(1.6) lev:(0.19) [4] conv:(4.5)
2. spectacle-prescrip=myope tear-prod-rate=reduced 6 ==> contact-lenses=none 6 <conf:(1)> lift:(1.6) lev:(0.09)
3. spectacle-prescrip=hypermetrope tear-prod-rate=reduced 6 ==> contact-lenses=none 6 <conf:(1)> lift:(1.6) lev
4. astigmatism=no tear-prod-rate=reduced 6 ==> contact-lenses=none 6 <conf:(1)> lift:(1.6) lev:(0.09) [2] conv
5. astigmatism=yes tear-prod-rate=reduced 6 ==> contact-lenses=none 6 <conf:(1)> lift:(1.6) lev:(0.09) [2] conv
6. contact-lenses=soft 5 ==> astigmatism=no 5 <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
7. contact-lenses=soft 5 ==> tear-prod-rate=normal 5 <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
8. tear-prod-rate=normal contact-lenses=soft 5 ==> astigmatism=no 5 <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5
9. astigmatism=no contact-lenses=soft 5 ==> tear-prod-rate=normal 5 <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5
10. contact-lenses=soft 5 ==> astigmatism=no tear-prod-rate=normal 5 <conf:(1)> lift:(4) lev:(0.16) [3] conv:(3.
Status
OK
Log
x 0

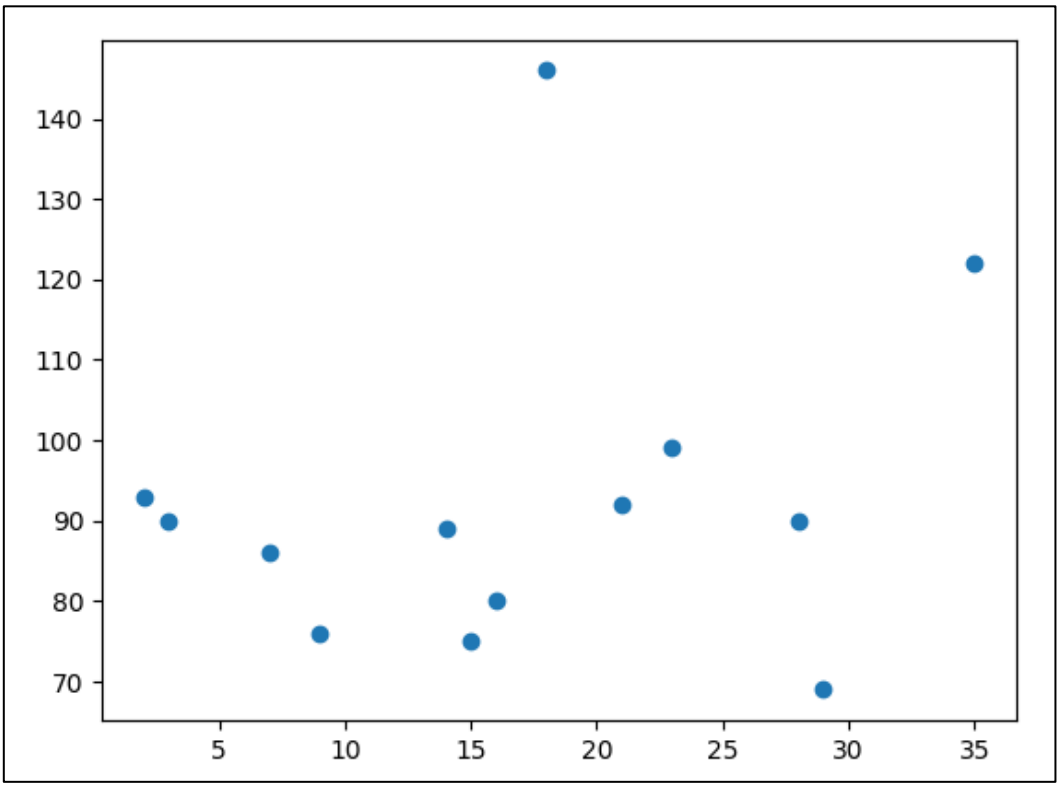

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Attribute Evaluator
Choose
CfsSubsetEval -P 1 -E 1
Search Method
Choose
BestFirst -D 1 -N 5
Attribute Selection Mode
Use full training set
Cross-validation
Folds
10
Seed
1
No class
Start
Stop
Result list (right-click for options)
01:10:51 - BestFirst + CfsSubsetEval
Attribute selection output
=== Run information ===
Evaluator: weka.attributeSelection.CfsSubsetEval -P 1 -E 1
Search: weka.attributeSelection.BestFirst -D 1 -N 5
Relation: contact-lenses
Instances: 24
Attributes: 5
age
spectacle-prescrip
astigmatism
tear-prod-rate
contact-lenses
Evaluation mode: evaluate on all training data
=== Attribute Selection on all input data ===
Search Method:
Best first.
Start set: no attributes
Search direction: forward
Stale search after 5 node expansions
Total number of subsets evaluated: 12
Merit of best subset found: 0.472
Attribute Subset Evaluator (supervised, Class (nominal): 5 contact-lenses):
CFS Subset Evaluator
Including locally predictive attributes
Selected attributes: 4 : 1
Status
OK
Log
x 0

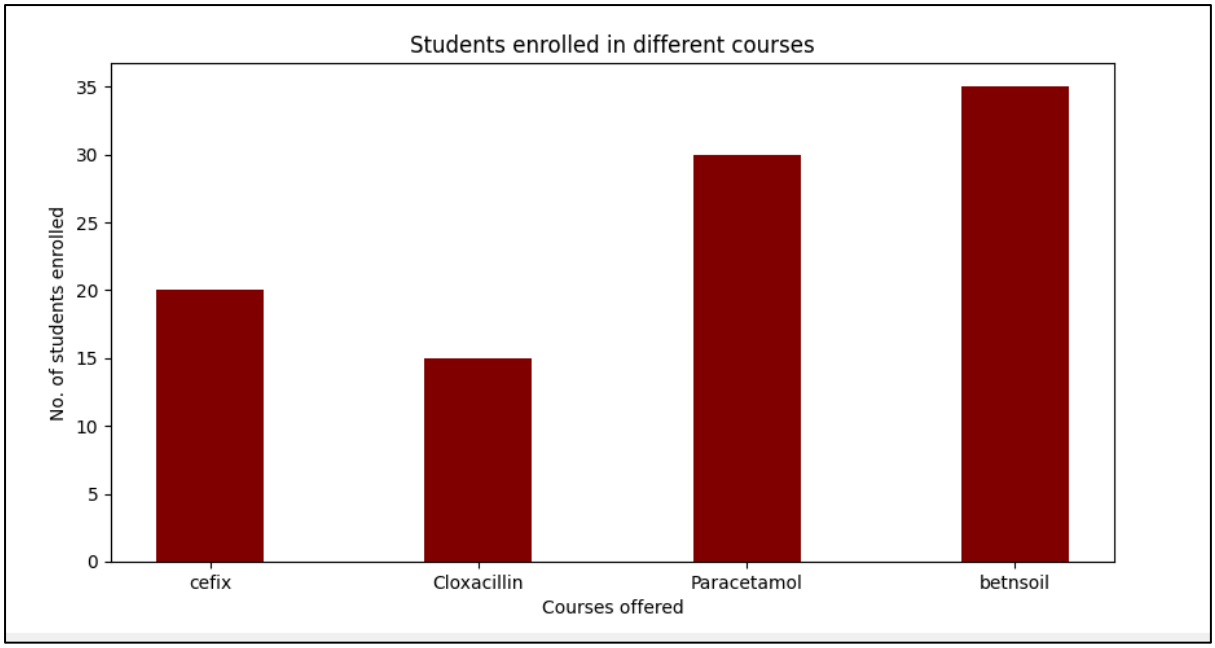

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Plot Matrix
age
spectacle-prescrip
astigmatism
tear-prod-rate
contact-lenses
Weka Explorer: Visualizing contact-lenses
X: age (Nom)
Y: age (Nom)
Colour: contact-lenses (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: contact-lenses
young
pre-presbyopic
presbyopic
PlotSize: [100]
PointSize: [1]
Jitter:
Class colour
soft hard none
Class Colour
soft hard none
Status

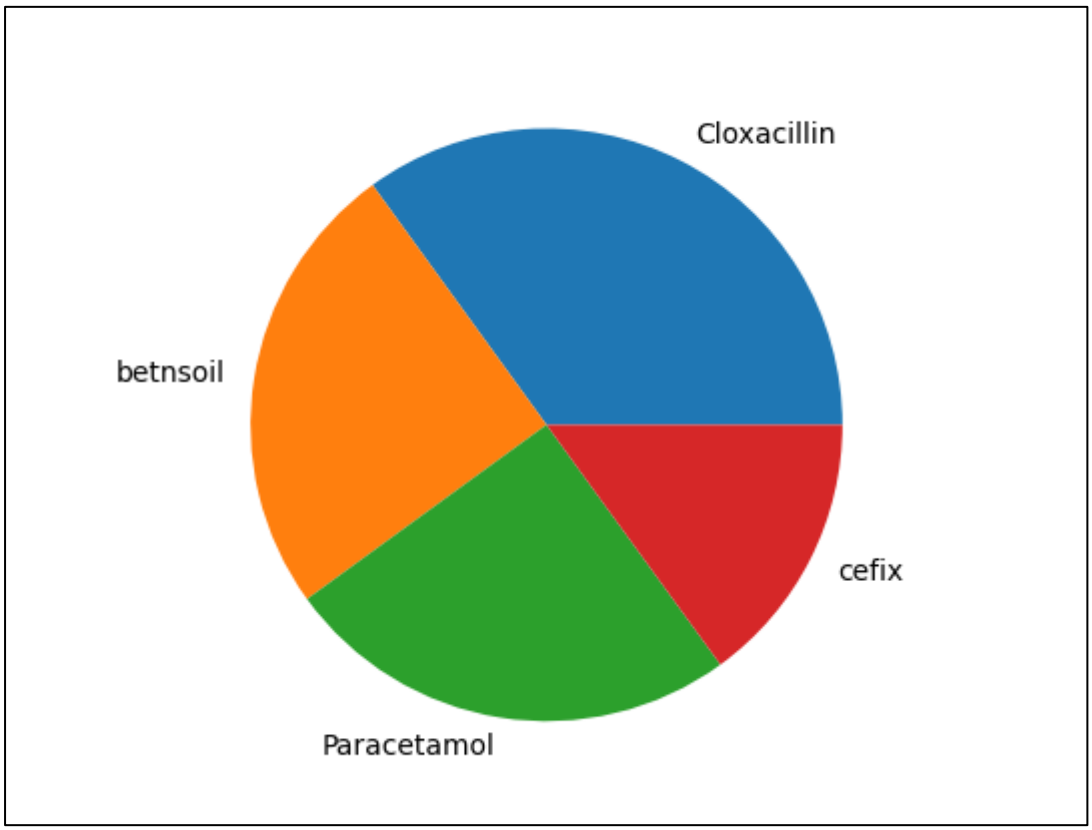
Experiment No: 07

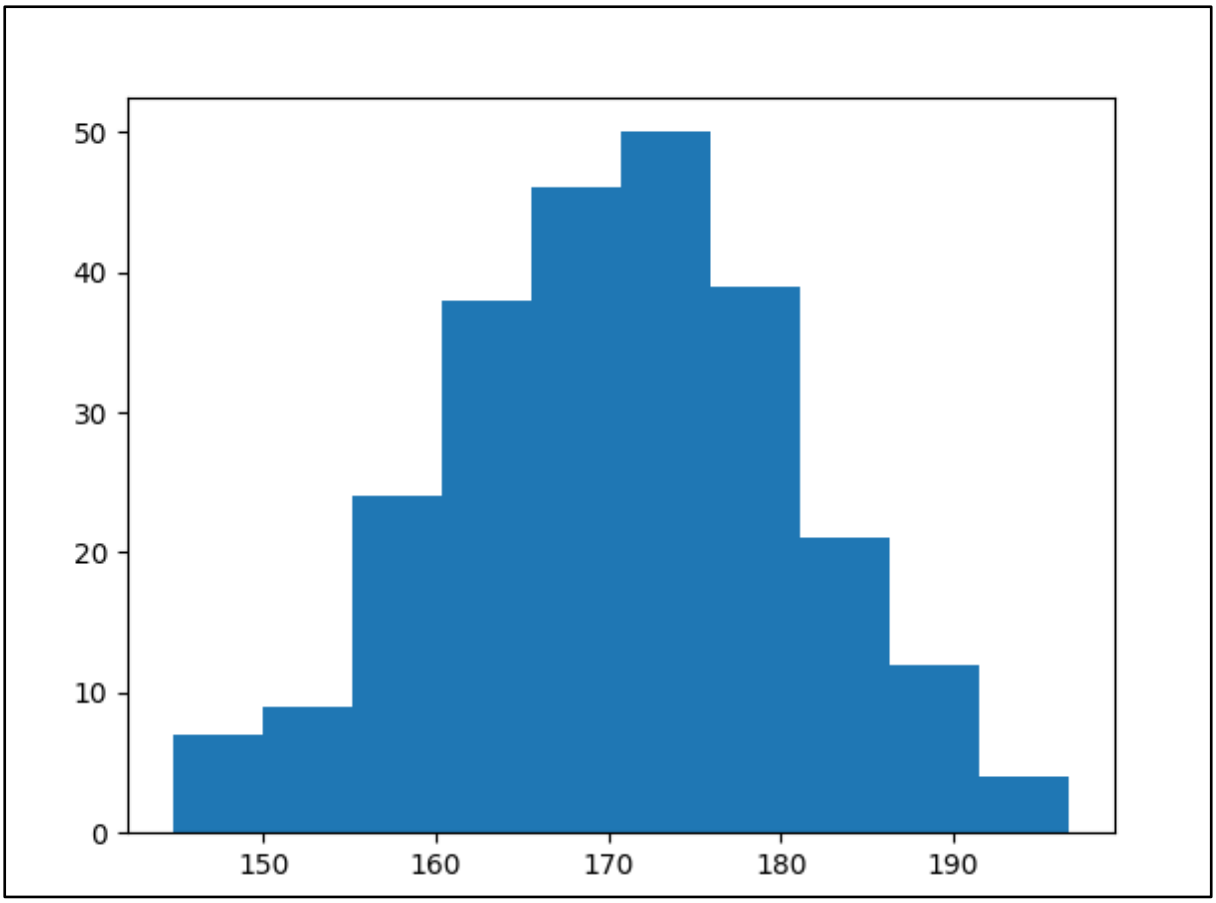
```
import numpy as np # to handle numeric data
import matplotlib.pyplot as plt # for visualization
import pandas as pd
data = pd.read_csv('pharama data.csv')
print(data)
datasubset = data.loc[:, ["orders","payment"]]
plt.figure(figsize=(5, 7))
plt.scatter(datasubset['orders'], datasubset['payment'], s=45,c='green')
plt.show()

import scipy.cluster.hierarchy as sch
plt.figure(figsize=(5, 7))
dendrogram = sch.dendrogram(sch.linkage(datasubset, method = 'ward'))
plt.title('Dendrogram') # title of the dendrogram

plt.xlabel('Pharma-Data') # label of the x-axis
plt.ylabel('Euclidean distances') # label of the y-axis
plt.show()
from sklearn.cluster import AgglomerativeClustering
cluster= AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean',
linkage = 'ward')
cluster. fit_predict(datasubset)
```

OUTPUT:

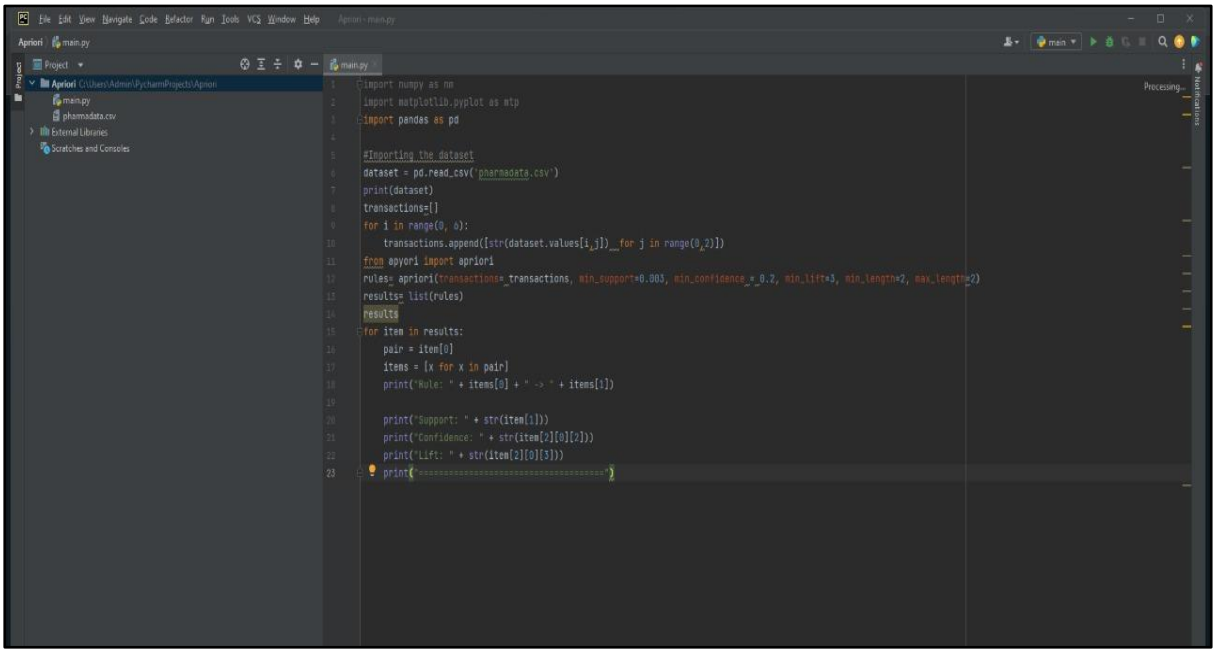



Students enrolled in different courses
No. of students enrolled
0
5
10
15
20
25
30
35
cefix
Cloxacillin
Paracetamol
betnsoil
Courses offered


Cloxacillin
betnsoil
cefix
Paracetamol

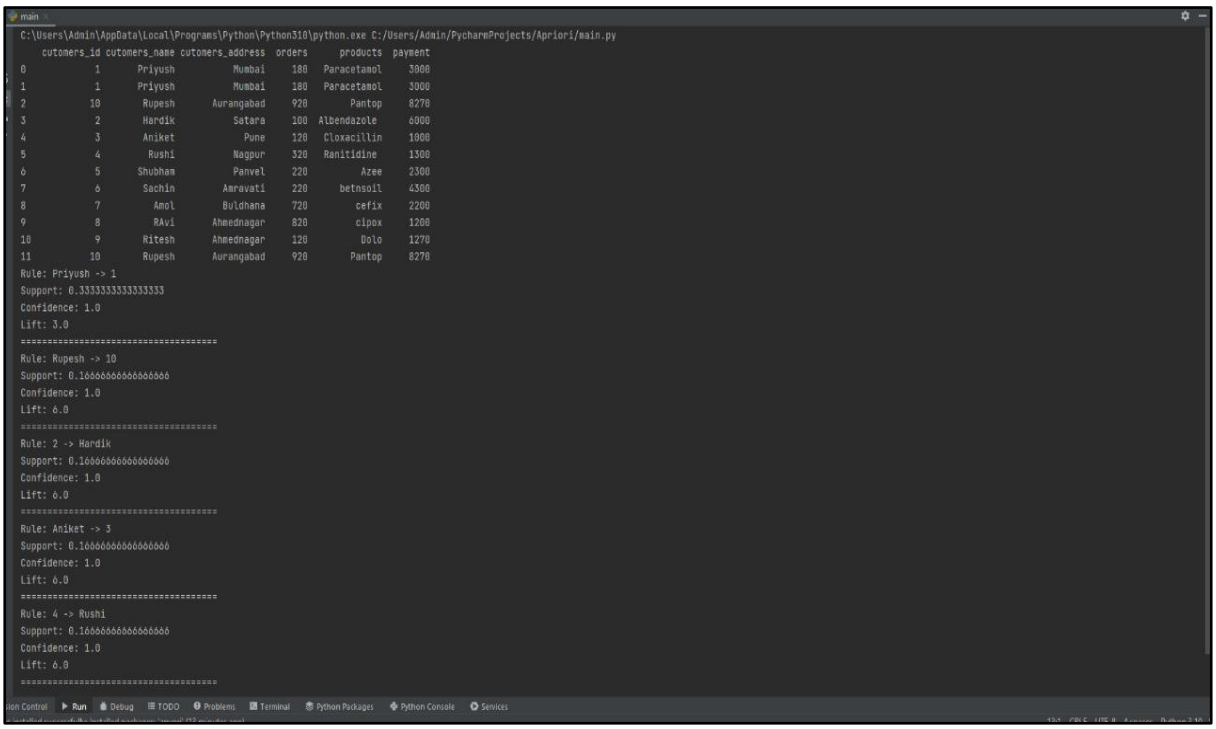

50
40
30
20
10
0
150
160
170
180
190

Experiment No: 08



```
import numpy as np
import matplotlib.pyplot as mtp
import pandas as pd

#Importing the dataset
dataset = pd.read_csv('pharmadata.csv')
print(dataset)
transactions=[]
for i in range(0, 6):
    transactions.append([str(dataset.values[i,j]) for j in range(0,2)])
from apyori import apriori
rules= apriori(transactions= transactions, min_support=0.003, min_confidence = 0.2, min_lift=3, min_length=2, max_length=2)
results= list(rules)
results
for item in results:
    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])

    print("Support: " + str(item[1]))
    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("=====================================")
```

OUTPUT:



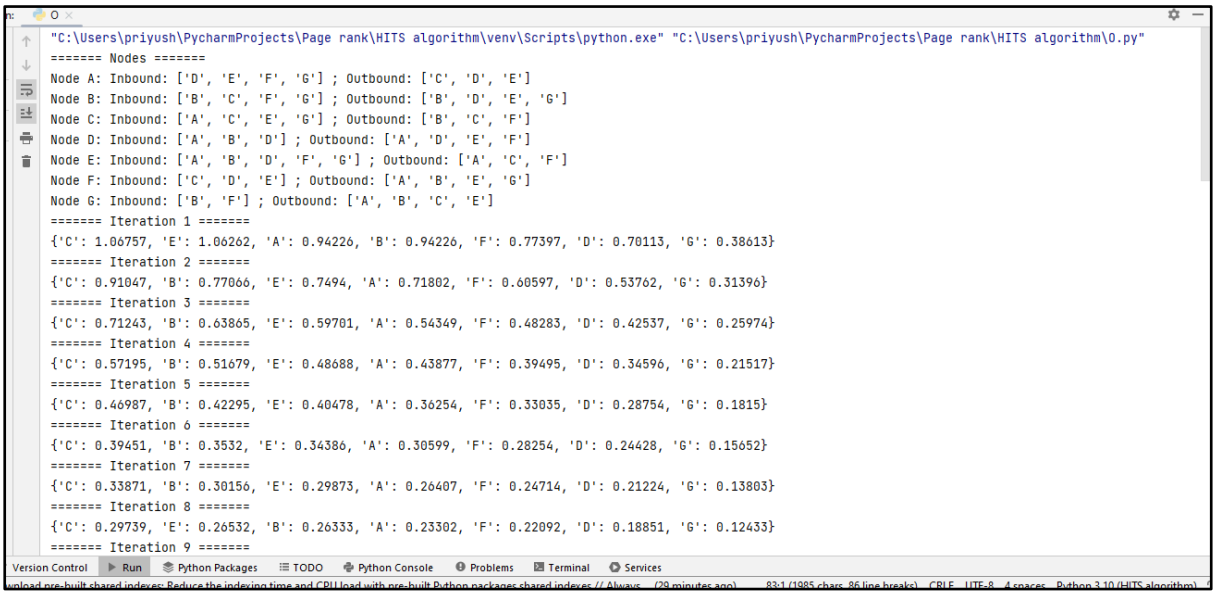
```
C:\Users\Admin\AppData\Local\Programs\Python\Python310\python.exe C:/Users/Admin/PycharmProjects/Apriori/main.py
    cutomers_id cutomers_name cutomers_address  orders     products  payment
0             1       Priyush          Mumbai     180   Paracetamol     3000
1             1       Priyush          Mumbai     180   Paracetamol     3000
2            10        Rupesh       Aurangabad     920       Pantop     8270
3             2        Hardik          Satara     100  Albendazole     6000
4             3        Aniket            Pune     120   Cloxacillin     1000
5             4         Rushi          Nagpur     320   Ranitidine     1300
6             5       Shubham          Panvel     220         Azee     2300
7             6        Sachin        Amravati     220      betnsoil     4300
8             7          Amol        Buldhana     720         cefix     2200
9             8          RAvi      Ahmednagar     820         cipox     1200
10            9         Ritesh      Ahmednagar     120          Dolo     1270
11           10        Rupesh       Aurangabad     920       Pantop     8270
Rule: Priyush -> 1
Support: 0.3333333333333333
Confidence: 1.0
Lift: 3.0
=====================================
Rule: Rupesh -> 10
Support: 0.16666666666666666
Confidence: 1.0
Lift: 6.0
=====================================
Rule: 2 -> Hardik
Support: 0.16666666666666666
Confidence: 1.0
Lift: 6.0
=====================================
Rule: Aniket -> 3
Support: 0.16666666666666666
Confidence: 1.0
Lift: 6.0
=====================================
Rule: 4 -> Rushi
Support: 0.16666666666666666
Confidence: 1.0
Lift: 6.0
=====================================
```

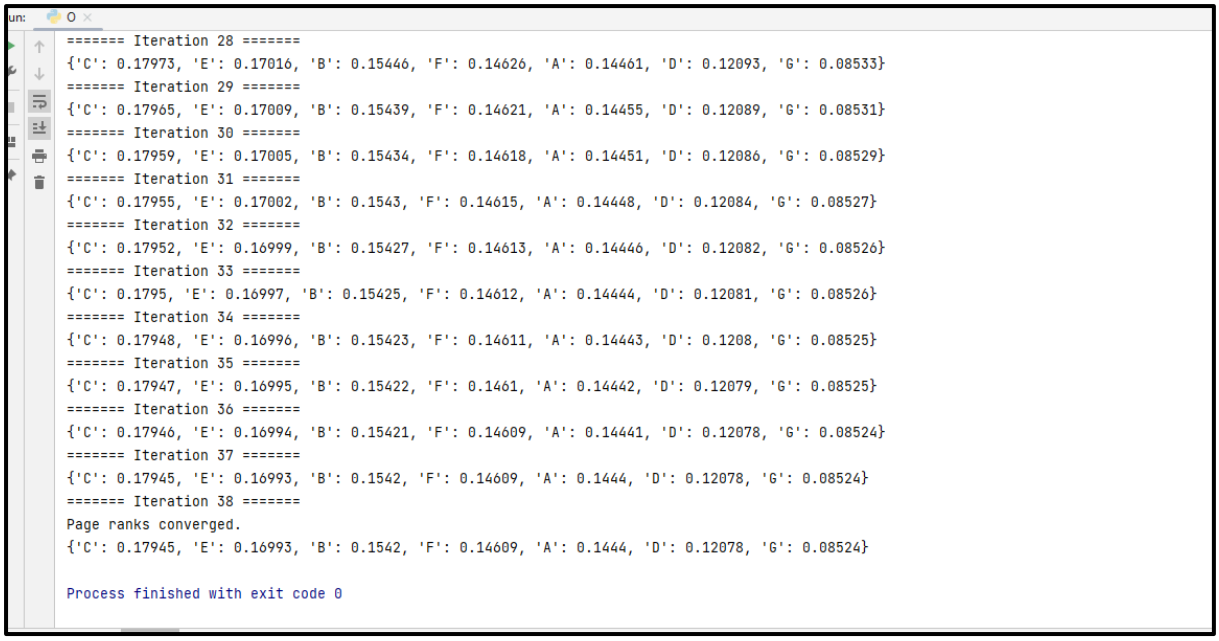
Experiment No: 09

```
import string

LETTERS = string.ascii_uppercase

graph = [
        [0, 0, 1, 1, 1, 0, 0],
        [0, 1, 0, 1, 1, 0, 1],
        [0, 1, 1, 0, 0, 1, 0],
        [1, 0, 0, 1, 1, 1, 0],
        [1, 0, 1, 0, 0, 1, 0],
        [1, 1, 0, 0, 1, 0, 1],
        [1, 1, 1, 0, 1, 0, 0],
]


class Node:
    def __init__(self, name):
        self.name = name
        self.inbound = []
        self.outbound = []

    def add_inbound(self, node):
        self.inbound.append(node)

    def add_outbound(self, node):
        self.outbound.append(node)

    def __repr__(self):
        return f"Node {self.name}: Inbound: {self.inbound} ; Outbound:
{self.outbound}"


def page_rank(nodes, limit=20, d=0.85):
    ranks = {}
    for node in nodes:
        ranks[node.name] = 1

    outbounds = {}
    for node in nodes:
        outbounds[node.name] = len(node.outbound)

    last_iteration_ranks = ranks.copy()

    i = 0
    while True:
        print(f"======= Iteration {i + 1} =======")
        for j, node in enumerate(nodes):
            ranks[node.name] = round((1 - d) / num_nodes + d * sum(
                [ranks[ib] / outbounds[ib] for ib in node.inbound]
            ), 5)

        ranks = dict(
            sorted(ranks.items(), key=lambda item: item[1], reverse=True))

        if ranks == last_iteration_ranks:
            print("Page ranks converged.")
            print(ranks)
            break
```

```
        else:
            last_iteration_ranks = ranks.copy()

        print(ranks)
        i += 1


def main():
    num_nodes = len(graph)

    names = list(LETTERS[:num_nodes])

    nodes = [Node(name) for name in names]

    for ri, row in enumerate(graph):
        for ci, col in enumerate(row):
            if col == 1:
                nodes[ci].add_inbound(names[ri])
                nodes[ri].add_outbound(names[ci])

    print("======== Nodes ========")
    for node in nodes:
        print(node)

    page_rank(nodes)


if __name__ == "__main__":
    num_nodes = len(graph)
    main()
```

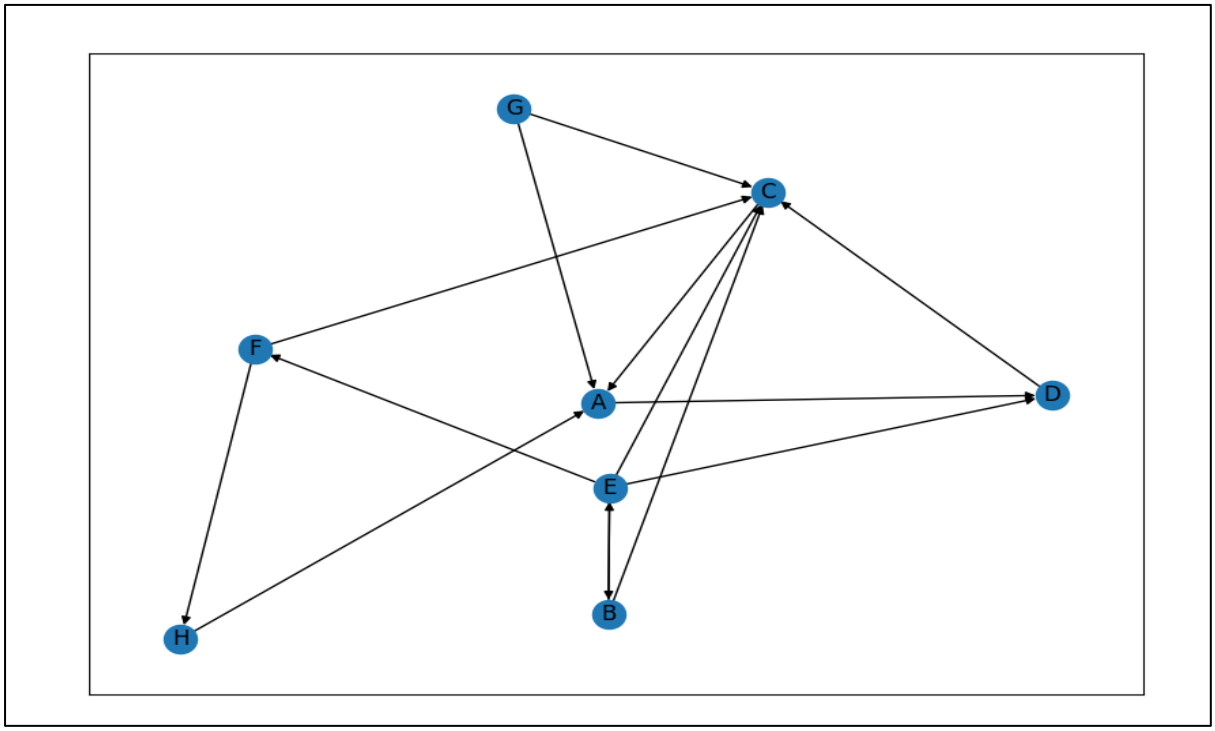
OUTPUT:




Run:
O
======= Iteration 28 =======
{'C': 0.17973, 'E': 0.17016, 'B': 0.15446, 'F': 0.14626, 'A': 0.14461, 'D': 0.12093, 'G': 0.08533}
======= Iteration 29 =======
{'C': 0.17965, 'E': 0.17009, 'B': 0.15439, 'F': 0.14621, 'A': 0.14455, 'D': 0.12089, 'G': 0.08531}
======= Iteration 30 =======
{'C': 0.17959, 'E': 0.17005, 'B': 0.15434, 'F': 0.14618, 'A': 0.14451, 'D': 0.12086, 'G': 0.08529}
======= Iteration 31 =======
{'C': 0.17955, 'E': 0.17002, 'B': 0.1543, 'F': 0.14615, 'A': 0.14448, 'D': 0.12084, 'G': 0.08527}
======= Iteration 32 =======
{'C': 0.17952, 'E': 0.16999, 'B': 0.15427, 'F': 0.14613, 'A': 0.14446, 'D': 0.12082, 'G': 0.08526}
======= Iteration 33 =======
{'C': 0.1795, 'E': 0.16997, 'B': 0.15425, 'F': 0.14612, 'A': 0.14444, 'D': 0.12081, 'G': 0.08526}
======= Iteration 34 =======
{'C': 0.17948, 'E': 0.16996, 'B': 0.15423, 'F': 0.14611, 'A': 0.14443, 'D': 0.1208, 'G': 0.08525}
======= Iteration 35 =======
{'C': 0.17947, 'E': 0.16995, 'B': 0.15422, 'F': 0.1461, 'A': 0.14442, 'D': 0.12079, 'G': 0.08525}
======= Iteration 36 =======
{'C': 0.17946, 'E': 0.16994, 'B': 0.15421, 'F': 0.14609, 'A': 0.14441, 'D': 0.12078, 'G': 0.08524}
======= Iteration 37 =======
{'C': 0.17945, 'E': 0.16993, 'B': 0.1542, 'F': 0.14609, 'A': 0.1444, 'D': 0.12078, 'G': 0.08524}
======= Iteration 38 =======
Page ranks converged.
{'C': 0.17945, 'E': 0.16993, 'B': 0.1542, 'F': 0.14609, 'A': 0.1444, 'D': 0.12078, 'G': 0.08524}
Process finished with exit code 0

Experiment No: 10

```
# 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')])

plt.figure(figsize =(10, 10))
nx.draw_networkx(G, with_labels = True)
plt.show()

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: ", hubs)
print("Authority Scores: ", authorities)
```




```
Run:  main
"C:\Users\priyush\PycharmProjects\Page rank\HITS algorithm\venv\Scripts\python.exe" "C:\Users\priyush\PycharmProjects\Page rank\HITS algorithm\main.py"
C:\Users\priyush\PycharmProjects\Page rank\HITS algorithm\venv\lib\site-packages\networkx\algorithms\link_analysis\hits_alg.py:78: FutureWarning:
 adjacency_matrix will return a scipy.sparse array instead of a matrix in Networkx 3.0.
  A = nx.adjacency_matrix(G, nodelist=list(G), dtype=float)
Hub Scores:  {'A': 0.04642540403219994, 'D': 0.13366037526115382, 'B': 0.15763599442967322, 'C': 0.03738913224642654, 'E': 0.25881445984686646, 'F':
 0.1576359944296732, 'H': 0.03738913224642654, 'G': 0.17104950750758036}
Authority Scores:  {'A': 0.10864044011724346, 'D': 0.13489685434358, 'B': 0.11437974073336447, 'C': 0.3883728003876181, 'E': 0.06966521184241478, 'F':
 0.11437974073336447, 'H': 0.06966521184241475, 'G': 0.0}

Process finished with exit code 0
```