

Experiment no 3

1) BIN_Mean

PROGRAM:

```
# import statsmodels.api as sm

import statistics

import math

from collections import OrderedDict

x =[]

print("enter the data")

x = list(map(float, input().split()))

print("enter the number of bins")

bi = int(input())

# X_dict will store the data in sorted order

X_dict = OrderedDict()

# x_old will store the original data

x_old ={}

# x_new will store the data after binning

x_new ={}

for i in range(len(x)):

    X_dict[i]= x[i]

    x_old[i]= x[i]

x_dict = sorted(X_dict.items(), key = lambda x: x[1])

# list of lists(bins)

binn =[]

# a variable to find the mean of each bin

avrg = 0

i = 0

k = 0

num_of_data_in_each_bin = int(math.ceil((len(x)/bi))

# performing binning

for g, h in X_dict.items():
```

```
if(i<num_of_data_in_each_bin):
    avrg = avrg + h
    i = i + 1
elif(i == num_of_data_in_each_bin):
    k = k + 1
    i = 0
    binn.append(round(avrg / num_of_data_in_each_bin, 3))
    avrg = 0
    avrg = avrg + h
    i = i + 1
rem = len(x)% bi
if(rem == 0):
    binn.append(round(avrg / num_of_data_in_each_bin, 3))
else:
    binn.append(round(avrg / rem, 3))

# store the new value of each data
i = 0
j = 0
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
        x_new[g]= binn[j]
        i = i + 1
    else:
        i = 0
        j = j + 1
        x_new[g]= binn[j]
        i = i + 1
print("number of data in each bin")
print(math.ceil(len(x)/bi))
for i in range(0, len(x)):
    print('index {2} old value {0} new value {1}'.format(x_old[i], x_new[i], i))
```

OUTPUT:

```
enter the data
34 84 42 55
enter the number of bins
4
number of data in each bin
1
index 0 old value 34.0 new value 34.0
index 1 old value 84.0 new value 84.0
index 2 old value 42.0 new value 42.0
index 3 old value 55.0 new value 55.0
```

2) BIN_Median

PROGRAM:

```
# import statsmodels.api as sm
import statistics
import math
from collections import OrderedDict
x=[]
print("enter the data")
x = list(map(float, input().split()))
print("enter the number of bins")
bi = int(input())
# X_dict will store the data in sorted order
X_dict = OrderedDict()
# x_old will store the original data
x_old={}
# x_new will store the data after binning
x_new={}
for i in range(len(x)):
    X_dict[i]= x[i]
    x_old[i]= x[i]

x_dict = sorted(X_dict.items(), key = lambda x: x[1])

# list of lists(bins)
binn=[]
# a variable to find the mean of each bin
avrg=[]
i = 0
k = 0
num_of_data_in_each_bin = int(math.ceil(len(x)/bi))

# performing binning
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
        avrg.append(h)
        i = i + 1
    elif(i == num_of_data_in_each_bin):
        k = k + 1
        i = 0
```

```
        binn.append(statistics.median(avrg))
        avrg=[]
        avrg.append(h)
        i = i + 1
    binn.append(statistics.median(avrg))
    # store the new value of each data
    i = 0
    j = 0
    for g, h in X_dict.items():
        if(i<num_of_data_in_each_bin):
            x_new[g]= round(binn[j], 3)
            i = i + 1
        else:
            i = 0
            j = j + 1
            x_new[g]= round(binn[j], 3)
            i = i + 1
    print("number of data in each bin")
    print(math.ceil(len(x)/bi))
    for i in range(0, len(x)):
        print('index {2} old value {0} new value {1}'.format(x_old[i], x_new[i], i))
```

OUTPUT:

```
enter the data
111 32 53 54
enter the number of bins
5
number of data in each bin
1
index 0 old value 111.0 new value 111.0
index 1 old value 32.0 new value 32.0
index 2 old value 53.0 new value 53.0
index 3 old value 54.0 new value 54.0
```

3) BIN_Boundary

PROGRAM:

```
# import statsmodels.api as sm

import statistics

import math

from collections import OrderedDict

x=[]

print("enter the data")

x = list(map(float, input().split()))

print("enter the number of bins")
```

```
bi = int(input())

# X_dict will store the data in sorted order
X_dict = OrderedDict()

# x_old will store the original data
x_old = {}

# x_new will store the data after binning
x_new = {}

for i in range(len(x)):
    X_dict[i] = x[i]
    x_old[i] = x[i]

x_dict = sorted(X_dict.items(), key = lambda x: x[1])

# list of lists(bins)
binn = []

# a variable to find the mean of each bin
avrg = []

i = 0
k = 0

num_of_data_in_each_bin = int(math.ceil(len(x)/bi))

for g, h in X_dict.items():
    if(i < num_of_data_in_each_bin):
        avrg.append(h)
        i = i + 1
    elif(i == num_of_data_in_each_bin):
        k = k + 1
        i = 0
        binn.append([min(avrg), max(avrg)])
        avrg = []
        avrg.append(h)
        i = i + 1

binn.append([min(avrg), max(avrg)])

i = 0
j = 0
```

```
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
        if(abs(h-binn[j][0]) >= abs(h-binn[j][1])):
            x_new[g]= binn[j][1]
            i = i + 1
        else:
            x_new[g]= binn[j][0]
            i = i + 1
    else:
        i = 0
        j = j + 1
        if(abs(h-binn[j][0]) >= abs(h-binn[j][1])):
            x_new[g]= binn[j][1]
        else:
            x_new[g]= binn[j][0]
        i = i + 1
print("number of data in each bin")
print(math.ceil(len(x)/bi))
for i in range(0, len(x)):
    print('index {2} old value {0} new value {1}'.format(x_old[i], x_new[i], i))
```

OUTPUT:

enter the data

43 53 5648 64

enter the number of bins

6

number of data in each bin

1

index 0 old value 43.0 new value 43.0

index 1 old value 53.0 new value 53.0

index 2 old value 5648.0 new value 5648.0

index 3 old value 64.0 new value 64.0

Experiment no 4

PROGRAM:

```
# Naive Bayes Algorithm
import pandas as pd
import csv

def pci(data):
    class_count = [0, 0]
    for i in range(len(data)):
        if data.iloc[i, -1] == 'Yes':
            class_count[0] += 1
        else:
            class_count[1] += 1
    return class_count[0], class_count[1]

def pcix(data, x, c1, c2):
    p1, p2 = 1, 1
    count_yes = [0 for i in range(len(data.columns) - 1)]
    count_no = [0 for i in range(len(data.columns) - 1)]
    for i in range(len(data)):
        for j in range(len(data.columns) - 1):
            if data.iloc[i, j] == x[j]:
                if data.iloc[i, -1] == 'Yes':
                    count_yes[j] += 1
                else:
                    count_no[j] += 1
    for i in range(len(count_yes)):
        p1 = p1 * count_yes[i] / c1
        p2 = p2 * count_no[i] / c2
    return p1, p2

data = pd.read_csv('NB.csv')
data = data.iloc[:, 1:]
X = input('Enter tuple to classify: ')
X = X.split(',')

```

```
c1, c2 = pci(data)
p1, p2 = pcix(data, X, c1, c2)
if (p1 * c1 / len(data)) > (p2 * c2 / len(data)):
    print('Buys')
else:
    print('No Buy')
```

OUTPUT:

Enter tuple to classify: Youth , Medium, Yes, Excellent

No buy

Experiment no 5

PROGRAM:

```
# importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import csv

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

#finding optimal number of clusters using the elbow method

from sklearn.cluster import KMeans

wcss_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):

    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)

    kmeans.fit(x)

    wcss_list.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss_list)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters(k)')
plt.ylabel('wcss_list')

plt.show()

#training the K-means model on a dataset

kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)

y_predict= kmeans.fit_predict(x)

#visualizing the clusters

plt.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first
cluster

plt.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second
cluster

plt.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster

plt.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth
cluster
```

```
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster

mtp.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:, 1], s = 300, c = 'yellow', label = 'Centroid')

mtp.title('Clusters of customers')

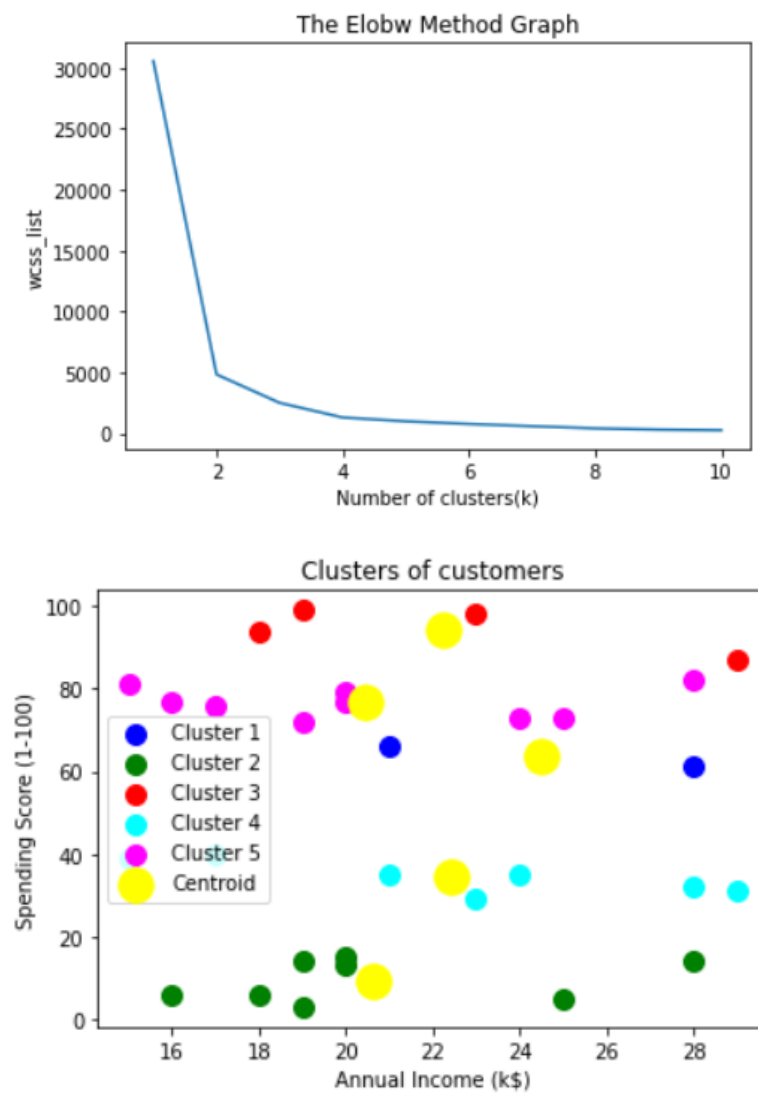
mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend()

mtp.show()
```

OUTPUT:



Experiment no 6

Classification:

The image displays two screenshots from the Weka software interface. The top screenshot shows the 'Weka Explorer' window with the 'Classifier' tab selected. The classifier used is 'J48 - C 0.25 - M 2'. The test options are set to 'Cross-validation' with 'Folds' set to 10. The classifier output shows a stratified cross-validation summary and a detailed accuracy by class table.

Classifier output

```

| | | pres > 61
| | | | age <= 30: tested_negative (40.0/13.0)
| | | | age > 30: tested_positive (60.0/17.0)
| | | | plas > 157: tested_positive (92.0/12.0)
Number of Leaves : 20
Size of the tree : 39
Time taken to build model: 0.08 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 567 73.8281 %
Incorrectly Classified Instances 201 26.1719 %
Kappa statistic 0.4164
Mean absolute error 0.3158
Root mean squared error 0.4463
Relative absolute error 69.4841 %
Root relative squared error 93.6293 %
Total Number of Instances 768

=== Detailed Accuracy By Class ===

```

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	PRC Area	Class
Weighted Avg.	0.738	0.327	0.735	0.738	0.736	0.417	0.751	tested_negative
							0.727	tested_positive

Confusion Matrix

```

a b <-- classified as
407 93 | a = tested_negative
108 160 | b = tested_positive

```

The bottom screenshot shows the 'Weka Classifier Tree Visualizer' window for the 'J48 - C 0.25 - M 2' classifier. It displays a decision tree structure for the 'pima_diabetes' dataset. The root node is 'plas', which splits on the value 127. The left branch leads to a node 'mass', which splits on 28.4. The right branch leads to a node 'mass', which splits on 29.8. The tree continues to split based on various attributes like 'age', 'preg', 'ped', and 'mass' until it reaches leaf nodes representing the final classification (tested_negative or tested_positive) with associated counts.

Clustering:

The screenshot shows the Weka Explorer interface with the SimpleKMeans clustering algorithm applied to the diabetes dataset. The 'Clusterer' tab is active, and the 'Cluster mode' is set to 'Use training set'. The 'Result list' on the left shows '17:21:47 - SimpleKMeans' as the selected result.

Clusterer output

Within cluster sum of squared errors: 149.5177664581119

Initial starting points (random):

Cluster 0: 1,126,56,29,152,20.7,0.801,21,tested_negative
Cluster 1: 8,95,72,0,0,36.8,0.485,57,tested_negative

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute	Full data	Cluster#	0	1
	(748.0)		(500.0)	(248.0)
preg	3.8451	3.298	4.8657	
plas	120.8945	109.98	141.2575	
pres	69.1055	68.184	70.8246	
skin	20.5365	19.664	22.1642	
insu	79.7595	60.792	100.3358	
mass	31.9924	30.3042	35.1425	
pedi	0.4719	0.4297	0.5505	
age	33.2409	31.19	37.0672	
class			tested_negative	tested_negative tested_positive

Time taken to build model (full training data) : 0.07 seconds

=== Model and evaluation on training set ===

Clustered Instances

Cluster	Count	Percentage
0	500	(65%)
1	248	(35%)

Association:

The screenshot shows the Weka Explorer interface with the Apriori association algorithm applied to the diabetes dataset. The 'Associate' tab is active, and the 'Associator' is set to 'Apriori'. The 'Result list' on the left shows '18:45:36 - FilteredAssociator' as the selected result.

Associator output

=== Run information ===

Scheme: weka.associations.FilteredAssociator -P "weka.filters.MultiFilter -P "weka.filters.unsupervised.attribute.ReplaceMissingValues \" -S 1" -c -1 -W weka.associations.Apriori -

Relation: pima_diabetes

Instances: 768

Attributes: 9

preg
plas
pres
skin
insu
mass
pedi
age
class

Experiment no 7

PROGRAM:

```
import numpy as np

X = np.array([[8,4],[10,15],[16,12],[22,12],[30,30],[85,70],[74,80],[60,76],[65,50],[85,92]])

import matplotlib.pyplot as plt

labels = range(1,11)

plt.figure(figsize=(10,7))

plt.subplots_adjust(bottom=0.1)

plt.scatter(X[:,0],X[:,1],label='True Position')

for label,x,y in zip(labels,X[:,0],X[:,1]):

    plt.annotate(label,xy=(x,y),xytext=(-3,3),textcoords='offset points',ha='right',va='bottom')

plt.show()
```

1) Single Linkage

```
from scipy.cluster.hierarchy import dendrogram,linkage

from matplotlib import pyplot as plt

linked = linkage(X,'single')

labelList = range(1,11)

plt.figure(figsize=(10,7))

dendrogram(linked,orientation='top',labels=labelList,
distance_sort='descending',show_leaf_counts=True)

plt.show()
```

2) Complete Linkage

```
from scipy.cluster.hierarchy import dendrogram,linkage

from matplotlib import pyplot as plt

linked = linkage(X,'complete')

labelList = range(1,11)

plt.figure(figsize=(10,7))

dendrogram(linked,orientation='top',labels=labelList,
distance_sort='descending',show_leaf_counts=True)

plt.show()
```

3) Average Linkage

```
from scipy.cluster.hierarchy import dendrogram,linkage

from matplotlib import pyplot as plt

linked = linkage(X,'average')

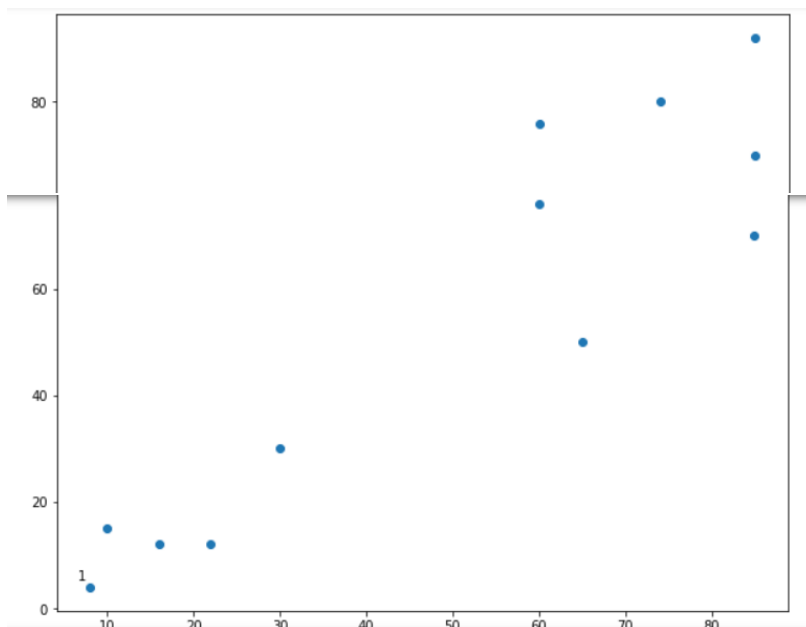
labelList = range(1,11)

plt.figure(figsize=(10,7))

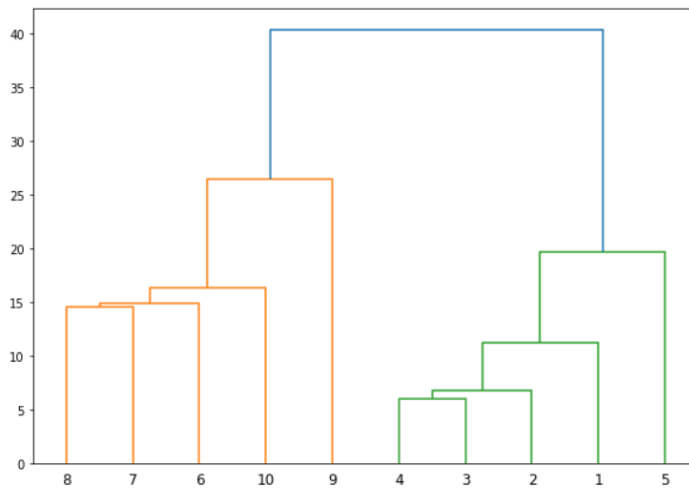
dendrogram(linked,orientation='top',labels=labelList,
distance_sort='descending',show_leaf_counts=True)

plt.show()
```

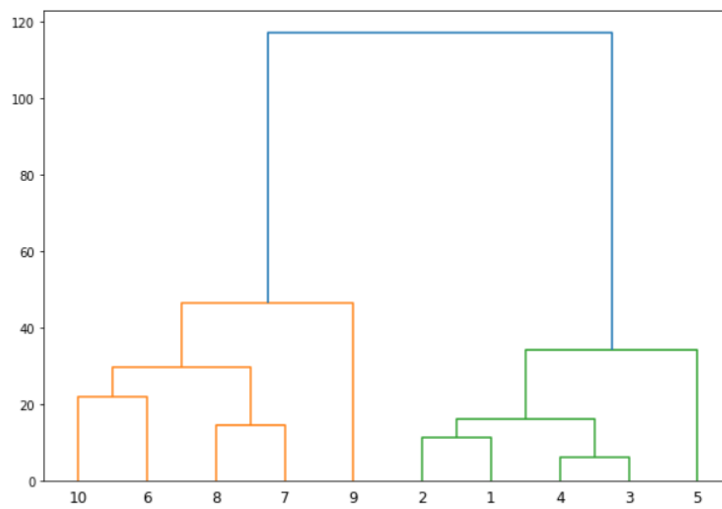
OUTPUT:



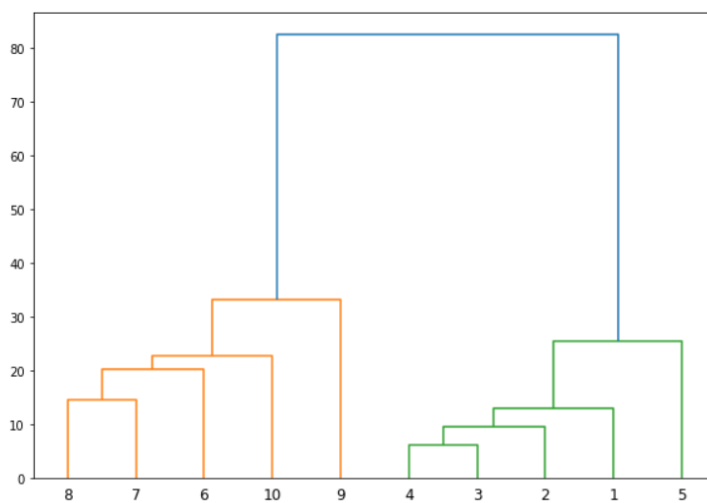
1) Single Linkage



2) Complete Linkage



3) Average Linkage



Experiment no 8

PROGRAM:

```
data = [  
    ['T100',['I1','I2','I5']],  
    ['T200',['I2','I4']],  
    ['T300',['I2','I3']],  
    ['T400',['I1','I2','I4']],  
    ['T500',['I1','I3']],  
    ['T600',['I2','I3']],  
    ['T700',['I1','I3']],  
    ['T800',['I1','I2','I3','I5']],  
    ['T900',['I1','I2','I3']]  
]  
  
init = []  
  
for i in data:  
    for q in i[1]:  
        if(q not in init):  
            init.append(q)  
    init = sorted(init)  
    print(init)  
  
sp = 0.4  
  
s = int(sp*len(init))  
  
s  
  
from collections import Counter  
  
c = Counter()  
  
for i in init:  
    for d in data:  
        if(i in d[1]):  
            c[i]+=1  
    print("C1:")  
  
for i in c:  
    print(str([i])+" ": "+str(c[i]))
```



```
print()

l = Counter()

for i in c:
    if(c[i] >= s):
        l[frozenset([i])]+=c[i]
print("L1:")

for i in l:
    print(str(list(i))+": "+str(l[i]))

print()

pl = l

pos = 1

for count in range (2,1000):
    nc = set()
    temp = list(l)
    for i in range(0,len(temp)):
        for j in range(i+1,len(temp)):
            t = temp[i].union(temp[j])
            if(len(t) == count):
                nc.add(temp[i].union(temp[j]))
    nc = list(nc)
    c = Counter()
    for i in nc:
        c[i] = 0
    for q in data:
        temp = set(q[1])
        if(i.issubset(temp)):
            c[i]+=1
    print("C"+str(count)+":")
    for i in c:
        print(str(list(i))+": "+str(c[i]))
    print()
    l = Counter()
```

```
for i in c:
    if(c[i] >= s):
        l[i]+=c[i]
    print("L"+str(count)+":")
    for i in l:
        print(str(list(i))+": "+str(l[i]))
    print()
    if(len(l) == 0):
        break
    pl = l
    pos = count
    print("Result: ")
    print("L"+str(pos)+":")
    for i in pl:
        print(str(list(i))+": "+str(pl[i]))
    print()
    from itertools import combinations
    for l in pl:
        c = [frozenset(q) for q in combinations(l,len(l)-1)]
        mmax = 0
        for a in c:
            b = l-a
            ab = l
            sab = 0
            sa = 0
            sb = 0
            for q in data:
                temp = set(q[1])
                if(a.issubset(temp)):
                    sa+=1
                if(b.issubset(temp)):
                    sb+=1
```

```
if(ab.issubset(temp)):
    sab+=1
    temp = sab/sa*100
    if(temp > mmax):
        mmax = temp
    temp = sab/sb*100
    if(temp > mmax):
        mmax = temp
    print(str(list(a))+ " -> " +str(list(b))+ " = " +str(sab/sa*100)+"%")
    print(str(list(b))+ " -> " +str(list(a))+ " = " +str(sab/sb*100)+"%")
    curr = 1
    print("choosing:", end=' ')
    for a in c:
        b = l-a
        ab = l
        sab = 0
        sa = 0
        sb = 0
        for q in data:
            temp = set(q[1])
            if(a.issubset(temp)):
                sa+=1
            if(b.issubset(temp)):
                sb+=1
            if(ab.issubset(temp)):
                sab+=1
            temp = sab/sa*100
            if(temp == mmax):
                print(curr, end = ' ')
                curr += 1
            temp = sab/sb*100
            if(temp == mmax):
```

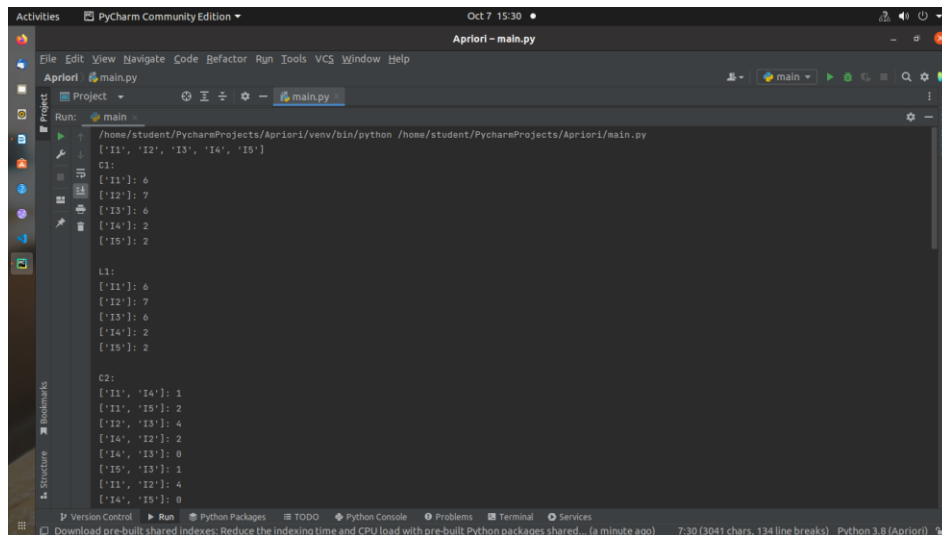
```
print(curr, end = ' ')
```

```
curr += 1
```

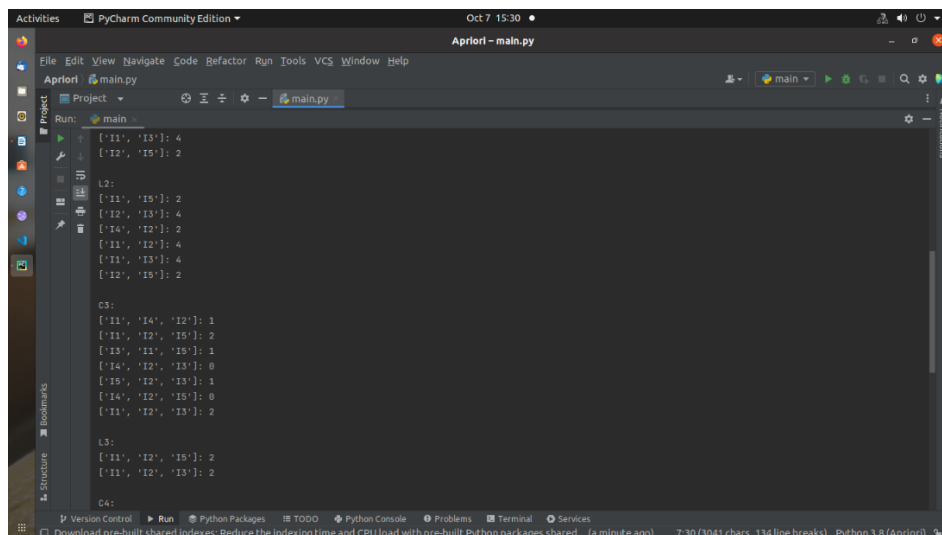
```
print()
```

```
print()
```

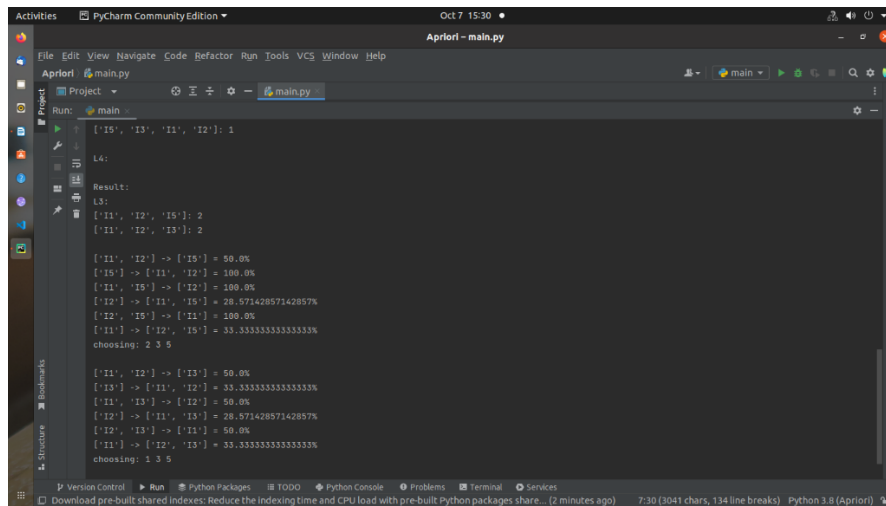
OUTPUT:



```
Apriori - main.py
Run: main
/home/student/PycharmProjects/Apriori/venv/bin/python /home/student/PycharmProjects/Apriori/main.py
['11', '12', '13', '14', '15']
C1:
['11']: 6
['12']: 7
['13']: 6
['14']: 2
['15']: 2
L1:
['11']: 6
['12']: 7
['13']: 6
['14']: 2
['15']: 2
C2:
['11', '14']: 1
['11', '15']: 2
['12', '13']: 4
['14', '12']: 2
['14', '13']: 0
['15', '13']: 1
['11', '12']: 4
['14', '15']: 0
```



```
Apriori - main.py
Run: main
['11', '13']: 4
['12', '15']: 2
L2:
['11', '15']: 2
['12', '13']: 4
['14', '12']: 2
['11', '12']: 4
['11', '13']: 4
['12', '15']: 2
C3:
['11', '14', '12']: 1
['11', '12', '15']: 2
['13', '11', '15']: 1
['14', '12', '13']: 0
['15', '12', '13']: 1
['14', '12', '15']: 0
['11', '12', '13']: 2
L3:
['11', '12', '15']: 2
['11', '12', '13']: 2
C4:
```



The screenshot shows the PyCharm Community Edition interface. The main editor displays a Python script named `main.py` with the following content:

```
[ '15', '13', '11', '12' ]: 1  
  
L4:  
  
Result:  
L5:  
[ '11', '12', '15' ]: 2  
[ '11', '12', '13' ]: 2  
  
[ '11', '12' ] -> [ '15' ] = 50.0%  
[ '15' ] -> [ '11', '12' ] = 100.0%  
[ '11', '15' ] -> [ '12' ] = 100.0%  
[ '12' ] -> [ '11', '15' ] = 28.57142857142857%  
[ '12', '15' ] -> [ '11' ] = 100.0%  
[ '11' ] -> [ '12', '15' ] = 33.33333333333333%  
choosing: 2 3 5  
  
[ '11', '12' ] -> [ '13' ] = 50.0%  
[ '13' ] -> [ '11', '12' ] = 33.33333333333333%  
[ '11', '13' ] -> [ '12' ] = 50.0%  
[ '12' ] -> [ '11', '13' ] = 28.57142857142857%  
[ '12', '13' ] -> [ '11' ] = 50.0%  
[ '11' ] -> [ '12', '13' ] = 33.33333333333333%  
choosing: 1 3 5
```

The Run console at the bottom shows the output of the script, which matches the content of the `main.py` file. The status bar at the bottom indicates the file is 7:30 (3041 chars, 134 line breaks) and is using Python 3.8 (Apriori).

Experiment no 9

PROGRAM:

```
import networkx as nx
import numpy as np
from numpy import array
import matplotlib.pyplot as plt
with open('./dataset/HITS.txt') as f:
    lines = f.readlines()
    G = nx.DiGraph()
for line in lines:
    t = tuple(line.strip().split(','))
    G.add_edge(*t)
h, a = nx.hits(G, max_iter=100)
h = dict(sorted(h.items(), key=lambda x:x[0]))
a = dict(sorted(a.items(), key=lambda x:x[0]))
print(np.round(list(a.values()), 3))
print(np.round(list(h.values()), 3))
pr = nx.pagerank(G)
pr = dict(sorted(pr.items(), key=lambda x: x[0]))
print(np.round(list(pr.values()), 3))
sim = nx.simrank_similarity(G)
lol = [[sim[u][v] for v in sorted(sim[u])]
        for u in sorted(sim)]
sim_array = np.round(array(lol), 3)
print(sim_array)
nx.draw(G, with_labels=True, node_size=2000, edge_color='#eb4034', width=3,
font_size=16, font_weight=500, arrowsize=20, alpha=0.8)
plt.savefig("graph.png")
```

OUTPUT:

```
[0.088 0.187 0.369 0.128 0.059 0.11 0. 0.059]
[0.043 0.144 0.03 0.187 0.268 0.144 0.154 0.03 ]
[0.241 0.137 0.218 0.24 0.077 0.035 0.019 0.034]
-----
[[1. 0.208 0.221 0.193 0.217 0.269 0. 0.171]
 [0.208 1. 0.355 0.369 0.302 0.553 0. 0.369]
 [0.221 0.355 1. 0.242 0.4 0.325 0. 0.427]
 [0.193 0.369 0.242 1. 0.229 0.548 0. 0.244]
 [0.217 0.302 0.4 0.229 1. 0.272 0. 0.498]
 [0.269 0.553 0.325 0.548 0.272 1. 0. 0.245]
 [0. 0. 0. 0. 0. 0. 1. 0. ]
 [0.171 0.369 0.427 0.244 0.498 0.245 0. 1. ]]
```

Experiment no 10

PROGRAM:

```
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)

hubs, authorities = nx.hits(G, max_iter=50, normalized=True)

print("Hub Scores:", hubs)

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

OUTPUT:

Hub Scores: {'A': 0.04642540403219997, 'D': 0.1336603752611538, 'B': 0.1576359944296732, 'C': 0.03738913224642651, 'E': 0.2588144598468665, 'F': 0.1576359944296732, 'H': 0.03738913224642651, 'G': 0.1710495075075803}

Authority Scores: {'A': 0.10864044011724336, 'D': 0.13489685434358006, 'B': 0.1143797407333645, 'C': 0.388372800387618, 'E': 0.06966521184241475, 'F': 0.1143797407333645, 'H': 0.06966521184241475, 'G': 0.0}

