**INDEX**

|  |  |  |
| --- | --- | --- |
| S. NO. | PROBLEM DESCRIPTION | PAGE |
| 1. | [Write](#_i2tdm2c45o5i) a program to implement APRIORI algorithm | 2 |
| 2. | [Write a program to implement FP-TREE-GROWTH algorithm.](#_r8qa3g5y6af0) | 4 |
| 3. | [Write](#_cx3qk3wrx4fa) a program to implement K-NEAREST NEIGHBOURS (KNN). | 8 |
| 4. | Create a NAÏVE BAYES Classifier for the given dataset. | 11 |
| 5. | Implement DATA PROCESSING and CLASSIFICATION over the dataset. | 14 |
| 6. | Extract FEATURES from Random Act of Pizza (RAOP) dataset. | 17 |
| 7. | Implement K-MEANS Algorithm over the given dataset |  |
| 8. | Implement DECISION TREE algorithm Using ID3 algorithm | 28 |
| 9. | Implement DECISION TREE algorithm using RANDOM FOREST procedure. | 33 |

PROGRAM NO. 1

[Write](#_i2tdm2c45o5i) a program to implement APRIORI algorithm

PROGRAM NO. 2

Write a program to implement FP-TREE-GROWTH algorithm.

from collections import OrderedDict

def find\_frequent\_items(transactions, threshold):

items = {}

for transaction in transactions:

transaction=list(set(transaction))

for item in transaction:

if item in items:

items[item] += 1

else:

items[item] = 1

for key in list(items.keys()):

if items[key] < threshold:

del items[key]

return items

transactions = [['m', 'o', 'n', 'k', 'e','y'],

['d','o','n','k','e','y'],

['m','u','c','k','y'],

['m','a','k','e'],

['c','o','o','k','i','e']]

patterns=find\_frequent\_items(transactions, 3)

d={}

for i in sorted(patterns.keys()):

d[i]=patterns[i]

s = [(k, d[k]) for k in sorted(d, key=d.get, reverse=True)]

d={}

for i in s:

d[i[0]]=i[1]

patterns=d

print("Transaction Table")

for i in range(len(transactions)):

print(i,"|", ''.join(transactions[i]))

print("\nWith min sup = 3\n\nTable 1\n", patterns)

table1=['' for i in range(len(transactions))]

for i in range(len(transactions)):

for item in patterns.keys():

if item in transactions[i]:

table1[i]=table1[i]+item

print("\n".join(table1))

ends = list(patterns.keys())[::-1]

print ("items\n\n","\n".join(ends))

condi = []

for i in ends:

l = []

for t in table1:

if i in t:

index = t.find(i)

if t[:index] not in l:

l.append(t[:index])

l.append(1)

else:

id = l.index(t[:index])

l[id+1]+=1

condi.append(l)

condi=condi[:-1]

print("Conditional Pattern Base")

for i in range(len(condi)):

print(ends[i],"|",condi[i])

fp = []

from difflib import SequenceMatcher

for i in condi:

if len(i)==2:

fp.append(i)

elif len(i)==4:#only 2 elements

pat = i[::2]

sum\_list = i[1::2]

val = sum(sum\_list)

c = 1

while pat[0][:c]==pat[1][:c]:

c+=1

fp.append([pat[0][:c-1],val])

else:

sum\_list = i[1::2]

val = sum(sum\_list)

st=''

for a in i[0]:

n=0

while(n<len(i)-1):

if a not in i[n]:

break

else:

st+= a

n+=2

a = i[0][0]

if st.count(a)==len(i)/2:

fp.append([a,val])

print("Conditional FP")

for i in range(len(fp)):

print (ends[i],"|",fp[i])

from itertools import combinations

ans=[]

for i in range(len(fp)):

l = list(fp[i][0])

l.append(ends[i])

c=2

a=[]

while c<=len(l):

com = combinations(l,c)

for j in list(com):

if ends[i] in j:

a.append(str(j) +" :" + str(fp[i][1]))

#a.append(list(com)[:])

c+=1

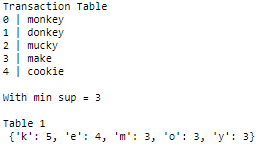
ans.append(a)

print("Items | FP generated")

for i in range(len(fp)):

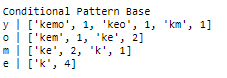
print (ends[i], "|", ans[i])

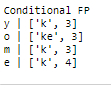
**OUTPUT**

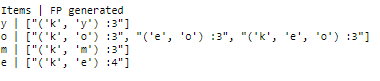












PROGRAM NO. 3

Write a program to implement K-NEAREST NEIGHBOURS (KNN).

pts\_table = [

[2,10],

[3,12],

[4,14],

[5,16],

[6,20],

[7,22],

[8,25],

[10,28],

[100,200],

[101,170]

]

print(len(pts\_table))

k = 2

print('Taking k = 2')

w, h = 4, len(pts\_table)

Matrix = [[0 for x in range(w)] for y in range(h)]

import math

centroid1 = pts\_table[0]

centroid2 = pts\_table[1]

centroid3 = pts\_table[2]

i=10

while i>0:

for x in range(len(pts\_table)):

temp = pts\_table[x]

distsq = (float)((centroid1[0]-temp[0])\*\*2 + (centroid1[1]-temp[1])\*\*2)

dist = math.sqrt(distsq)

Matrix[x][0] = dist

distsq = (float)((centroid2[0]-temp[0])\*\*2 + (centroid2[1]-temp[1])\*\*2)

dist = math.sqrt(distsq)

Matrix[x][1] = dist

distsq = (float)((centroid3[0]-temp[0])\*\*2 + (centroid3[1]-temp[1])\*\*2)

dist = math.sqrt(distsq)

Matrix[x][2] = dist

if Matrix[x][0] <= Matrix[x][1] and Matrix[x][0] <= Matrix[x][2]:

Matrix[x][3] = 1

elif Matrix[x][1] <= Matrix[x][0] and Matrix[x][0] <= Matrix[x][2]:

Matrix[x][3] = 2

elif Matrix[x][2] <= Matrix[x][0] and Matrix[x][2] <= Matrix[x][1]:

Matrix[x][3] = 3

sumG=0

for x in range(0,len(pts\_table)):

sumG+= Matrix[x][0]

centroid1[0] = sumG/len(Matrix)

centroid1[1] = 0

sumG=0

for x in range(0,len(pts\_table)):

sumG+= Matrix[x][1]

centroid2[0] = sumG/len(Matrix)

centroid2[1] = 0

sumG=0

for x in range(0,len(pts\_table)):

sumG+= Matrix[x][2]

centroid3[0] = sumG/len(Matrix)

centroid3[1] = 0

print("\n\n------- MATRIX -----------")

print(Matrix)

print("\n\n Centroid1",centroid1,"Centroid2",centroid2,"Centroid3",centroid3)

print("\nFirst Cluster")

for x in range(0,len(pts\_table)):

if Matrix[x][3]==1:

print(pts\_table[x],end='')

print("\n\nSecond Cluster")

for x in range(0,len(pts\_table)):

if Matrix[x][3]==2:

print(pts\_table[x],end='')

print("\n\nThird Cluster")

for x in range(0,len(pts\_table)):

if Matrix[x][3]==3:

print(pts\_table[x],end='')

i=i-1

**OUTPUT**

------- MATRIX -----------

[[0.0, 2.23606797749979, 4.47213595499958, 1], [2.23606797749979, 0.0, 2.23606797749979, 2], [4.47213595499958, 2.23606797749979, 0.0, 3], [6.708203932499369, 4.47213595499958, 2.23606797749979, 3], [10.770329614269007, 8.54400374531753, 6.324555320336759, 3], [13.0, 10.770329614269007, 8.54400374531753, 3], [16.15549442140351, 13.92838827718412, 11.704699910719626, 3], [19.697715603592208, 17.46424919657298, 15.231546211727817, 3], [213.78493866500511, 211.5490486861144, 209.31316251014889, 3], [188.1515346735179, 185.9247159470736, 183.69812192834198, 3]]

Centroid1 [47.49764208427864, 0] Centroid2 [45.71250073765308, 0] Centroid3 [44.37603615365917, 0]

First Cluster [47.49764208427864, 0]

Second Cluster [45.71250073765308, 0]

Third Cluster [44.37603615365917, 0][5, 16][6, 20][7, 22][8, 25][10, 28][100, 200][101, 170]

------- MATRIX -----------

[[0.0, 1.785141346625558, 3.1216059306194666, 1], [1.785141346625558, 0.0, 1.3364645839939087, 2], [3.1216059306194666, 1.3364645839939087, 0.0, 3], [45.40979610968817, 43.74365915550964, 42.50261430987833, 3], [46.06576058804298, 44.464398284898635, 43.274936751738366, 3], [46.08751473432188, 44.52704474095255, 43.37012887414144, 3], [46.7446652594473, 45.24635578571477, 44.13859995799957, 3], [46.7982175075148, 45.380422088571095, 44.33634921413447, 3], [206.776443500488, 207.23689964907146, 207.59100499294044, 1], [178.22037566603308, 178.76439123797562, 179.18223483836704, 1]]

Centroid1 [62.100952064278125, 0] Centroid2 [61.24847768733132, 0] Centroid3 [60.8853939453813, 0]

First Cluster [62.100952064278125, 0][100, 200][101, 170]

Second Cluster [61.24847768733132, 0]

Third Cluster [60.8853939453813, 0][5, 16][6, 20][7, 22][8, 25][10, 28]

------- MATRIX -----------

[[0.0, 0.8524743769468088, 1.2155581188968227, 1], [0.8524743769468088, 0.0, 0.3630837419500139, 2], [1.2155581188968227, 0.3630837419500139, 0.0, 3], [59.300242214066785, 58.47983620139688, 58.13069117454618, 3], [59.559355457546985, 58.75707860987939, 58.41580666685772, 3], [59.33055636339403, 58.53970730532297, 58.203399219022735, 3], [59.597927935972066, 58.82516787921834, 58.496708392508495, 3], [59.14819698016337, 58.39868547552159, 58.080317810568566, 3], [203.5591752646737, 203.7196124126228, 203.78898990575524, 1], [174.39362353683, 174.58574834783792, 174.66877688618894, 1]]

Centroid1 [67.69571102484906, 0] Centroid2 [67.25213943506967, 0] Centroid3 [67.13633319162948, 0]

First Cluster [67.69571102484906, 0][100, 200][101, 170]

Second Cluster [67.25213943506967, 0]

Third Cluster [67.13633319162948, 0][5, 16][6, 20][7, 22][8, 25][10, 28]

------- MATRIX -----------

[[0.0, 0.4435715897793955, 0.5593778332195853, 1], [0.4435715897793955, 0.0, 0.11580624344018986, 2], [0.5593778332195853, 0.11580624344018986, 0.0, 3], [64.70511711535171, 64.27541415069494, 64.16325975588518, 3], [64.85646273781573, 64.43465360637254, 64.3245772323296, 3], [64.55981208779951, 64.14296770888511, 64.03419843907376, 3], [64.71922368788339, 64.31031043023302, 64.2036284282393, 3], [64.13107726105098, 63.73231103524066, 63.62830007618387, 3], [202.5921200002361, 202.66332270931542, 202.68206777140298, 1], [173.23156659264026, 173.31739120096972, 173.3399778750082, 1]]

Centroid1 [69.97983289057765, 0] Centroid2 [69.7435748674931, 0] Centroid3 [69.70511936547827, 0]

First Cluster [69.97983289057765, 0][100, 200][101, 170]

Second Cluster [69.7435748674931, 0]

Third Cluster [69.70511936547827, 0][5, 16][6, 20][7, 22][8, 25][10, 28]

------- MATRIX -----------

[[0.0, 0.23625802308455945, 0.2747135250993864, 1], [0.23625802308455945, 0.0, 0.03845550201482695, 2], [0.2747135250993864, 0.03845550201482695, 0.0, 3], [66.92068949500893, 66.6913074292496, 66.65397566612805, 3], [67.03296962470216, 66.80750958453471, 66.77081872622077, 3], [66.71176321253311, 66.48876737579597, 66.45247922116133, 3], [66.83187626532666, 66.61282937556193, 66.57718645232706, 3], [66.19350688384489, 65.9795024075489, 65.9446834736964, 3], [202.24047674310313, 202.27568134058774, 202.28143709361945, 1], [172.8069754596049, 172.849541833538, 172.8564998891536, 1]]

Centroid1 [70.92492292323078, 0] Centroid2 [70.79798528719164, 0] Centroid3 [70.78502495494209, 0]

First Cluster [70.92492292323078, 0][100, 200][101, 170]

Second Cluster [70.79798528719164, 0]

Third Cluster [70.78502495494209, 0][5, 16][6, 20][7, 22][8, 25][10, 28]

------- MATRIX -----------

[[0.0, 0.12693763603914476, 0.13989796828869316, 1], [0.12693763603914476, 0.0, 0.0129603322495484, 2], [0.13989796828869316, 0.0129603322495484, 0.0, 3], [67.83874602639645, 67.71539609168278, 67.70280280994541, 3], [67.93559903752566, 67.8142971450645, 67.80191338312267, 3], [67.60470228276282, 67.48468660892425, 67.4724344343865, 3], [67.70927502857, 67.59132308314683, 67.57928202187932, 3], [67.0510718273885, 66.93575289022728, 66.92398119338786, 3], [202.1023505727236, 202.12065125386638, 202.12252414533953, 1], [172.63982814279447, 172.6619868202399, 172.66425431157856, 1]]

Centroid1 [71.31484085224893, 0] Centroid2 [71.24639918614406, 0] Centroid3 [71.24200506001782, 0]

First Cluster [71.31484085224893, 0][100, 200][101, 170]

Second Cluster [71.24639918614406, 0]

Third Cluster [71.24200506001782, 0][5, 16][6, 20][7, 22][8, 25][10, 28]

------- MATRIX -----------

[[0.0, 0.06844166610487434, 0.07283579223111758, 1], [0.06844166610487434, 0.0, 0.004394126126243236, 2], [0.07283579223111758, 0.004394126126243236, 0.0, 3], [68.21772582884235, 68.15119518489715, 68.14692388047627, 3], [68.30833357325156, 68.24289418509197, 68.23869301394474, 3], [67.97351509117436, 67.90876091039632, 67.90460377714722, 3], [68.07179351350757, 68.00813929238973, 68.00405284989505, 3], [67.40556140806643, 67.34331008547551, 67.33931380531916, 3], [202.04662421167004, 202.0563524360533, 202.05697778836551, 1], [172.57232881788235, 172.58411503203385, 172.58487263618443, 1]]

Centroid1 [71.47371599027306, 0] Centroid2 [71.4367602918569, 0] Centroid3 [71.43526676696897, 0]

First Cluster [71.47371599027306, 0][100, 200][101, 170]

Second Cluster [71.4367602918569, 0]

Third Cluster [71.43526676696897, 0][5, 16][6, 20][7, 22][8, 25][10, 28]

------- MATRIX -----------

[[0.0, 0.03695569841616475, 0.03844922330409872, 1], [0.03695569841616475, 0.0, 0.0014935248879339724, 2], [0.03844922330409872, 0.0014935248879339724, 0.0, 3], [68.37217941206413, 68.33625039521596, 68.3347983855834, 3], [68.46026209104767, 68.42491941898024, 68.42349111865305, 3], [68.12385818194969, 68.08888365151999, 68.08747023741194, 3], [68.21959118621166, 68.18520775304944, 68.18381824008185, 3], [67.55011293589934, 67.51648328489189, 67.51512425174556, 3], [202.0241294484488, 202.02935099293083, 202.0295621553297, 1], [172.54507077115548, 172.55139855139143, 172.55165444335876, 1]]

Centroid1 [71.5370608948497, 0] Centroid2 [71.51709432712839, 0] Centroid3 [71.51658615803562, 0]

First Cluster [71.5370608948497, 0][100, 200][101, 170]

Second Cluster [71.51709432712839, 0]

Third Cluster [71.51658615803562, 0][5, 16][6, 20][7, 22][8, 25][10, 28]

------- MATRIX -----------

[[0.0, 0.01996656772131189, 0.020474736814080075, 1], [0.01996656772131189, 0.0, 0.0005081690927681848, 2], [0.020474736814080075, 0.0005081690927681848, 0.0, 3], [68.43376704905947, 68.41435403279121, 68.41385995629376, 3], [68.52084610346867, 68.50174924094887, 68.50126321319415, 3], [68.18381207402193, 68.16491370504023, 68.1644327306201, 3], [68.27853328210733, 68.25995364605315, 68.25948078597699, 3], [67.60776481718975, 67.58959161330102, 67.58912910031479, 3], [202.01519473174164, 202.0180088892317, 202.018080537591, 1], [172.53424234253848, 172.5376530700341, 172.53773990572768, 1]]

Centroid1 [71.56146017046628, 0] Centroid2 [71.55066989342144, 0] Centroid3 [71.55049691356253, 0]

First Cluster [71.56146017046628, 0][100, 200][101, 170]

Second Cluster [71.55066989342144, 0]

Third Cluster [71.55049691356253, 0][5, 16][6, 20][7, 22][8, 25][10, 28]

------- MATRIX -----------

[[0.0, 0.010790277044833374, 0.010963256903750107, 1], [0.010790277044833374, 0.0, 0.00017297985891673306, 2], [0.010963256903750107, 0.00017297985891673306, 0.0, 3], [68.45749031351185, 68.44699893540367, 68.44683074797616, 3], [68.54418326658825, 68.533862604382, 68.53369715413703, 3], [68.20690683019355, 68.19669334864753, 68.1965296168936, 3], [68.30123878087257, 68.29119741154508, 68.29103643935066, 3], [67.6299739636199, 67.62015205786612, 67.6199946044546, 3], [202.01175843904724, 202.0132777406304, 202.01330210128543, 1], [172.53007745693225, 172.5319189127804, 172.53194843865322, 1]]

Centroid1 [71.57033825847142, 0] Centroid2 [71.56450642681588, 0] Centroid3 [71.56444753395132, 0]

First Cluster [71.57033825847142, 0][100, 200][101, 170]

Second Cluster [71.56450642681588, 0]

Third Cluster [71.56444753395132, 0][5, 16][6, 20][7, 22][8, 25][10, 28]

PROGRAM NO. 4

Create a NAÏVE BAYES Classifier for the given dataset.

import csv

import math

import random

#Loading the data from CSV file

def opencsv(filename):

lines = csv.reader(open(filename, "r"))

dataset = list(lines)

for i in range(len(dataset)):

dataset[i] = [x for x in dataset[i]]

return dataset

filename = 'nb\_dataset.csv'

SomeDataset = opencsv(filename)

print(SomeDataset)

def mean(numbers):

return sum(numbers)/float(len(numbers))

def sd(numbers):

avg = mean(numbers)

variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)

return math.sqrt(variance)

gender=[]

gender1=['M','F']

gender2=[0,0]

for i in SomeDataset:

gender.append(i[1])

for i in range(len(gender2)):

gender2[i]=gender.count(gender1[i])

gender2

height=[]

for i in SomeDataset:

height.append(float(i[2]))

height

height1=[[0,1.6],[1.61,1.7],[1.71,1.75],[1.75,1.9],[1.9,3.0]]

output=[]

for i in SomeDataset:

output.append(i[3])

output1=['Short','Medium','Tall']

output2=[0,0,0]

for i in range(len(output2)):

output2[i]=round(float(output.count(output1[i])/len(output)),2)

output2

p\_g\_o=[[0,0,0],[0,0,0]]

for i in range(len(gender)):

for j in range(len(gender1)):

if gender[i]==gender1[j]:

if output[i]==output1[0]:

p\_g\_o[j][0]+=1

elif output[i]==output1[1]:

p\_g\_o[j][1]+=1

else:

p\_g\_o[j][2]+=1

break

print("Gender vs Output\n")

for i in p\_g\_o:

print(i)

p\_h\_o=[[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]]

for i in range(len(height)):

for j in range(len(height1)):

if(height[i]>=height1[j][0] and height[i]<=height1[j][1]):

if output[i]==output1[0]:

p\_h\_o[j][0]+=1

elif output[i]==output1[1]:

p\_h\_o[j][1]+=1

else:

p\_h\_o[j][2]+=1

break

for i in p\_h\_o:

print(i)

pho=[[],[],[],[],[]]

count=0

for i in p\_h\_o:

a=i.index(max(i))

if(a==0):

pho[count].append(output1[0])

elif(a==1):

pho[count].append(output1[1])

else:

pho[count].append(output1[2])

if(sum(i)==0):

pho[count].append(float(0))

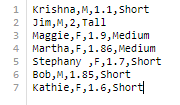
else:

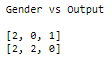
pho[count].append(round(float(i[a]/sum(i)),2))

count+=1

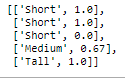
pho

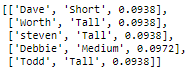
**OUTPUT**





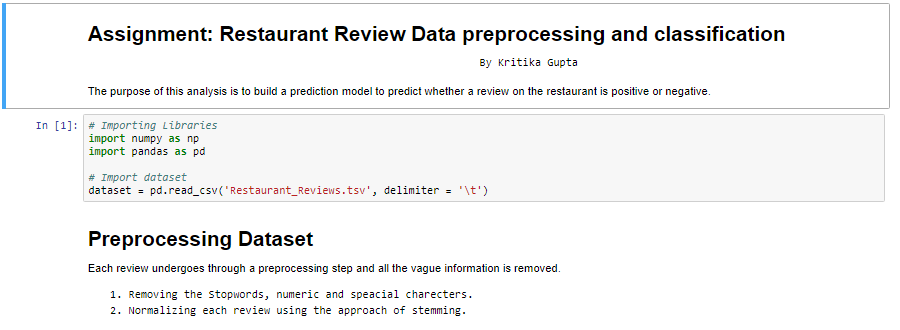


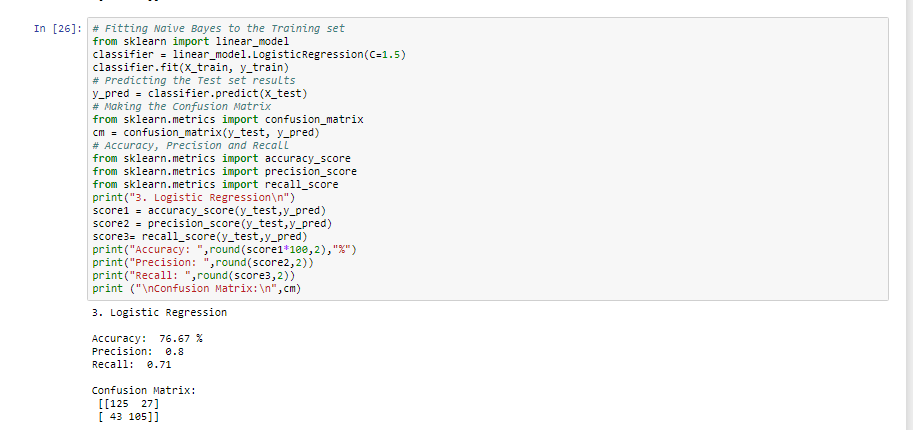
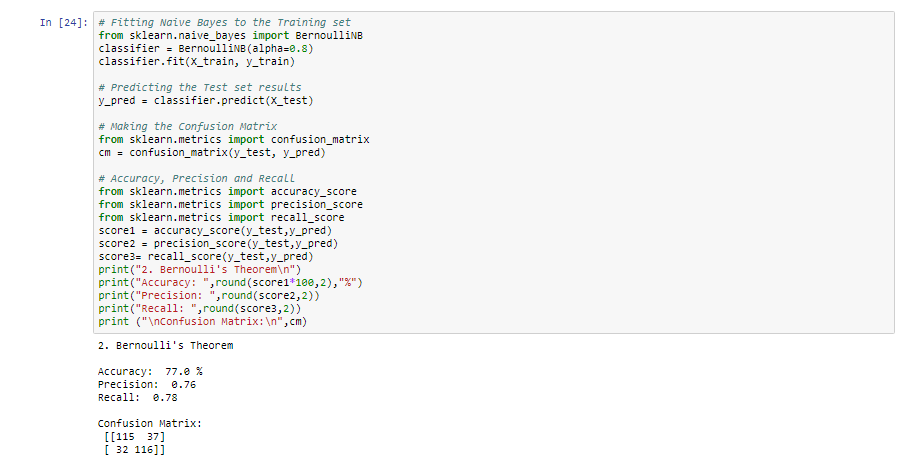
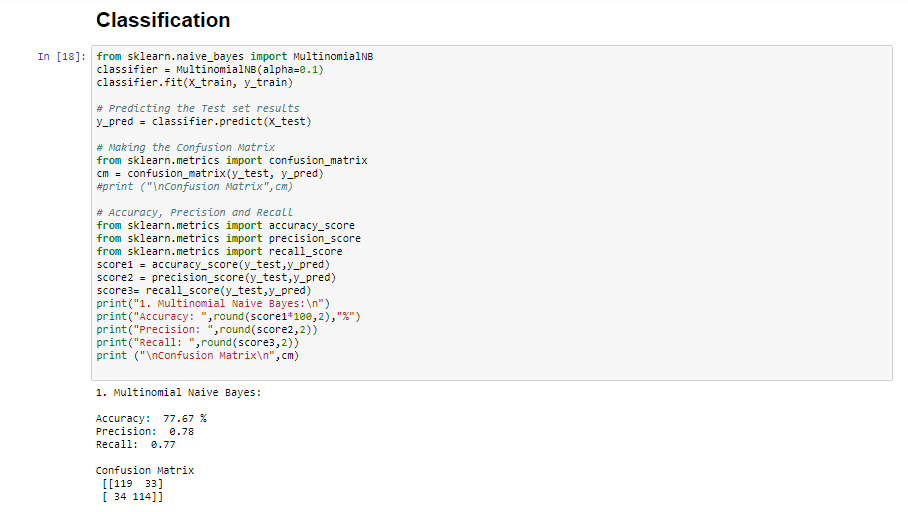
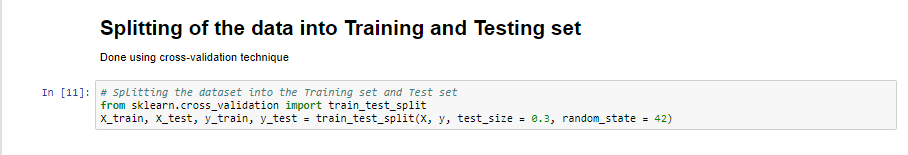
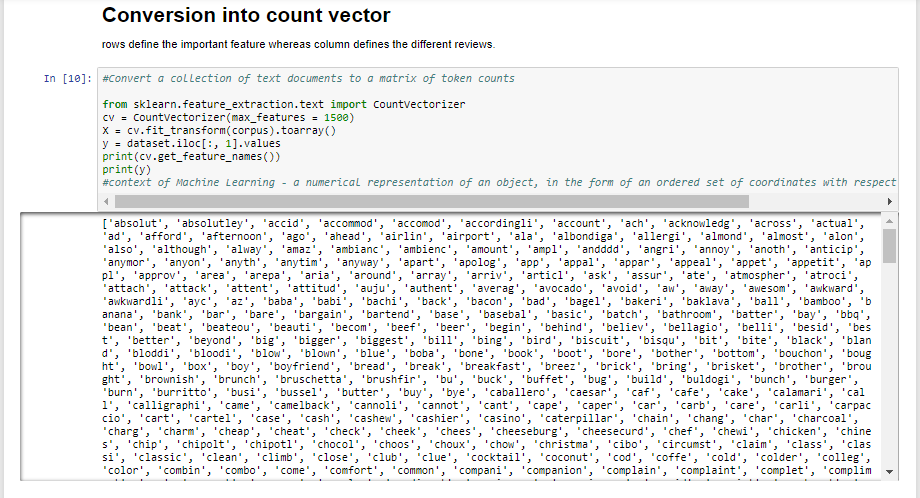
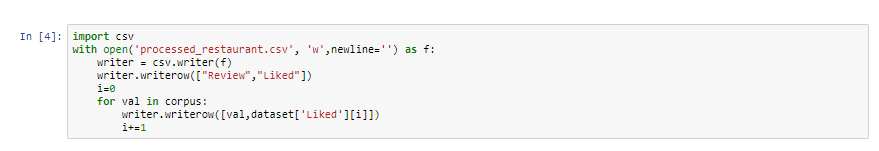
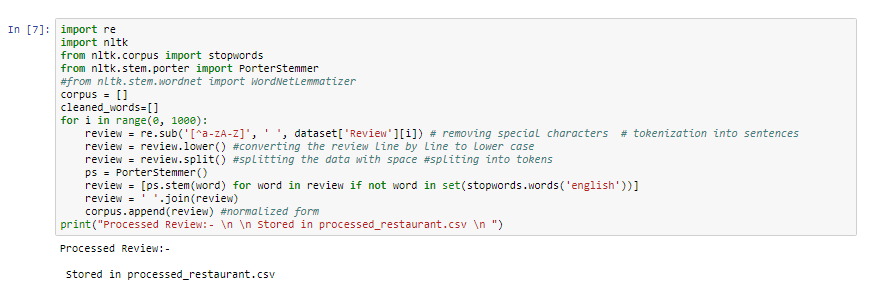


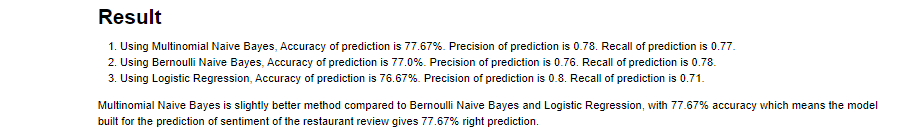


PROGRAM NO. 5

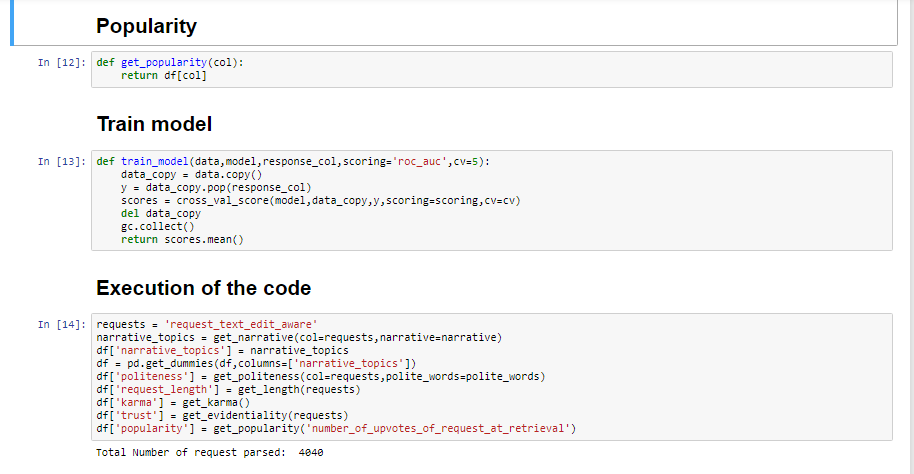
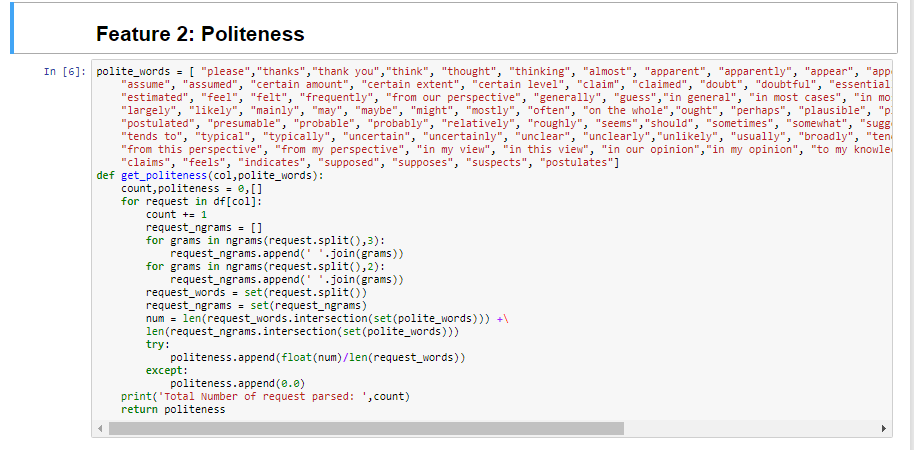
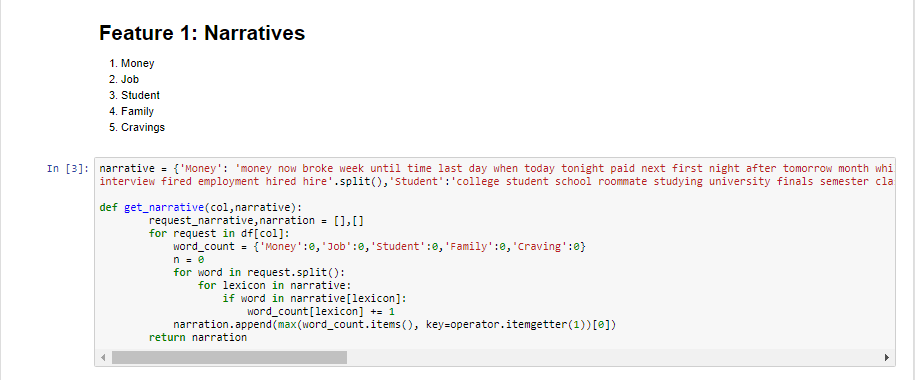
Implement DATA PROCESSING and CLASSIFICATION over the dataset.





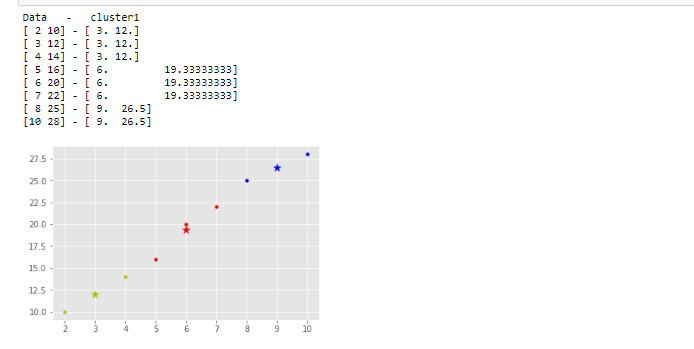
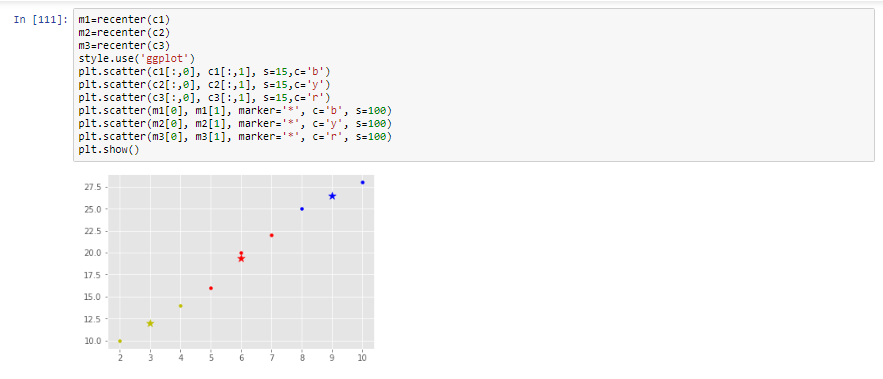
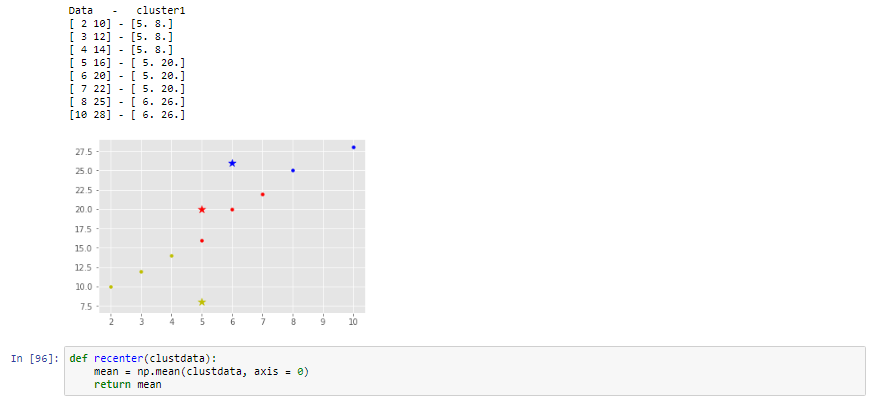
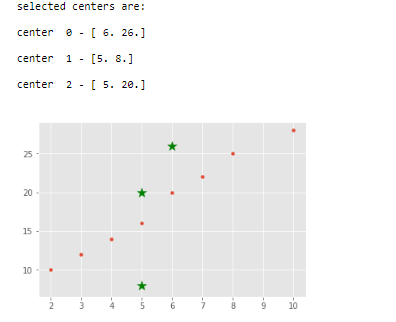
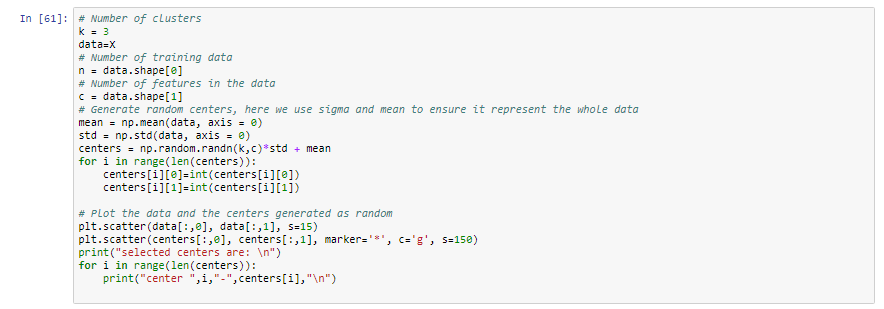
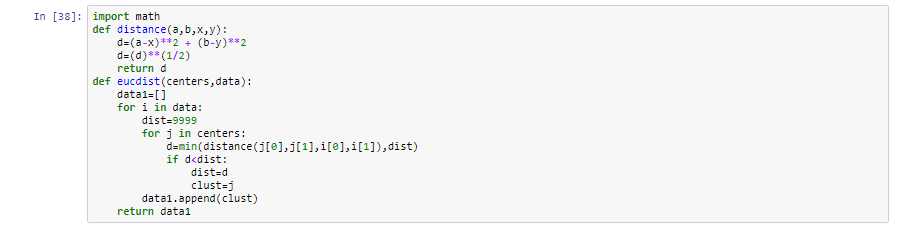
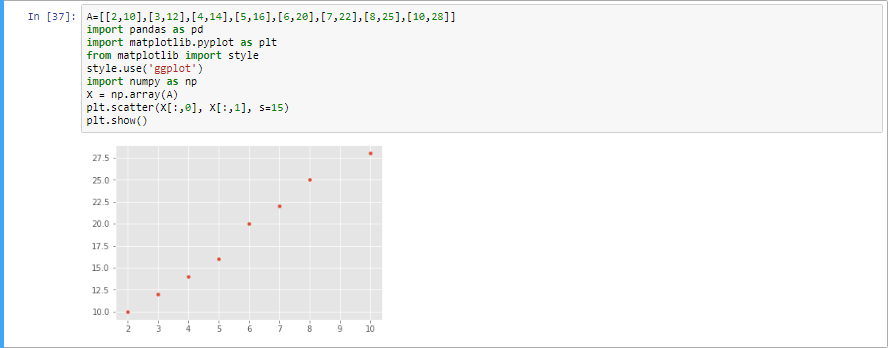


PROGRAM NO. 6

Extract FEATURES from Random Act of Pizza (RAOP) dataset. 

PROGRAM NO. 7

Implement K-MEANS Algorithm over the given dataset.



PROGRAM NO. 8

Implement DECISION TREE algorithm Using ID3 algorithm.

# In[15]:

import pandas as pd

import numpy as np

from pprint import pprint

eps = np.finfo(float).eps

from numpy import log2 as log

# In[16]:

df = pd.read\_csv('dataset.csv')

df

# In[17]:

entropy\_node = 0

values = df.play.unique()

for value in values:

fraction = df.play.value\_counts()[value]/len(df.play)

entropy\_node += -fraction\*np.log2(fraction)

# In[18]:

def find\_entropy(df):

Class = df.keys()[-1]

entropy = 0

values = df[Class].unique()

for value in values:

fraction = df[Class].value\_counts()[value]/len(df[Class])

entropy += -fraction\*np.log2(fraction)

return entropy

# In[19]:

def find\_entropy\_attribute(df,attribute):

Class = df.keys()[-1]

target\_variables = df[Class].unique()

variables = df[attribute].unique()

entropy2 = 0

for variable in variables:

entropy = 0

for target\_variable in target\_variables:

num = len(df[attribute][df[attribute]==variable][df[Class] ==target\_variable])

den = len(df[attribute][df[attribute]==variable])

fraction = num/(den+eps)

entropy += -fraction\*log(fraction+eps)

fraction2 = den/len(df)

entropy2 += -fraction2\*entropy

return abs(entropy2)

# In[20]:

def find\_winner(df):

Entropy\_att = []

IG = []

for key in df.keys()[:-1]:

IG.append(find\_entropy(df)-find\_entropy\_attribute(df,key))

return df.keys()[:-1][np.argmax(IG)]

# In[21]:

def get\_subtable(df, node,value):

return df[df[node] == value].reset\_index(drop=True)

# In[22]:

def buildTree(df,tree=None):

Class = df.keys()[-1]

node = find\_winner(df)

attValue = np.unique(df[node])

if tree is None:

tree={}

tree[node] = {}

for value in attValue:

subtable = get\_subtable(df,node,value)

clValue,counts = np.unique(subtable['play'],return\_counts=True)

if len(counts)==1:

tree[node][value] = clValue[0]

else:

tree[node][value] = buildTree(subtable)

return tree

# In[23]:

tree = buildTree(df)

pprint(tree)

**OUTPUT**





PROGRAM NO. 9

Implement DECISION TREE algorithm using RANDOM FOREST procedure.