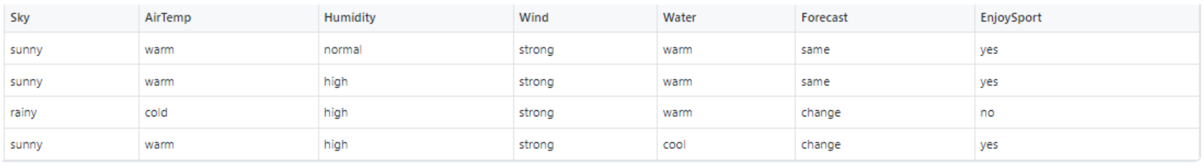
NEELESH GANGOPADHYAY 6th – 'B'
1BM18CS058

ML LAB RECORD

LAB 1: CANDIDATE ELIMINATION

DATA SET:



Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

PROGRAM:

```
import numpy as np
import pandas as pd

data = pd.read_csv('enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(l
en(specific_h))]
    print(general_h)

    for i, h in enumerate(concepts):
        print("For Loop Starts")
        if target[i] == "yes":
            print("If instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("If instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
        print("\n")
        print("\n")
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?'
, '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

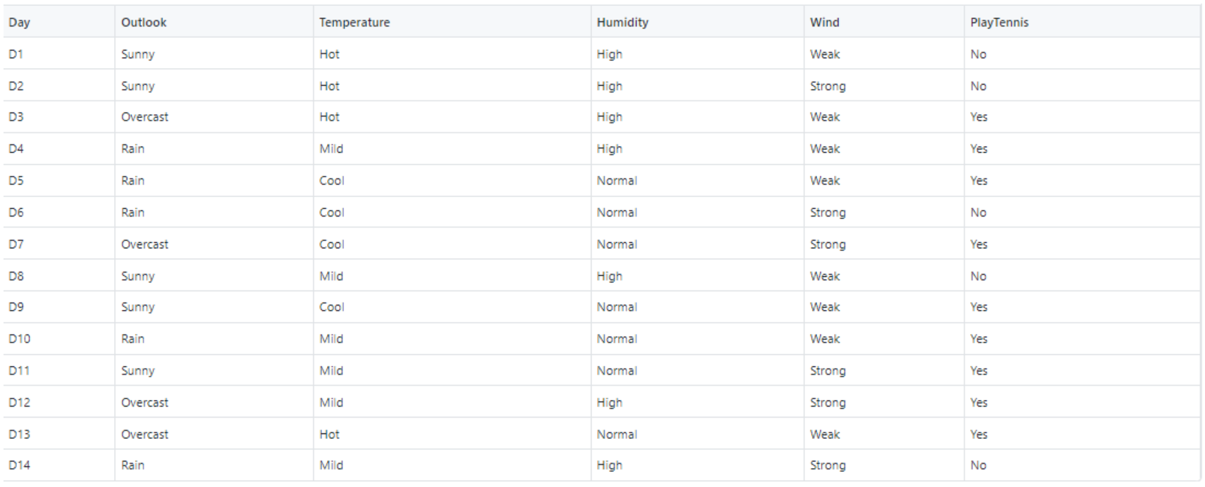
OUTPUT:

```
[['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]
['Yes' 'No' 'Yes' 'Yes']
initialization of specific_h and general_h
['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
 steps of Candidate Elimination Algorithm 1
['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

For Loop Starts
 steps of Candidate Elimination Algorithm 2
['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

For Loop Starts
 steps of Candidate Elimination Algorithm 3
['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

For Loop Starts
 steps of Candidate Elimination Algorithm 4
['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
Final General_h:
[]
```

NEELESH GANGOPADHYAY
1BM18CS058

LAB 2: DECISION TREE

DATA SET:



Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

PROGRAM:

```
import math
import csv
def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""

def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic

def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy

def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]


    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node

def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)


def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)

'''Main program'''
dataset,features=load_csv("id3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3_test.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end="   ")
    classify(node1,xtest,features)
```

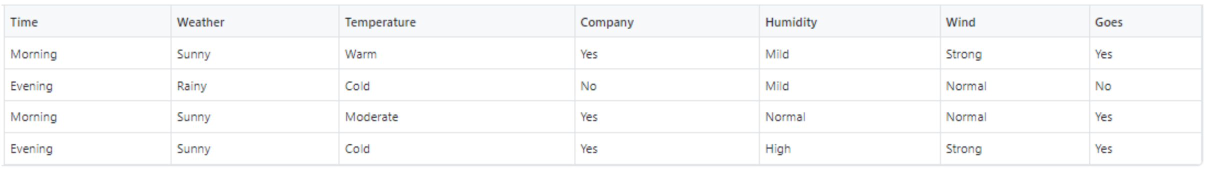
OUTPUT:

```
The decision tree for the dataset using ID3 algorithm is
 Day
   D11
     Yes
   D3
     Yes
   D7
     Yes
   D14
     No
   D8
     No
   D13
     Yes
   D2
     No
   D12
     Yes
   D10
     Yes
   D6
```

```
     No
   D1
     No
   D5
     Yes
   D4
     Yes
   D9
     Yes
The test instance: ['T1', 'Rain', 'Cool', 'Normal', 'Strong']
The label for test instance:   The test instance: ['T2', 'Sunny', 'Mild
', 'Normal', 'Strong']
The label for test instance:
```

LAB 3: FIND S ALGORITHM

DATA SET:



Time	Weather	Temperature	Company	Humidity	Wind	Goes
Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

PROGRAM:

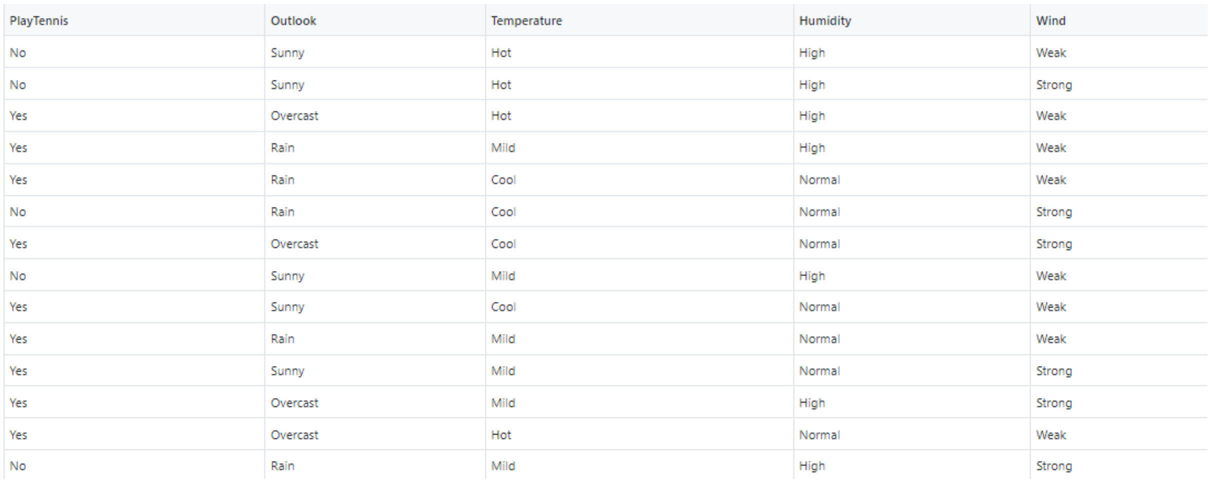
```
import pandas as pd
import numpy as np

#to read the data in the csv file
data = pd.read_csv("data.csv")
print(data,"n")

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

#obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))
```

OUTPUT:

```
  Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes n
n The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong'
]
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]
n The target is:  ['Yes' 'No' 'Yes' 'Yes']
n The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

LAB 4: NAÏVE BAYES CLASSIFIER

DATA SET:



PlayTennis	Outlook	Temperature	Humidity	Wind
No	Sunny	Hot	High	Weak
No	Sunny	Hot	High	Strong
Yes	Overcast	Hot	High	Weak
Yes	Rain	Mild	High	Weak
Yes	Rain	Cool	Normal	Weak
No	Rain	Cool	Normal	Strong
Yes	Overcast	Cool	Normal	Strong
No	Sunny	Mild	High	Weak
Yes	Sunny	Cool	Normal	Weak
Yes	Rain	Mild	Normal	Weak
Yes	Sunny	Mild	Normal	Strong
Yes	Overcast	Mild	High	Strong
Yes	Overcast	Hot	Normal	Weak
No	Rain	Mild	High	Strong

PROGRAM:

```
import pandas as pd

data = pd.read_csv('PlayTennis.csv')
data.head()
y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values

print(f'Target Values: {y}')
print(f'Features: \n{X}')
y_train = y[:8]
y_val = y[8:]

X_train = X[:8]
X_val = X[8:]

print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
class NaiveBayesClassifier:


    def __init__(self, X, y):

        self.X, self.y = X, y

        self.N = len(self.X)

        self.dim = len(self.X[0])

        self.attrs = [[] for _ in range(self.dim)]

        self.output_dom = {}

        self.data = []

        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])

            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1

            else:
                self.output_dom[self.y[i]] += 1

            self.data.append([self.X[i], self.y[i]])
    def classify(self, entry):

        solve = None
        max_arg = -1

        for y in self.output_dom.keys():

            prob = self.output_dom[y]/self.N

            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] an
d x[1] == y]
                n = len(cases)
                prob *= n/self.N

            if prob > max_arg:
                max_arg = prob
                solve = y

        return solve
```

```
nbc = NaiveBayesClassifier(X_train, y_train)

total_cases = len(y_val)

good = 0
bad = 0
predictions = []

for i in range(total_cases):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)

    if y_val[i] == predict:
        good += 1
    else:
        bad += 1

print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)
```

OUTPUT:

```
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Ye
s', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
Number of instances in training set: 8
Number of instances in testing set: 6
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```

NEELESH GANGOPADHYAY 6th – 'B'
1BM18CS058

LAB 5: BAYESIAN NETWORKS

PROGRAM:

```python
import bayespy as bp
import numpy as np
import csv
from colorama import init
from colorama import Fore, Back, Style
init()

# Define Parameter Enum values
# Age
ageEnum = {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1,
           'MiddleAged': 2, 'Youth': 3, 'Teen': 4}
# Gender
genderEnum = {'Male': 0, 'Female': 1}
# FamilyHistory
familyHistoryEnum = {'Yes': 0, 'No': 1}
# Diet(Calorie Intake)
dietEnum = {'High': 0, 'Medium': 1, 'Low': 2}
# LifeStyle
lifeStyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}
# Cholesterol
cholesterolEnum = {'High': 0, 'BorderLine': 1, 'Normal': 2}
# HeartDisease
heartDiseaseEnum = {'Yes': 0, 'No': 1}
import pandas as pd


data = pd.read_csv("heart_disease_data.csv")
data =np.array(data, dtype='int8')
N = len(data)
# Input data column assignment
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:, 0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:, 1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:, 2])
```

```
p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:, 3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:, 4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:, 5])
# Prepare nodes and establish edges
# np.ones(2) -> HeartDisease has 2 options Yes/No
# plates(5, 2, 2, 3, 4, 3) -> corresponds to options present for domain val
ues
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.MultiMixture(
    [age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Ca
tegorical, p_heartdisease)
heartdisease.observe(data[:, 6])
p_heartdisease.update()
#print("Sample Probability")
#print("Probability(HeartDisease|Age=SuperSeniorCitizen, Gender=Female, Fam
ilyHistory=Yes, DietIntake=Medium, LifeStyle=Sedetary, Cholesterol=High)")
#print(bp.nodes.MultiMixture([ageEnum['SuperSeniorCitizen'], genderEnum['Fe
male'], familyHistoryEnum['Yes'], dietEnum['Medium'], lifeStyleEnum['Sedeta
ry'], cholesterolEnum['High']], bp.nodes.Categorical, p_heartdisease).get_m
oments()[0] [heartDiseaseEnum['Yes']])

# Interactive Test
m = 0
while m == 0:
    print("\n")
    res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))),
int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHist
ory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(
        dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))),
int(input('Enter Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categor
ical, p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']]
    print("Probability(HeartDisease) = " + str(res))

# print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1 "))
```

OUTPUT:

```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged':
2, 'Youth': 3, 'Teen': 4}3
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}1
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}1
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary':
3}0
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 0


Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged':
2, 'Youth': 3, 'Teen': 4}0
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}0
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary':
3}0
```

LAB 6: INFERENCING WITH BAYESIAN NETWORKS

PROGRAM:

```python
from pgmpy.models import  BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
#Define a structure with nodes and edges
cancer_model = BayesianModel({('Pollution','Cancer'),('Smoker','Cancer'
),('Cancer','Xray'),('Cancer','Dyspnoea')})
print('Bayesian network nodes are :')
print('\t',cancer_model.nodes())
print('Bayesian network edges are :')
print('\t',cancer_model.edges())
#Creation of the conditional Probality Table
cpd_poll = TabularCPD(variable = 'Pollution',variable_card = 2,values =
[[0.9],[0.1]])
cpd_smoke = TabularCPD(variable = 'Smoker',variable_card = 2,values = [
[0.3],[0.7]])
cpd_cancer = TabularCPD(variable = 'Cancer',variable_card = 2,values =
[[0.03,0.05,0.001,0.02],[0.97,0.95,0.999,0.98]],evidence = ['Smoker','P
ollution'],evidence_card = [2,2])
cpd_xray = TabularCPD(variable = 'Xray',variable_card = 2,values = [[0.
9,0.2],[0.1,0.8]],evidence = ['Cancer'],evidence_card = [2])
cpd_dysp = TabularCPD(variable = 'Dyspnoea',variable_card = 2,values =
[[0.65,0.3],[0.35,0.7]],evidence = ['Cancer'],evidence_card = [2])
#Associating the parameters with the model structure
cancer_model.add_cpds(cpd_poll,cpd_smoke,cpd_cancer,cpd_xray,cpd_dysp)
print('Model generated by adding conditional probality distributions(cp
ds)')
```

```
#checking the correctness of the model
print('Checking for the correctness of the model:',end ='')
print(cancer_model.check_model)
print("All local independencies are as follows")
cancer_model.get_independencies()
print('Displaying CPDs')
print(cancer_model.get_cpds('Pollution'))
print(cancer_model.get_cpds('Smoker'))
print(cancer_model.get_cpds('Cancer'))
print(cancer_model.get_cpds('Xray'))
print(cancer_model.get_cpds('Dyspnoea'))
## infrencing with Bayesian Network
# Cmputing the probality of Cancer given smoke
cancer_infer = VariableElimination(cancer_model)

print('\n Inferencing with the Bayesian network')

print('\n Probality of Cancer given Smoker')
q = cancer_infer.query(variables = ['Cancer'],evidence = {'Smoker': 1})
print(q)

print('\n Probality of Cancer given Smoker,Pollution')
q = cancer_infer.query(variables = ['Cancer'],evidence = {'Smoker': 1,'
Pollution':1})
print(q)
```

OUTPUT:

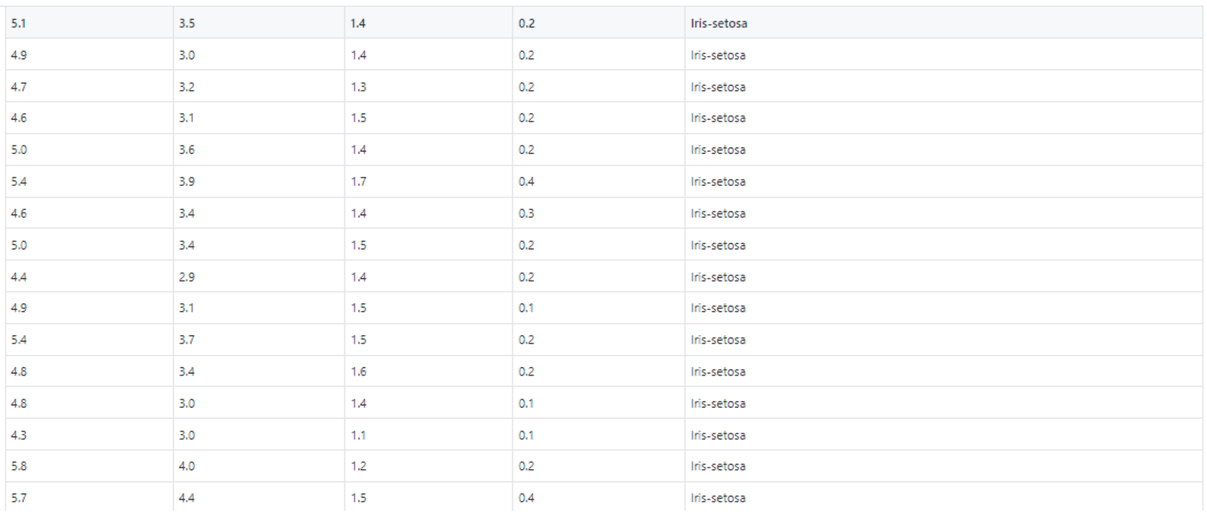
```
Bayesian network nodes are :
        ['Cancer', 'Dyspnoea', 'Smoker', 'Xray', 'Pollution']
Bayesian network edges are :
        [('Cancer', 'Dyspnoea'), ('Cancer', 'Xray'), ('Smoker', 'Cancer
'), ('Pollution', 'Cancer')]
Model generated by adding conditional probality distributions(cpds)
Checking for the correctness of the model:<bound method BayesianModel.c
heck_model of <pgmpy.models.BayesianModel.BayesianModel object at 0x000
0026D7AA367C0>>
(Dyspnoea ⍰ Xray, Smoker, Pollution | Cancer)
(Dyspnoea ⍰ Smoker, Pollution | Xray, Cancer)
(Dyspnoea ⍰ Xray, Pollution | Smoker, Cancer)
(Dyspnoea ⍰ Xray, Smoker | Cancer, Pollution)
(Dyspnoea ⍰ Pollution | Xray, Smoker, Cancer)
(Dyspnoea ⍰ Smoker | Xray, Cancer, Pollution)
(Dyspnoea ⍰ Xray | Smoker, Cancer, Pollution)
(Smoker ⍰ Pollution)
(Smoker ⍰ Xray, Dyspnoea | Cancer)
(Smoker ⍰ Dyspnoea | Xray, Cancer)
(Smoker ⍰ Xray | Cancer, Dyspnoea)
(Smoker ⍰ Xray, Dyspnoea | Cancer, Pollution)
(Smoker ⍰ Dyspnoea | Xray, Cancer, Pollution)
(Smoker ⍰ Xray | Pollution, Cancer, Dyspnoea)
(Xray ⍰ Smoker, Dyspnoea, Pollution | Cancer)
(Xray ⍰ Dyspnoea, Pollution | Smoker, Cancer)
(Xray ⍰ Smoker, Pollution | Cancer, Dyspnoea)
```

```
(Xray  Smoker, Dyspnoea | Cancer, Pollution)
(Xray  Pollution | Smoker, Cancer, Dyspnoea)
(Xray  Dyspnoea | Smoker, Cancer, Pollution)
(Xray  Smoker | Pollution, Cancer, Dyspnoea)
(Pollution  Smoker)
(Pollution  Xray, Dyspnoea | Cancer)
(Pollution  Dyspnoea | Xray, Cancer)
(Pollution  Xray, Dyspnoea | Smoker, Cancer)
(Pollution  Xray | Cancer, Dyspnoea)
(Pollution  Dyspnoea | Xray, Smoker, Cancer)
(Pollution  Xray | Smoker, Cancer, Dyspnoea)
Displaying CPDs
+--------------+-----+
| Pollution(0) | 0.9 |
+--------------+-----+
| Pollution(1) | 0.1 |
+--------------+-----+
+-----------+-----+
| Smoker(0) | 0.3 |
+-----------+-----+
| Smoker(1) | 0.7 |
+-----------+-----+
+-----------+--------------+--------------+--------------+-------------
-+
| Smoker    | Smoker(0)    | Smoker(0)    | Smoker(1)    | Smoker(1)
|
+-----------+--------------+--------------+--------------+-------------
-+
| Pollution | Pollution(0) | Pollution(1) | Pollution(0) | Pollution(1)
|
+-----------+--------------+--------------+--------------+-------------
-+
| Cancer(0) | 0.03         | 0.05         | 0.001        | 0.02
|
+-----------+--------------+--------------+--------------+-------------
-+
| Cancer(1) | 0.97         | 0.95         | 0.999        | 0.98
|
+-----------+--------------+--------------+--------------+-------------
-+
+---------+-----------+-----------+
| Cancer  | Cancer(0) | Cancer(1) |
+---------+-----------+-----------+
| Xray(0) | 0.9       | 0.2       |
+---------+-----------+-----------+
| Xray(1) | 0.1       | 0.8       |
+---------+-----------+-----------+
+-------------+-----------+-----------+
| Cancer      | Cancer(0) | Cancer(1) |
+-------------+-----------+-----------+
| Dyspnoea(0) | 0.65      | 0.3       |
+-------------+-----------+-----------+
| Dyspnoea(1) | 0.35      | 0.7       |
+-------------+-----------+-----------+
Inferencing with the Bayesian network

 Probality of Cancer given Smoker
```

```
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 |
+-----------+---------------+
| Cancer(1) |        0.9971 |
+-----------+---------------+

 Probality of Cancer given Smoker,Pollution
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 |
+-----------+---------------+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```

LAB 7: K MEANS

DATA SET:



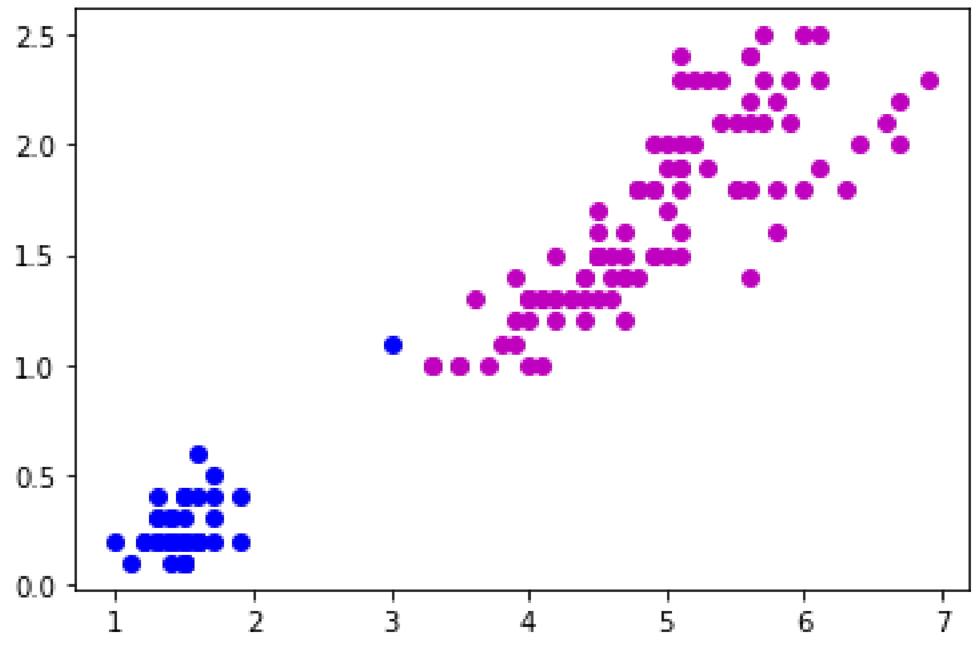
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5.0	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3.0	1.4	0.1	Iris-setosa
4.3	3.0	1.1	0.1	Iris-setosa
5.8	4.0	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa

PROGRAM:

```python
import math;
import sys;
import pandas as pd
import numpy as np
from random import choice
from matplotlib import pyplot
from random import shuffle, uniform;



def ReadData(fileName):
    f = open(fileName,'r')
    lines = f.read().splitlines()
    f.close()
    items = []

    for i in range(1,len(lines)):
        line = lines[i].split(',')
        itemFeatures = []

        for j in range(len(line)-1):
            v = float(line[j])
            itemFeatures.append(v)
        items.append(itemFeatures)

    shuffle(items)

    return items
def FindColMinMax(items):
    n = len(items[0])
    minima = [float('inf') for i in range(n)]
    maxima = [float('-inf') -1 for i in range(n)]

    for item in items:
        for f in range(len(item)):
            if(item[f] < minima[f]):
                minima[f] = item[f]

            if(item[f] > maxima[f]):
                maxima[f] = item[f]

    return minima,maxima

def EuclideanDistance(x,y):
    S = 0
    for i in range(len(x)):
        S += math.pow(x[i]-y[i],2)

    return math.sqrt(S)

def InitializeMeans(items,k,cMin,cMax):
    f = len(items[0])
    means = [[0 for i in range(f)] for j in range(k)]

    for mean in means:
        for i in range(len(mean)):
            mean[i] = uniform(cMin[i]+1,cMax[i]-1)
            return means

def UpdateMean(n,mean,item):
    for i in range(len(mean)):
        m = mean[i]
        m = (m*(n-1)+item[i])/float(n)
        mean[i] = round(m,3)

    return mean


def FindClusters(means,items):
    clusters = [[] for i in range(len(means))]
    for item in items:
        index = Classify(means,item)
        clusters[index].append(item)

    return clusters
def Classify(means,item):

    minimum = float('inf');
    index = -1

    for i in range(len(means)):
        dis = EuclideanDistance(item,means[i])

        if(dis < minimum):
            minimum = dis
            index = i

    return index

def CalculateMeans(k,items,maxIterations=100000):
    cMin, cMax = FindColMinMax(items)

    means = InitializeMeans(items,k,cMin,cMax)

    clusterSizes = [0 for i in range(len(means))]

    belongsTo = [0 for i in range(len(items))]

    for e in range(maxIterations):
        noChange = True;
        for i in range(len(items)):
            item = items[i];
            index = Classify(means,item)
            clusterSizes[index] += 1
            cSize = clusterSizes[index]
            means[index] = UpdateMean(cSize,means[index],item)
            if(index != belongsTo[i]):
                noChange = False
            belongsTo[i] = index

        if (noChange):
            break

    return means

def CutToTwoFeatures(items,indexA,indexB):
    n = len(items)
    X = []
    for i in range(n):
        item = items[i]
        newItem = [item[indexA],item[indexB]]
        X.append(newItem)

    return X

def PlotClusters(clusters):
    n = len(clusters)
    X = [[] for i in range(n)]
    for i in range(n):
        cluster = clusters[i]
        for item in cluster:
            X[i].append(item)
            colors = ['r','b','g','c','m','y']

    for x in X:
        c = choice(colors)
        colors.remove(c)

        Xa = []
        Xb = []

        for item in x:
            Xa.append(item[0])
            Xb.append(item[1])

        pyplot.plot(Xa,Xb,'o',color=c)

    pyplot.show()


def main():
    items = ReadData('Iris.csv')
    k = 3
    items = CutToTwoFeatures(items,2,3)
    print(items)
    means = CalculateMeans(k,items)
    print("\nMeans = ", means)

    clusters = FindClusters(means,items)

    PlotClusters(clusters)
    newItem = [1.5,0.2]
    print(Classify(means,newItem))
if __name__ == "__main__":
    main()
```

OUTPUT:

```
[[4.5, 1.3], [4.9, 1.8], [4.2, 1.3], [5.8, 1.8], [5.0, 1.7], [6.1, 2.5], [1
.4, 0.2], [4.4, 1.4], [5.6, 1.8], [4.2, 1.5], [4.0, 1.2], [6.7, 2.2], [1.3,
0.4], [1.6, 0.2], [6.9, 2.3], [5.1, 2.3], [6.3, 1.8], [1.5, 0.1], [3.9, 1.2
], [4.5, 1.5], [5.7, 2.1], [1.7, 0.3], [4.3, 1.3], [1.5, 0.4], [1.3, 0.2],
[4.9, 1.5], [4.1, 1.3], [6.0, 2.5], [6.4, 2.0], [1.5, 0.2], [5.6, 2.4], [5.
6, 2.1], [1.0, 0.2], [5.1, 2.0], [4.3, 1.3], [4.2, 1.3], [5.1, 1.9], [4.7,
1.4], [1.5, 0.4], [4.6, 1.3], [1.3, 0.2], [4.0, 1.3], [5.5, 1.8], [5.1, 1.9
], [4.1, 1.0], [1.3, 0.3], [1.6, 0.2], [1.4, 0.2], [4.1, 1.3], [1.6, 0.2],
[1.5, 0.4], [1.4, 0.2], [4.8, 1.8], [1.7, 0.2], [1.6, 0.4], [3.0, 1.1], [4.
9, 1.5], [5.2, 2.3], [5.0, 1.9], [4.8, 1.8], [5.6, 2.2], [4.8, 1.8], [1.3,
0.2], [5.6, 1.4], [1.4, 0.3], [4.4, 1.4], [1.2, 0.2], [1.5, 0.1], [4.9, 2.0
], [1.3, 0.2], [3.6, 1.3], [1.5, 0.3], [1.4, 0.2], [1.4, 0.1], [4.7, 1.4],
[1.4, 0.3], [1.5, 0.2], [1.5, 0.1], [4.5, 1.5], [5.7, 2.5], [4.4, 1.3], [4.
7, 1.5], [4.5, 1.6], [1.2, 0.2], [5.8, 1.6], [4.5, 1.5], [3.3, 1.0], [5.7,
2.3], [1.5, 0.1], [4.5, 1.7], [5.1, 1.5], [1.4, 0.2], [6.7, 2.0], [5.3, 2.3
], [3.5, 1.0], [4.9, 1.8], [4.0, 1.0], [5.9, 2.1], [6.0, 1.8], [5.1, 1.6],
[6.1, 1.9], [3.9, 1.1], [1.4, 0.2], [5.4, 2.3], [1.3, 0.3], [1.7, 0.4], [1.
5, 0.2], [4.5, 1.5], [5.0, 2.0], [3.7, 1.0], [5.4, 2.1], [4.6, 1.5], [1.4,
0.2], [3.3, 1.0], [1.1, 0.1], [5.0, 1.5], [4.2, 1.2], [5.5, 1.8], [5.9, 2.3
], [1.7, 0.5], [4.0, 1.3], [6.6, 2.1], [5.5, 2.1], [5.1, 1.8], [5.2, 2.0],
[1.5, 0.2], [3.9, 1.4], [4.6, 1.4], [4.7, 1.6], [1.9, 0.4], [3.5, 1.0], [4.
8, 1.4], [1.5, 0.2], [4.7, 1.2], [1.6, 0.2], [1.5, 0.2], [1.6, 0.6], [4.5,
1.5], [5.6, 2.4], [1.9, 0.2], [1.4, 0.3], [5.1, 2.4], [6.1, 2.3], [1.6, 0.2
], [5.3, 1.9], [4.0, 1.3], [3.8, 1.1], [5.8, 2.2], [4.4, 1.2]]

Means =  [[4.923, 1.684], [1.495, 0.26], [0, 0]]
```



LAB 8: EM ALGORITHM USING K-MEANS

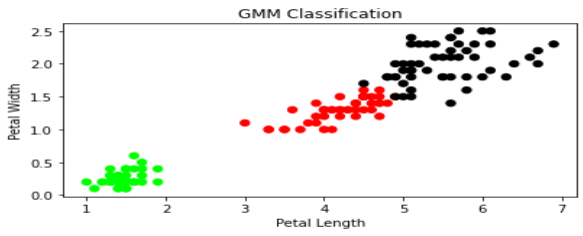
PROGRAM:

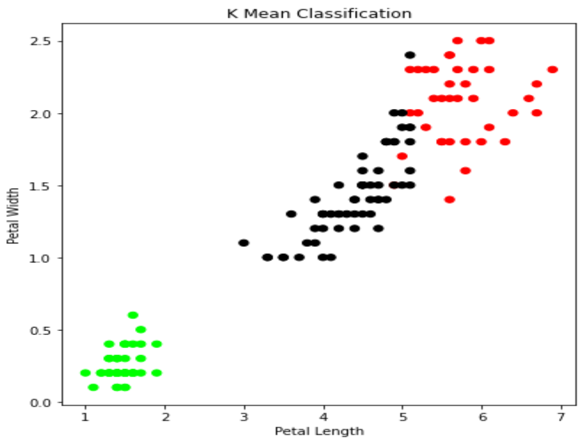
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']
```

```
model = KMeans(n_clusters=3)
model.fit(X)


plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s
=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.label
s_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.la
bels_))


from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

OUTPUT:

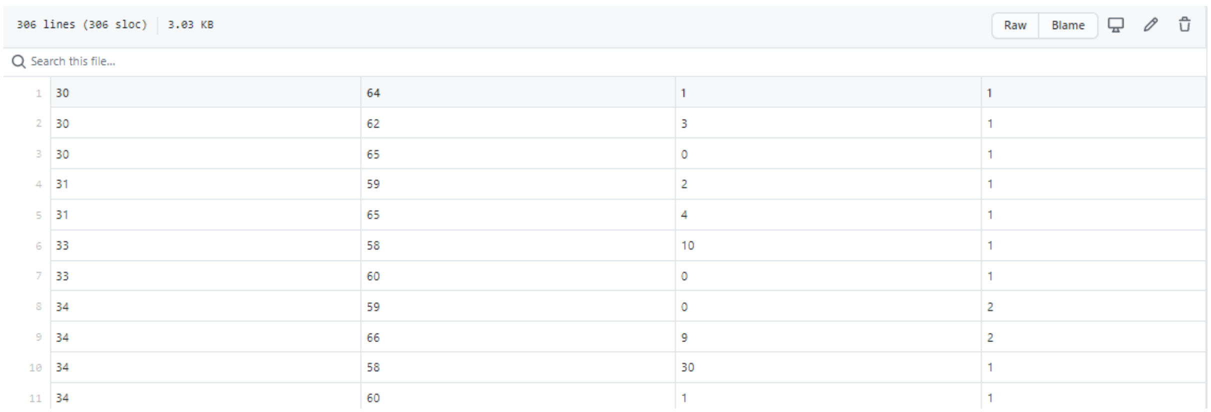
```
The accuracy score of K-Mean:  0.09333333333333334
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [ 2  0 48]
 [36  0 14]]
The accuracy score of EM:  0.3333333333333333
The Confusion matrix of EM:  [[ 0 50  0]
 [45  0  5]
 [ 0  0 50]]
```





LAB 9: KNN CLASSIFIER

DATA SET:


306 lines (306 sloc) | 3.03 KB

Raw Blame

Search this file...

30	64	1	1
30	62	3	1
30	65	0	1
31	59	2	1
31	65	4	1
33	58	10	1
33	60	0	1
34	59	0	2
34	66	9	2
34	58	30	1
34	60	1	1

PROGRAM:

```
import math
data = []
with open('haberman.csv', 'r') as f:
    for line in f.readlines():
        atributes = line.strip('\n').split(',')
        data.append([int(x) for x in atributes])
def info_dataset(data, verbose=True):
    label1, label2 = 0, 0
    data_size = len(data)
    for datum in data:
        if datum[-1] == 1:
            label1 += 1
        else:
            label2 += 1
    if verbose:
        print('Total of samples: %d' % data_size)
        print('Total label 1: %d' % label1)
        print('Total label 2: %d' % label2)
    return [len(data), label1, label2]
info_dataset(data)
p = 0.6
_, label1, label2 = info_dataset(data,False)
train_set, test_set = [], []
max_label1, max_label2 = int(p * label1), int(p * label2)
total_label1, total_label2 = 0, 0
for sample in data:
    if (total_label1 + total_label2) < (max_label1 + max_label2):
        train_set.append(sample)
        if sample[-1] == 1 and total_label1 < max_label1:
            total_label1 += 1
        else:
            total_label2 += 1
    else:
        test_set.append(sample)
def euclidian_dist(p1, p2):
    dim, sum_ = len(p1), 0
    for index in range(dim - 1):
        sum_ += math.pow(p1[index] - p2[index], 2)
    return math.sqrt(sum_)
def knn(train_set, new_sample, K):
    dists, train_size = {}, len(train_set)

    for i in range(train_size):
        d = euclidian_dist(train_set[i], new_sample)
        dists[i] = d

    k_neighbors = sorted(dists, key=dists.get)[:K]

    qty_label1, qty_label2 = 0, 0
    for index in k_neighbors:
        if train_set[index][-1] == 1:
            qty_label1 += 1
        else:
            qty_label2 += 1
    if qty_label1 > qty_label2:
        return 1
    else:
        return 2

print(test_set[0])
print(knn(train_set, test_set[0], 12))
```

```
correct, K = 0, 15
for sample in test_set:
    label = knn(train_set, sample, K)
    if sample[-1] == label:
        correct += 1
print("Train set size: %d" % len(train_set))
print("Test set size: %d" % len(test_set))
print("Correct predicitons: %d" % correct)
print("Accuracy: %.2f%%" % (100 * correct / len(train_set)))
```

OUTPUT:

```
Total of samples: 306
Total label 1: 225
Total label 2: 81
[306, 225, 81]
[55, 58, 0, 1]
1
Train set size: 183
Test set size: 123
Correct predicitons: 93
Accuracy: 50.82%
```

LAB 10: LINEAR REGRESSION

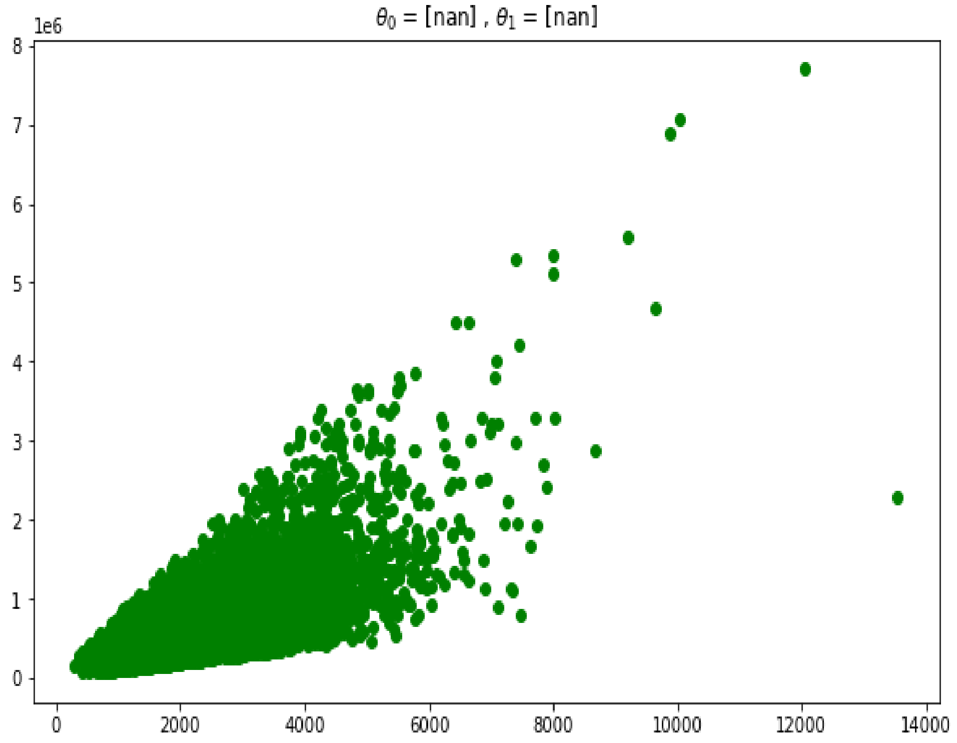
PROGRAM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

dataset = pd.read_csv('house_data.csv')

Y = dataset[['price']]

X = dataset.drop(['price', 'id', 'date'],  axis=1)

x = X[['sqft_living']]
y = Y

xg = x.values.reshape(-1,1)
yg = y.values.reshape(-1,1)
```

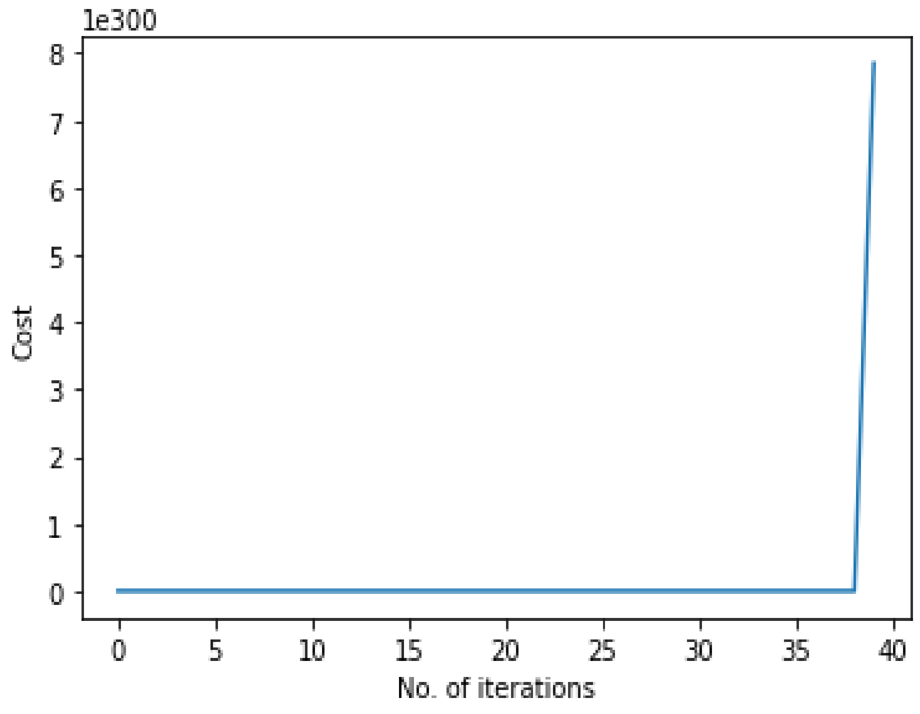
```
xg = np.concatenate((np.ones(len(x)).reshape(-1,1), x), axis=1)

def computeCost(x, y, theta):
    m = len(y)
    h_x = x.dot(theta)
    j = np.sum(np.square(h_x - y))*(1/(2*m))
    return j
def gradientDescent(x, y, theta, alpha, iteration):
    print('Running Gradient Descent...')
    j_hist = []
    m = len(y)
    for i in range(iteration):
        j_hist.append(computeCost(x, y, theta))
        h_x = x.dot(theta)
        theta = theta - ((alpha/m) *((np.dot(x.T, (h_x-y) ))))
        #theta[0] = theta[0] - ((alpha/m) *(np.sum((h_x-y))))
    return theta, j_hist

theta = np.zeros((2,1))
iteration = 2000
alpha = 0.001

theta, cost = gradientDescent(xg, yg, theta, alpha, iteration)
print('Theta found by Gradient Descent: slope = {} and intercept {}'.fo
rmat(theta[1], theta[0]))

theta.shape

plt.figure(figsize=(10,6))
plt.title('$\\theta_0$ = {} , $\\theta_1$ = {}'.format(theta[0], theta[
1]))
plt.scatter(x,y, marker='o', color='g')
plt.plot(x,np.dot(x.values, theta.T))
plt.show()

plt.plot(cost)
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
```

OUTPUT:

```
Running Gradient Descent...
<ipython-input-2-8b794ecd2ce3>:23: RuntimeWarning: overflow encountered in
square
  j = np.sum(np.square(h_x - y))*(1/(2*m))
C:\Users\neelesh\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:90:
RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
<ipython-input-2-8b794ecd2ce3>:32: RuntimeWarning: invalid value encountere
d in subtract
  theta = theta - ((alpha/m) *((np.dot(x.T, (h_x-y) ))))
Theta found by Gradient Descent: slope = [nan] and intercept [nan]
```

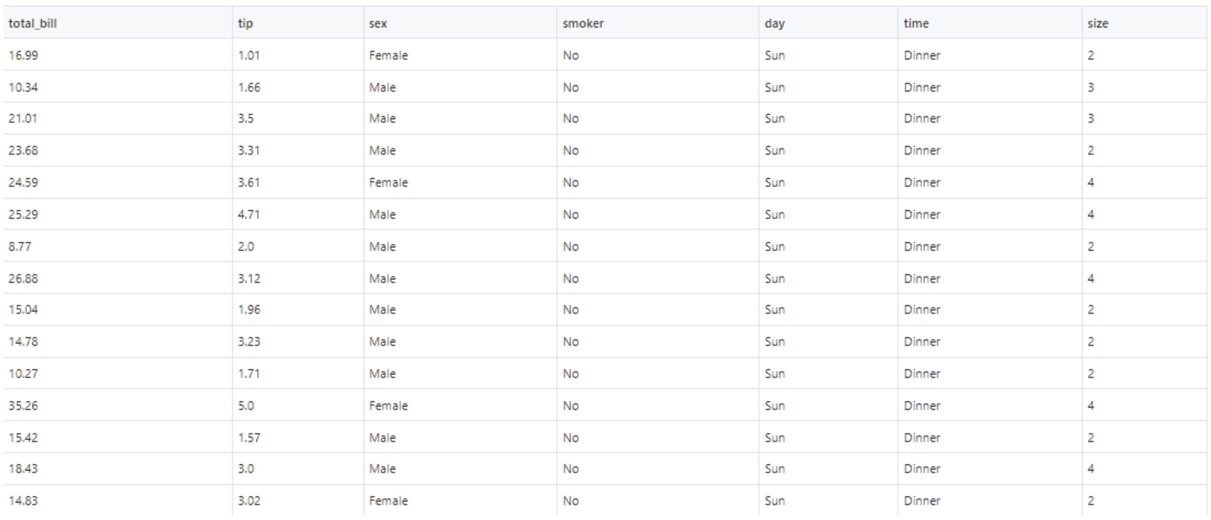


```
Text(0, 0.5, 'Cost')
```



LAB 11: LOCALLY WEIGHTED REGRESSION

DATA SET:



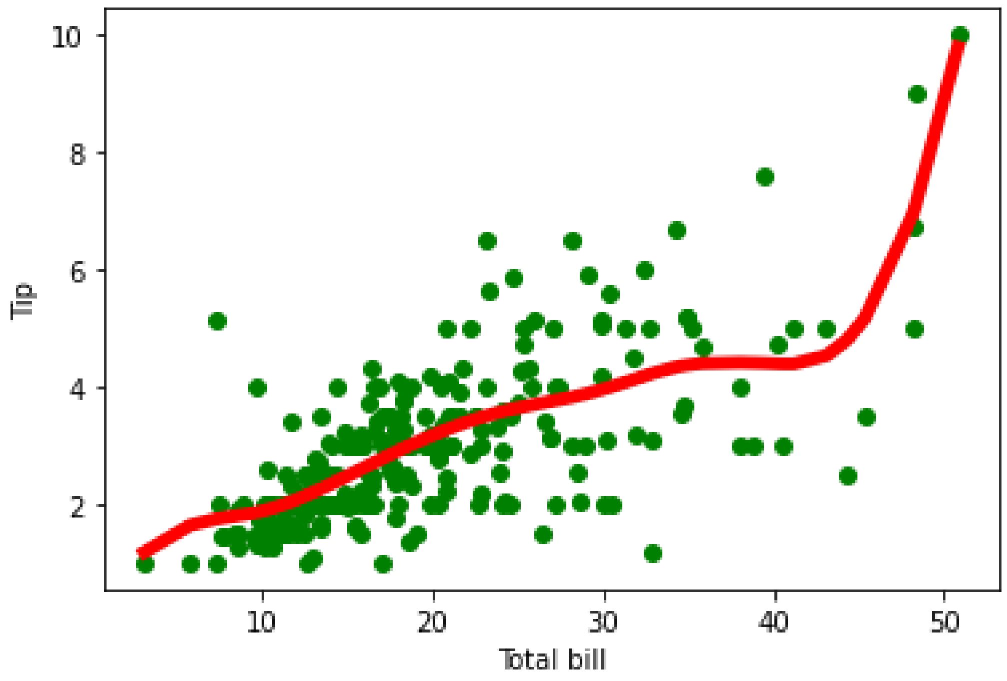
total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2.0	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2
14.78	3.23	Male	No	Sun	Dinner	2
10.27	1.71	Male	No	Sun	Dinner	2
35.26	5.0	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3.0	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2

PROGRAM

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();

# load data points
data = pd.read_csv('data10_tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips dat
a
```

```
tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D ar
ray
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```

OUTPUT:


Tip
Total bill