Name: Manas Agarwal

USN: 1BM18CS052

Section: 6-B

ML LAB PROGRAMS

---all csv files are taken from here----

https://drive.google.com/drive/folders/1jqAPIa-C4E9JzwmPIr4kWRUAw0xCd7ny

Program 1: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Date:10/03/2021

```
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("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  Whether 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

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']]

The target is:  ['Yes' 'No' 'Yes' 'Yes']

The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

Program 2: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples. DATE:24/03/2021

```
import numpy as np import
pandas as pd

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

#making an array of all the attributes concepts =
np.array(data.iloc[:,0:-1]) print("The attributes
are: ",concepts)

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

```
#training function to implement candidate_elimination algorithm def
learn(concepts, target):
specific_h = concepts[0].copy() print("\n Initialization of
specific_h and general_h") print(specific_h) general_h =
[["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
 print(general_h) for i, h in
 enumerate(concepts): if target[i]
 == "yes":
      for x in range(len(specific_h)):
         if h[x]!= specific_h[x]: specific_h[x]
            ='?' general_h[x][x] ='?'
         print(specific_h)
    print(specific_h) if
    target[i] == "no":
    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("\n Steps of Candidate Elimination Algorithm",i+1)
    print(specific_h) print(general_h)
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)

#obtaining the final hypothesis print("\nFinal
Specific_h:", s_final, sep="\n") print("\nFinal
General_h:", g_final, sep="\n")
```

Output:

```
   sky  temp humidity    wind water forcast enjoysport
0  sunny  warm   normal  strong  warm    same        yes
```

```
1  sunny  warm    high  strong  warm   same         yes
2  rainy  cold    high  strong  warm  change   no
3  sunny  warm    high  strong  cool  change yes
```

```
The attributes are:  [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
['rainy' 'cold' 'high' 'strong' 'warm' 'change']
['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

The target is:  ['yes' 'yes' 'no' 'yes']

Initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']

Steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

```
['sunny' 'warm' '?' 'strong' 'warm' 'same']

Steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
['sunny' 'warm' '?' 'strong' '?' '?']

Steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Program 3: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. DATE:31/03/2021

```
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:")
    classify(node1,xtest,features)
```

Output:

```
The decision tree for the dataset using ID3 algorithm is
Outlook sunny
```

```
     Humidity
       normal yes
       high
         no
   overcast
     yes rain
     Wind
       strong no
       weak
         yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance: no
The test instance: ['sunny', 'mild', 'normal', 'strong'] The
label for test instance:
yes
```

Program 4: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. DATE:21/04/2021 **import pandas as pd**

```
data = pd.read_csv('PlayTennis.csv')
data.head()
```

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak

```
y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values
```

```
print(f'Target Values: {y}')
print(f'Features: \n{X}')
```

```
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', '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']]
```

```
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)}")
```

```
Number of instances in training set: 8
Number of instances in testing set: 6
```

```
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] and 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:

```
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

Program 5: Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

DATE:28/04/2021

(With Built-in)

```
import numpy as np import pandas as pd import csv from
pgmpy.estimators import MaximumLikelihoodEstimator from
pgmpy.models import BayesianModel from
pgmpy.inference import VariableElimination

#read Cleveland Heart Disease data
heartDisease = pd.read_csv('/content/heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
```

```
#display the data print('Sample instances from the dataset
are given below') print(heartDisease.head())
```

```
Sample instances from the dataset are given below age  sex  cp  trestbps
   chol  fbs  restecg  thalach exang  oldpeak  slope  \
0   63    1   1      145   233    1       2      150     0      2.3      3
1   67    1   4      160   286    0       2      108     1      1.5      2
2   67    1   4      120   229    0       2      129     1      2.6      2
3   37    1   3      130   250    0       0      187     0      3.5      3
4   41    0   2      130   204    0       2      172     0      1.4      1
  ca thal  heartdisease
0  0    6            0
1  3    3            2
2  2    7            1
3  0    3            0
4  0    3            0
```

```
#display the Attributes names and datatyes
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
Attributes and datatypes
age             int64
sex             int64 cp
int64 trestbps
int64 chol
int64 fbs
int64 restecg
int64 thalach
int64 exang
int64 oldpeak
float64 slope
int64 ca
object thal
object heartdisease
int64 dtype: object
```

```
#Create Model-Bayesian Network
model =
BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','
heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])
```

```
#Learning CPDs using Maximum Likelihood Estimators print('\n
Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
```

```
Learning CPD using Maximum likelihood estimators
```

```
#Inferencing with Bayesian Network print('\n
Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
```

```
Inferencing with Bayesian Network:
```

```
#computing the Probability of HeartDisease given restecg print('\n 1.Probability of HeartDisease given evidence= restecg :1')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
```

```
1.Probability of HeartDisease given evidence= restecg :1
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+
```

New Section

```
#computing the Probability of HeartDisease given cp print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

```
2.Probability of HeartDisease given evidence= cp:2
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

(Without Built-in)

```
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 values 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.Categorical,
p_heartdisease) heartdisease.observe(data[:, 6]) p_heartdisease.update()

#print("Sample Probability")
#print("Probability(HeartDisease|Age=SuperSeniorCitizen, Gender=Female,
FamilyHistory=Yes, DietIntake=Medium, LifeStyle=Sedetary, Cholesterol=High)")
#print(bp.nodes.MultiMixture([ageEnum['SuperSeniorCitizen'], genderEnum['Female'],
familyHistoryEnum['Yes'], dietEnum['Medium'], lifeStyleEnum['Sedetary'],
cholesterolEnum['High']], bp.nodes.Categorical, p_heartdisease).get_moments()[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 FamilyHistory: ' + str(familyHistoryEnum))),
int(input('Enter dietEnum: ' + str( dietEnum))), int(input('Enter LifeStyle: ' +
str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)))],
bp.nodes.Categorical, 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}1
Enter Gender: {'Male': 0, 'Female': 1}1
Enter FamilyHistory: {'Yes': 0, 'No': 1}1
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}2
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
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}2
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
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}0
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 1
```

(Bayesian Graph)

```
# Starting with defining the network structure 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:') print('\t',
cancer_model.nodes()) print('Bayesian
network edges:') print('\t',
cancer_model.edges())
```

```
Bayesian network nodes:
             ['Pollution', 'Cancer', 'Smoker', 'Xray', 'Dyspnoea'] Bayesian
network edges:
                    [('Pollution', 'Cancer'), ('Cancer', 'Xray'), ('Cancer', 'Dyspnoea'), ('Smoker', 'Cancer')]
```

```
#Creation of Conditional Probability 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',
                        'Pollution'], 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 bt adding conditional probability distribution(cpds)')

# Checking if the cpds are valid for the model. print('Checking for
Correctness of model:', end='')
print(cancer_model.check_model())
```

```
Model generated bt adding conditional probability distribution(cpds)
Checking for Correctness of model:True
```

```
'''print('All local dependencies 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'))
```

```
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 Bayesian Network
#Computing the probability of Cancer given smoke

cancer_infer = VariableElimination(cancer_model)
print('\nInferencing with Bayesian Network')

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

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

```
Inferencing with Bayesian Network

Probability of Cancer given Smoker
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 | +-----------+---------------
+
| Cancer(1) |        0.9971 | +-----------+---------------
+
```

Probability of Cancer given Smoker, Pollution

```
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 | +-----------+---------------
+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```