Lab Report for ML Lab Test – 1

USN – 1BM18CS067

Name – Parag Gattani

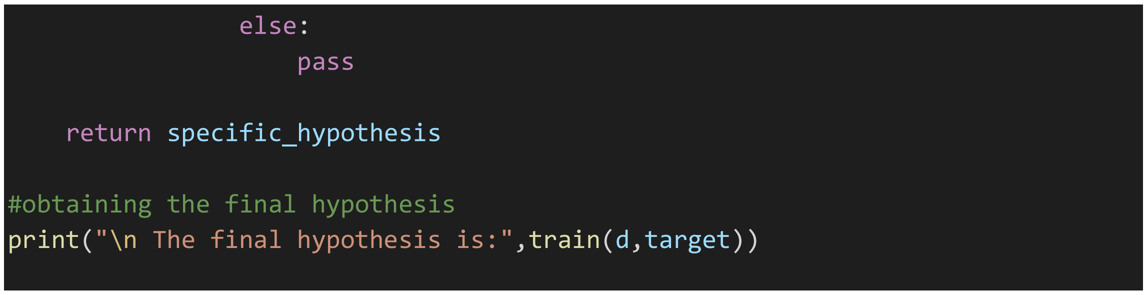
Date – 25 May, 2021

Program 1

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

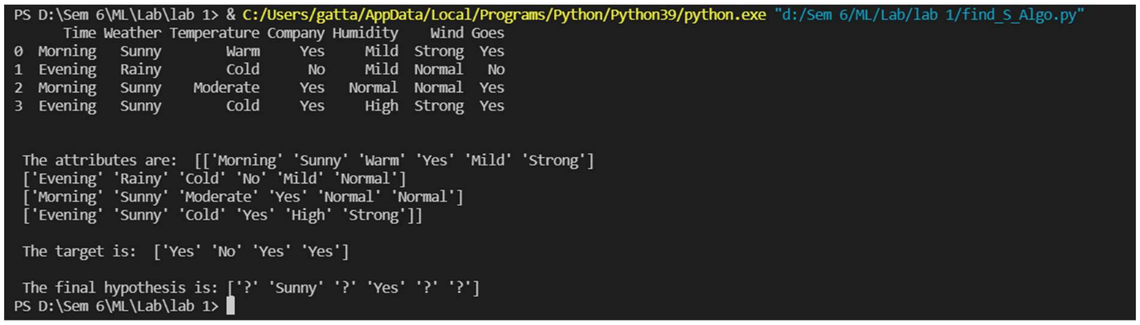
Code –

```python
#!/usr/bin/env python
# coding: utf-8

# In[3]:


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] = '?'
```



```python
                else:
                    pass

    return specific_hypothesis

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

OUTPUT –



```
PS D:\Sem 6\ML\Lab\lab 1> & C:/Users/gatta/AppData/Local/Programs/Python/Python39/python.exe "d:/Sem 6/ML/Lab/lab 1/find_S_Algo.py"
      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


 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' '?' '?']
PS D:\Sem 6\ML\Lab\lab 1> 
```

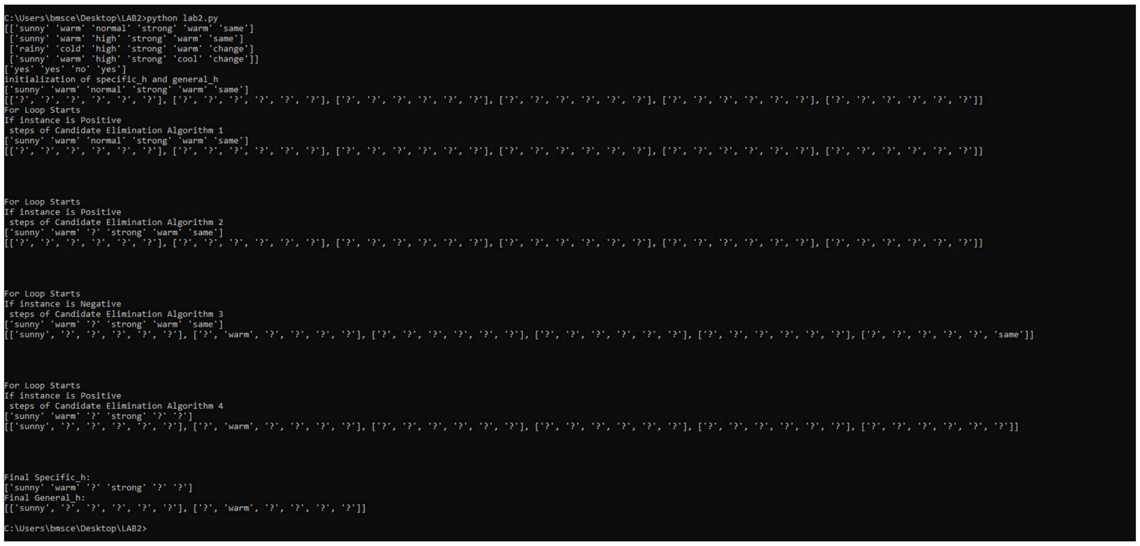
Program – 2

Question - 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.

Code –

```python
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(len(spec
ific_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 –



```
C:\Users\bmsce\Desktop\LAB2>python lab2.py
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Negative
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



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

C:\Users\bmsce\Desktop\LAB2>
```

Program – 3

Question - 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.

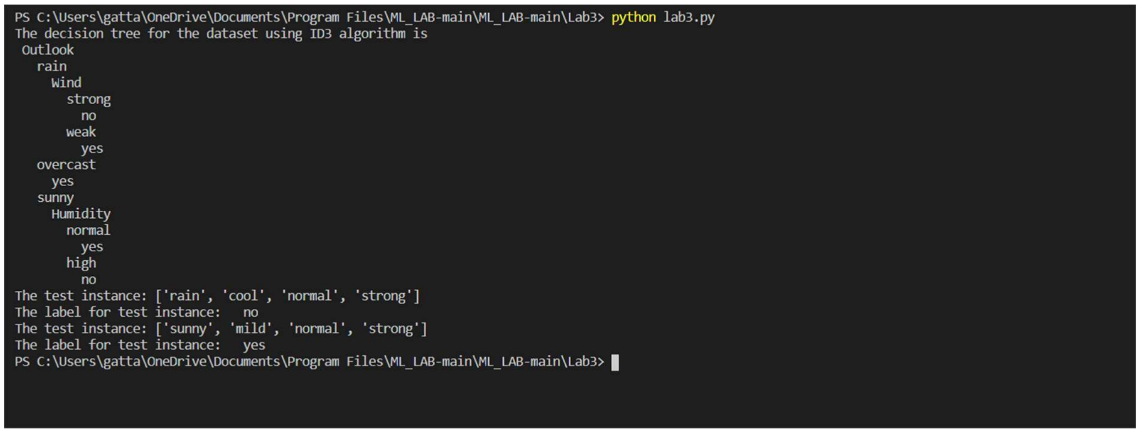
Code –

```
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))
```

```python
    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
```

```python
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 –



```
PS C:\Users\gatta\OneDrive\Documents\Program Files\ML_LAB-main\ML_LAB-main\Lab3> python lab3.py
The decision tree for the dataset using ID3 algorithm is
 Outlook
   rain
     Wind
       strong
         no
       weak
         yes
   overcast
     yes
   sunny
     Humidity
       normal
         yes
       high
         no
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
PS C:\Users\gatta\OneDrive\Documents\Program Files\ML_LAB-main\ML_LAB-main\Lab3>
```

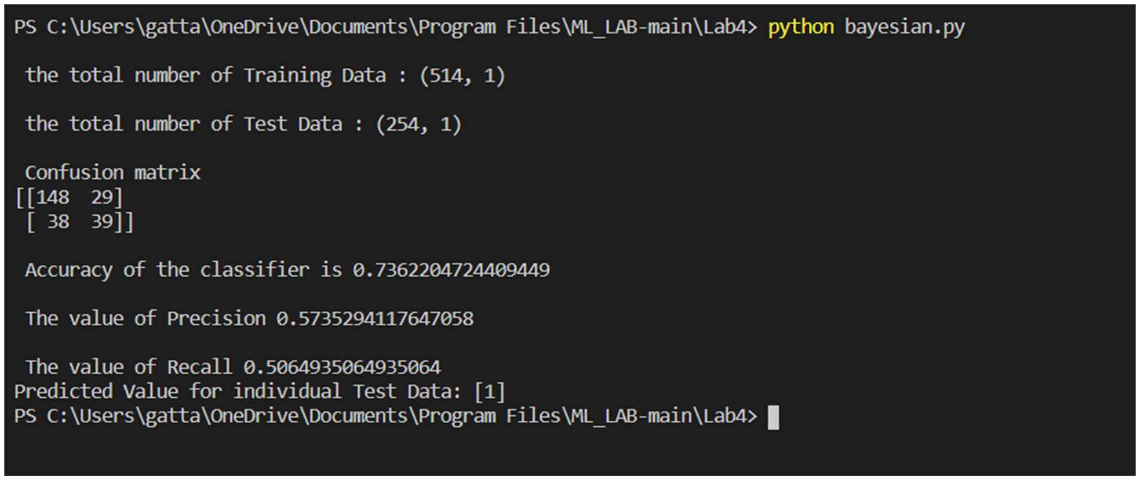
Program – 4

Question - 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

Code –

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

df = pd.read_csv("pima_indian.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness',
'insulin', 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']

X = df[feature_col_names].values # these are factors for the prediction
y = df[predicted_class_names].values # this is what we want to predict

#splitting the dataset into train and test data

xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)

print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)


# Training Naive Bayes (NB) classifier on training data.

clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])

#printing Confusion matrix, accuracy, Precision and Recall

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicte
d))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

OUTPUT –



```
PS C:\Users\gatta\OneDrive\Documents\Program Files\ML_LAB-main\Lab4> python bayesian.py

 the total number of Training Data : (514, 1)

 the total number of Test Data : (254, 1)

 Confusion matrix
[[148  29]
 [ 38  39]]

 Accuracy of the classifier is 0.7362204724409449

 The value of Precision 0.5735294117647058

 The value of Recall 0.5064935064935064
Predicted Value for individual Test Data: [1]
PS C:\Users\gatta\OneDrive\Documents\Program Files\ML_LAB-main\Lab4> 
```

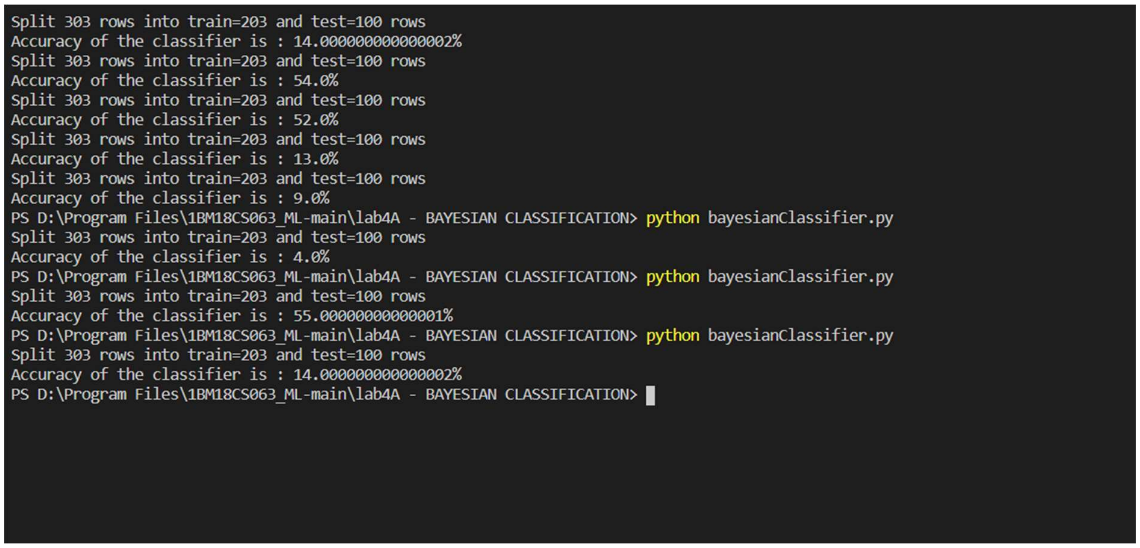
Program – 4A

Code –

```
import csv
import random
import math

def loadcsv(filename):
    lines = csv.reader(open(filename, "r"))
    dataset = list(lines)
    for i in range(len(dataset)):
       #converting strings into numbers for processing
        dataset[i] = [float(x) for x in dataset[i]]

    return dataset

def splitdataset(dataset, splitratio):
    #67% training size
    trainsize = int(len(dataset) * splitratio)
    trainset = []
    copy = list(dataset)
    while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for training data
        index = random.randrange(len(copy))
        trainset.append(copy.pop(index))
    return [trainset, copy]

def separatebyclass(dataset):
    separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

def summarize(dataset): #creates a dictionary of classes
```

```
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dat
aset)]
    del summaries[-1] #excluding labels +ve or -ve
    return summaries

def summarizebyclass(dataset):
    separated = separatebyclass(dataset)
    #print(separated)
    summaries = {}
    for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
        summaries[classvalue] = summarize(instances) #summarize is used to cal
 to mean and std
    return summaries

def calculateprobability(x, mean, stdev):
    if((2*math.pow(stdev,2)) == 0):
        x = (2*math.pow(stdev,2)) + 1
    else:
        x = (2*math.pow(stdev,2))
    exponent = math.exp(-(math.pow(x-mean,2)/x))
    if((math.sqrt(2*math.pi) * stdev) == 0):
        y = (math.sqrt(2*math.pi) * stdev) + 1
    else:
        y = (math.sqrt(2*math.pi) * stdev)
    return (1 / y) * exponent

def calculateclassprobabilities(summaries, inputvector):
    probabilities = {} # probabilities contains the all prob of all class of t
est data
    for classvalue, classsummaries in summaries.items():#class and attribute i
nformation as mean and sd
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i] #take mean and sd of every attribu
te for class 0 and 1 seperaely
            x = inputvector[i] #testvector's first attribute
            probabilities[classvalue] *= calculateprobability(x, mean, stdev)#
use normal dist
    return probabilities

def predict(summaries, inputvector): #training and test data is passed
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():#assigns that class w
hich has he highest prob
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel

def getpredictions(summaries, testset):
    predictions = []
    for i in range(len(testset)):
        result = predict(summaries, testset[i])
        predictions.append(result)
    return predictions

def getaccuracy(testset, predictions):
    correct = 0
    for i in range(len(testset)):
        if testset[i][-1] == predictions[i]:
            correct += 1
    return (correct/float(len(testset))) * 100.0

def main():
    # filename = 'naivedata.csv'
    filename = 'heart.csv'
    splitratio = 0.67
    dataset = loadcsv(filename)

    trainingset, testset = splitdataset(dataset, splitratio)
    print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset
), len(trainingset), len(testset)))
    # prepare model
    summaries = summarizebyclass(trainingset)
    #print(summaries)
    # test model
    predictions = getpredictions(summaries, testset) #find the predictions of
test data with the training data
    accuracy = getaccuracy(testset, predictions)
    print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

OUTPUT –



```
Split 303 rows into train=203 and test=100 rows
Accuracy of the classifier is : 14.000000000000002%
Split 303 rows into train=203 and test=100 rows
Accuracy of the classifier is : 54.0%
Split 303 rows into train=203 and test=100 rows
Accuracy of the classifier is : 52.0%
Split 303 rows into train=203 and test=100 rows
Accuracy of the classifier is : 13.0%
Split 303 rows into train=203 and test=100 rows
Accuracy of the classifier is : 9.0%
PS D:\Program Files\1BM18CS063_ML-main\lab4A - BAYESIAN CLASSIFICATION> python bayesianClassifier.py
Split 303 rows into train=203 and test=100 rows
Accuracy of the classifier is : 4.0%
PS D:\Program Files\1BM18CS063_ML-main\lab4A - BAYESIAN CLASSIFICATION> python bayesianClassifier.py
Split 303 rows into train=203 and test=100 rows
Accuracy of the classifier is : 55.00000000000001%
PS D:\Program Files\1BM18CS063_ML-main\lab4A - BAYESIAN CLASSIFICATION> python bayesianClassifier.py
Split 303 rows into train=203 and test=100 rows
Accuracy of the classifier is : 14.000000000000002%
PS D:\Program Files\1BM18CS063_ML-main\lab4A - BAYESIAN CLASSIFICATION>
```

Program – 5

Question - Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

Code –

```
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('heart.csv')

heartDisease = heartDisease.replace('?',np.nan)

#display the data

print('Sample instances from the dataset are given below')

print(heartDisease.head())

#display the Attributes names and datatyes

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

#Create Model- Bayesian Network

model=BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heart
disease'),('heartdisease','restecg'),('heartdisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators

print('\n Learning CPD using Maximum likelihood estimators')

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network

print('\n Inferencing with Bayesian Network:')

HeartDiseasetest_infer = VariableElimination(model)

#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)

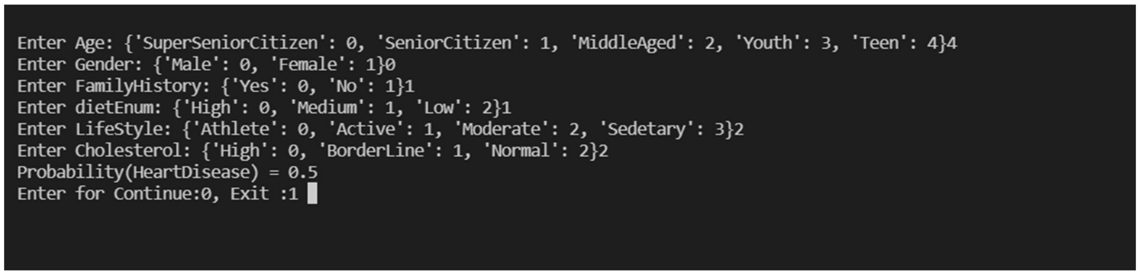
#computing the Probability of HeartDisease given cp

print('\n 2.Probability of HeartDisease given evidence= cp:2 ')

q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})

print(q2)

OUTPUT –



```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}4
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}2
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1
```