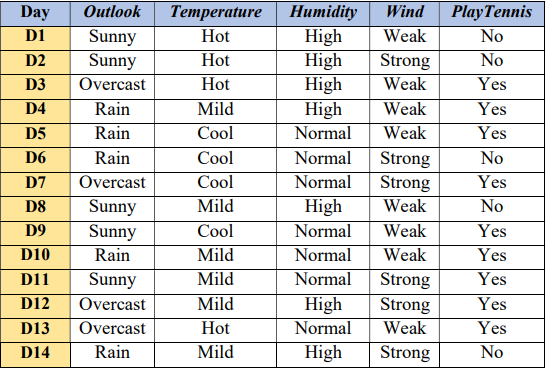
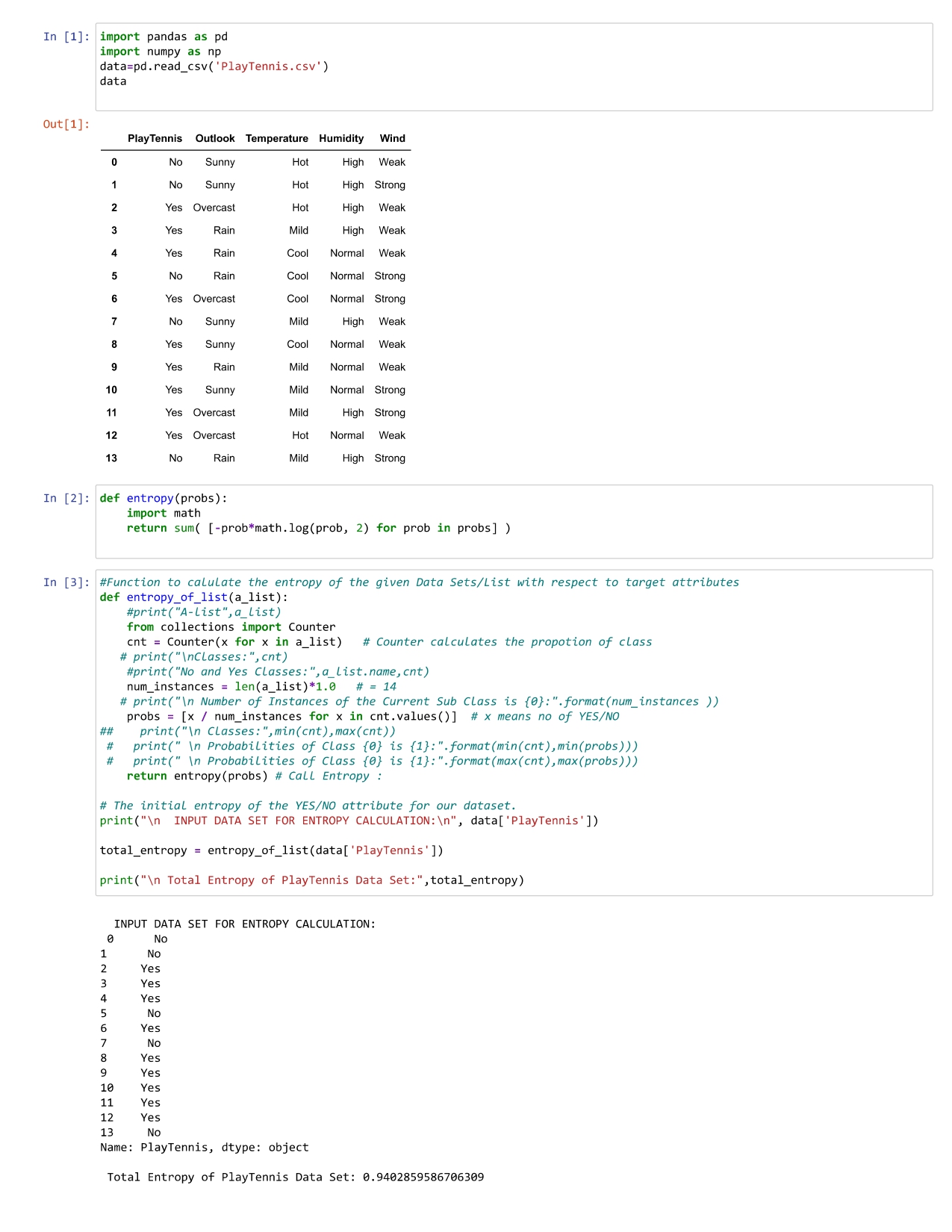
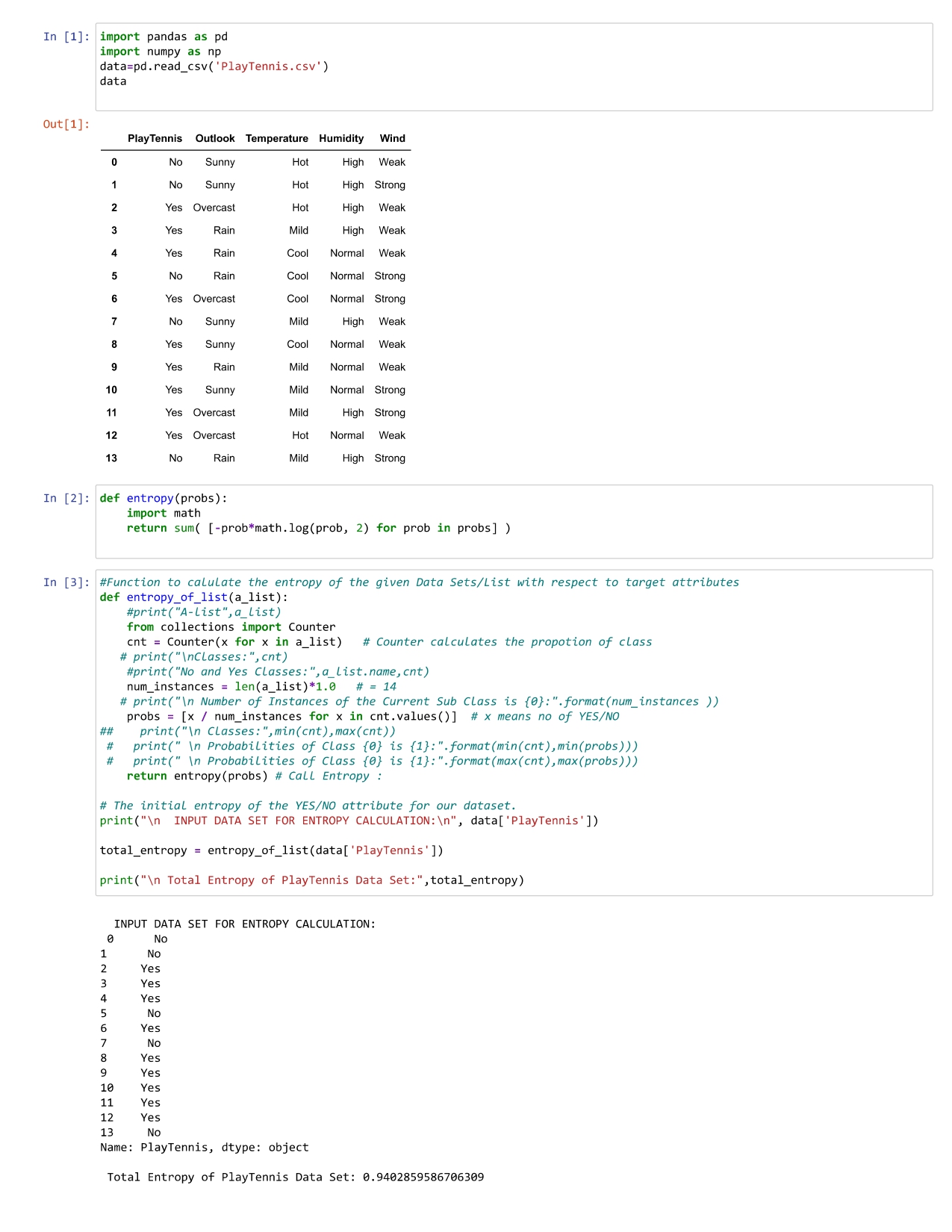
**Practical : 3**

**Aim:** 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.

**Data Set:**









# Practical: 4

## AIM: 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.

import numpy as np

# Data

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X / np.amax(X, axis=0) # Maximum of X array longitudinally

y = y / 100

# Sigmoid Function

def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

# Variable initialization

epoch = 5000 # Setting training iterations

lr = 0.1 # Setting learning rate

inputlayer\_neurons = 2 # Number of features in dataset

hiddenlayer\_neurons = 3 # Number of hidden layer neurons

output\_neurons = 1 # Number of neurons at output layer

# Weight and bias initialization

wh = np.random.uniform(size=(inputlayer\_neurons, hiddenlayer\_neurons))

bh = np.random.uniform(size=(1, hiddenlayer\_neurons))

wout = np.random.uniform(size=(hiddenlayer\_neurons, output\_neurons))

bout = np.random.uniform(size=(1, output\_neurons))

# Training

for i in range(epoch):

# Forward Propagation

hinp1 = np.dot(X, wh)

hinp = hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1 = np.dot(hlayer\_act, wout)

outinp = outinp1 + bout

output = sigmoid(outinp)

# Backpropagation

EO = y - output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act) # How much hidden layer weights contributed to error

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \* lr # Dot product of next layer error and current layer output

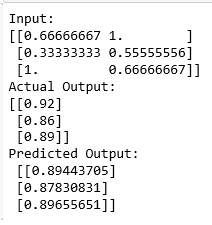
wh += X.T.dot(d\_hiddenlayer) \* lr

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n", output)

**Output:**



# Practical: 5

## AIM: 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.

 **Input:**

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("ml5.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

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)

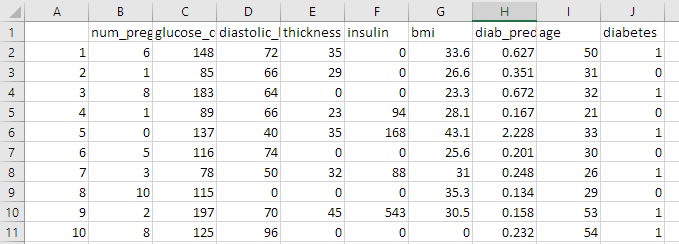
clf = GaussianNB().fit(xtrain,ytrain.ravel()) predicted = clf.predict(xtest)

predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])

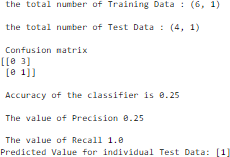
print('\n Confusion matrix') print(metrics.confusion\_matrix(ytest,predicted))

print('\n Accuracy of the classifier is',metrics.accuracy\_score(ytest,predicted)) 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)

 **ml5.csv**



 **Output:**



# Practical: 6

## AIM: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

 **Input:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer from sklearn.naive\_bayes import MultinomialNB

from sklearn import metrics

msg = pd.read\_csv('ml6.csv', names=['message', 'label']) msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})

X = msg.message y = msg.labelnum

xtrain, xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.2) # Specify the test\_size

print('The dimensions of the dataset', msg.shape)

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

count\_vect = CountVectorizer()

xtrain\_dtm = count\_vect.fit\_transform(xtrain) xtest\_dtm = count\_vect.transform(xtest)

print('\n The words or Tokens in the text documents:') print(count\_vect.get\_feature\_names\_out())

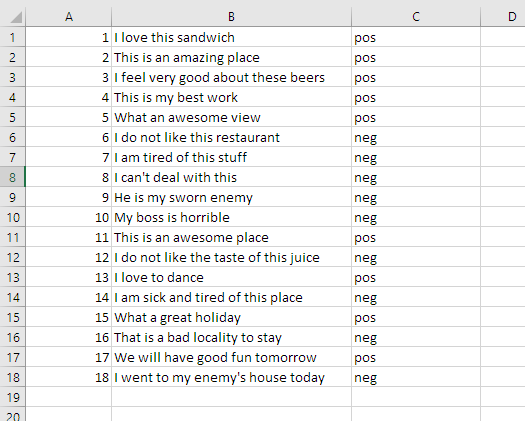
clf = MultinomialNB() clf.fit(xtrain\_dtm, ytrain) predicted = clf.predict(xtest\_dtm)

print('\n Accuracy of the classifier is', metrics.accuracy\_score(ytest, predicted)) print('\n Confusion matrix:')

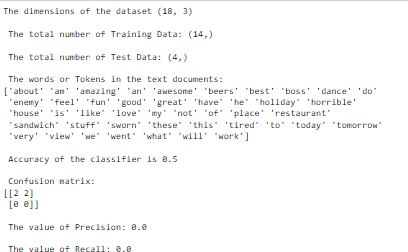
print(metrics.confusion\_matrix(ytest, predicted))

print('\n The value of Precision:', metrics.precision\_score(ytest, predicted)) print('\n The value of Recall:', metrics.recall\_score(ytest, predicted))

 **ml6.csv**



 **Output:**



# Practical: 7

## AIM: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API

 **Input :**

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

heartDisease = pd.read\_csv('ml7.csv') heartDisease = heartDisease.replace('?', np.nan)

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

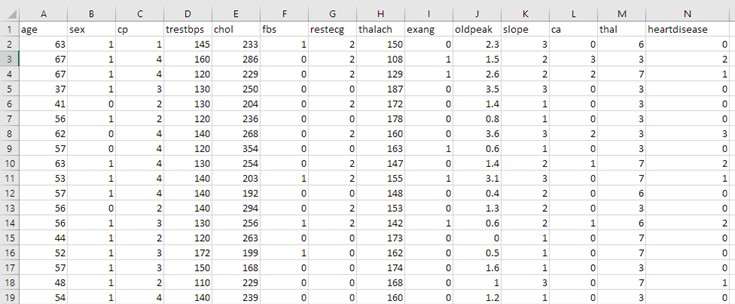
print('\nAttributes and datatypes') print(heartDisease.dtypes)

model = BayesianModel([('age', 'heartdisease'), ('sex', 'heartdisease'), ('exang', 'heartdisease'), ('cp', 'heartdisease'), ('heartdisease', 'restecg'), ('heartdisease', 'chol')]) print('\nLearning CPD using Maximum likelihood estimators')

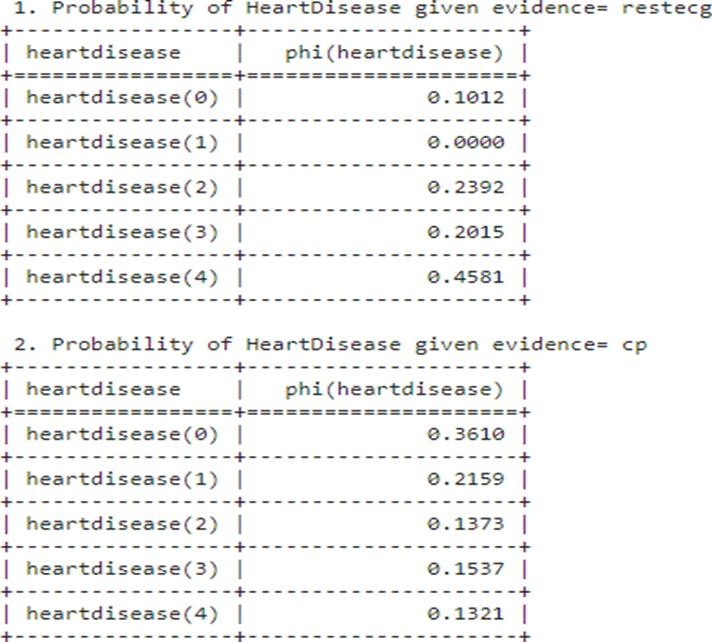
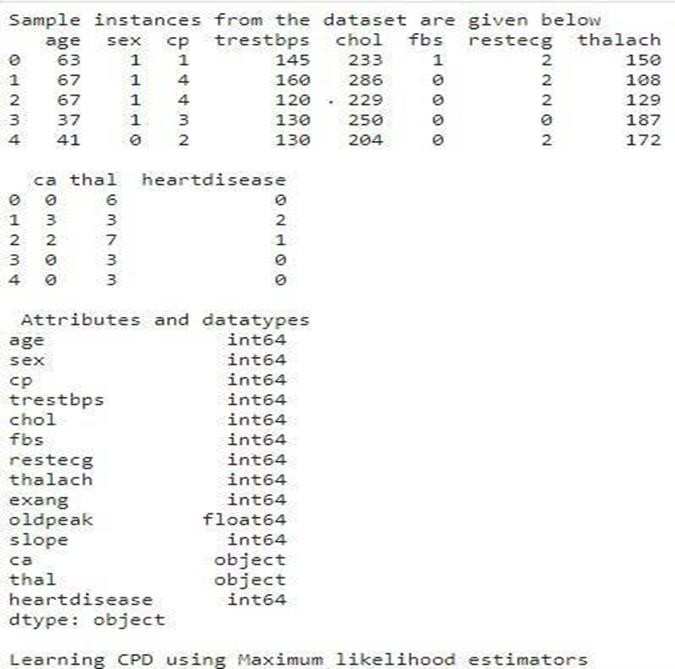
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator) print('\nInferencing with Bayesian Network:') HeartDiseasetest\_infer = VariableElimination(model)

print('\n1. Probability of HeartDisease given evidence= restecg') print(HeartDiseasetest\_infer.query(variables=['heartdisease'], evidence={'restecg': 1})) print('\n2. Probability of HeartDisease given evidence= cp') print(HeartDiseasetest\_infer.query(variables=['heartdisease'], evidence={'cp': 2}))

 **ml7.csv**



 **Output :**



# Practical: 8

## AIM: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

 **Input:**

import matplotlib.pyplot as plt from sklearn import datasets

from sklearn.cluster import KMeans 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'])

plt.subplot(1, 3, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40) plt.title('Real Clusters')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

plt.subplot(1, 3, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40) plt.title('K-Means Clustering')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

from sklearn import preprocessing

scaler = preprocessing.StandardScaler() scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns) from sklearn.mixture import GaussianMixture gmm = GaussianMixture(n\_components=40) gmm.fit(xs)

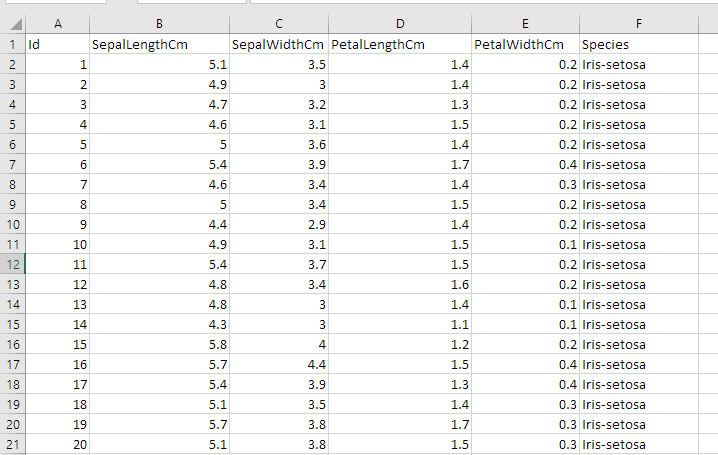
plt.subplot(1, 3, 3)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[0], s=40) plt.title('GMM Clustering')

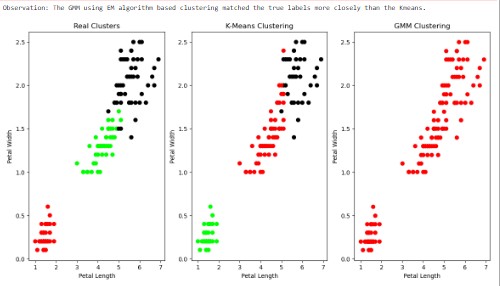
plt.xlabel('Petal Length') plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')

 **ml8.csv**



 **Output:**



# Practical: 9

## AIM: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API

 **Input:**

from sklearn.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix from sklearn import datasets

iris=datasets.load\_iris() x = iris.data

y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width') print(x)

print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica') print(y)

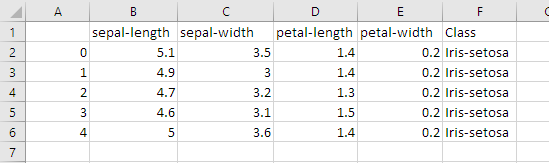
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3)

classifier = KNeighborsClassifier(n\_neighbors=5) classifier.fit(x\_train, y\_train)

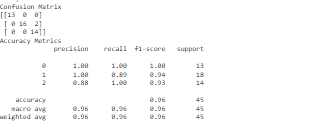
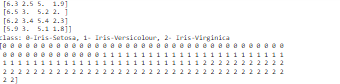
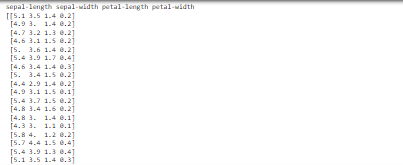
y\_pred=classifier.predict(x\_test)

print('Confusion Matrix') print(confusion\_matrix(y\_test,y\_pred)) print('Accuracy Metrics') print(classification\_report(y\_test,y\_pred))

 **ml9.csv**



 **Output:**



# Practical: 10

## AIM: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

 **Input :**

import numpy as np

from bokeh.plotting import figure, show from bokeh.layouts import gridplot

def local\_regression(x0, X, Y, tau): # Add a bias term

x0 = np.r\_[1, x0] # Add one to avoid the loss in information X = np.c\_[np.ones(len(X)), X]

# Fit model: normal equations with kernel

xw = X.T \* radial\_kernel(x0, X, tau) # XTranspose \* W

beta = np.linalg.pinv(xw @ X) @ xw @ Y # @ Matrix Multiplication or Dot Product # Predict value

return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction

def radial\_kernel(x0, X, tau):

return np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau \* tau)) # Weight or Radial Kernal Bias Function

n = 1000

# Generate dataset

X = np.linspace(-3, 3, num=n)

print("The Data Set (10 Samples) X:\n", X[1:10]) Y = np.log(np.abs(X \*\* 2 - 1) + 0.5)

print("The Fitting Curve Data Set (10 Samples) Y:\n", Y[1:10]) # Jitter X

X += np.random.normal(scale=0.1, size=n) print("Normalized (10 Samples) X:\n", X[1:10])

domain = np.linspace(-3, 3, num=300)

print("Xo Domain Space (10 Samples):\n", domain[1:10])

def plot\_lwr(tau):

# Prediction through regression

prediction = [local\_regression(x0, X, Y, tau) for x0 in domain]

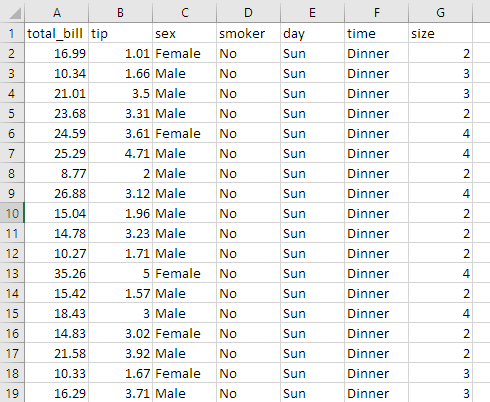
plot = figure(width=400, height=400) # Use 'width' and 'height' instead of 'plot\_width' and 'plot\_height'

plot.title.text = 'tau=%g' % tau plot.scatter(X, Y, alpha=0.3)

plot.line(domain, prediction, line\_width=2, color='red') return plot

show(gridplot([[plot\_lwr(10.0), plot\_lwr(1.0)], [plot\_lwr(0.1), plot\_lwr(0.01)]]))

##  ml10.csv



 **Output:**

