Practical - 7

AIM: Apply classification techniques in any programming language.

A. Apply classification technique with quality measures.

- **Program:**

```
def accuracy(y_true, y_pred):
    return round(float(sum(y_pred == y_true))/float(len(y_true)) * 100 ,2)

def pre_processing(df):
    X = df.drop([df.columns[-1]], axis = 1)
    y = df[df.columns[-1]]

    return X, y

def train_test_split(x, y, test_size = 0.25):
    x_test = x.sample(frac = test_size)
    y_test = y[x_test.index]
    x_train = x.drop(x_test.index)
    y_train = y.drop(y_test.index)

    return x_train, x_test, y_train, y_test


def fclass_prior():
  for outcome in np.unique(y_train):
outcome_count = sum(y_train == outcome)
class_priors[outcome] = outcome_count / train_size

def flikelihoods():
  for feature in features:
    for outcome in np.unique(y_train):
      likelihoods[feature][outcome]['mean'] = X_train[feature][y_train[y_train ==
outcome].index.values.tolist()].mean()
      likelihoods[feature][outcome]['variance'] = X_train[feature][y_train[y_train ==
outcome].index.values.tolist()].var()

def fit():
  for feature in features:
    likelihoods[feature] = {}

    for outcome in np.unique(y_train):
      likelihoods[feature].update({outcome:{}})
class_priors.update({outcome: 0})

fclass_prior()
flikelihoods()
```

```
def predict(X):
  results = []
  X = np.array(X)

  for query in X:
probs_outcome = {}

    for outcome in np.unique(y_train):
      prior = class_priors[outcome]
      likelihood = 1
evidence_temp = 1

      for feat, feat_val in zip(features, query):
        mean = likelihoods[feat][outcome]['mean']
        var = likelihoods[feat][outcome]['variance']
        likelihood *= (1/math.sqrt(2*math.pi*var)) * np.exp(-(feat_val - mean)**2 / (2*var))

posterior_numerator = (likelihood * prior)
probs_outcome[outcome] = posterior_numerator

    result = max(probs_outcome, key = lambda x: probs_outcome[x])
results.append(result)

  return np.array(results)

df = pd.read_csv("iris.csv")
X,y  = pre_processing(df)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)

features = list(X_train.columns)
likelihoods = {}
class_priors = {}
train_size ,num_feats = X_train.shape

fit()
print("Train Accuracy of the model:",accuracy_score(y_train, predict(X_train)))
print("Test Accuracy of the model:",accuracy_score(y_test, predict(X_test)))

d = np.array([[5.7, 2.9, 4.2, 1.3]])
print("\nPrediction:", predict(d))
```

- Output:

```
Train Accuracy of the model: 96.3
Test Accuracy of the model: 93.33

Prediction: ['Iris-versicolor']
```

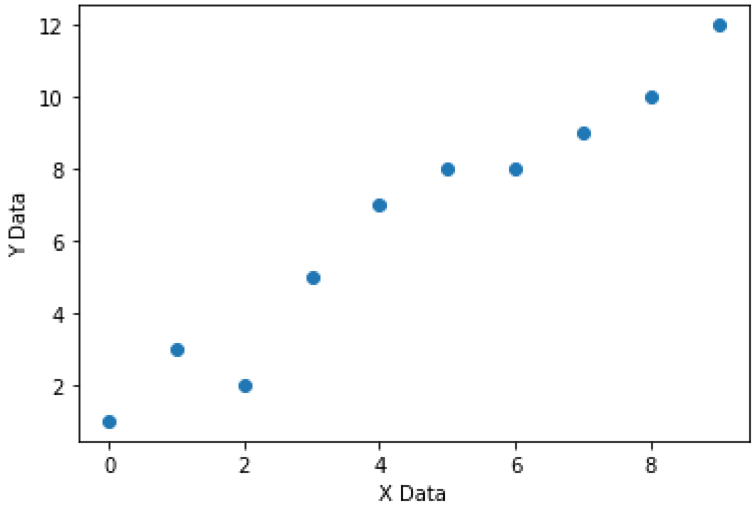
B. Implement regression technique.

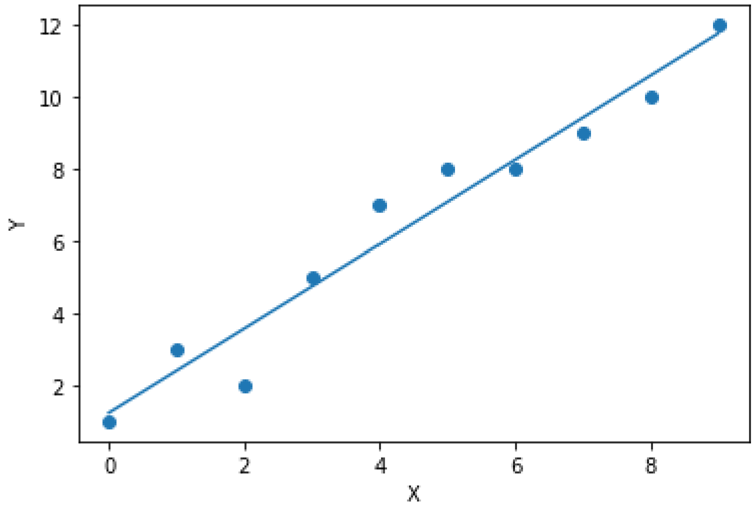
- **Program:**

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

X = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
m = len(y)

plt.scatter(X,y)
plt.xlabel('X Data')
plt.ylabel('Y Data')
plt.show()

X = X[:,np.newaxis]
y = y[:,np.newaxis]
th = np.zeros([2,1])
iter = 500
ap = 0.05
ones = np.ones((m,1))
X = np.hstack((ones, X))

def gradDescent(X, y, th, ap, iter):
    for _ in range(iter):
        temp = np.dot(X, th) - y
        temp = np.dot(X.T, temp)
th = th - (ap/m) * temp
    return th

th = gradDescent(X, y, th, ap, iter)
print(th)

def computeCost(X, y, th):
    temp = np.dot(X, th) - y
    return np.sum(np.power(temp, 2)) / (2*m)

J = computeCost(X, y, th)
print("Cost of the model:",J)

def score(X,y,th):
    temp = np.dot(X, th) - y
    return 100-np.mean(np.abs(temp))

print("Accuracy of the model:", score(X,y,th), "%")
```

```
plt.scatter(X[:,1], y)
plt.xlabel('X')
plt.ylabel('Y')
plt.plot(X[:,1], np.dot(X, th))
plt.show()

hours = 9.25
pred = np.dot([1.,hours], th)[0]
print("Prediction:",pred)

y_pred = np.dot(X,th)
err = np.abs(y - y_pred)

print("Mean Absolute Error:", np.mean(err))
```

○ Output:





```
[[1.2355268 ],  [1.16983042]]
Cost of the model: 0.2812122225945924
Accuracy of the model: 99.38301695766678 %
Prediction: 12.056458219710747
Mean Absolute Error: 0.6169830423332237
```