```
#vinayak arjun
#VU1F1920047
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
import warnings
import seaborn as sns
warnings.filterwarnings("ignore")


inputData = pd.read_csv('diabetes.csv')


from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,roc_curv

def model(algorithm,dtrain_x,dtrain_y,dtest_x,dtest_y,of_type):

    print ("*******************************************************************************
    print ("MODEL - OUTPUT")
    print ("*******************************************************************************
    algorithm.fit(dtrain_x,dtrain_y)
    predictions = algorithm.predict(dtest_x)
    print (algorithm)
    print ("\naccuracy_score :",accuracy_score(dtest_y,predictions))

    print ("\nclassification report :\n",(classification_report(dtest_y,predictions)))

    plt.figure(figsize=(13,10))
    plt.subplot(221)
    sns.heatmap(confusion_matrix(dtest_y,predictions),annot=True,fmt = "d",linecolor="k",l
    plt.title("CONFUSION MATRIX",fontsize=20)
    predicting_probabilites = algorithm.predict_proba(dtest_x)[:,1]
    fpr,tpr,thresholds = roc_curve(dtest_y,predicting_probabilites)
    plt.subplot(222)
    plt.plot(fpr,tpr,label = ("Area_under the curve :",auc(fpr,tpr)),color = "r")
    plt.plot([1,0],[1,0],linestyle = "dashed",color ="k")
    plt.legend(loc = "best")
    plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
    if  of_type == "feat":

        dataframe = pd.DataFrame(algorithm.feature_importances_,dtrain_x.columns).reset_in
        dataframe = dataframe.rename(columns={"index":"features",0:"coefficients"})
        dataframe = dataframe.sort_values(by="coefficients",ascending = False)
        plt.subplot(223)
        ax = sns.barplot(x = "coefficients" ,y ="features",data=dataframe,palette="husl")
        plt.title("FEATURE IMPORTANCES",fontsize =20)
        for i,j in enumerate(dataframe["coefficients"]):
            ax.text(.011,i,j,weight = "bold")
        plt.show()
    elif of_type == "none" :
        plt.show()
```

```
        return (algorithm)

def run_classifiers(train_X,train_Y,test_X,test_Y):
    print ("*************************************")
    print ("GRADIENT BOOST CLASSIFIER")
    print ("*************************************")

    from sklearn.ensemble import GradientBoostingClassifier
    gbc = GradientBoostingClassifier()
    model(gbc,train_X,train_Y,test_X,test_Y,"feat")
```

PCA

```
print ("**************************")
print ("PCA OF DATA AND  PREDICTION")
print ("**************************")
original = inputData.copy()
inputData = original.copy()
n_components = 2;
splitRatio = 0.2
pca = PCA(n_components = n_components).fit(inputData.loc[:, inputData.columns != 'Outcome'
pcaTransformedData = pca.transform(inputData.loc[:, inputData.columns != 'Outcome'])
pcaDictionary = {}
# Create the dictionarry
for i in range(n_components):
    key = "PCA_component_"+str(i)
    value = pcaTransformedData[:,i]
    pcaDictionary[key] = value
p = pd.DataFrame.from_dict(pcaDictionary)
p["Outcome"] = inputData.Outcome
train , test = train_test_split(p,test_size = splitRatio,random_state = 123)
#Seperating Predictor and target variables
train_X = train[[x for x in train.columns if x not in ["Outcome"]]]
train_Y = train[["Outcome"]]
test_X  = test[[x for x in test.columns if x not in ["Outcome"]]]
test_Y  = test[["Outcome"]]
run_classifiers(train_X,train_Y,test_X,test_Y)
```

```
**************************
PCA OF DATA AND  PREDICTION
**************************
************************************
GRADIENT BOOST CLASSIFIER
************************************
*****************************************************************************************
MODEL - OUTPUT
*****************************************************************************************
GradientBoostingClassifier()

accuracy_score : 0.7272727272727273

classification report :
              precision    recall  f1-score   support

           0       0.74      0.86      0.80        96
           1       0.69      0.50      0.58        58

    accuracy                           0.73       154
   macro avg       0.72      0.68      0.69       154
weighted avg       0.72      0.73      0.72       154
```