**Name** : B.M.G.G.K. Rajapaksha

**Index No.** : S14210

**Faculty** : Faculty of Science, Bioinformatics

**CS 4104 – Data Analytics – Assignment 2**

**a. Screenshots with an explanation of the tools you used for the above training process.**

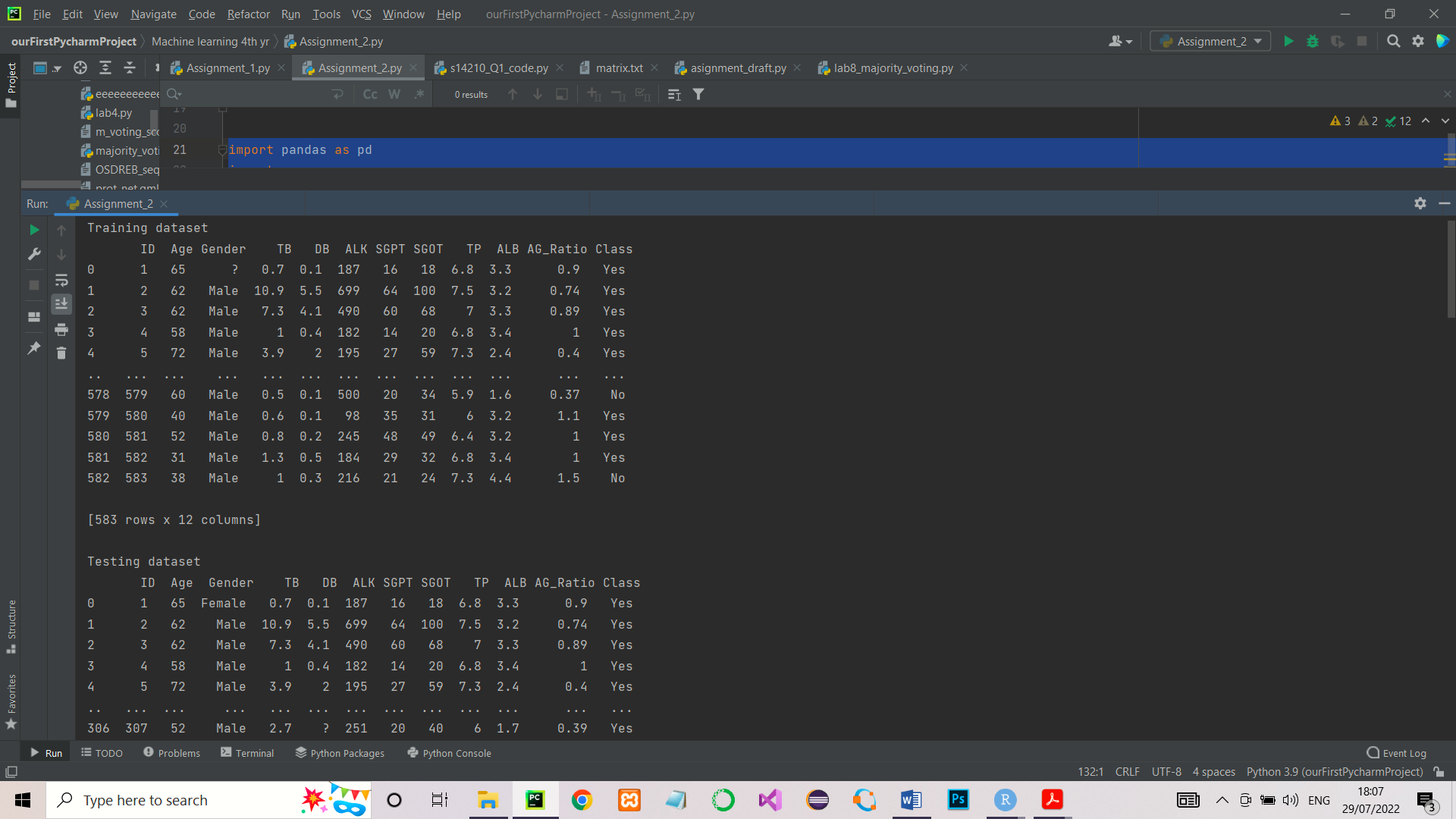
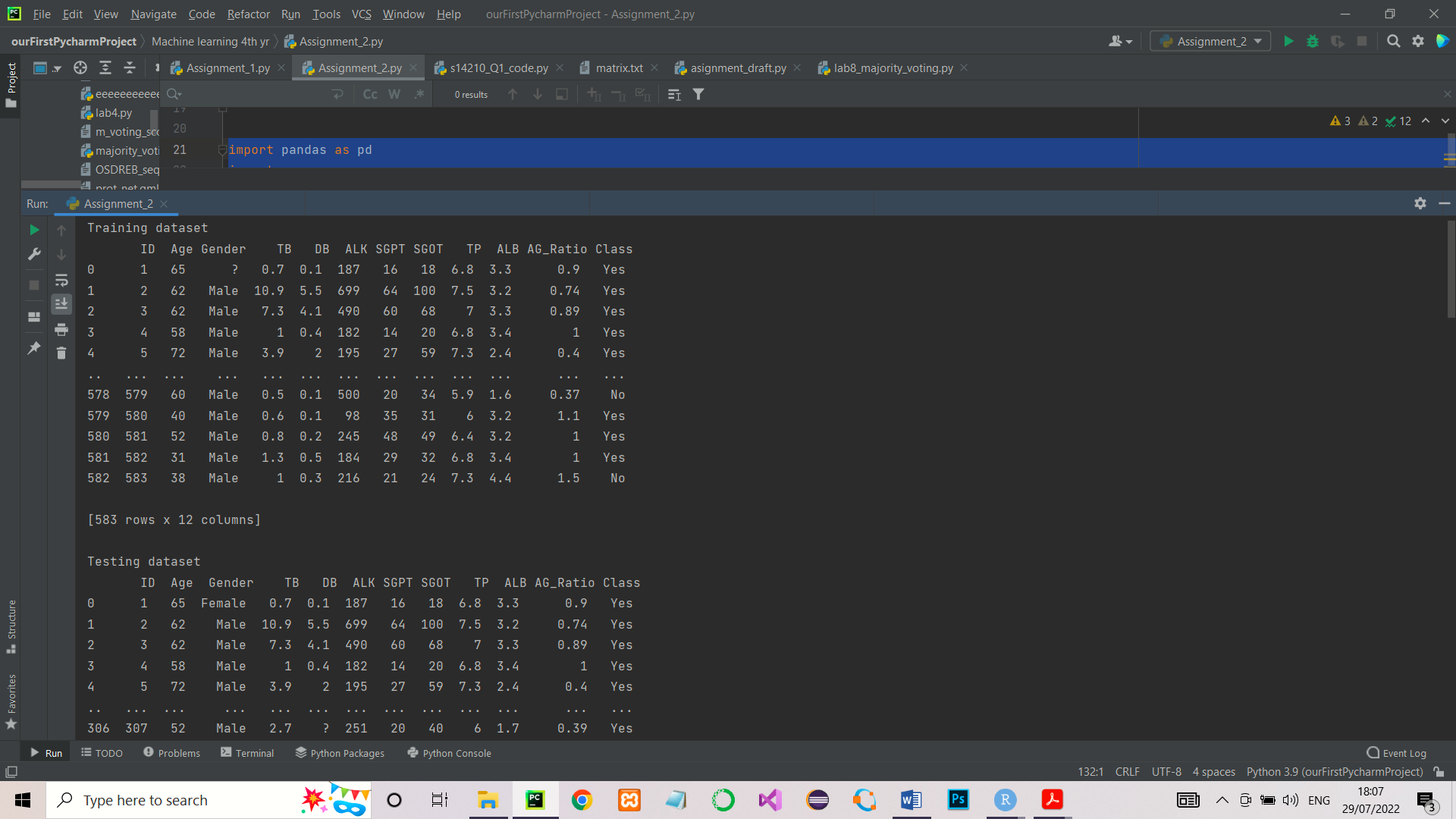
import pandas as pd  
import numpy as np  
from sklearn.preprocessing import StandardScaler  
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import GridSearchCV  
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay, recall\_score, accuracy\_score, precision\_score  
import matplotlib.pyplot as plt

**pandas –** It is a fast powerful, flexible and easy to use open source data analysis and manipulation tool which can be used for python programming. In here it is used to represent the training and testing datasets using pandas dataframe structure.

datasets = pd.ExcelFile('SCS4204\_IS4103\_CS4104 \_dataset.xlsx')  
training = pd.read\_excel(datasets, 'Training Dataset')  
testing = pd.read\_excel(datasets, 'Testing Dataset')

**Output**



**numpy –** NumPy is a fundamental package for scientific computing with python. It supports for large, multi – dimensional arrays and matrices.

Here **np.NaN** was used which is the IEEE 754 floating-point representation for Not a Number. It was used under data pre-processing part to replace ‘?’ values from the dataset.

# Replace '?' values from the mean value of the respective columns (training dataset)  
  
empty\_not\_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT', 'TP', 'ALB', 'AG\_Ratio', 'Class']  
  
accepted = empty\_not\_accepted  
  
for column in accepted:  
 if column not in ('Gender', 'Class'):  
 training[column] = training[column].replace('?', np.NaN)  
 mean = int(training[column].mean(skipna=True))  
 training[column] = training[column].replace(np.NaN, mean)  
  
 else:  
 training[column] = training[column].replace('?', np.NaN)  
 training = training.dropna(how='any', axis=0)

**sklearn –** Sklearn is a highly recommended and wide using machine learning library package for the python programming. It consists set of machine learning tools that are used in most machine learning problems.

**sklearn.preprocessing –** This package provide several common utility functions and transformer classes to change raw feature vectors into more suitable representation.

**standardScaler –** This is a class under **sklearn.preprocessing** to standardizefeatures by removing the mean and scaling to unit variance. Standardising of the datasets is essential requirement for machine learning estimators.

standardScaler calculates standard score of a sample x (here x = training and testing datasets) as follows;

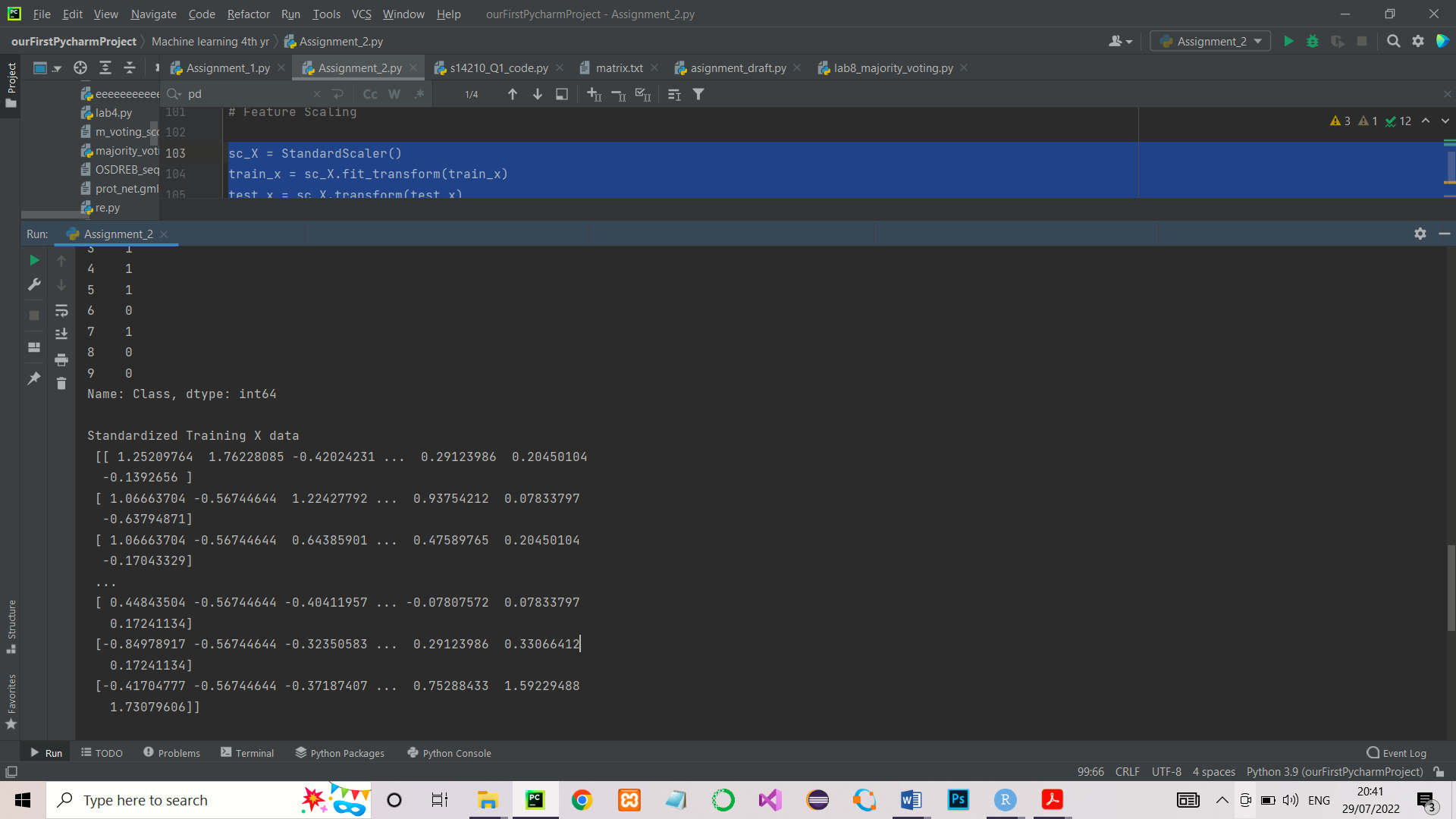
Standard score (Z) = (x-u)/s

Where;

* u = mean of the training samples or zero if with\_mean = False. (in this problem, with\_mean = True (default) )
* s = standard deviation of the training sample or one if with\_std=False (in this problem, with\_std = True (default) )

sc\_X = StandardScaler()  
train\_x = sc\_X.fit\_transform(train\_x)  
test\_x = sc\_X.transform(test\_x)  
  
print('Standardized Training X data\n', train\_x, '\n')

**Output**



**sklearn.model\_selection –** It is a method for setting a blueprint to analyse data and then using it to measure new data.

**sklearn.neighbors –** This provides functionality for unsupervised and supervised neighbours – based learning methods.

**KNeighborsClassifier –** This is the classifier that implement the k-nearest neighbours vote under sklearn.neighbors. Classification is computed from a simple majority vote of the nearest neighbours of each point. The majority voted class is selected as the class of the candidate data point.

# Define the model  
knn = KNeighborsClassifier()

**GridSearchCV –** This is a library function under sklearn.model\_selection. It is used to loop through predefined hypeparameters and fit the model (here knn) on the training dataset. At the end it gives the best set of hyperparameters from the given set.

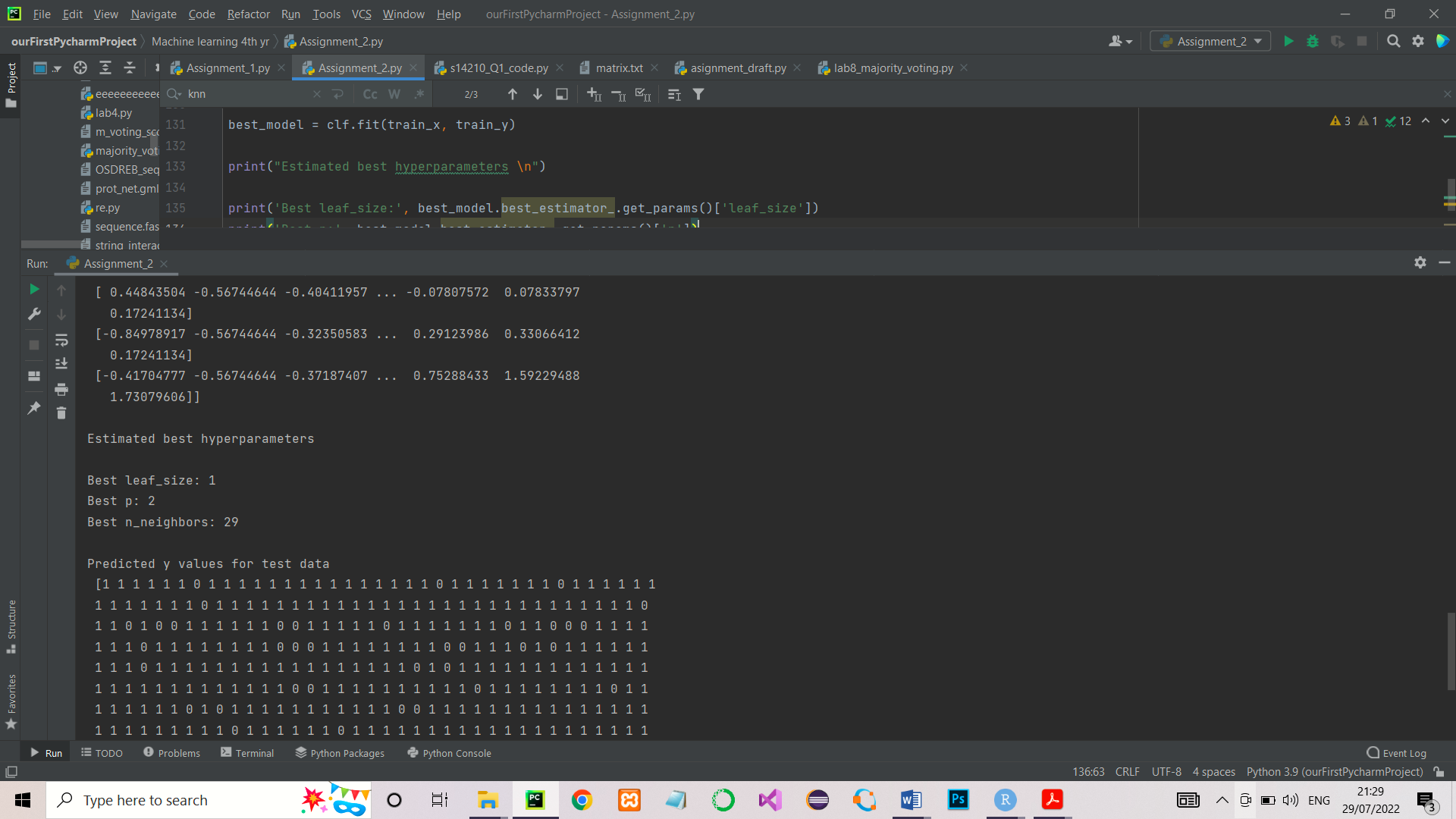
Used parameters in GridSearchCV

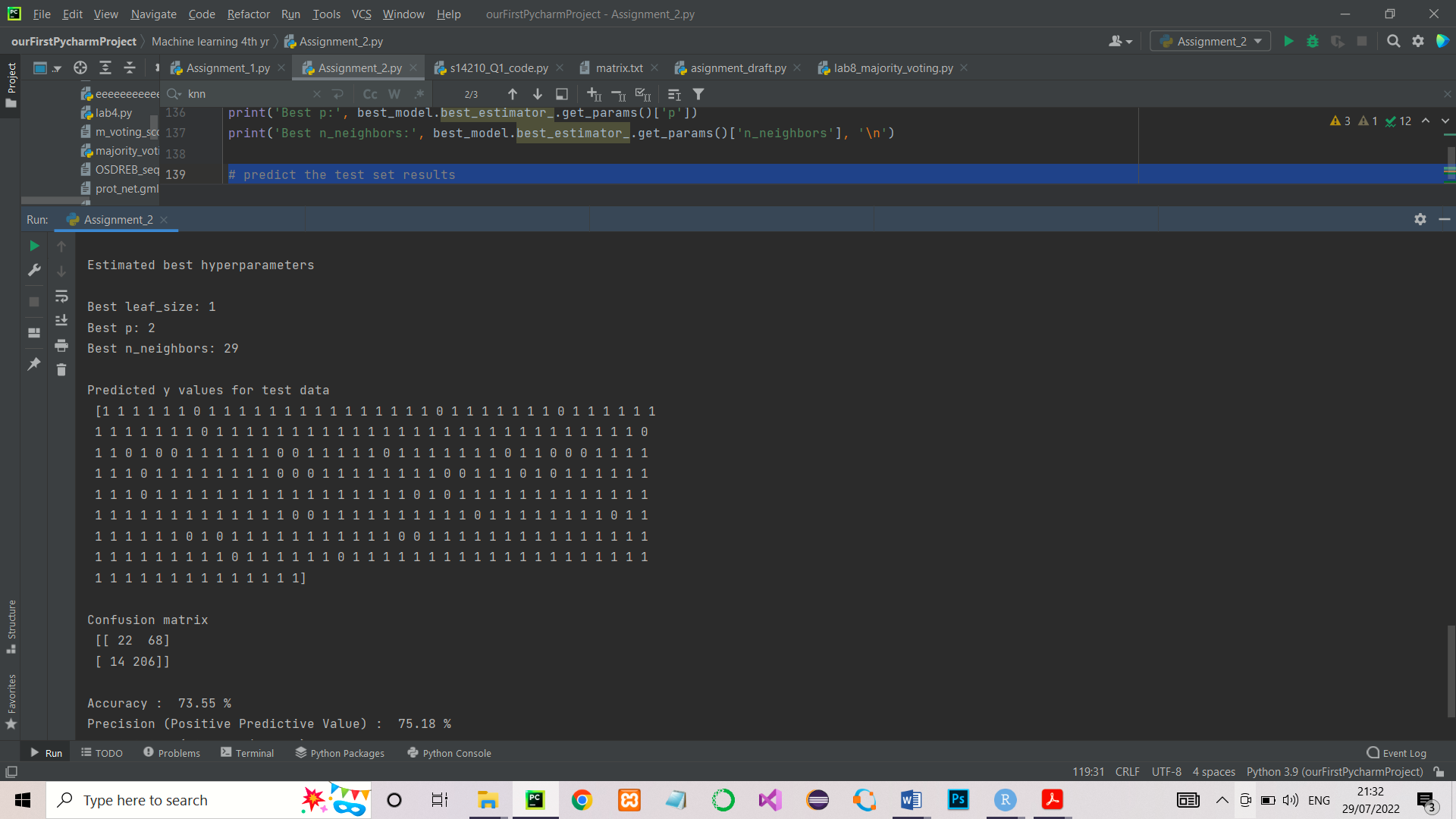
* **cv** = Class validation. This determines the cross validation splitting strategy. Here 10 is used as the cv value. Hence it split the dataset into 10 equal sets. Then select one set as a testing set and others as the training set. This is repeats until it covers the all 10 folds.
* **hyperparameters;**
  + leaf\_size (controls the minimum no. of points in a given node)
  + n\_neighbors (number of nearest neighbours)
  + p (power parameter for the Minkkowski metric)

are selected as hyperparameters which want to tune.

# List of Hyperparameters that we want to tune  
  
leaf\_size = list(range(1, 30))  
n\_neighbors = list(range(1, 30))  
p = [1, 2]  
  
# get hyperparameters into a dictionary  
  
hyperparameters = dict(leaf\_size=leaf\_size, n\_neighbors=n\_neighbors, p=p)  
  
# Define the model  
knn = KNeighborsClassifier()  
  
# Use GridSearch  
clf = GridSearchCV(knn, hyperparameters, cv=10) # cv = class validation  
  
# here cv = 10 means we have to divide the dataset into 5 sets/folds  
  
# Fit the model  
  
best\_model = clf.fit(train\_x, train\_y)  
  
print("Estimated best hyperparameters \n")

**Output**





**sklearn.metrics –** This module implements functions assessing prediction error for specific purposes.

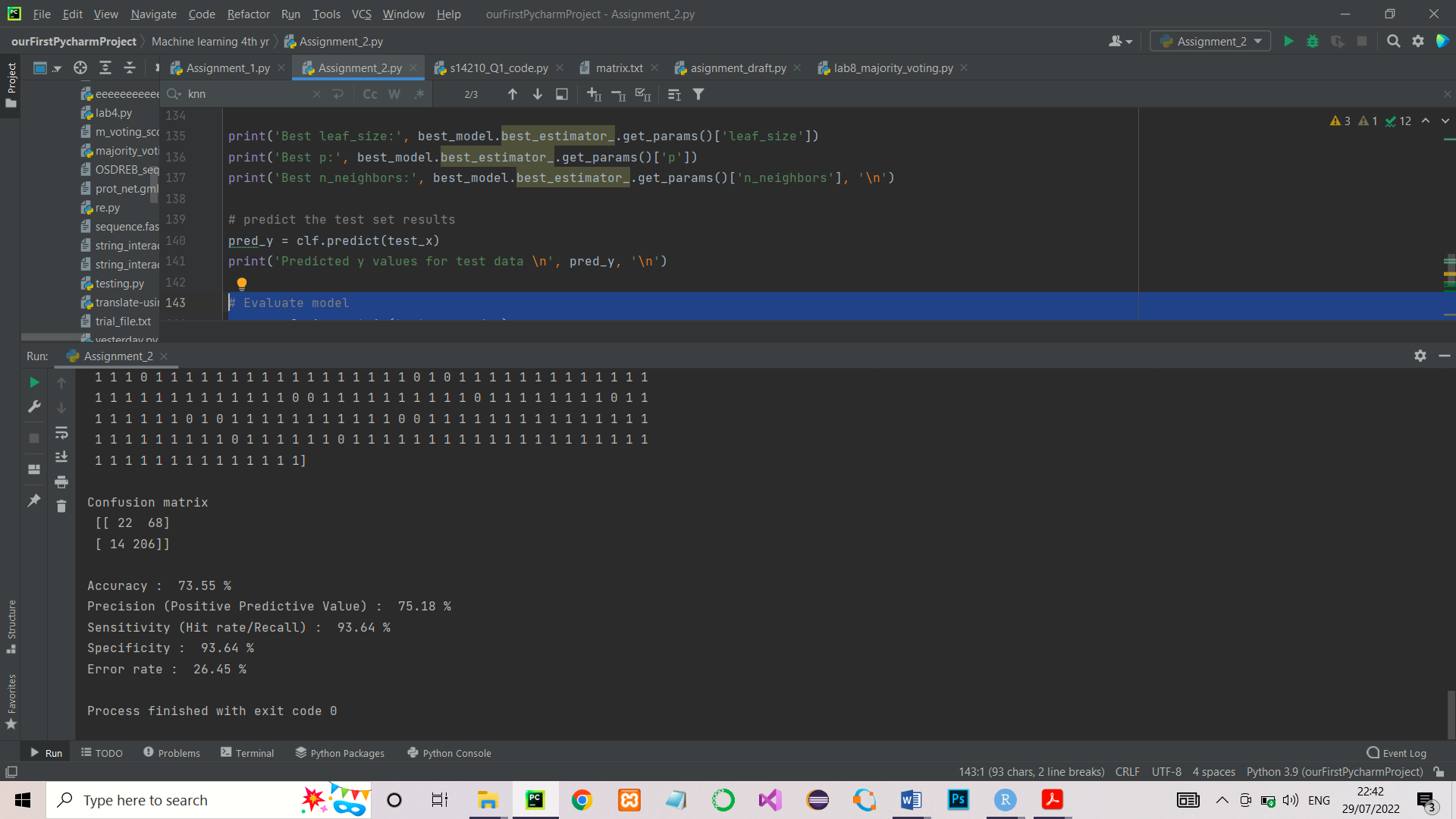
**confusion\_matrix –** Compute confusion matrix to evaluate the accuracy of a classification. According to the definition;

Confusion matrix is such that Ci,j = no of observations known to be in group i and predicted to be in group j.

Since this problem is a binary classification,

* C0,0 = true negative’
* C1,0 = false negative
* C1,1 = true positive
* C0,1 = false positive

# Evaluate model  
cm = confusion\_matrix(test\_y, pred\_y)  
print('Confusion matrix \n', cm, '\n')

Output

**ConfusionMatrixDisplay –** For confusion matrix visualization.

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=['No', 'Yes'])

**matplotlib –** Is a plotting library for python programming.

**matplotlob.pyplot –** pyplot is an API(Aplication Programming Interface) for python’s matplotlib that effectively makes matplotlib a viable open source alternative to MATLAB. Here these two are used to visualize the confusion matrix as a plot.

# Confusion matrix plot  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=['No', 'Yes'])  
disp.plot()  
plt.show()

**recall\_score, accuracy\_score, precision\_score –** These are used to calculate the Specificity, Accuracy and Precision of the KNN model.

print('Accuracy : ', round((accuracy\_score(test\_y, pred\_y)\*100), 2), '%')  
print('Precision (Positive Predictive Value) : ', round((precision\_score(test\_y, pred\_y)\*100), 2), '%')  
print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))\*100, 2), '%')  
print('Specificity : ', round((recall\_score(test\_y, pred\_y)\*100), 2), '%')  
print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))\*100, 2), '%')

**b. Brief explanation of the pre-processing steps you followed.**

1. First, the nominal data columns (Gender and Class) were selected and each nominal categories were replaced by 0 and 1 in both training and testing datasets.

* Male = 1
* Female = 0
* No = 0
* Yes = 1

# give numerical labels for 'Gender' and 'Class' attribute values in testing dataset  
  
training['Gender'].replace(['Male', 'Female'], [0, 1], inplace=True)  
training['Class'].replace(['Yes', 'No'], [1, 0], inplace=True)  
  
testing['Gender'].replace(['Male', 'Female'], [0, 1], inplace=True)  
testing['Class'].replace(['Yes', 'No'], [1, 0], inplace=True)

1. Then the complete two datasets were scanned through a loop to check the ‘?’ symbolic data entries these entries are replaced by **NaN** values. Except the **Gender** and **Class** columns, all other Columns were scanned again through a loop and replace all NaN values from the mean values of their respective column.

Replacement of the NaN values in Gender and Class columns by their means is not meaningful. Hence these rows were completely removed from the training and testing datasets.

# Replace '?' values from the mean value of the respective columns (training dataset)  
  
empty\_not\_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT', 'TP', 'ALB', 'AG\_Ratio', 'Class']  
  
accepted = empty\_not\_accepted  
  
for column in accepted:  
 if column not in ('Gender', 'Class'):  
 training[column] = training[column].replace('?', np.NaN)  
 mean = int(training[column].mean(skipna=True))  
 training[column] = training[column].replace(np.NaN, mean)  
  
 else:  
 training[column] = training[column].replace('?', np.NaN)  
 training = training.dropna(how='any', axis=0)  
  
# Replace '?' values from the mean value of the respective columns (testing dataset)  
  
empty\_not\_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT', 'TP', 'ALB', 'AG\_Ratio', 'Class']  
  
accepted = empty\_not\_accepted  
  
for column in accepted:  
 if column not in ('Gender', 'Class'):  
 testing[column] = testing[column].replace('?', np.NaN)  
 mean = int(testing[column].mean(skipna=True))  
 testing[column] = testing[column].replace(np.NaN, mean)  
 else:  
 testing[column] = testing[column].replace('?', np.NaN)  
 testing = testing.dropna(how='any', axis=0)

1. Both training and testing datasets were spit into X and Y variables**.** Only the **Class** column was selected in to the Y variable and rest of the columns except the **Id** column of the both training and testing data sets were selected to the X.

# Split the training dataset into x and y  
  
train\_x = training.iloc[:, 1:11] # [all rows,column 1-11]  
train\_y = training.iloc[:, 11] # [all rows,column 11]  
  
print('First 10 values of train\_x\n', train\_x.head(10), '\n')  
print('First 10 values of train\_y\n', train\_y.head(10), '\n')  
  
# Split the testing dataset into x and y  
  
test\_x = testing.iloc[:, 1:11] # [all rows,column 1-11]  
test\_y = testing.iloc[:, 11] # [all rows,column 11]  
  
print('First 10 values of test\_x\n', test\_x.head(10), '\n')  
print('First 10 values of test\_y\n', test\_y.head(10), '\n')

1. Finally both training and testing X data sets were standardized by using  **standardScaler()** function. Standardising of the datasets is essential requirement for machine learning estimators.

standardScaler calculates standard score of a sample x (here x = training and testing datasets) as follows;

Standard score (Z) = (x-u)/s

Where;

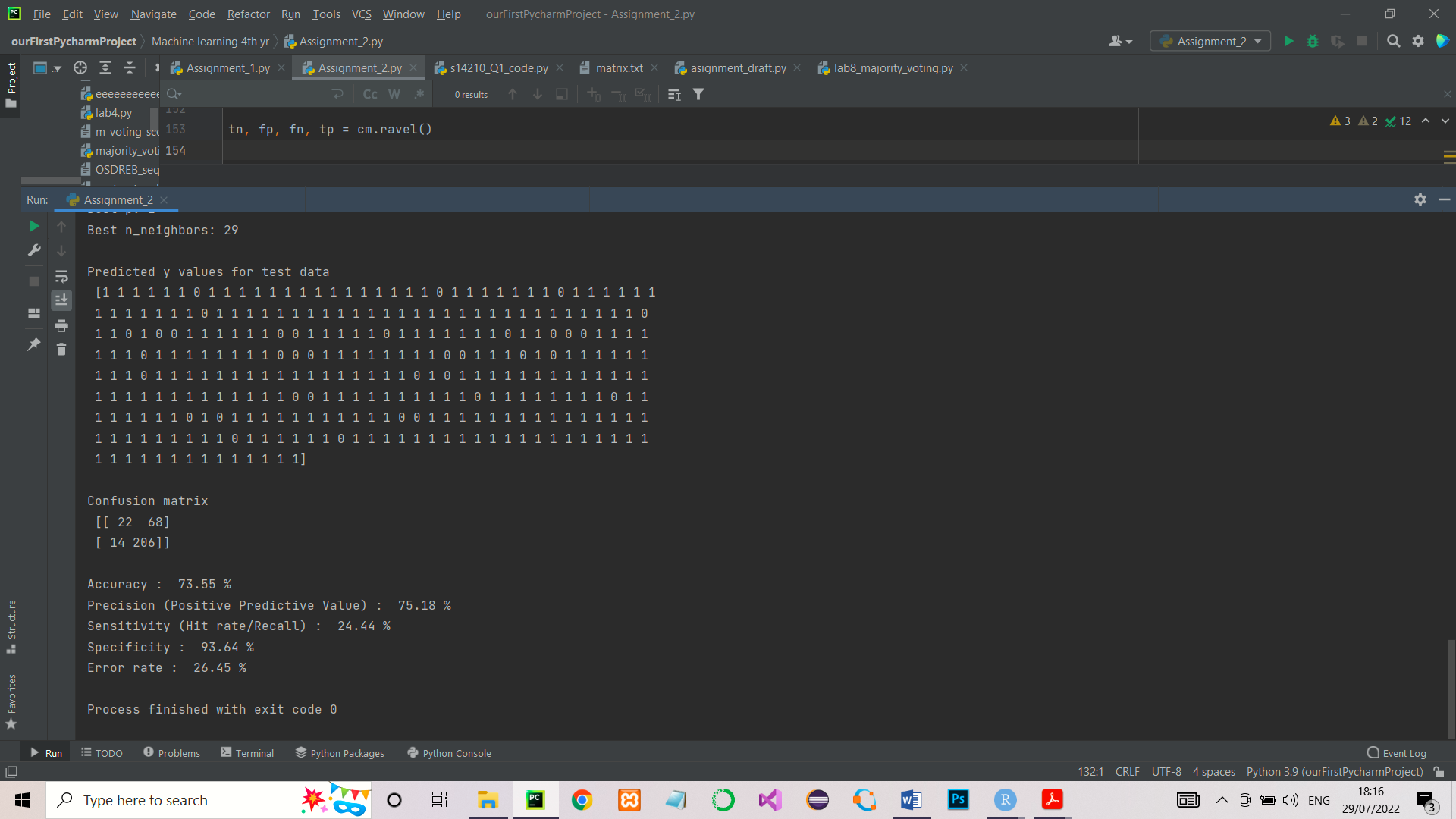
* u = mean of the training samples or zero if with\_mean = False. (in this problem, with\_mean = True (default) )
* s = standard deviation of the training sample or one if with\_std=False (in this problem, with\_std = True (default) )

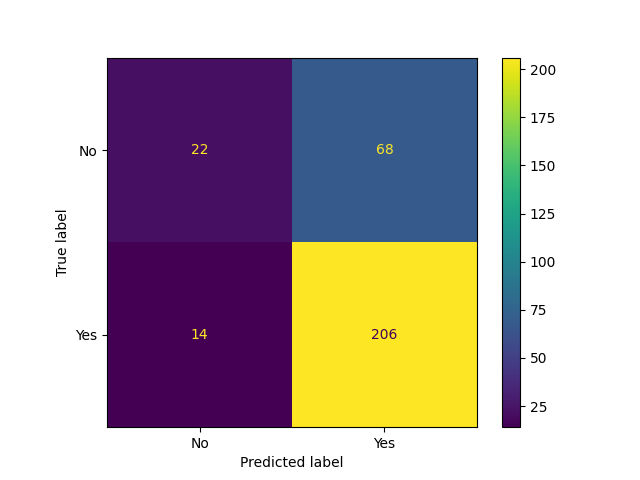
# Feature Scaling  
  
sc\_X = StandardScaler()  
train\_x = sc\_X.fit\_transform(train\_x)  
test\_x = sc\_X.transform(test\_x)  
  
print('Standardized Training X data\n', train\_x, '\n')

**c. Generated *Confusion matrix* for the Test dataset.**

# predict the test set results  
pred\_y = clf.predict(test\_x)  
print('Predicted y values for test data \n', pred\_y, '\n')  
  
# Evaluate model  
cm = confusion\_matrix(test\_y, pred\_y)  
print('Confusion matrix \n', cm, '\n')  
  
# Confusion matrix plot  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=['No', 'Yes'])  
disp.plot()  
plt.show()

**Output**



****

**d. List of below measures calculated for the Test dataset.**

**I. Accuracy :** 73.55%

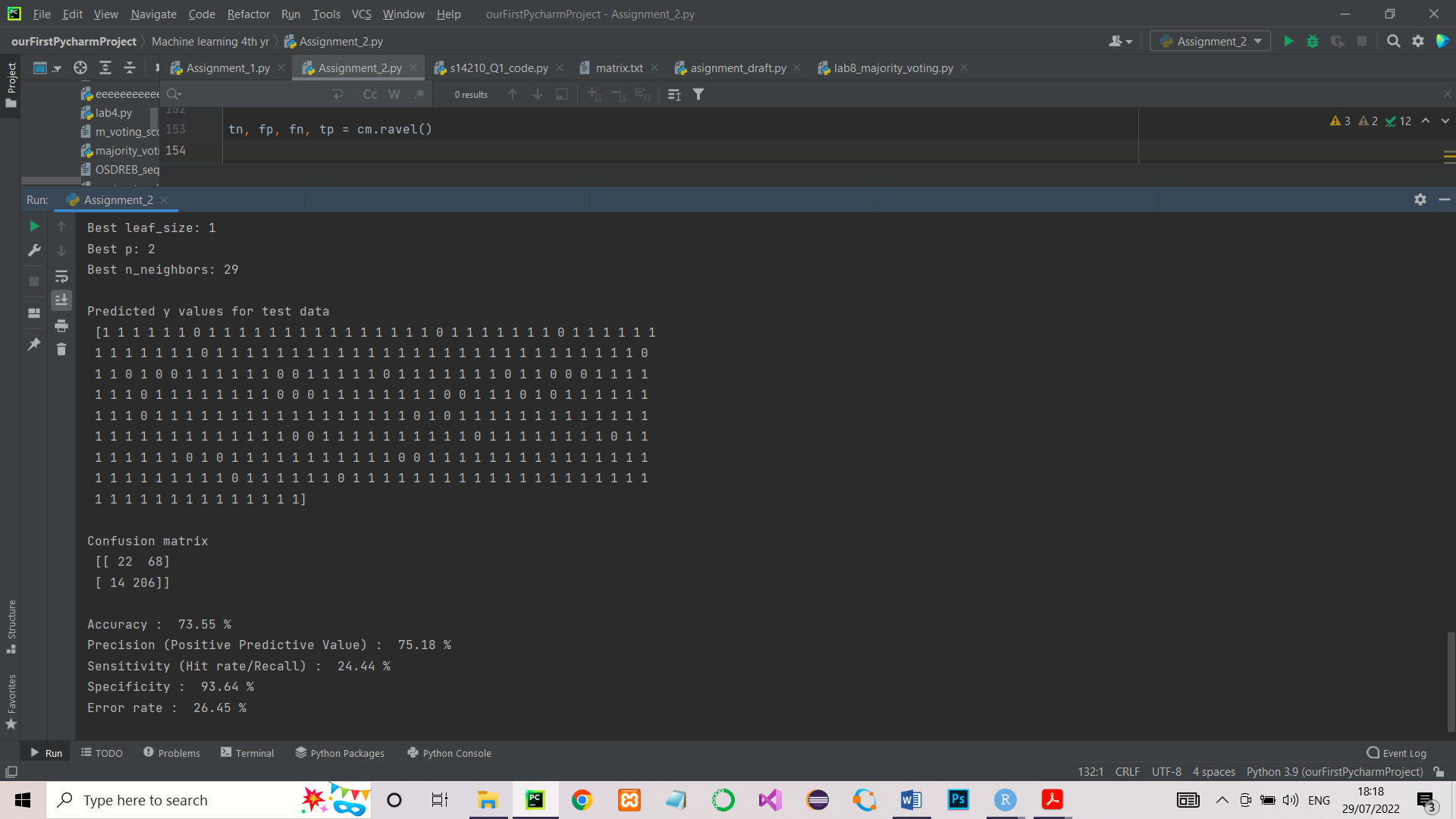
**II. Precision :** 75.18%

**III. Sensitivity :** 24.44%

**IV. Specificity :** 93.64%

**V. Error Rate :** 26.45%

print('Accuracy : ', round((accuracy\_score(test\_y, pred\_y)\*100), 2), '%')  
print('Precision (Positive Predictive Value) : ', round((precision\_score(test\_y, pred\_y)\*100), 2), '%')  
print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))\*100, 2), '%')  
print('Specificity : ', round((recall\_score(test\_y, pred\_y)\*100), 2), '%')  
print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))\*100, 2), '%')

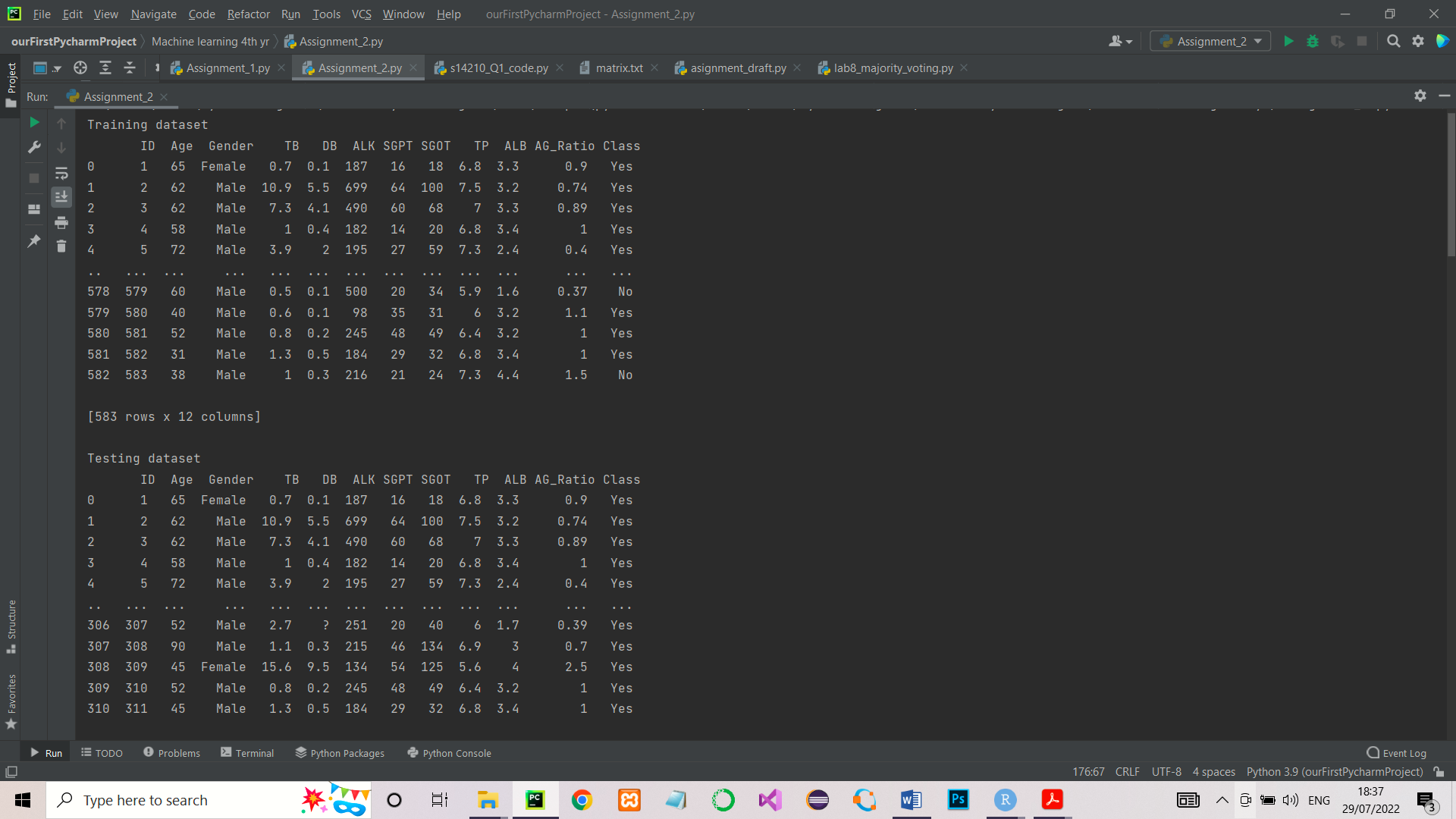


**Manual calculations**

# tn = true negative = 22  
# tp = true positive = 206  
# fn = false negative = 14  
# fp = false positive = 68  
  
# Manual calculations  
  
# print('Accuracy : ', round(((tp+tn)/(tp+fn+fp+tn))\*100, 2), '%')  
# print('Precision (Positive Predictive Value) : ', round((tp/(tp+fp))\*100, 2), '%')  
# print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))\*100, 2), '%')  
# print('Specificity : ', round((tn/(tn+fp))\*100, 2), '%')  
# print('FN Rate (Miss rate) : ', round((fn/(tp+fn))\*100, 2), '%')  
# print('TP Rate (False Alarm Rate) : ', round((fp/(fp+tn))\*100, 2), '%')  
# print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))\*100, 2), '%')

**e. Append your full code lines at the end of the PDF file**

*"""  
Name : B.M.G.G.K. Rajapaksha  
Faculty : Faculty of Science, Bioinformatics  
Index NO. : S14210  
Date : 25/07/2022  
  
Task : Assignment 2; A supervised machine learning algorithm (KNN) to train and test the  
 SCS4204\_IS4103\_CS4104 \_dataset.xlsx dataset  
  
Input : 1. SCS4204\_IS4103\_CS4104 \_dataset.xlsx dataset  
 (It contains training and testing data)  
  
Output : 1. Accuracy  
 2. Precision  
 3. Sensitivity  
 4. Specificity  
 5. Error Rate  
  
"""*import pandas as pd  
import numpy as np  
from sklearn.preprocessing import StandardScaler  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay, recall\_score, accuracy\_score, precision\_score  
import matplotlib.pyplot as plt  
  
# load the datasets  
  
datasets = pd.ExcelFile('SCS4204\_IS4103\_CS4104 \_dataset.xlsx')  
training = pd.read\_excel(datasets, 'Training Dataset')  
testing = pd.read\_excel(datasets, 'Testing Dataset')  
  
print('Training dataset\n', training, '\n')  
print('Testing dataset\n', testing, '\n')  
  
print('Length of the testing dataset - ', len(testing))  
print('Length of the training dataset - ', len(training), '\n')  
  
# Pre processing the dataset  
# convert categorical values into numerical values  
# both 'Gender' and 'Class' attributes are categorical variable. Hence need to convert them into numerical values before  
# preprocessing  
  
# give numerical labels for 'Gender' and 'Class' attribute values in testing dataset  
  
training['Gender'].replace(['Male', 'Female'], [0, 1], inplace=True)  
training['Class'].replace(['Yes', 'No'], [1, 0], inplace=True)  
  
testing['Gender'].replace(['Male', 'Female'], [0, 1], inplace=True)  
testing['Class'].replace(['Yes', 'No'], [1, 0], inplace=True)  
  
# Replace '?' values from the mean value of the respective columns (training dataset)  
  
empty\_not\_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT', 'TP', 'ALB', 'AG\_Ratio', 'Class']  
  
accepted = empty\_not\_accepted  
  
for column in accepted:  
 if column not in ('Gender', 'Class'):  
 training[column] = training[column].replace('?', np.NaN)  
 mean = int(training[column].mean(skipna=True))  
 training[column] = training[column].replace(np.NaN, mean)  
  
 else:  
 training[column] = training[column].replace('?', np.NaN)  
 training = training.dropna(how='any', axis=0)  
  
# Replace '?' values from the mean value of the respective columns (testing dataset)  
  
empty\_not\_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT', 'TP', 'ALB', 'AG\_Ratio', 'Class']  
  
accepted = empty\_not\_accepted  
  
for column in accepted:  
 if column not in ('Gender', 'Class'):  
 testing[column] = testing[column].replace('?', np.NaN)  
 mean = int(testing[column].mean(skipna=True))  
 testing[column] = testing[column].replace(np.NaN, mean)  
 else:  
 testing[column] = testing[column].replace('?', np.NaN)  
 testing = testing.dropna(how='any', axis=0)  
  
# Split the training dataset into x and y  
  
train\_x = training.iloc[:, 1:11] # [all rows,column 1-11]  
train\_y = training.iloc[:, 11] # [all rows,column 11]  
  
print('First 10 values of train\_x\n', train\_x.head(10), '\n')  
print('First 10 values of train\_y\n', train\_y.head(10), '\n')  
  
# Split the testing dataset into x and y  
  
test\_x = testing.iloc[:, 1:11] # [all rows,column 1-11]  
test\_y = testing.iloc[:, 11] # [all rows,column 11]  
  
print('First 10 values of test\_x\n', test\_x.head(10), '\n')  
print('First 10 values of test\_y\n', test\_y.head(10), '\n')  
  
# Feature Scaling  
  
sc\_X = StandardScaler()  
train\_x = sc\_X.fit\_transform(train\_x)  
test\_x = sc\_X.transform(test\_x)  
  
print('Standardized Training X data\n', train\_x, '\n')  
  
# hyperparameter tuning  
  
from sklearn.model\_selection import GridSearchCV  
  
# List of Hyperparameters that we want to tune  
  
leaf\_size = list(range(1, 30))  
n\_neighbors = list(range(1, 30))  
p = [1, 2]  
  
# get hyperparameters into a dictionary  
  
hyperparameters = dict(leaf\_size=leaf\_size, n\_neighbors=n\_neighbors, p=p)  
  
# Define the model  
knn = KNeighborsClassifier()  
  
# Use GridSearch  
clf = GridSearchCV(knn, hyperparameters, cv=10) # cv = class validation  
  
# here cv = 10 means we have to divide the dataset into 5 sets/folds  
  
# Fit the model  
  
best\_model = clf.fit(train\_x, train\_y)  
  
print("Estimated best hyperparameters \n")  
  
print('Best leaf\_size:', best\_model.best\_estimator\_.get\_params()['leaf\_size'])  
print('Best p:', best\_model.best\_estimator\_.get\_params()['p'])  
print('Best n\_neighbors:', best\_model.best\_estimator\_.get\_params()['n\_neighbors'], '\n')  
  
# predict the test set results  
pred\_y = clf.predict(test\_x)  
print('Predicted y values for test data \n', pred\_y, '\n')  
  
# Evaluate model  
cm = confusion\_matrix(test\_y, pred\_y)  
print('Confusion matrix \n', cm, '\n')  
  
# Confusion matrix plot  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=['No', 'Yes'])  
disp.plot()  
plt.show()  
  
tn, fp, fn, tp = cm.ravel()  
  
# tn = true negative  
# tp = true positive  
# fn = false negative  
# fp = false positive  
  
# Manual calculations  
  
# print('Accuracy : ', round(((tp+tn)/(tp+fn+fp+tn))\*100, 2), '%')  
# print('Precision (Positive Predictive Value) : ', round((tp/(tp+fp))\*100, 2), '%')  
# print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))\*100, 2), '%')  
# print('Specificity : ', round((tn/(tn+fp))\*100, 2), '%')  
# print('FN Rate (Miss rate) : ', round((fn/(tp+fn))\*100, 2), '%')  
# print('TP Rate (False Alarm Rate) : ', round((fp/(fp+tn))\*100, 2), '%')  
# print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))\*100, 2), '%')  
  
# same calculations using sklearn  
  
print('Accuracy : ', round((accuracy\_score(test\_y, pred\_y)\*100), 2), '%')  
print('Precision (Positive Predictive Value) : ', round((precision\_score(test\_y, pred\_y)\*100), 2), '%')  
print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))\*100, 2), '%')  
print('Specificity : ', round((recall\_score(test\_y, pred\_y)\*100), 2), '%')  
print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))\*100, 2), '%')

Output

