**CS 634 – Data Mining**

**Project Report**

**Topic – Supervised Data Mining (Classification)**

**Name – Tanish Bugnait**

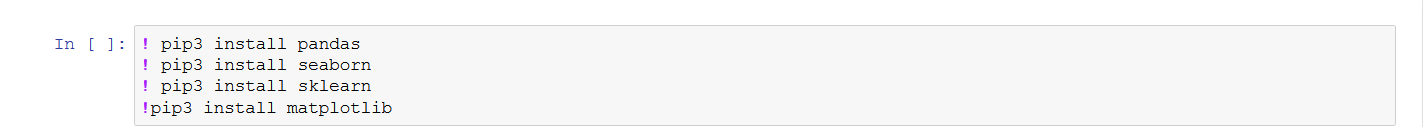
**NJIT ID – 31518500**

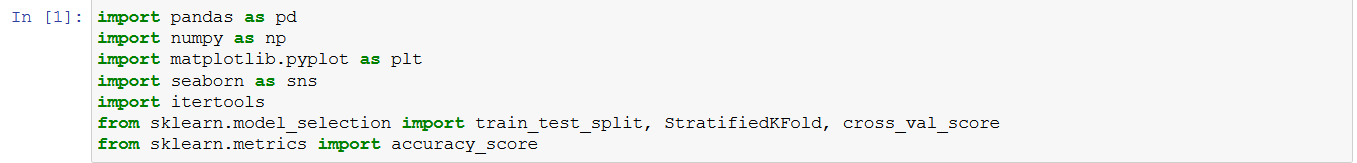
The following project demonstrates Supervised Data Mining (Classification). I had selected option 1 and implemented the following 3 evaluating classifiers:

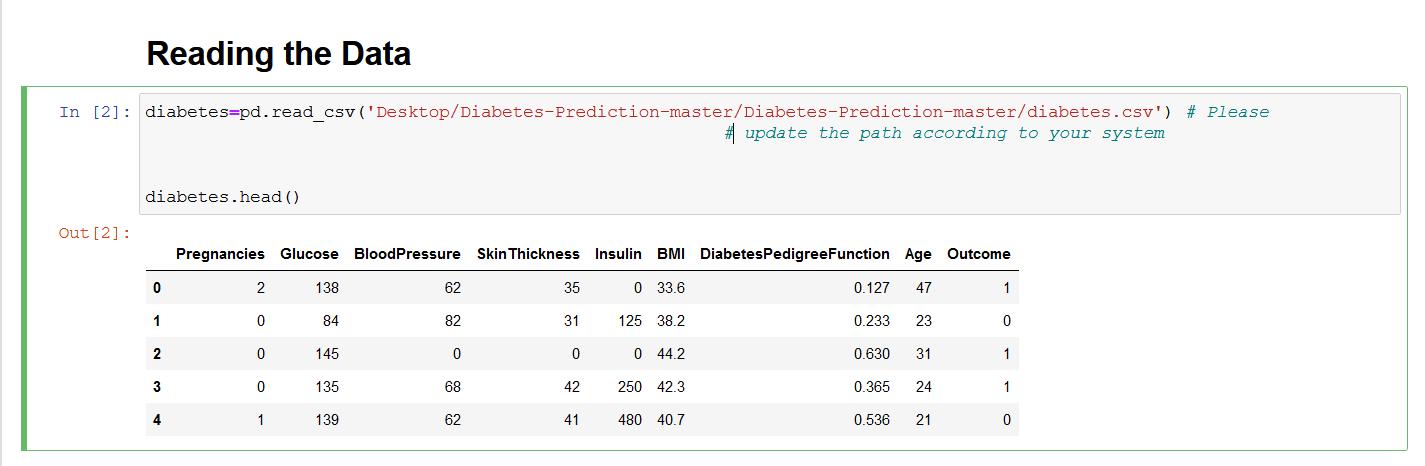
* Random Forest
* Naïve Bayes
* KNN (K- Nearest Neighbors)

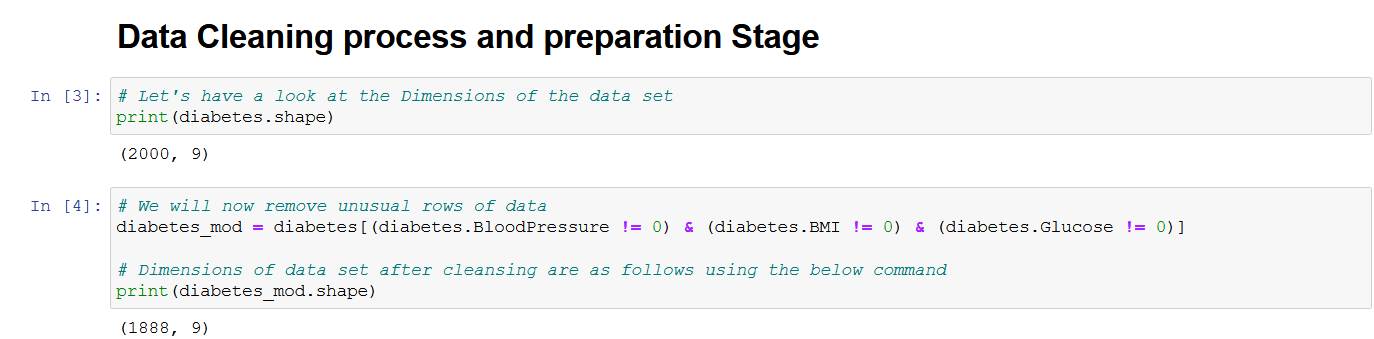
I have calculated all possible metrics that were taught in the class using the formulas given in the PPT. The Dataset that I have chosen is for Diabetes prediction and the code showcases a step-by-step implementation for all the parameters that was asked to be performed.

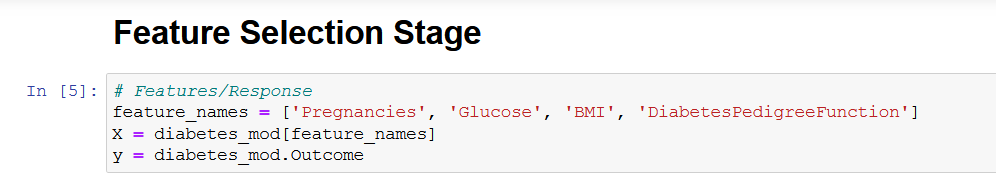
**Implementation of the code**

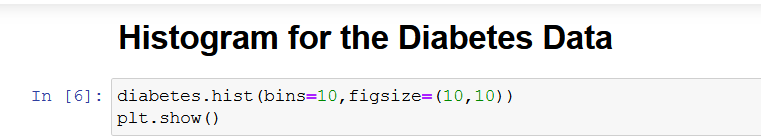


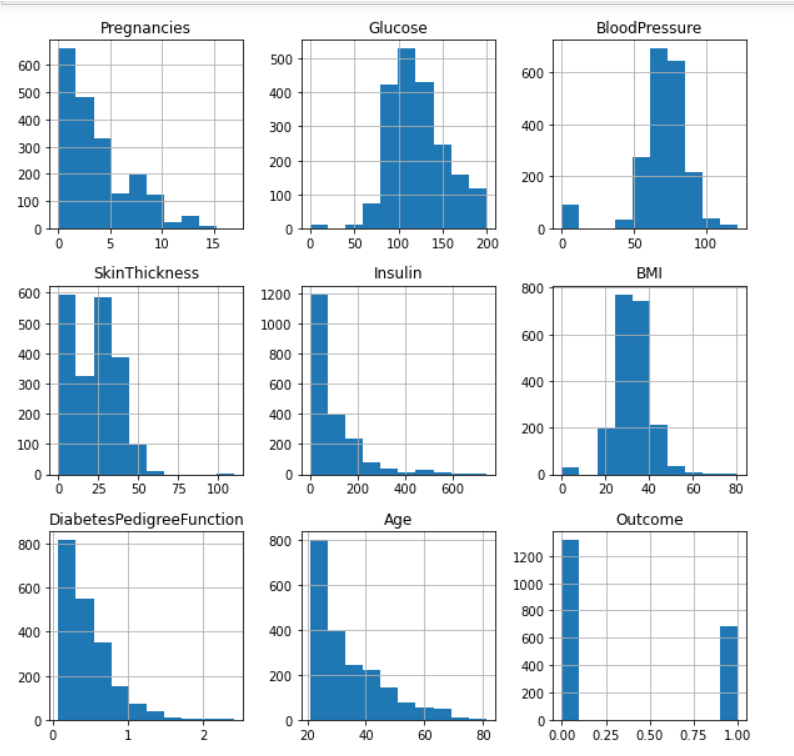


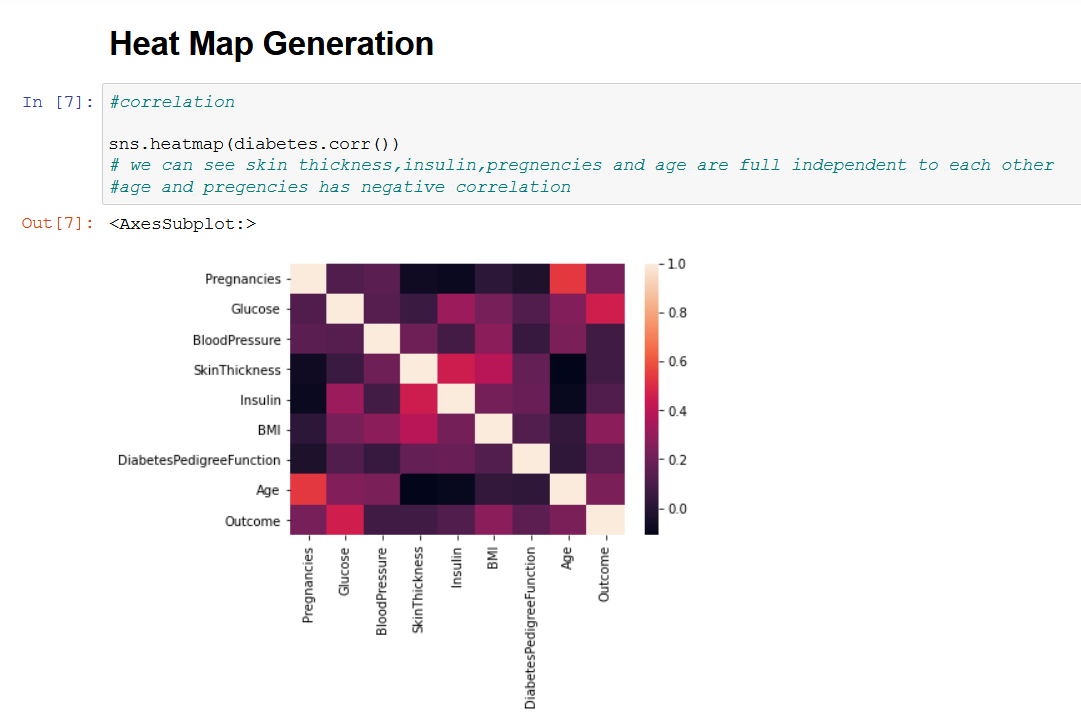


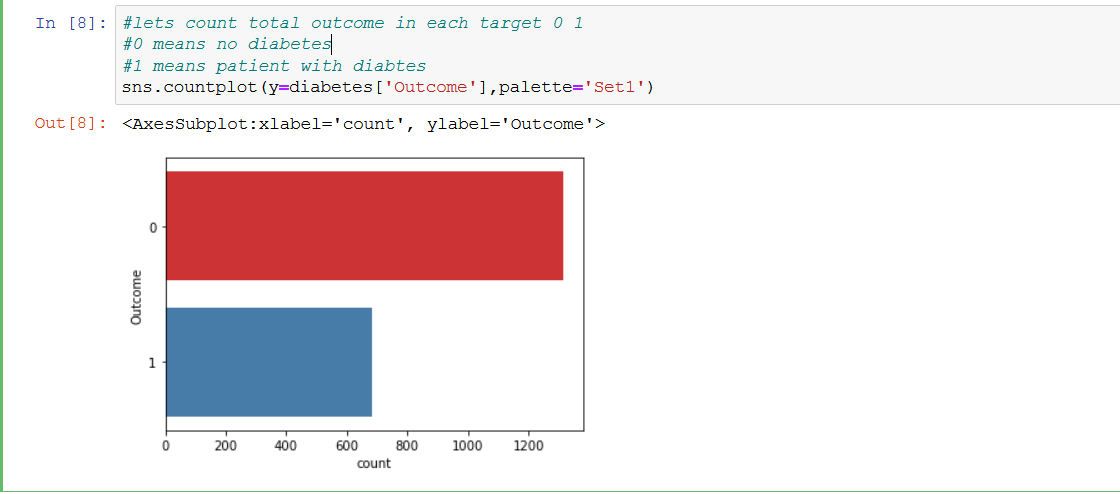


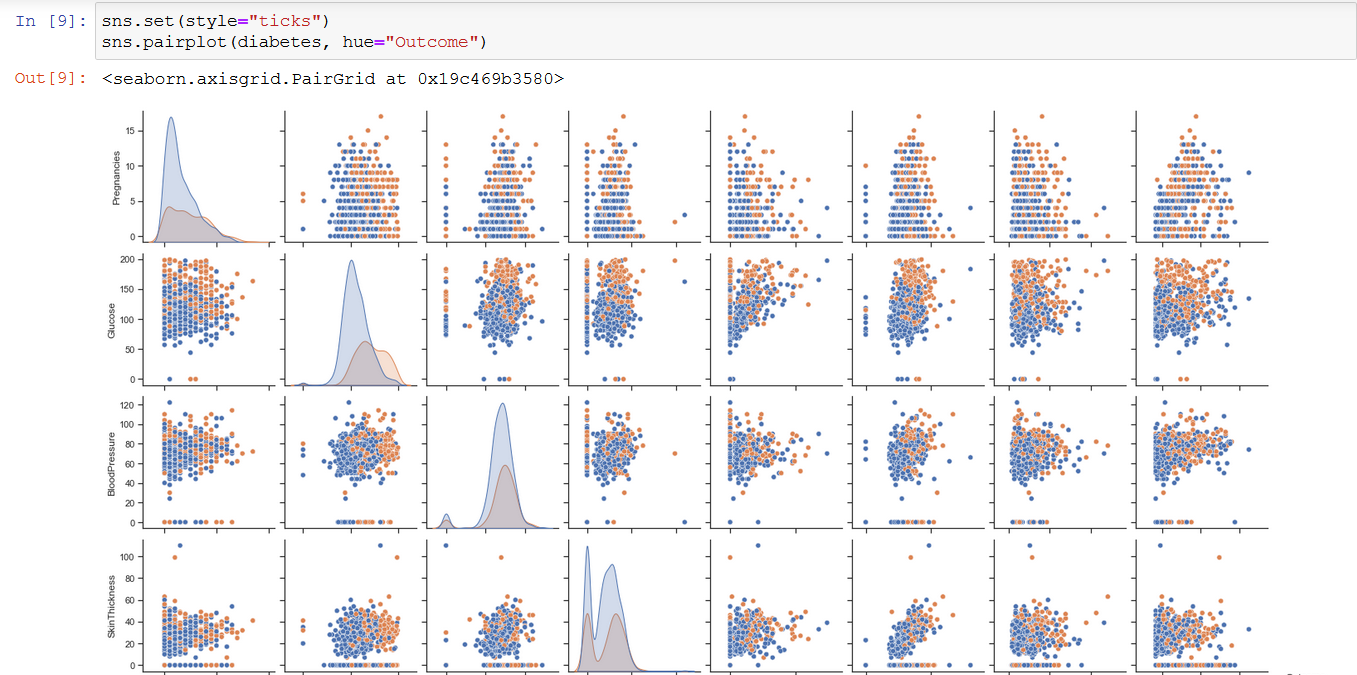


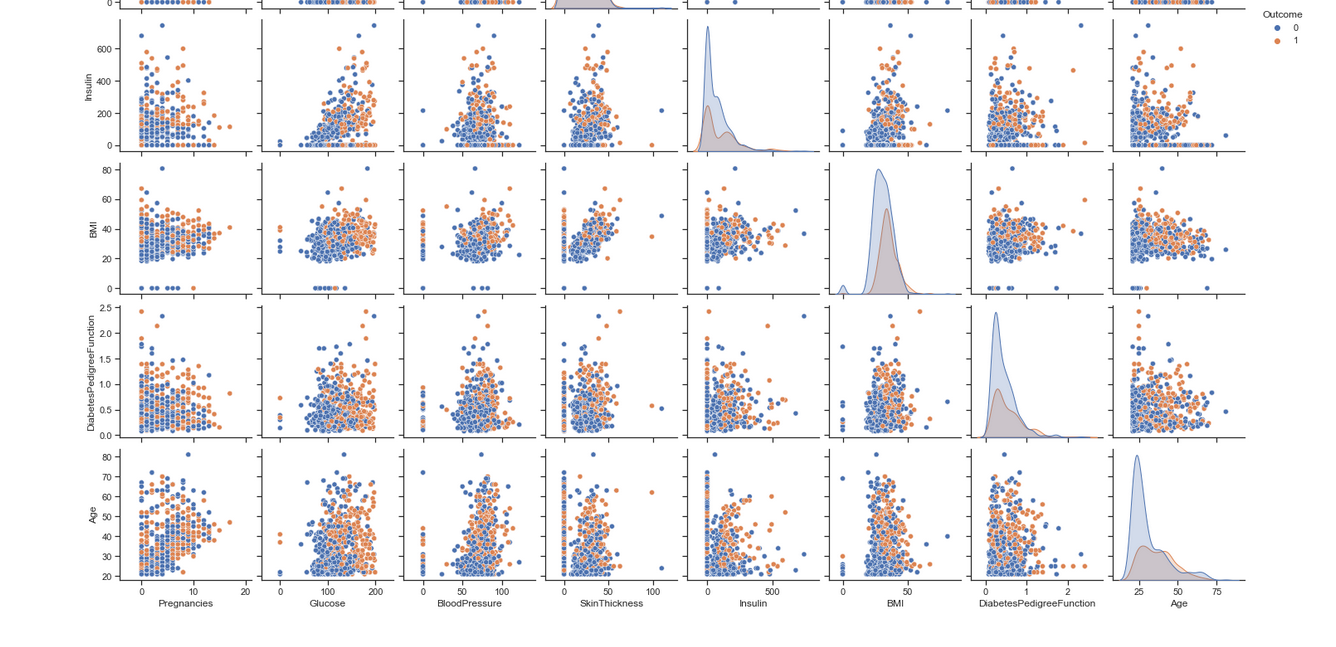


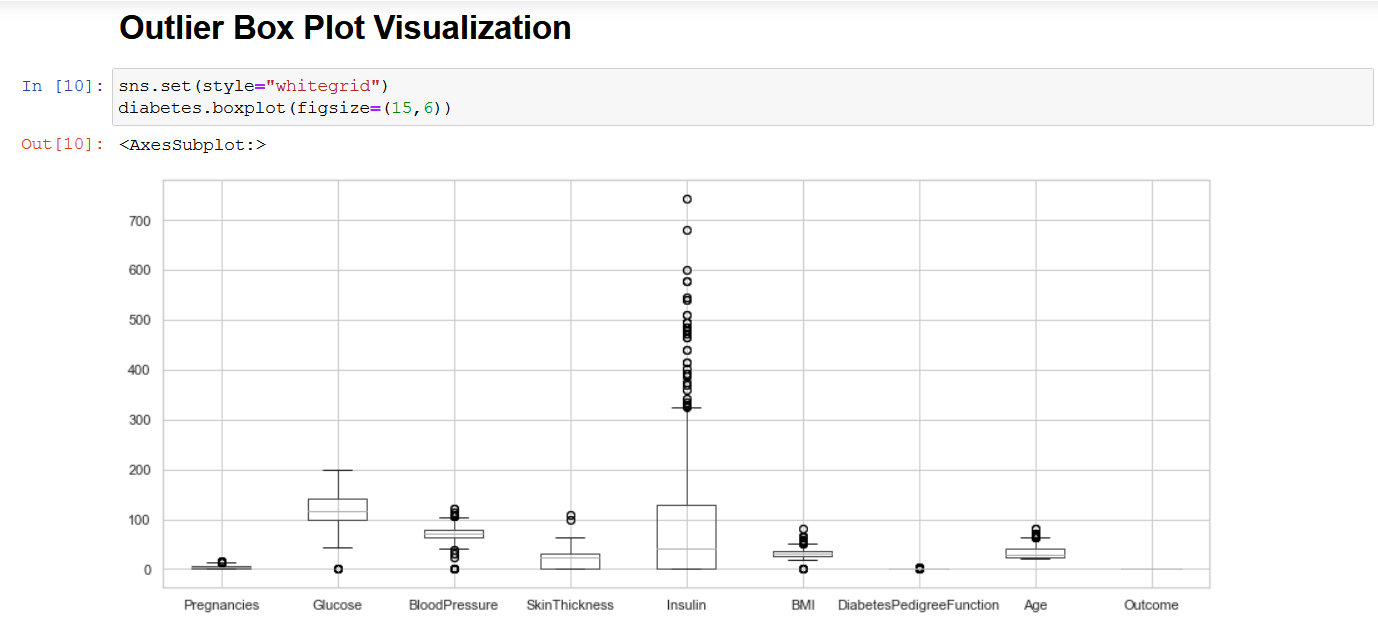






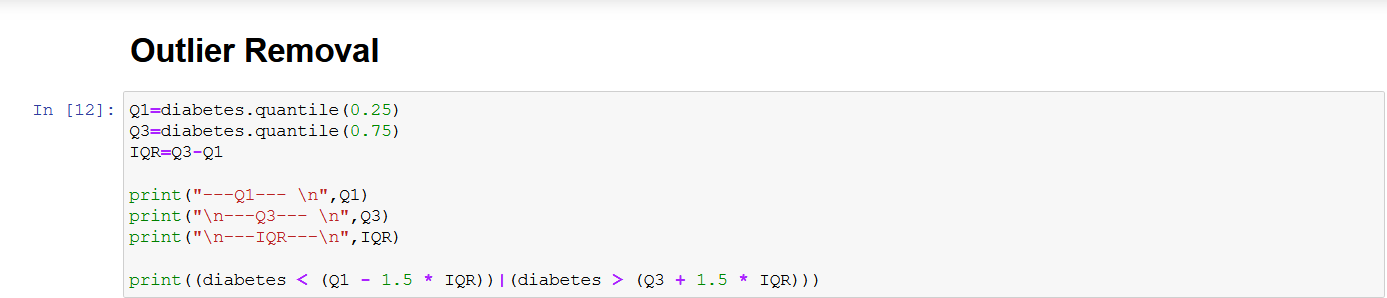


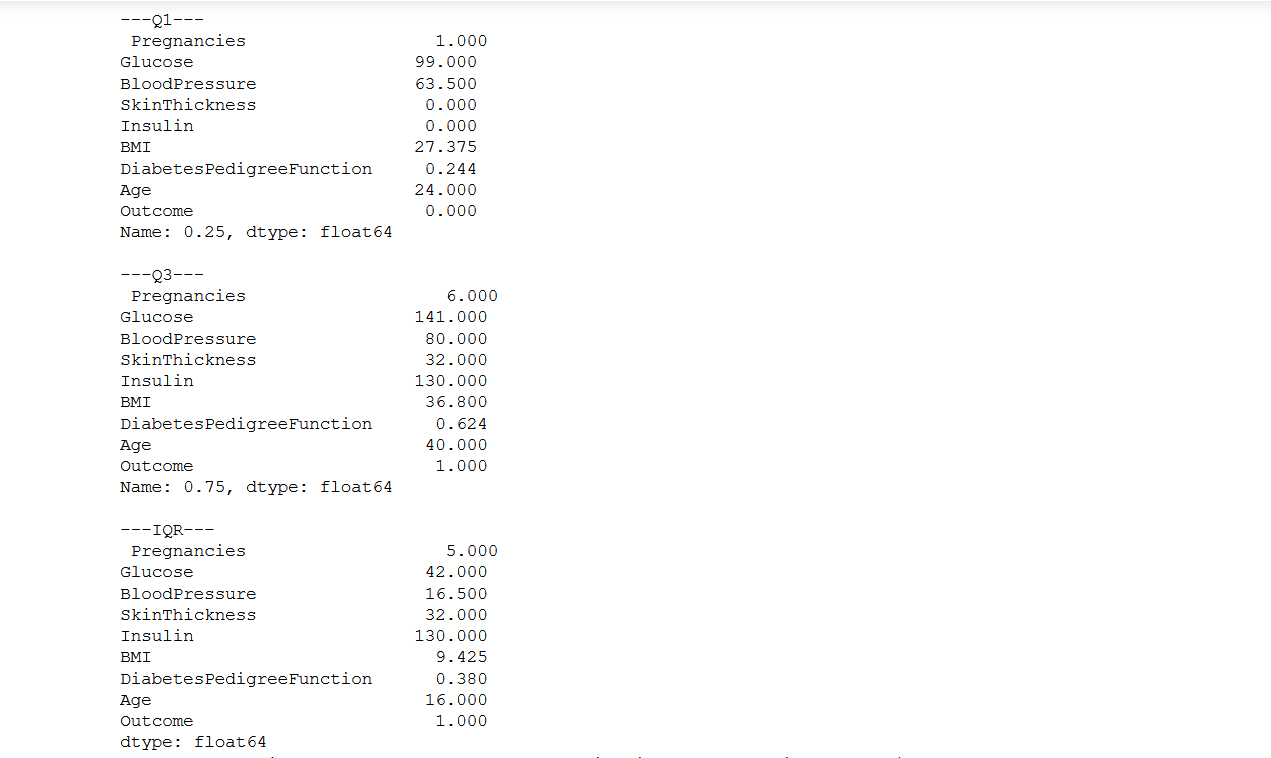


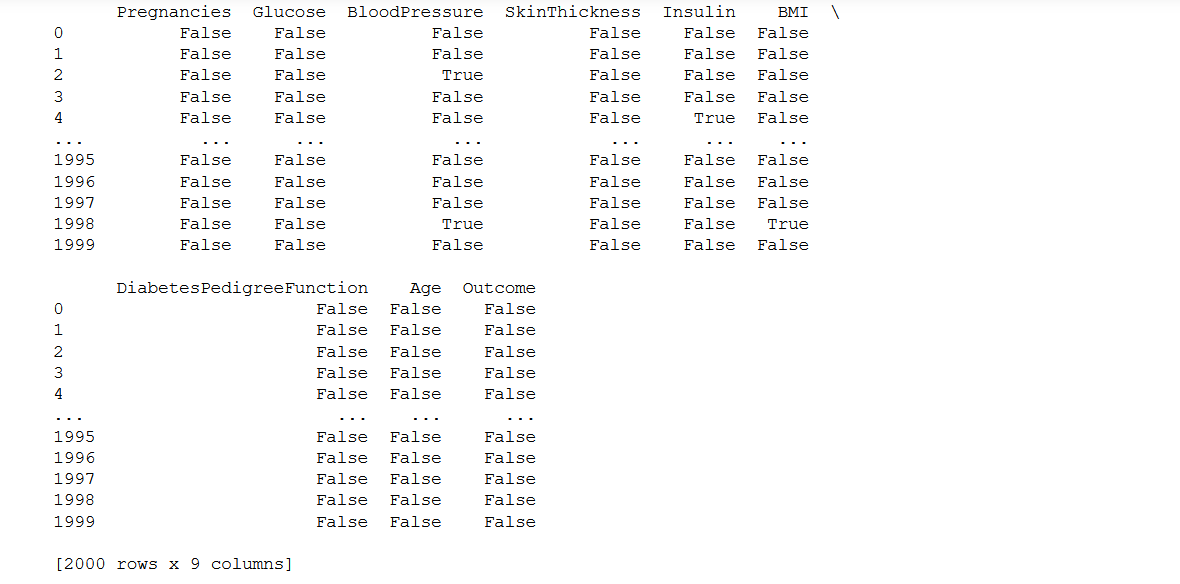


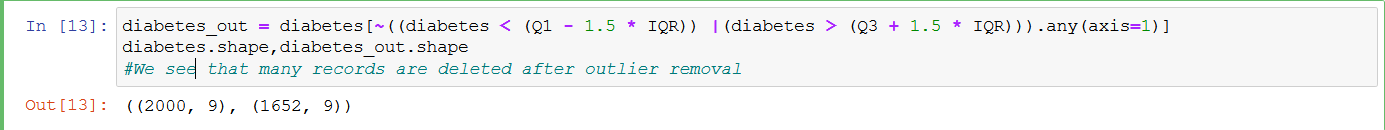




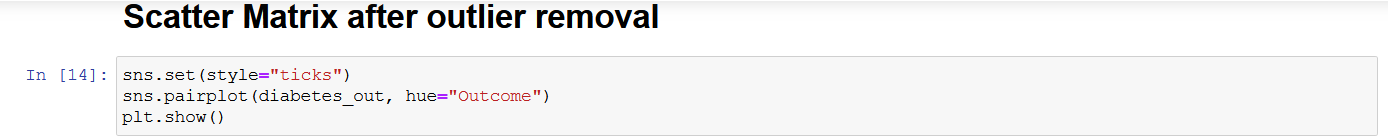


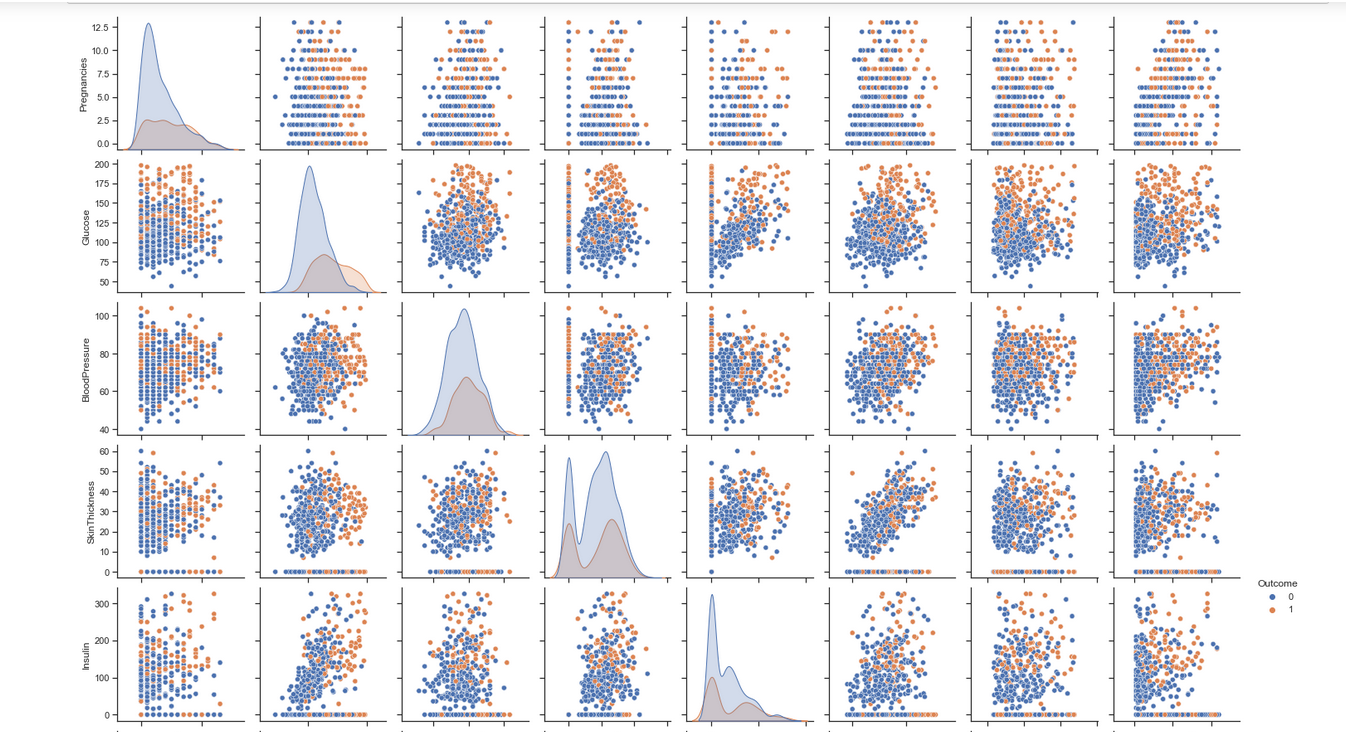


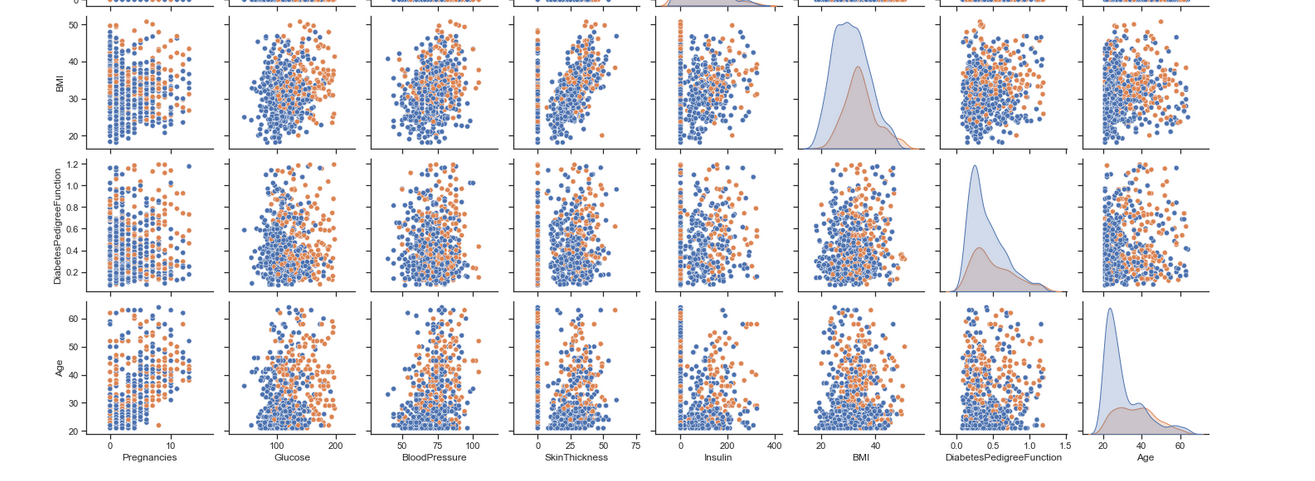




Now that the Outliers have been removed, we will have a look at the newly generated Scatter Matrix.



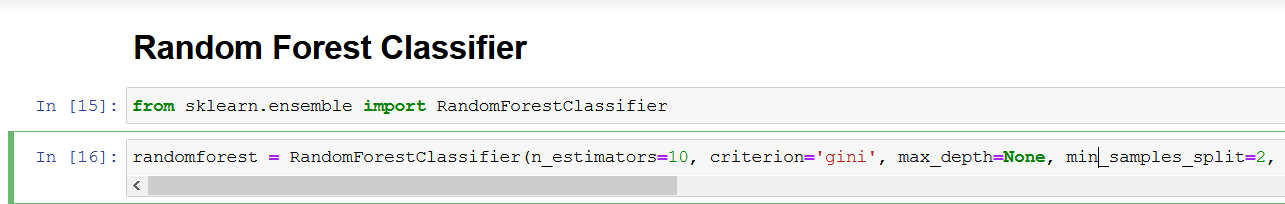


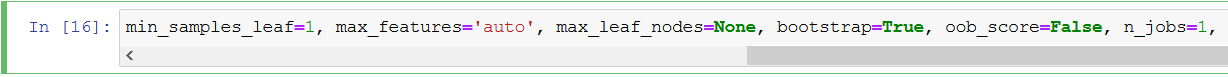


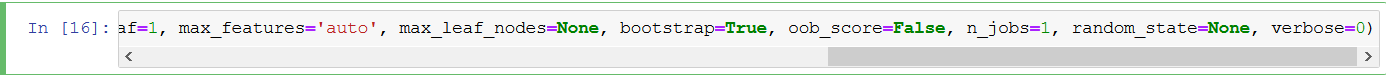
Now, it’s time to implement the 3 different classifiers and calculate different metrics as mentioned in the PPT.

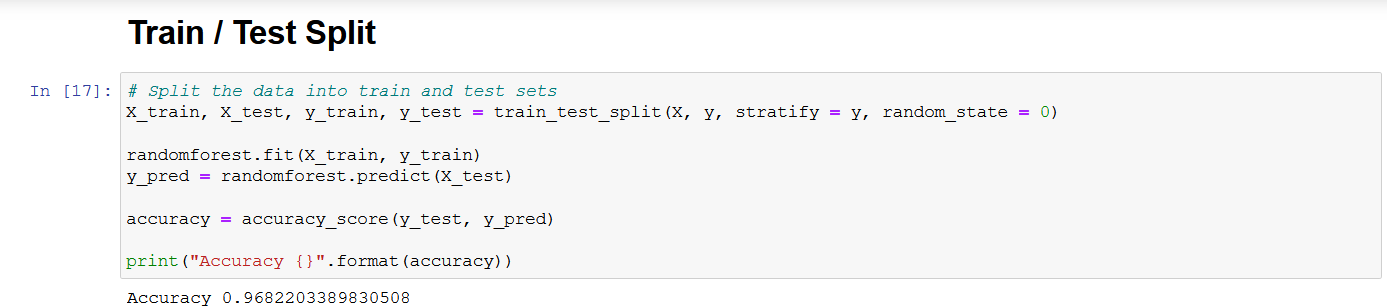
We will first have a look at the Random Forest Classifier.

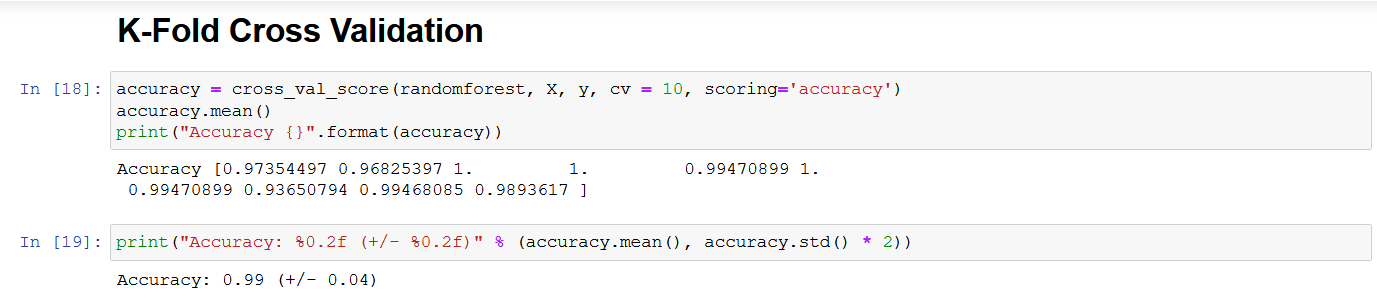
1. **Random Forest Classifier**



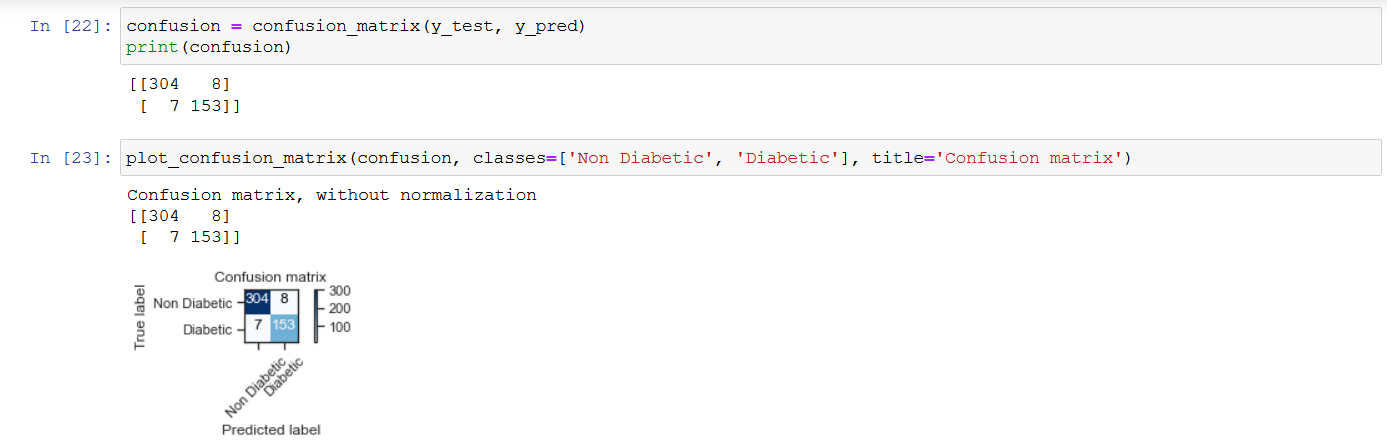


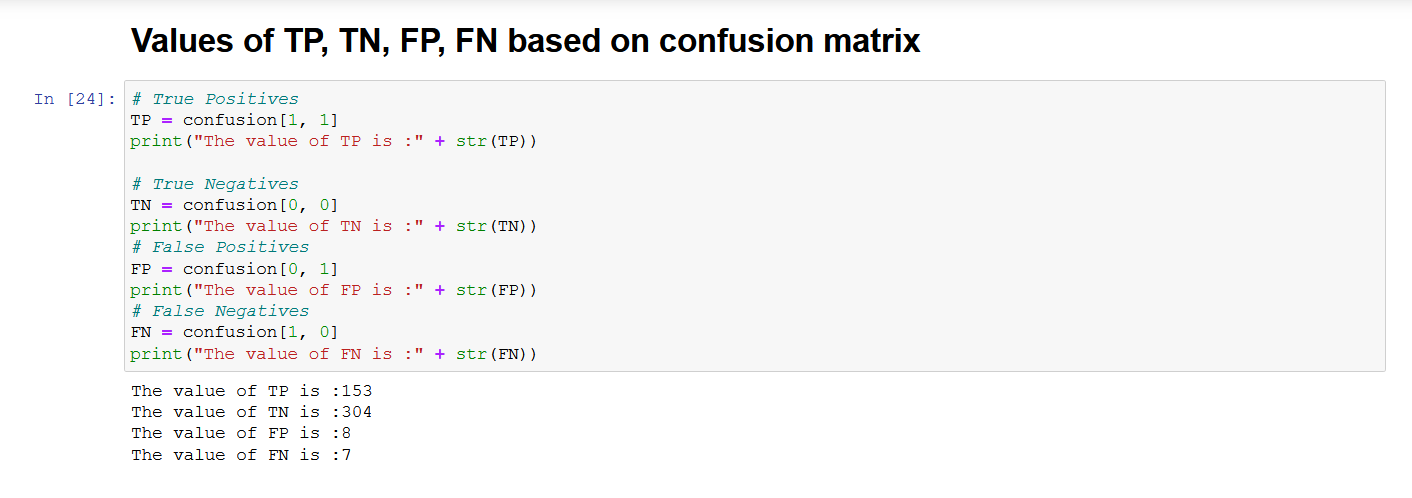


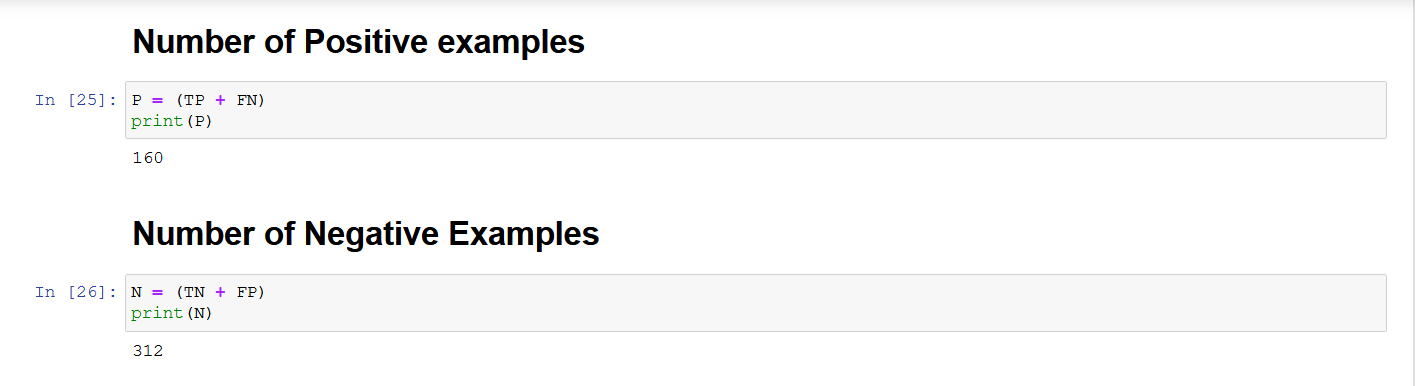






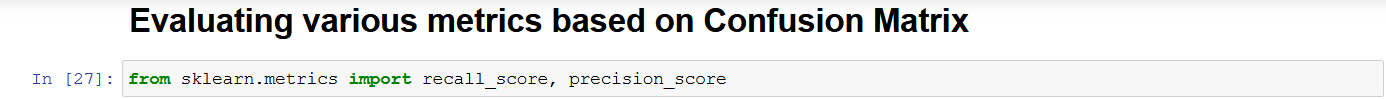




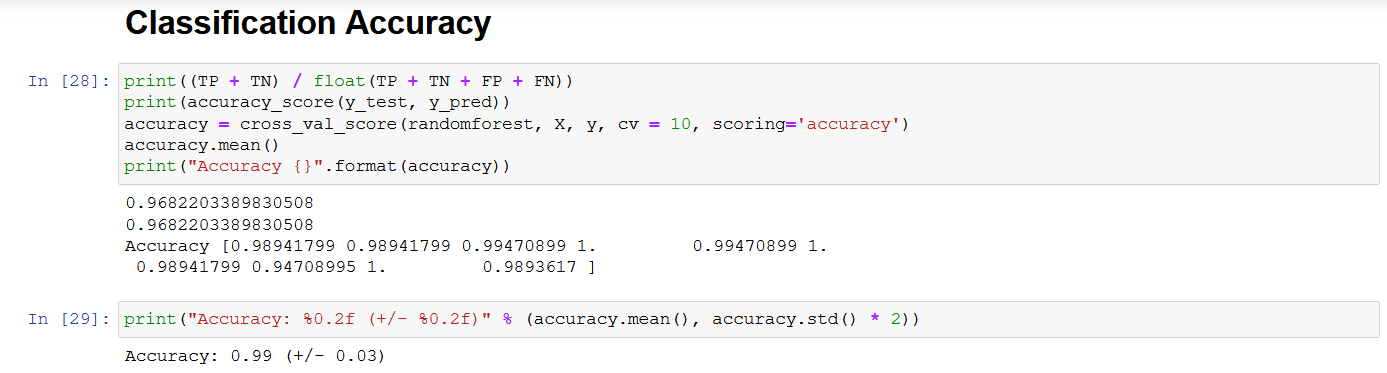


Up until now I have imported the necessary libraries required to run my Random Forest Classifier with the help of scikit learn and also made the confusion matrix.

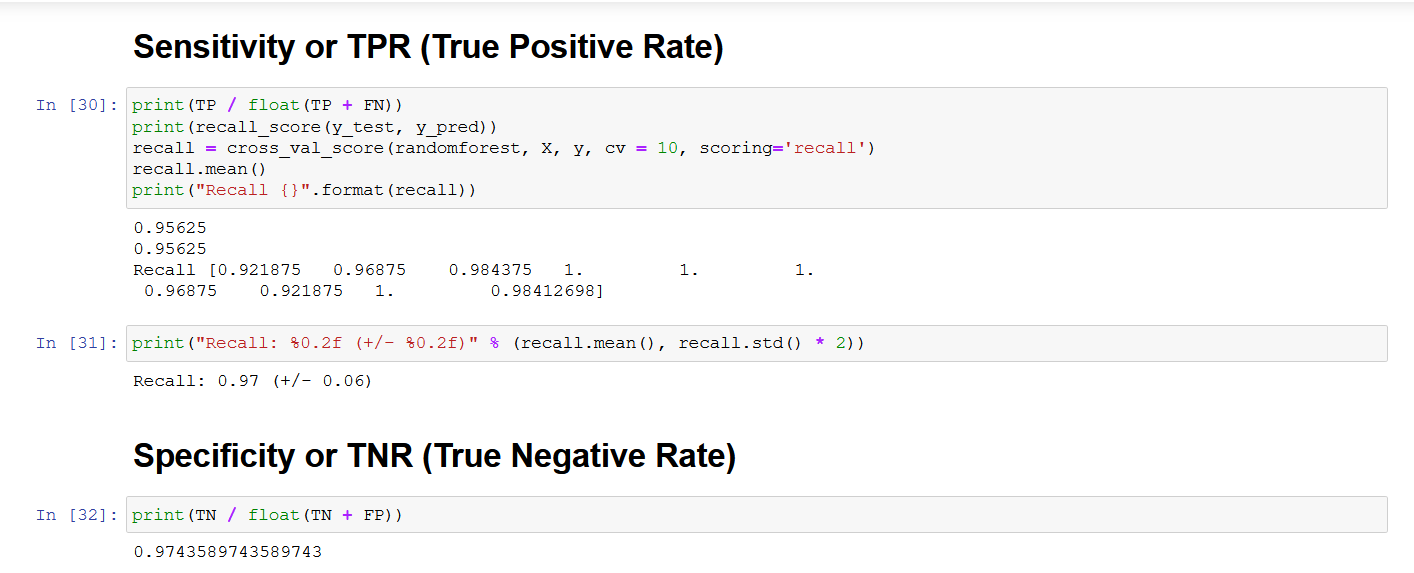
Now, it’s time to calculate the various metrics values by using the given formula.



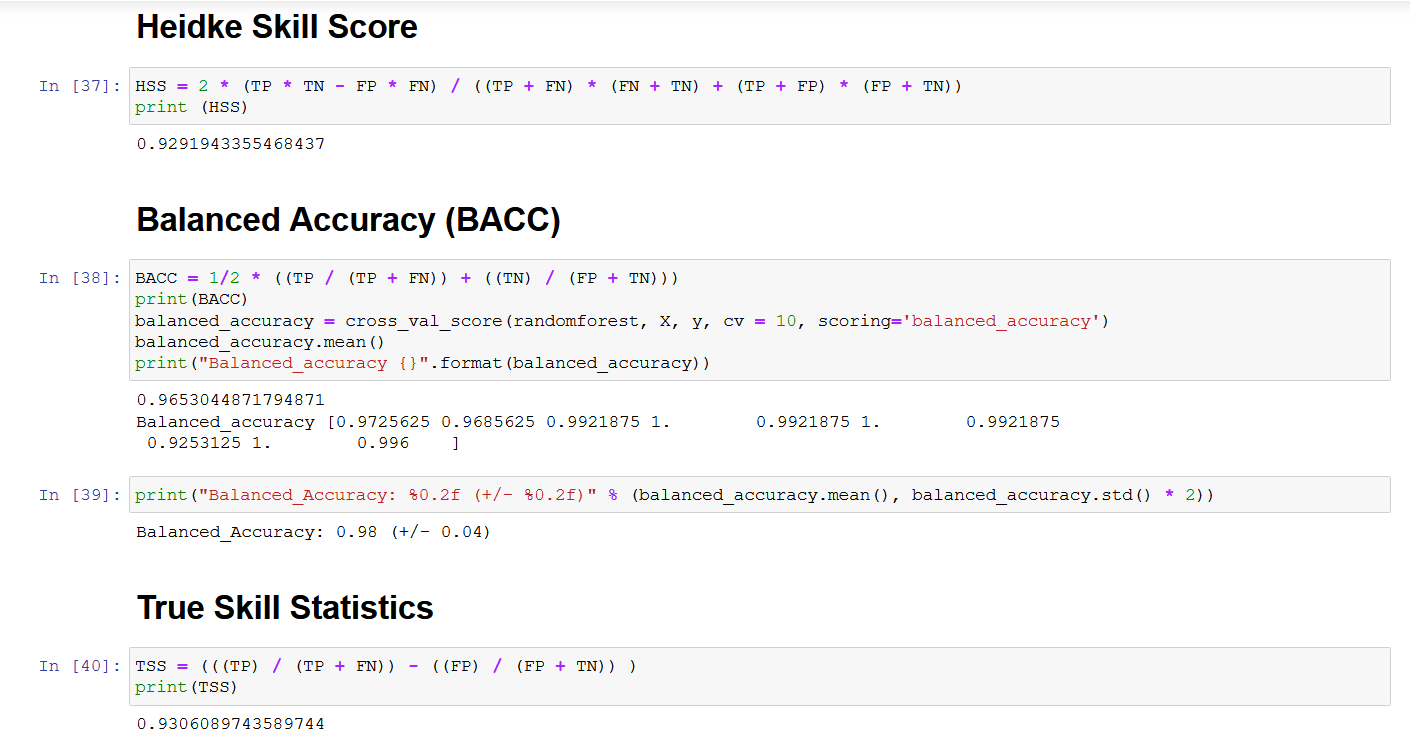
At the end of the classifier, I have provided a tabular format showcasing the different values for 10 Folds and also the average for each metric.



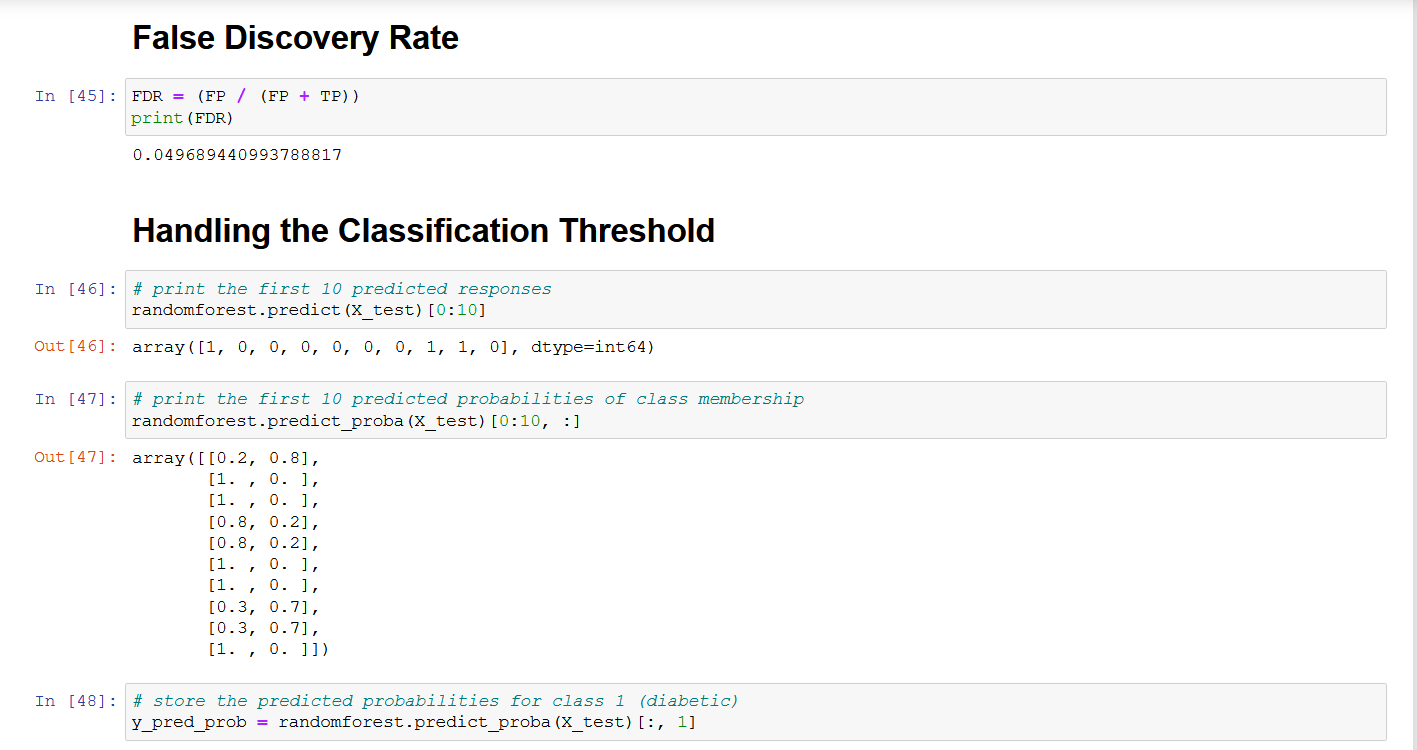
We see 10 different values of Accuracy for 10 different folds along with the mean for the same. The tabular format provided at the end of this classifier would give a clear understanding of the output.

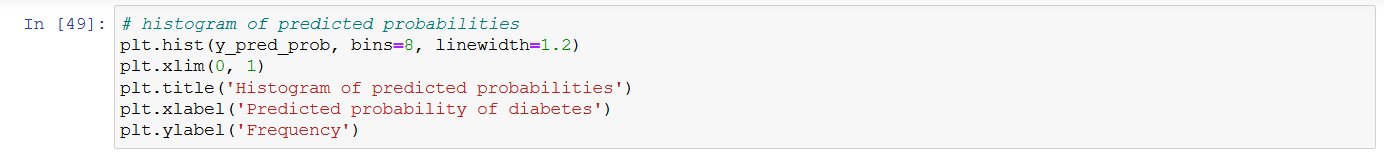


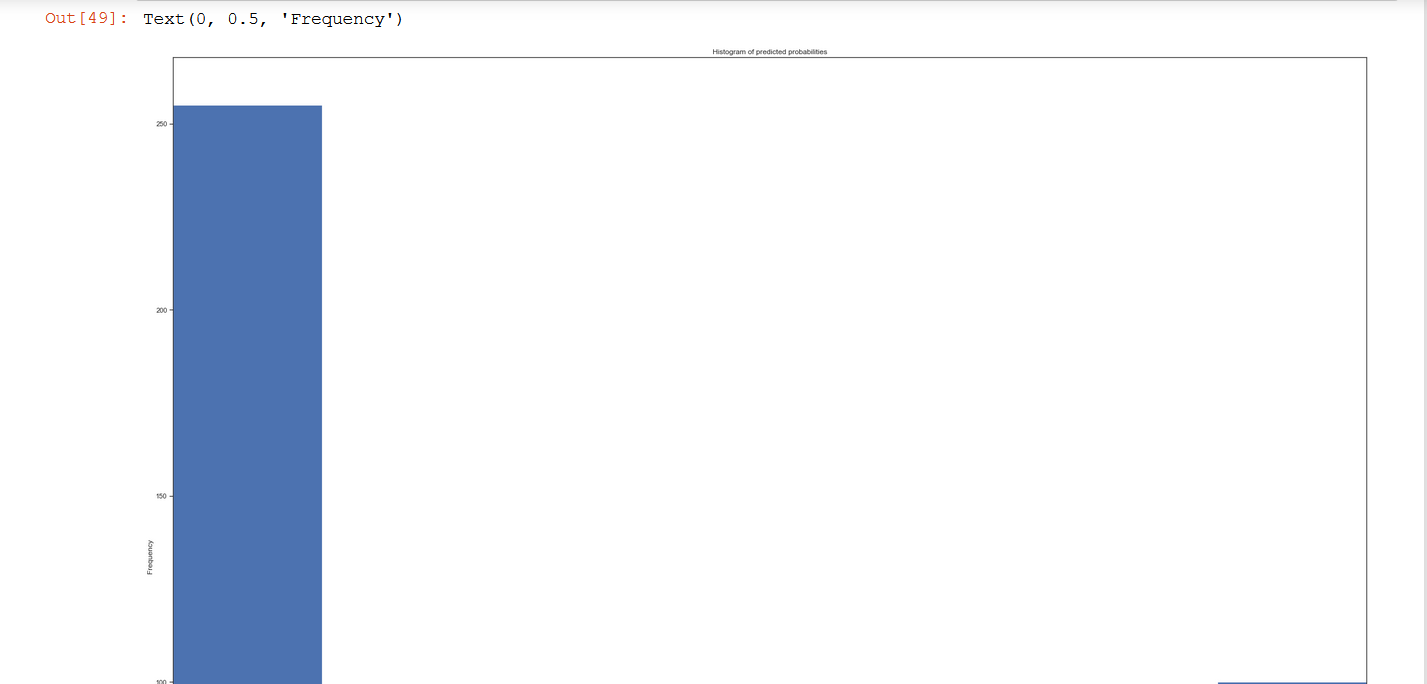


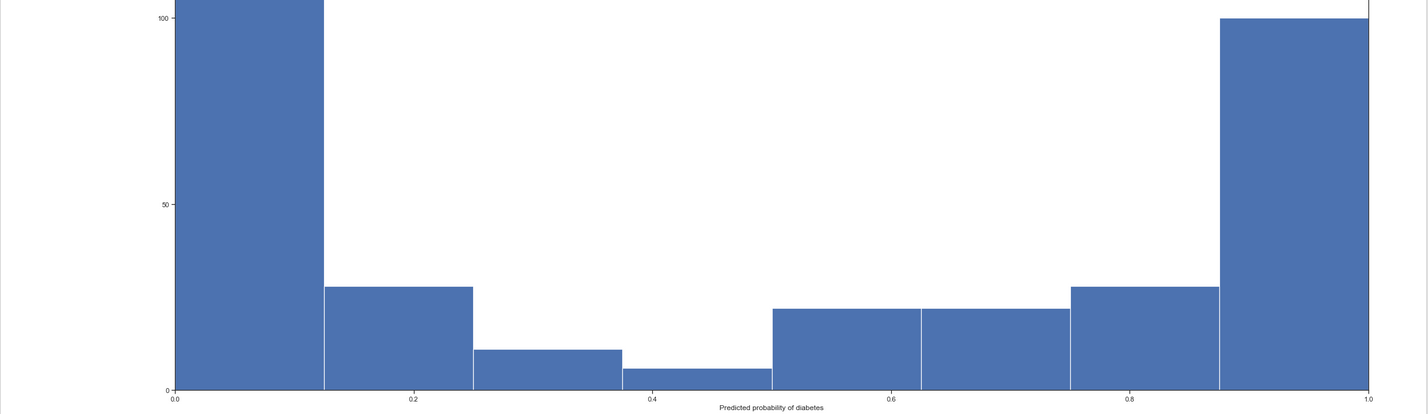


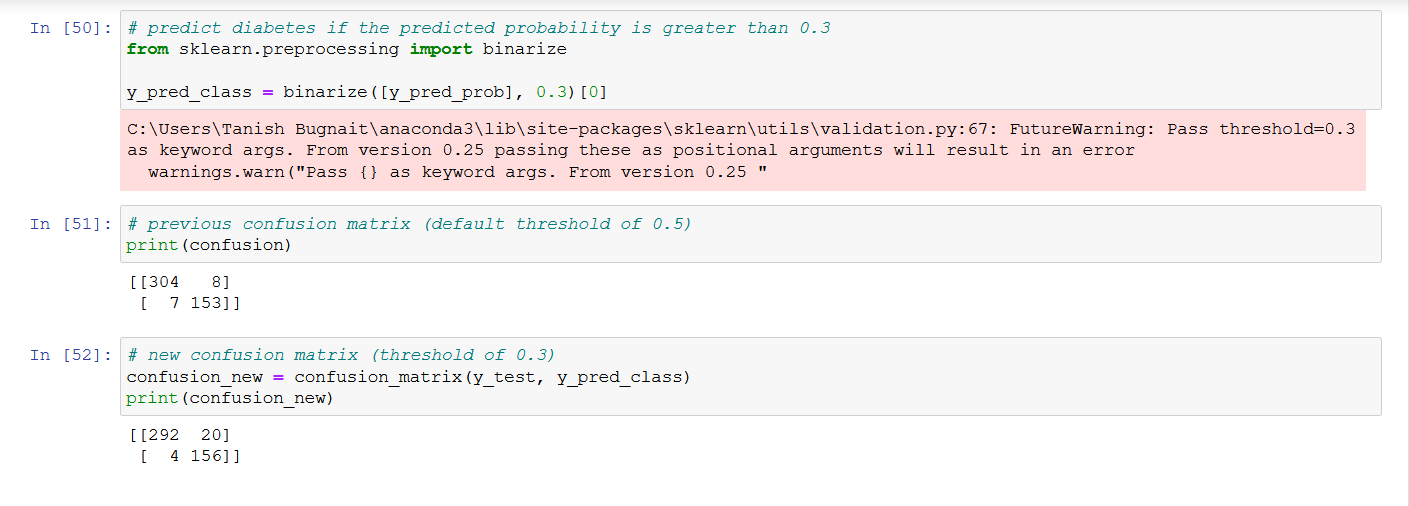




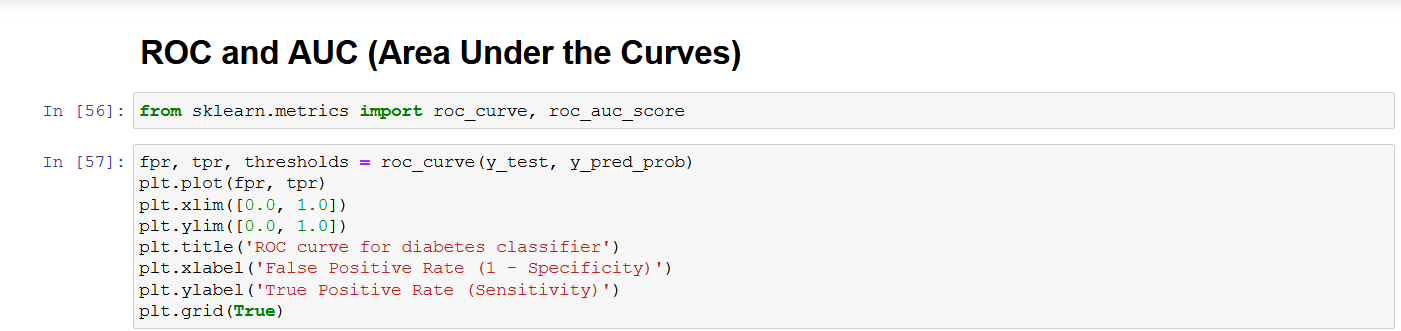




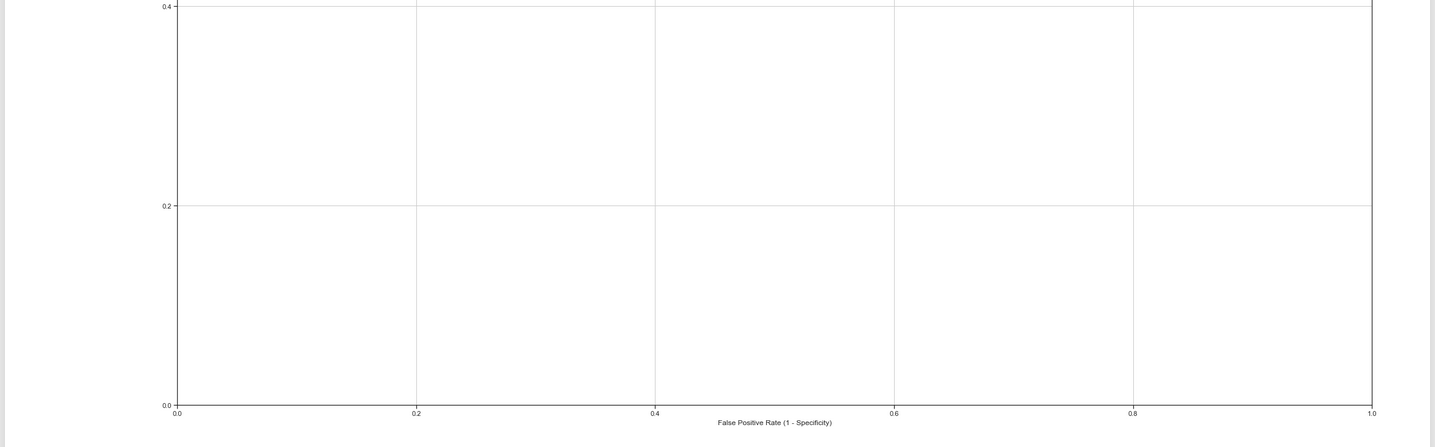




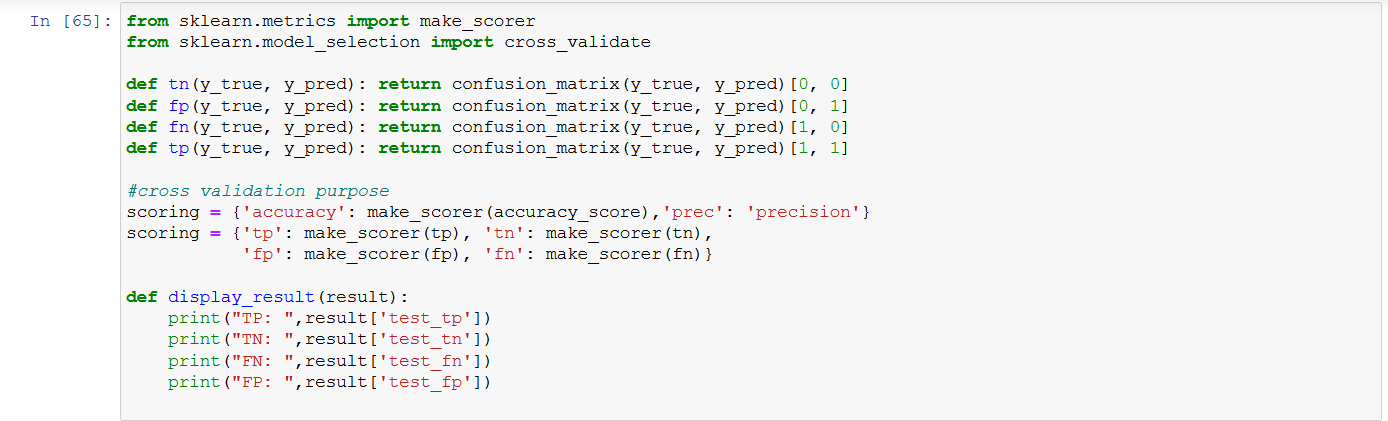


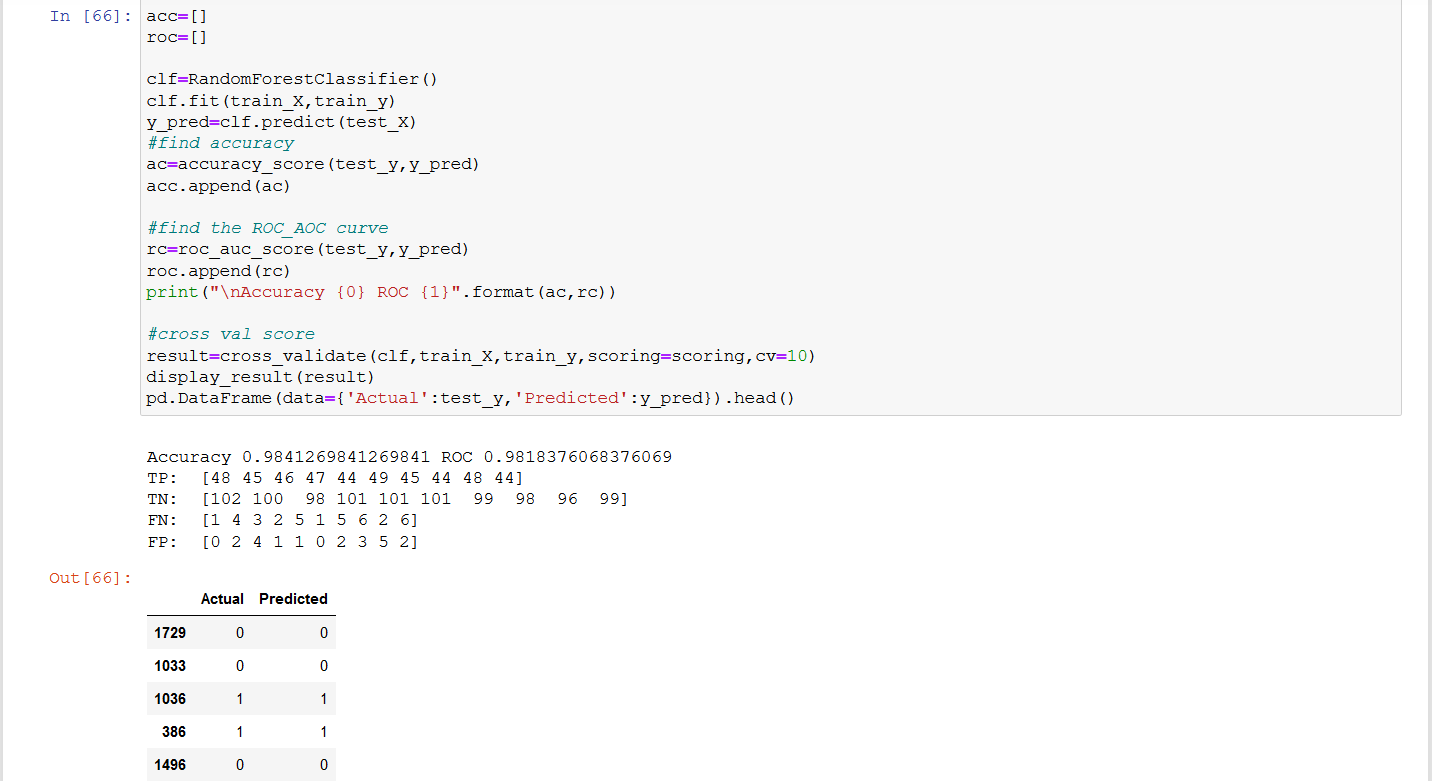




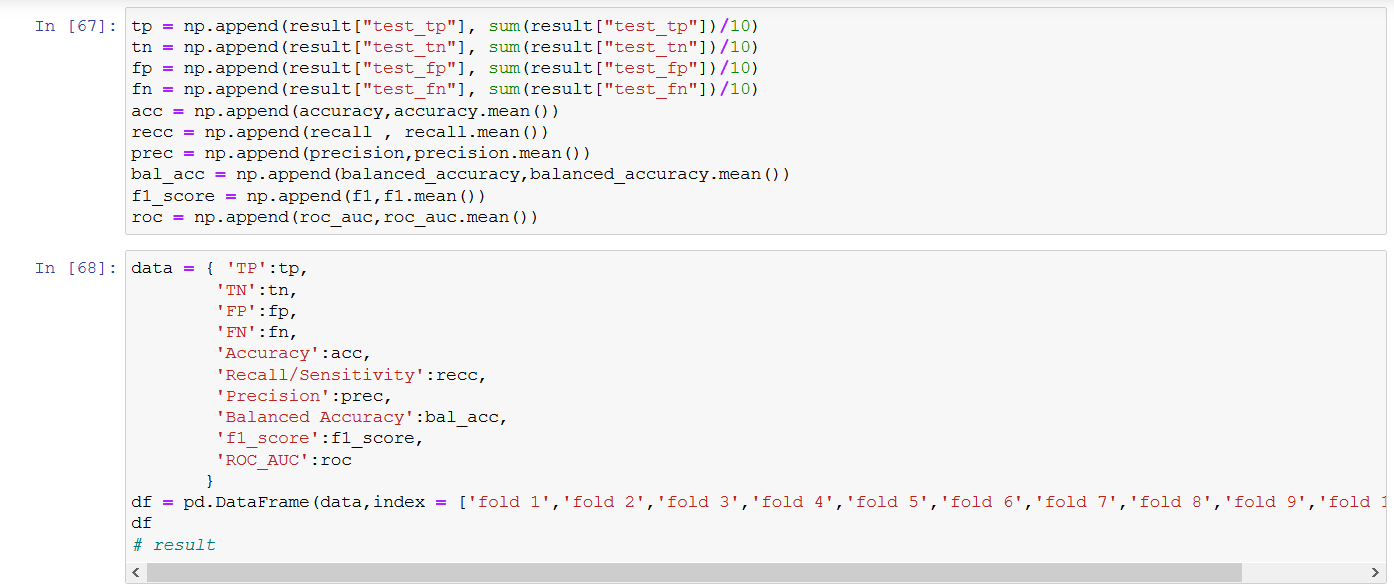


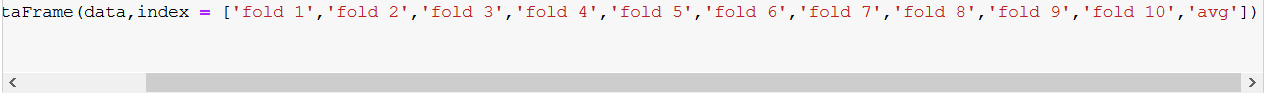


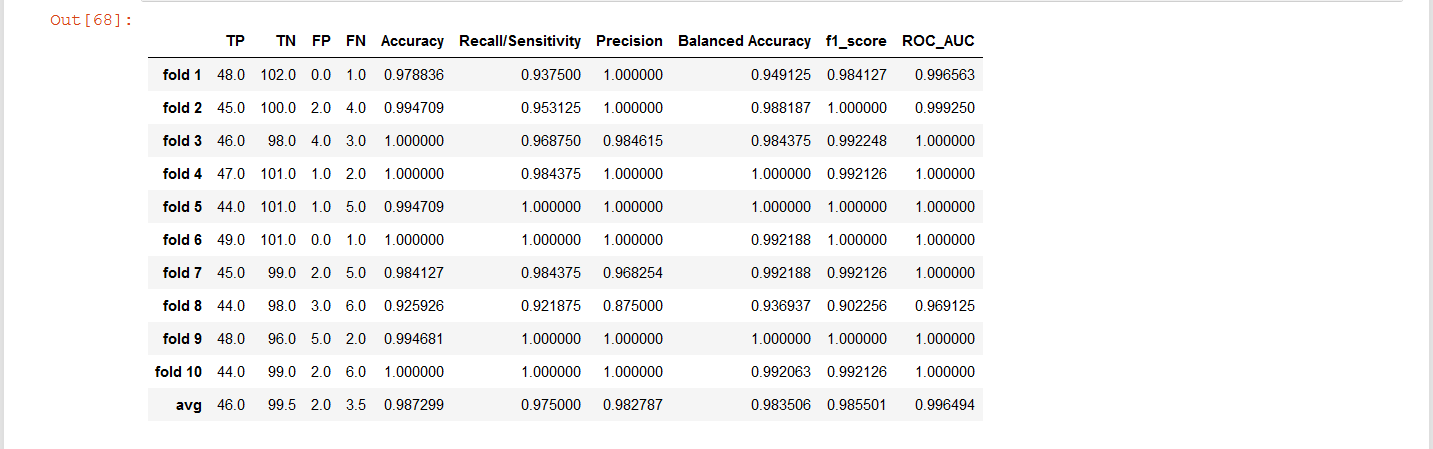




Now, we have a look at the various values in a tabular format to give a better clarity.



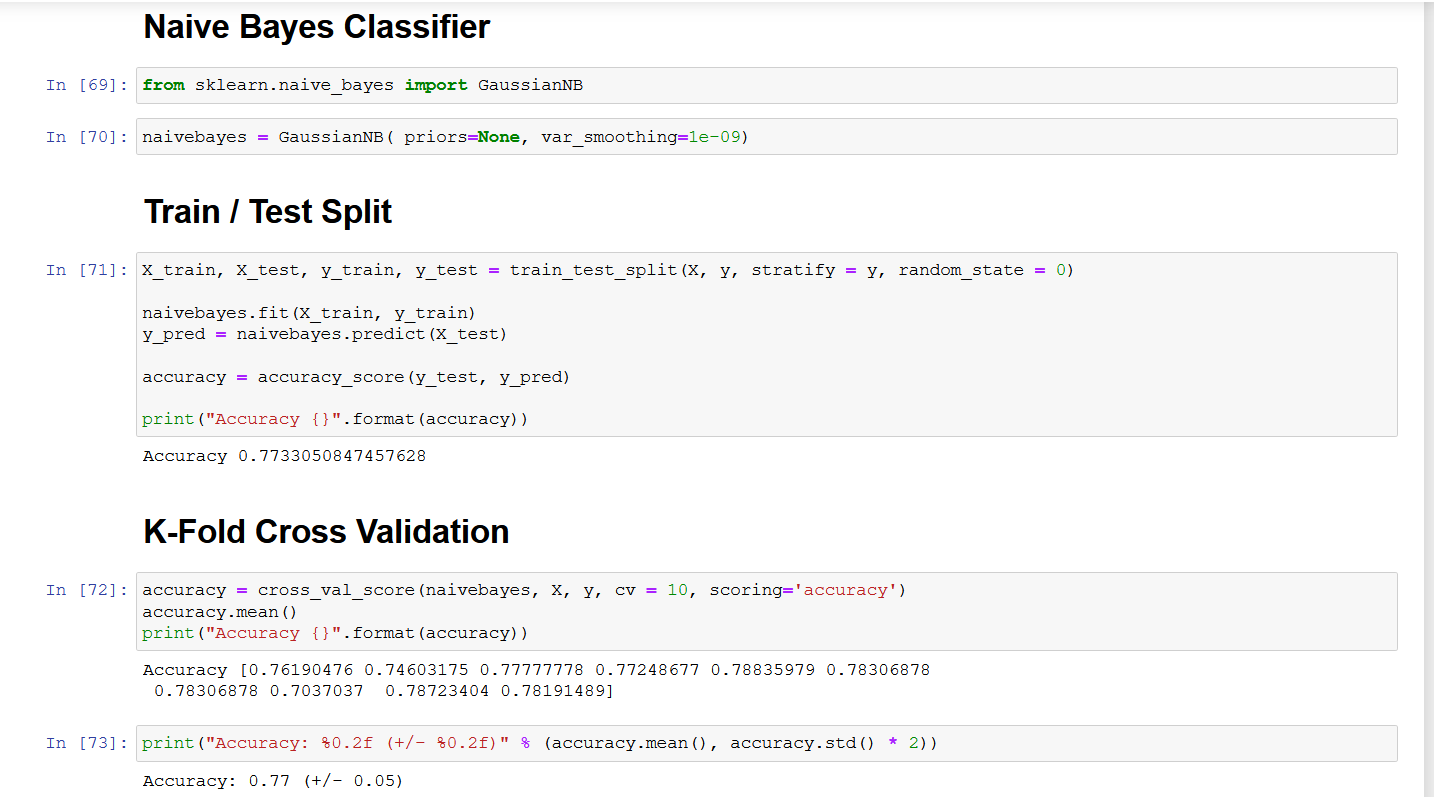




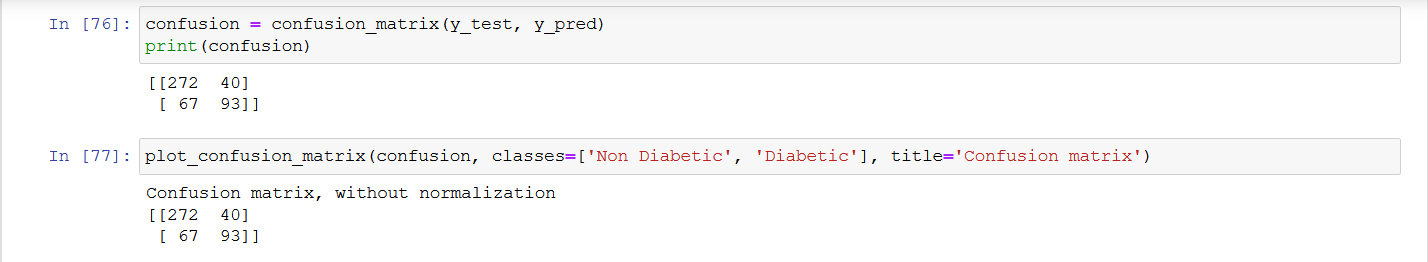
From the above screenshots we can see that I have calculated various metrics values for 10 folds and at the end I have also calculated the average as it was desired for the project.

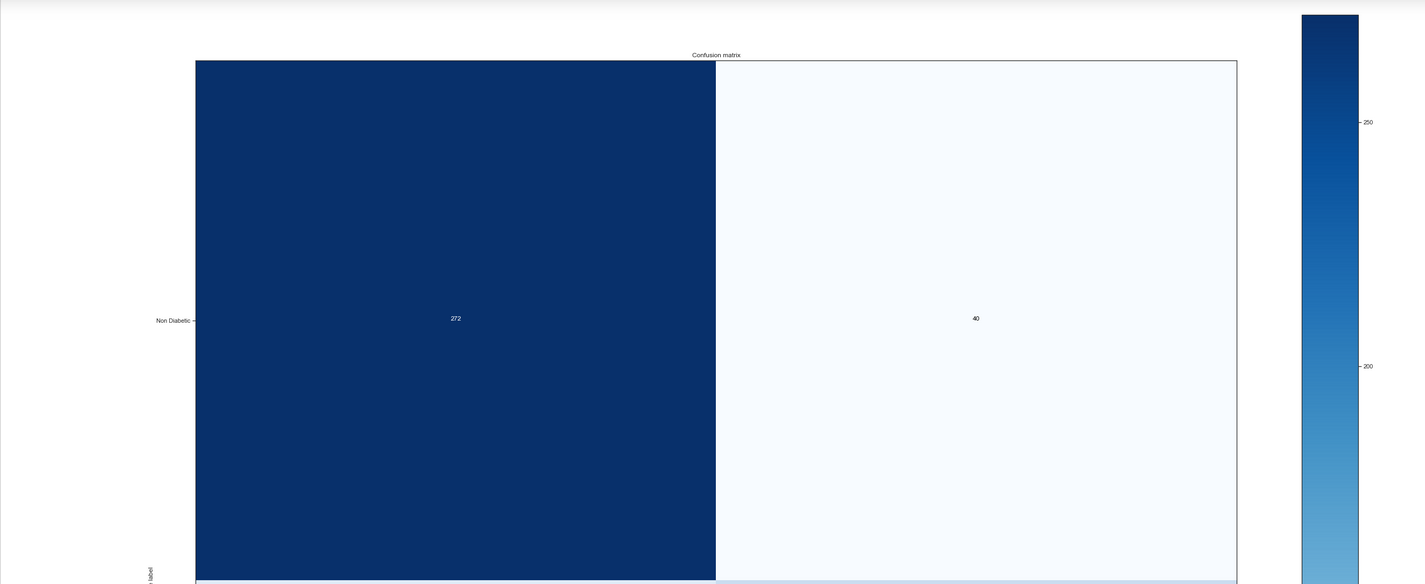
Now, that we have seen Random Forest classifier we will shift our focus to Naïve Bayes Classifier.

1. **Naïve Bayes**

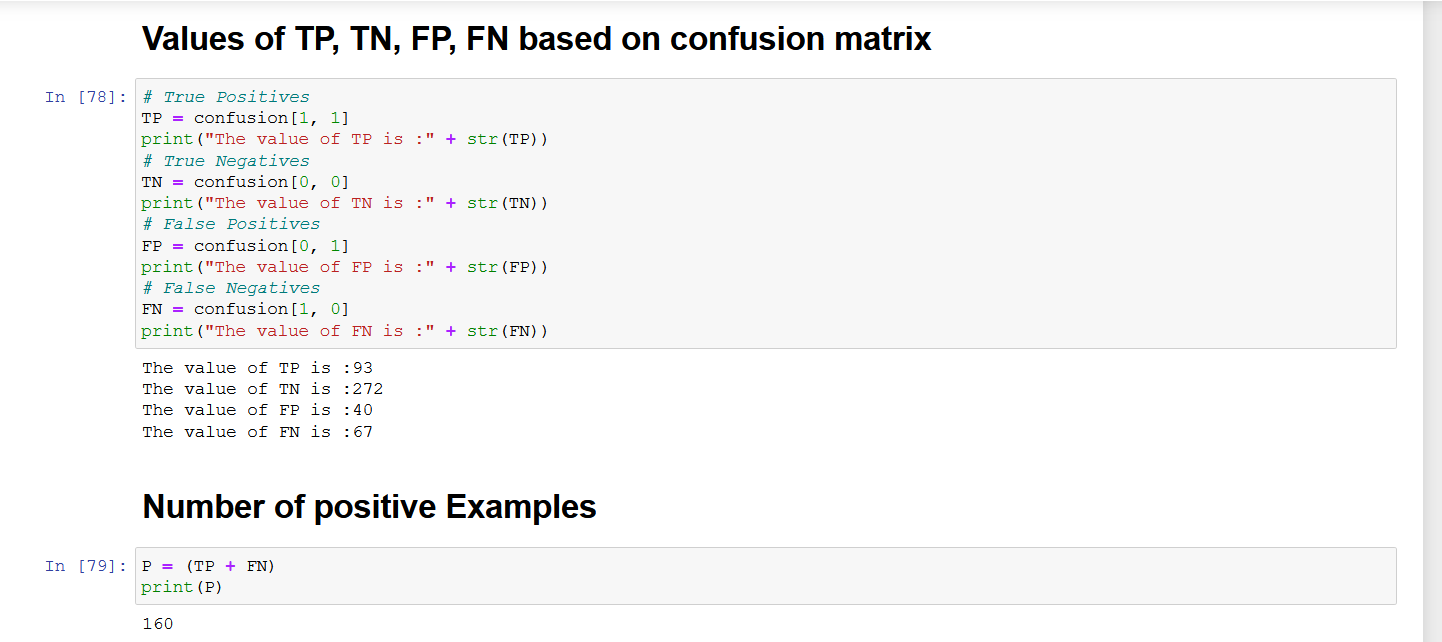


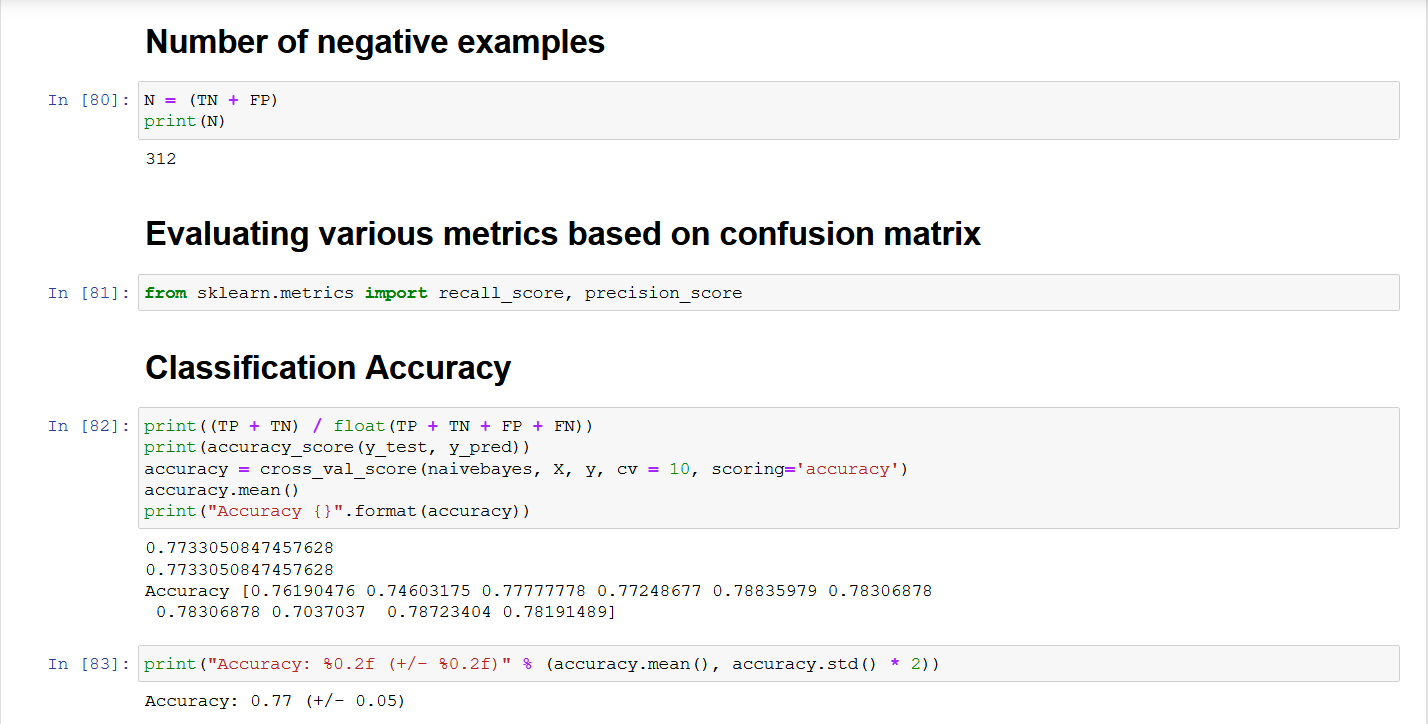


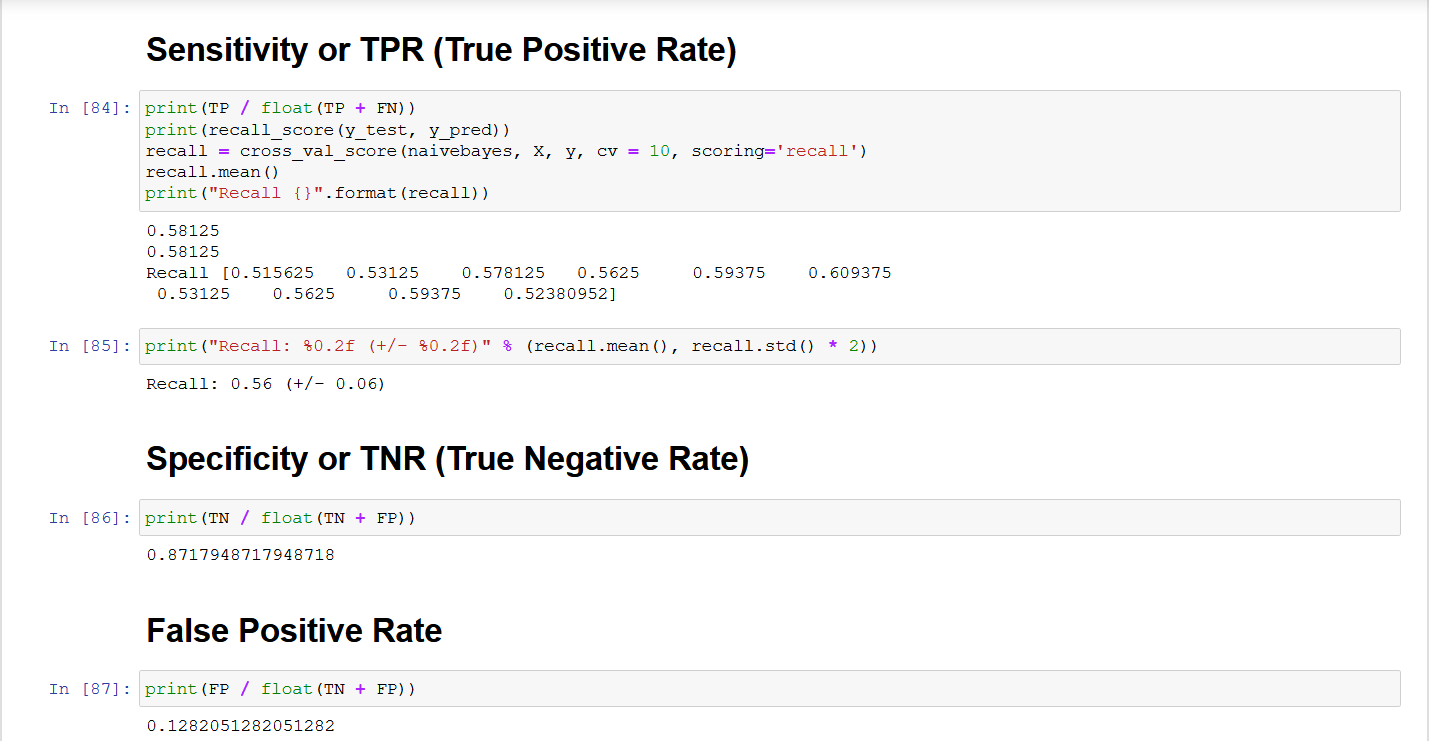






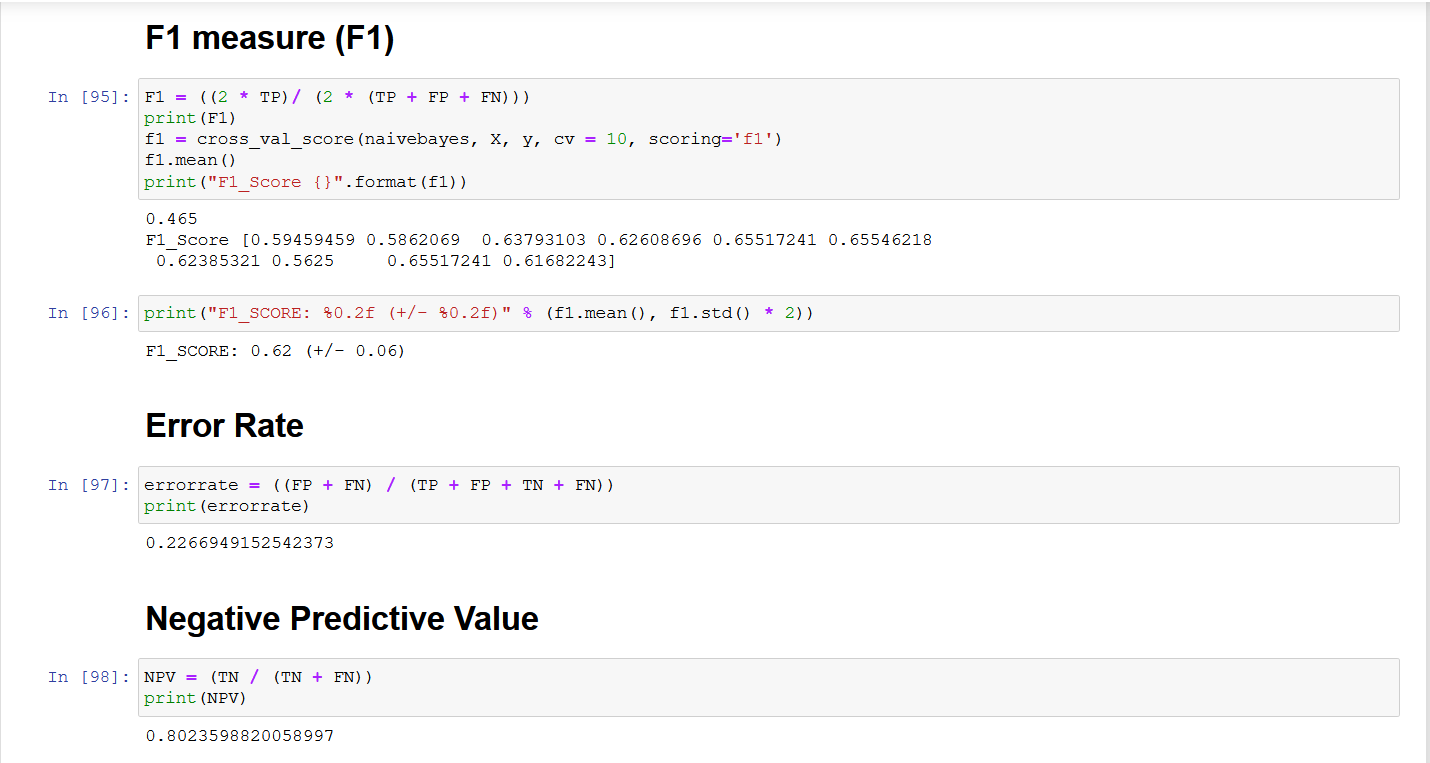


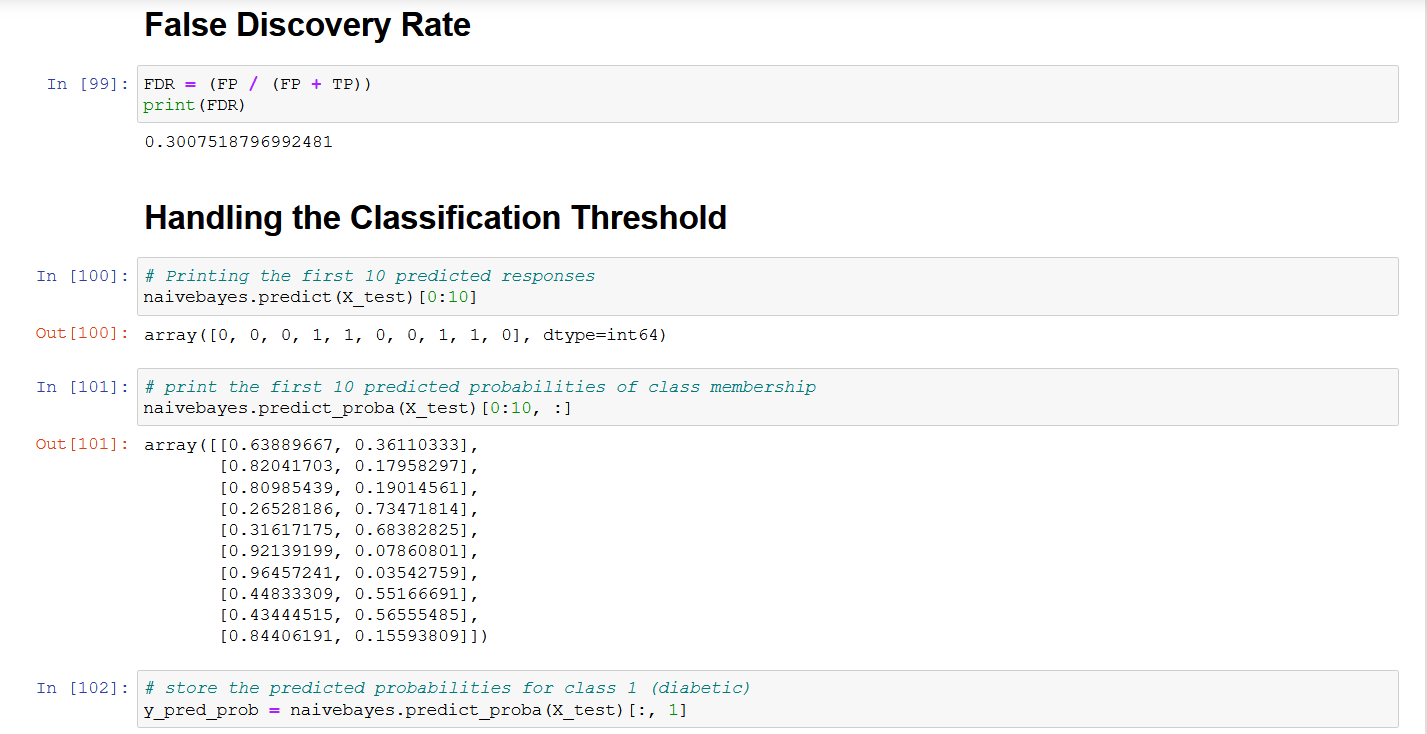




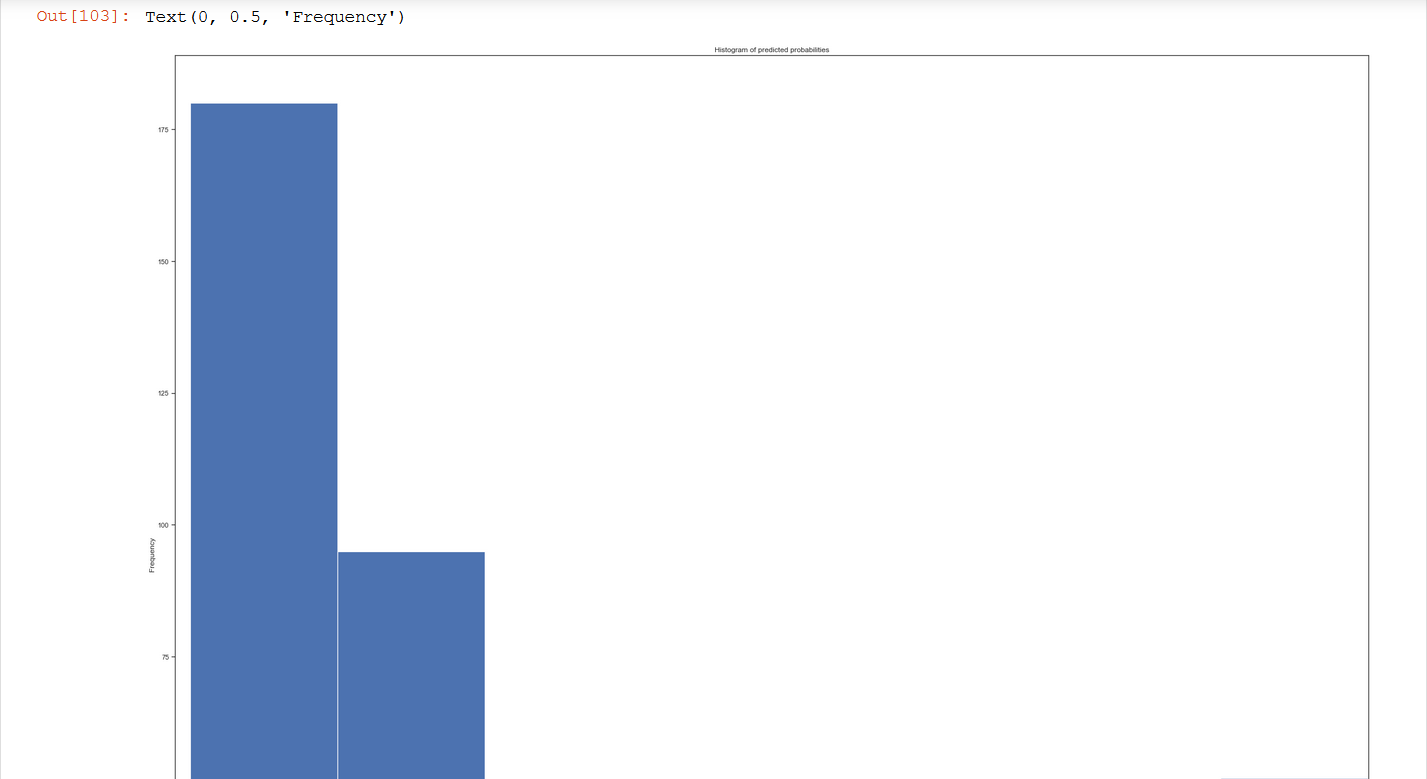


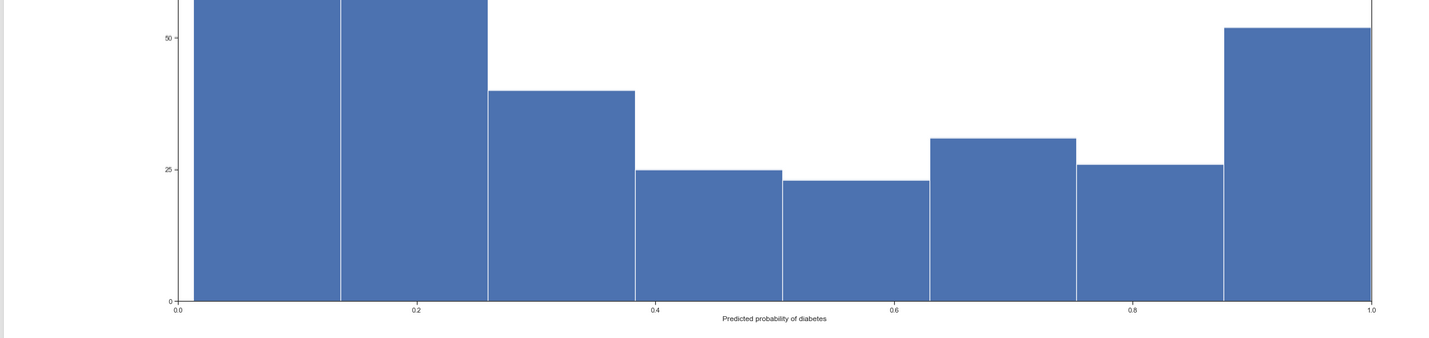


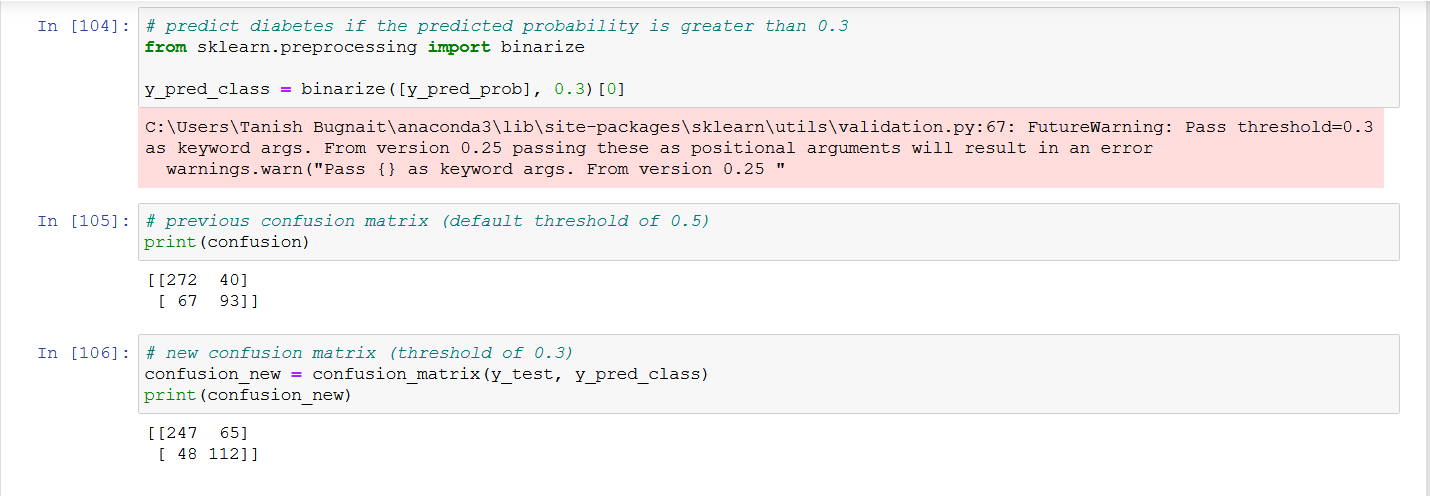


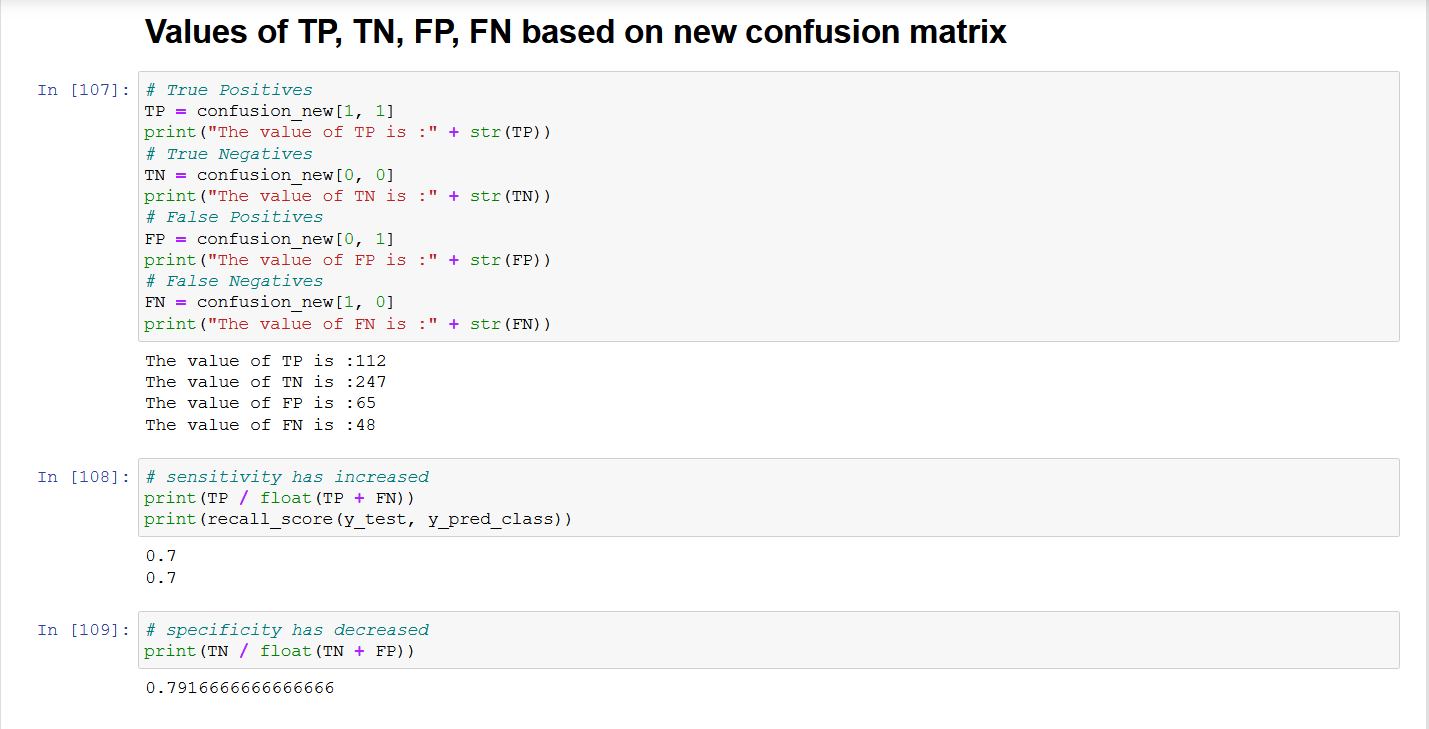


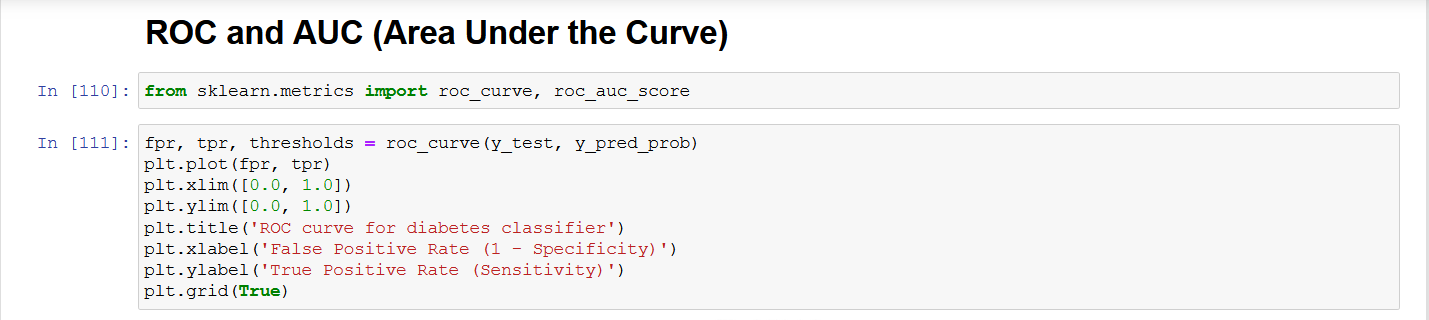


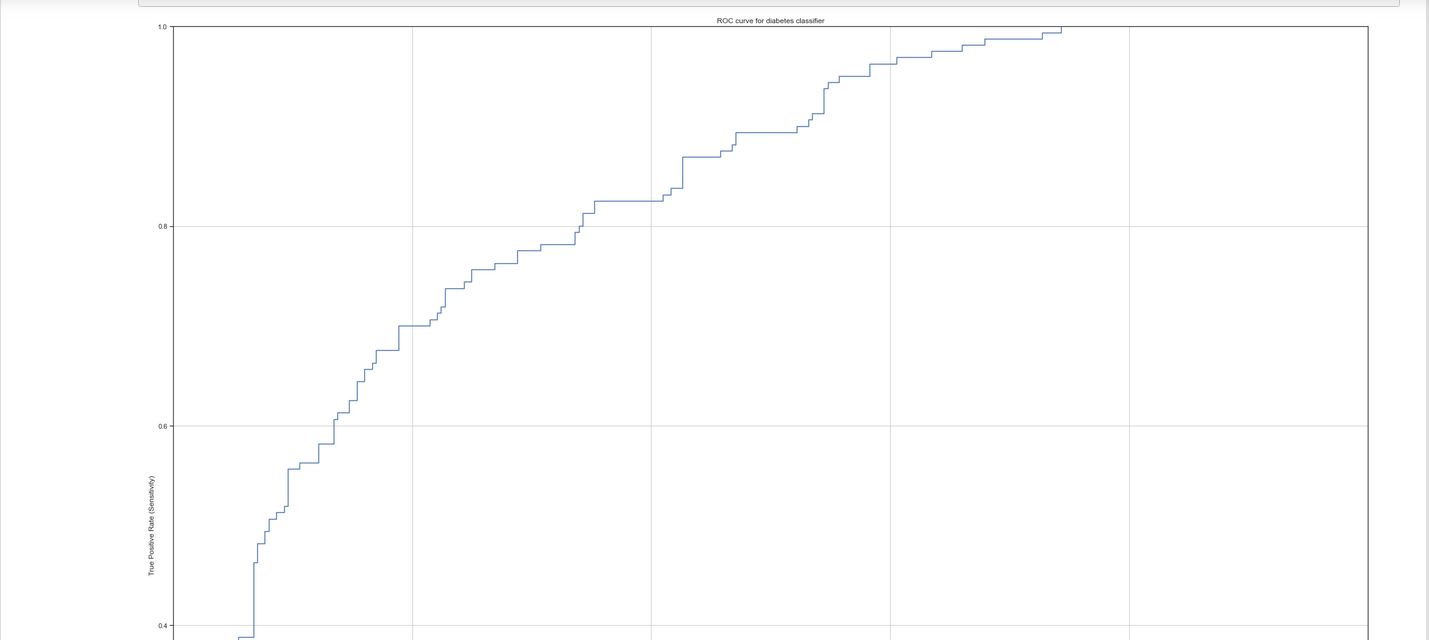


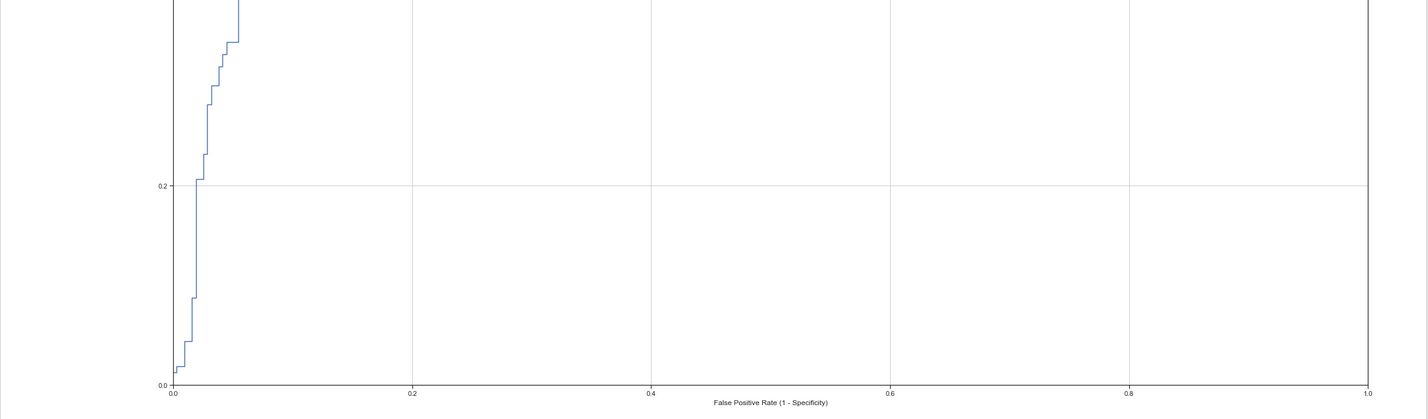


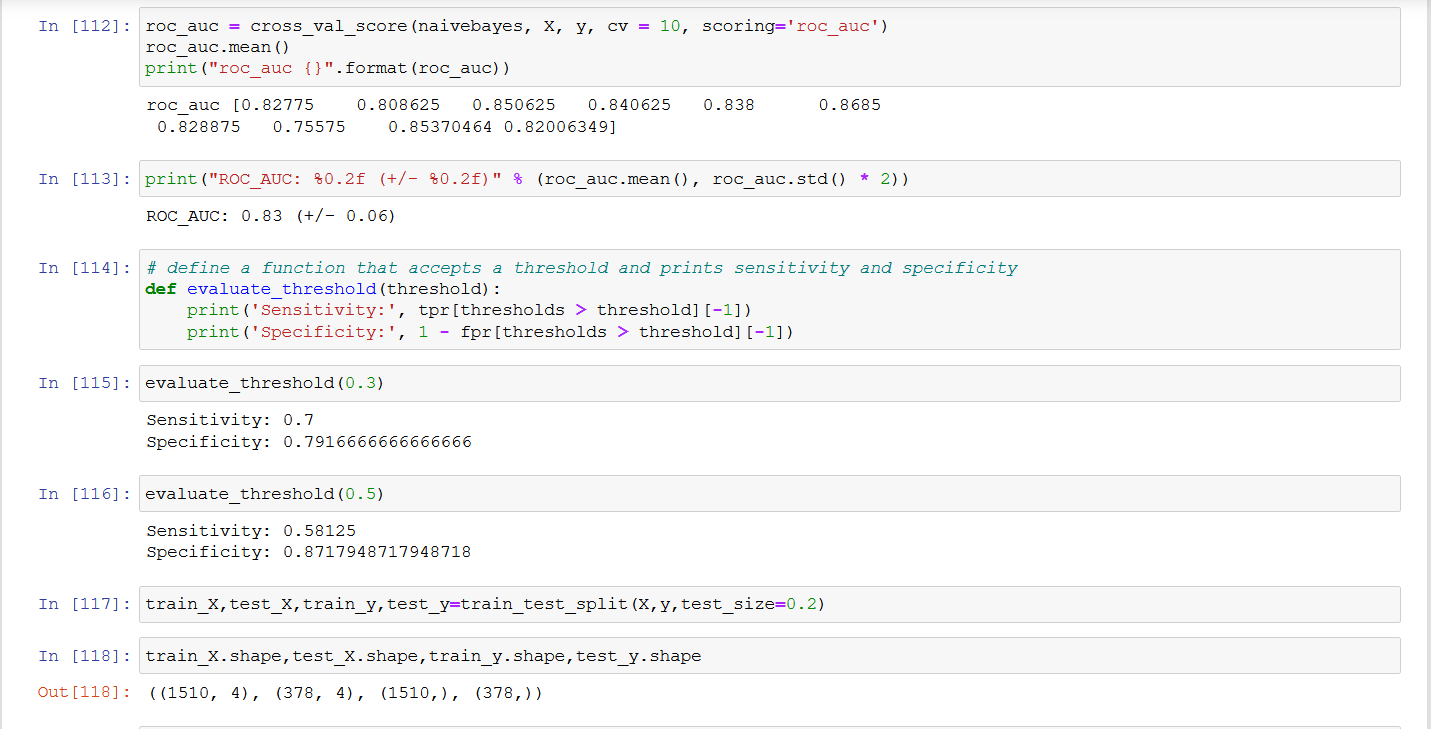


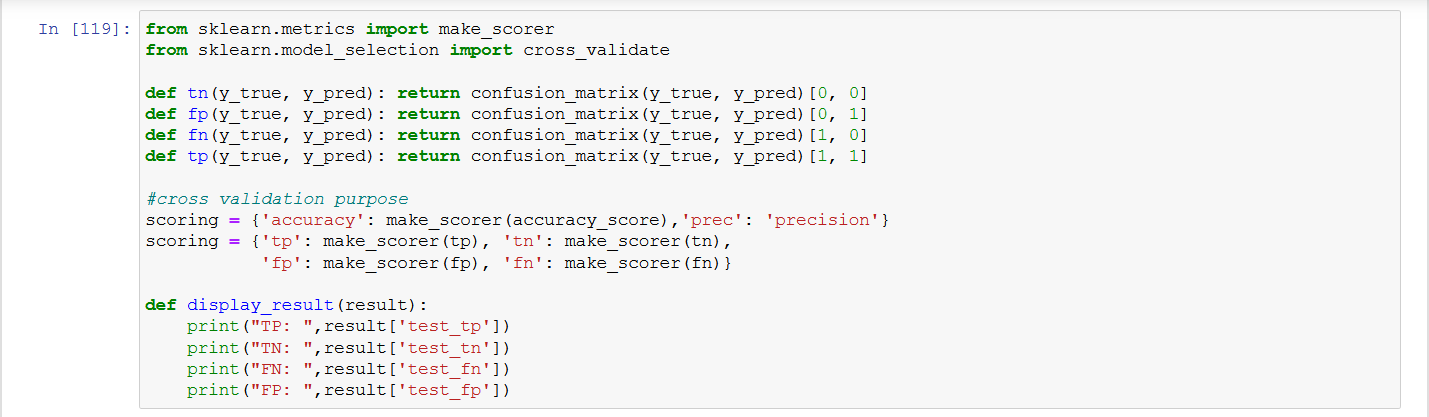




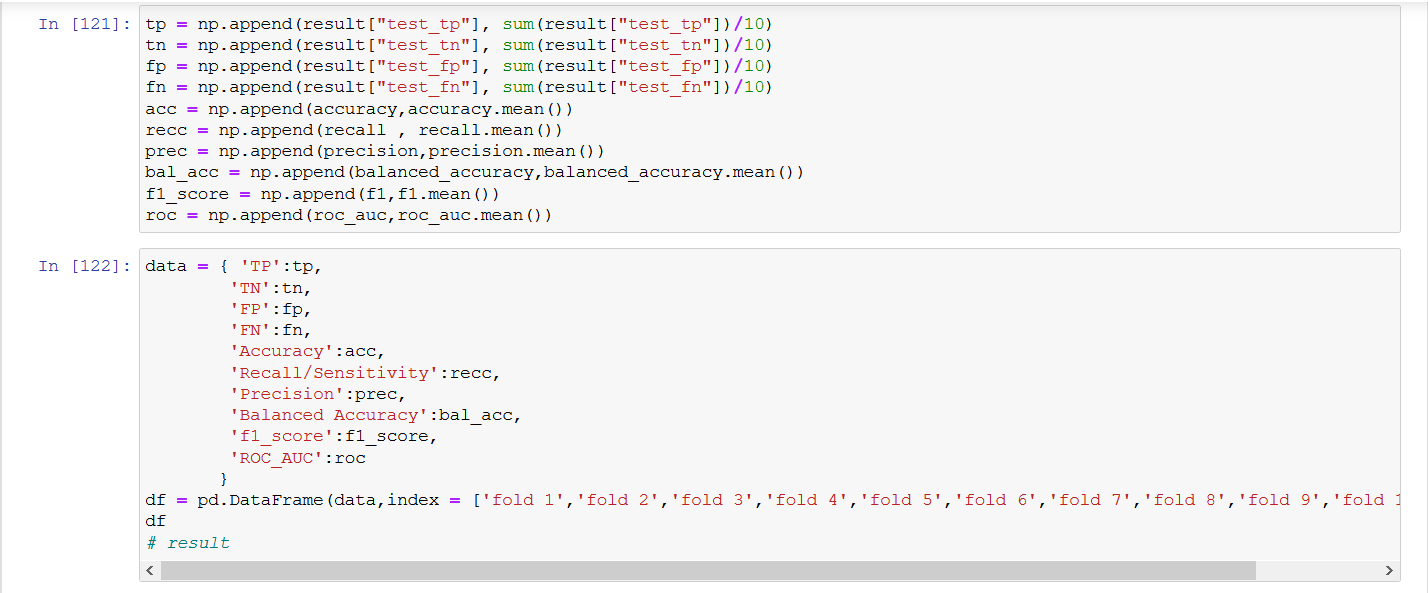


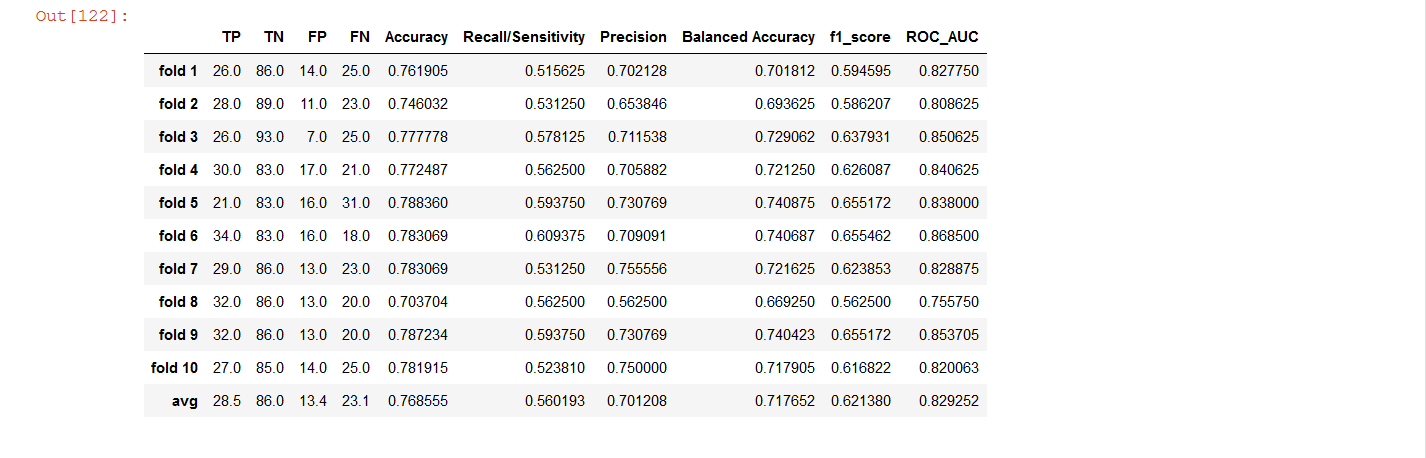




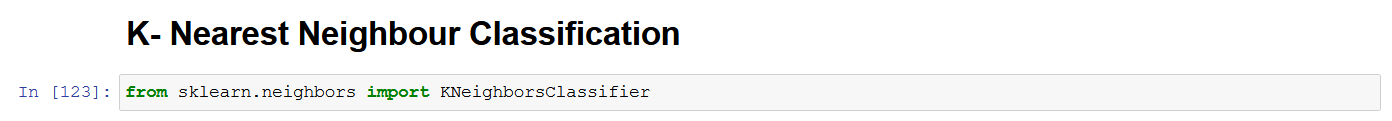








1. **KNN Classifier**

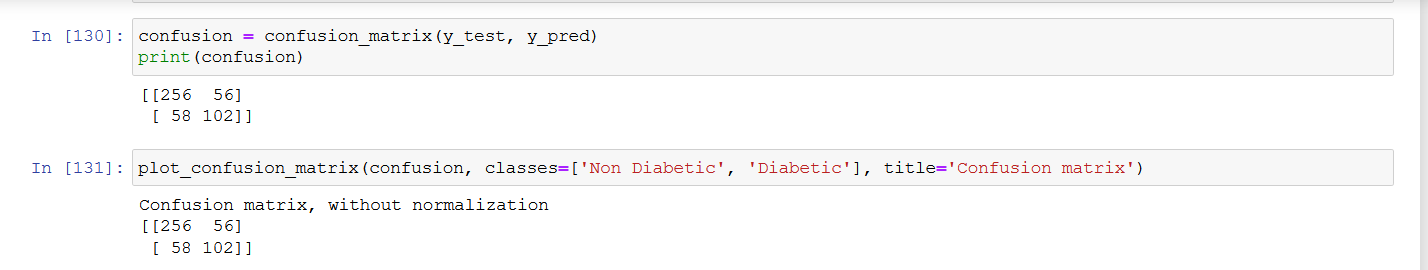






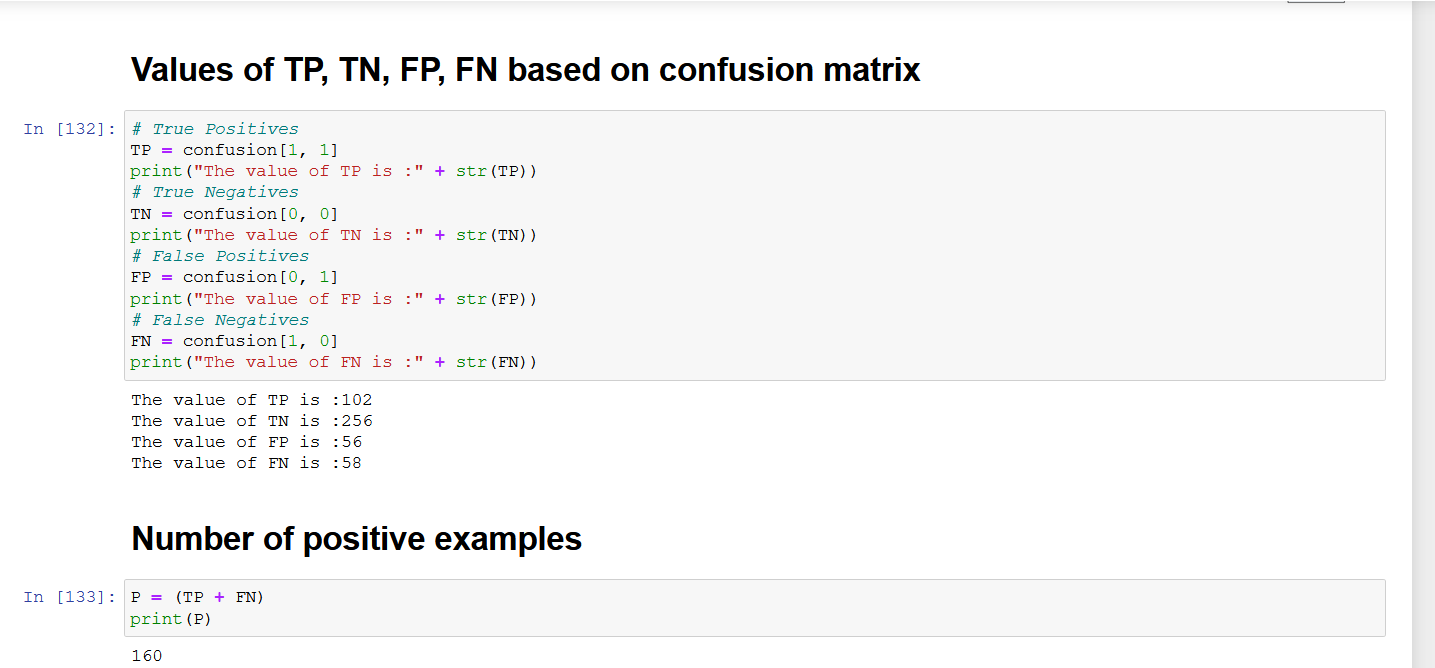


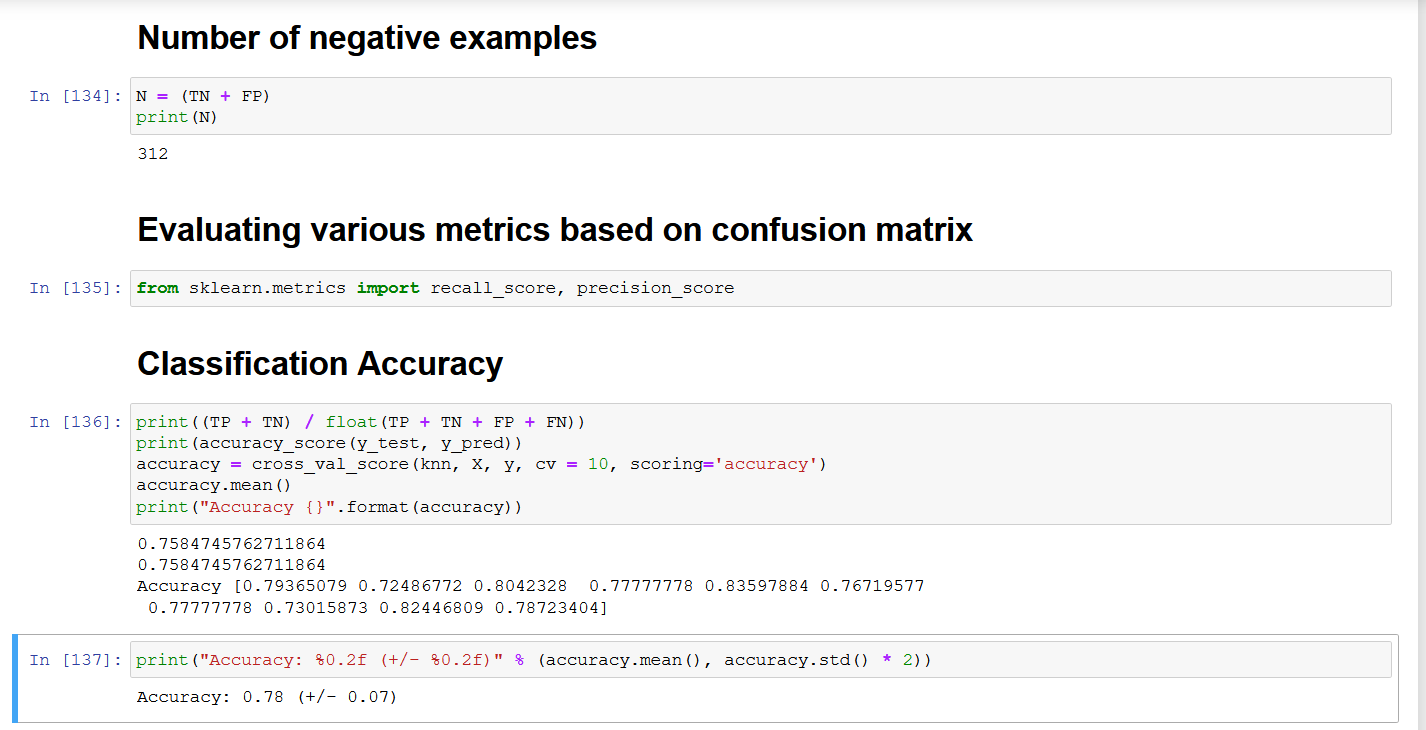






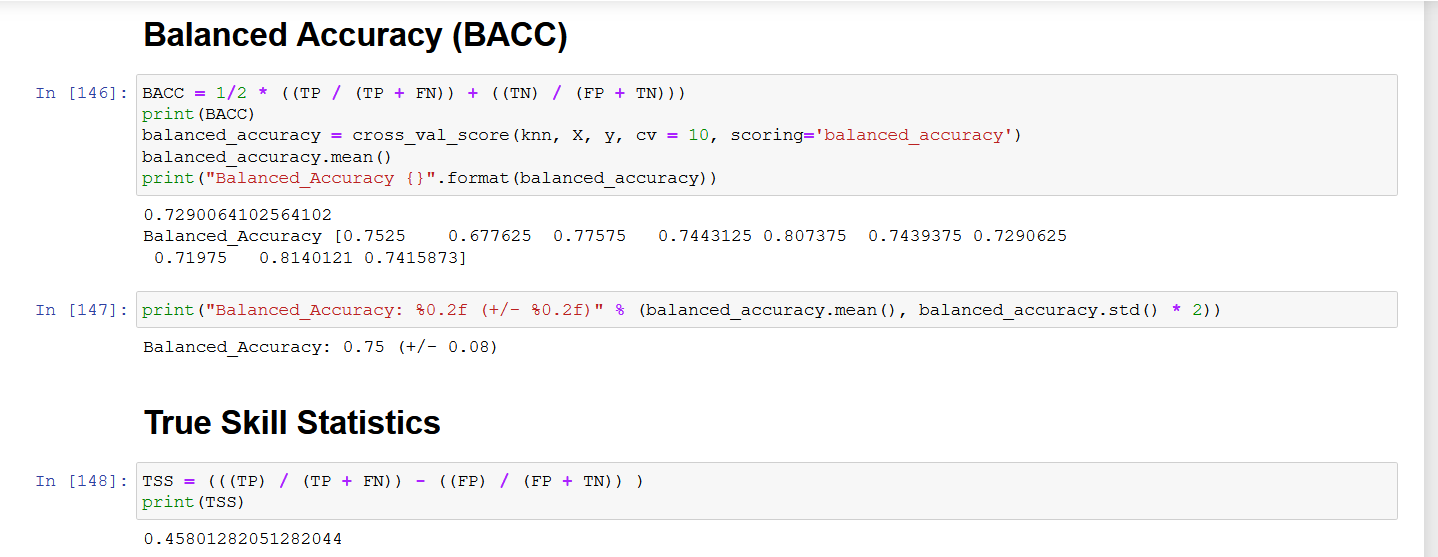




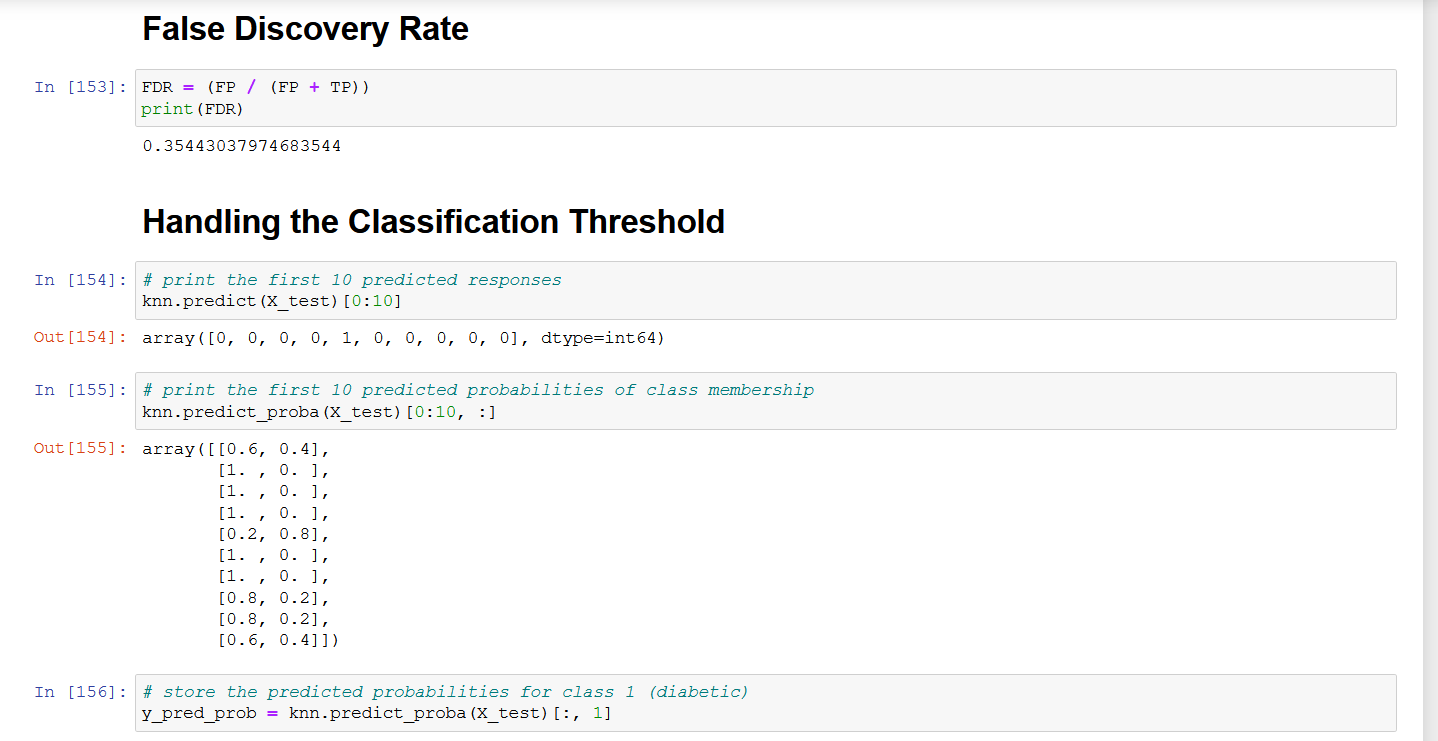


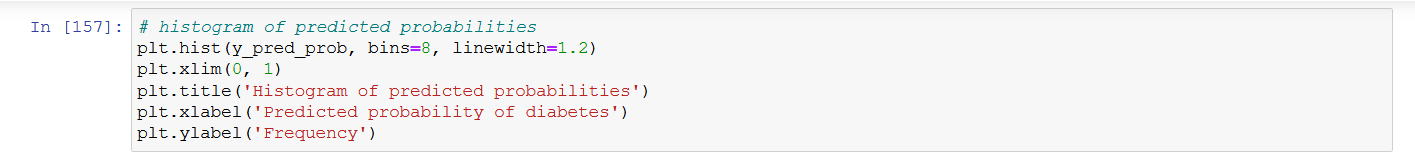


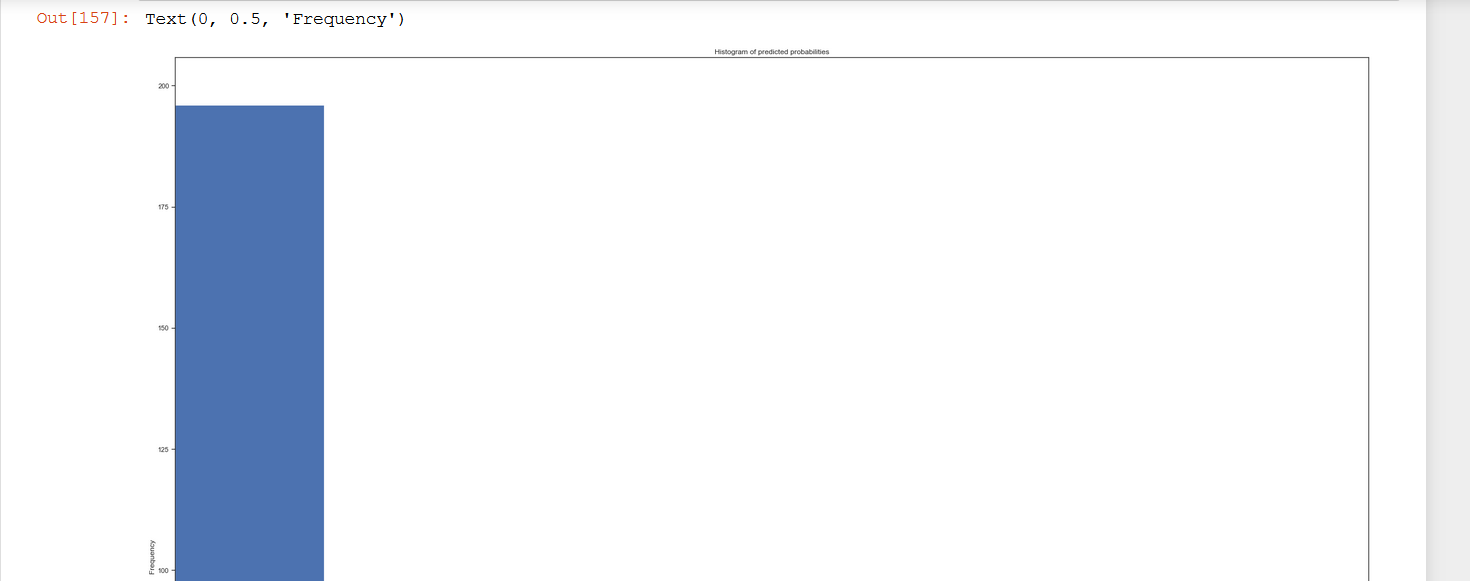


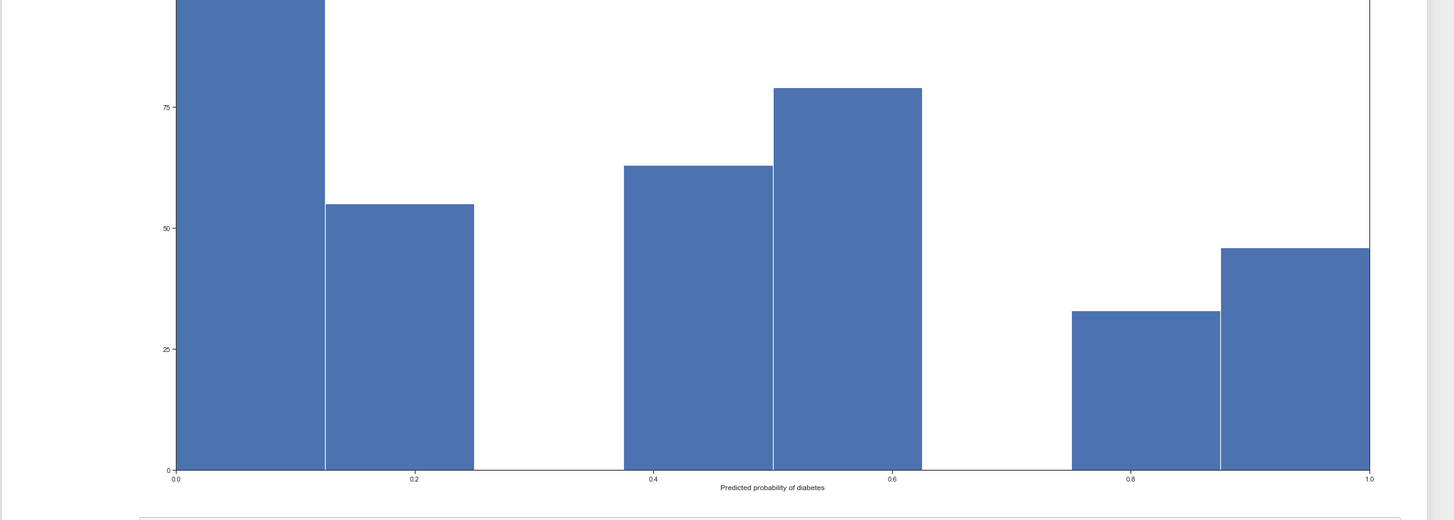


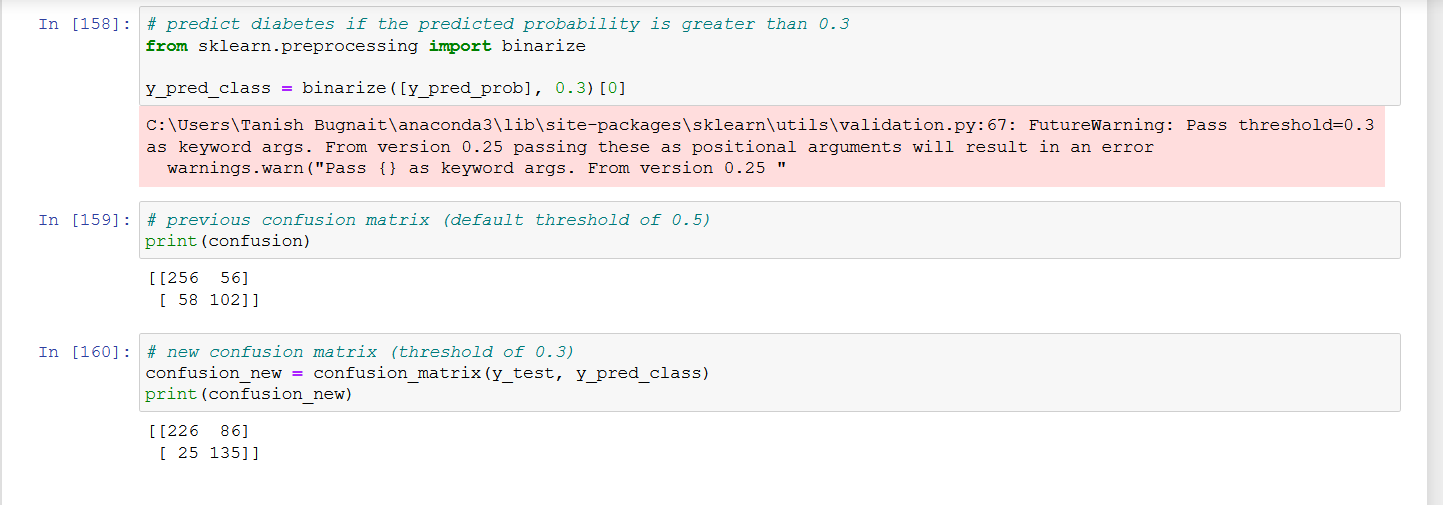


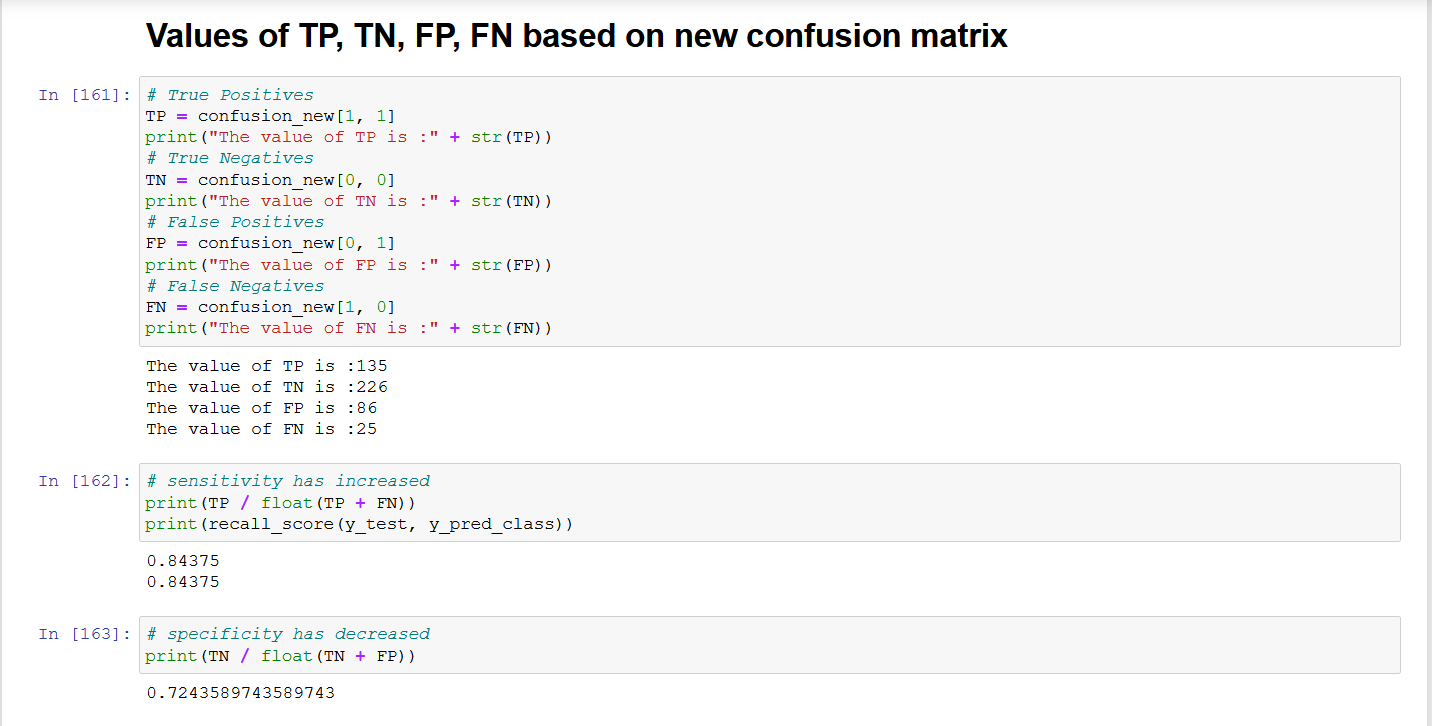


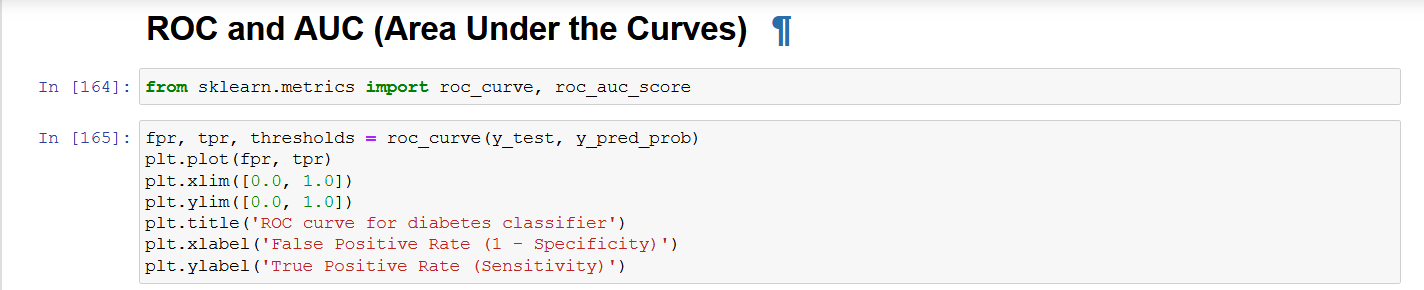


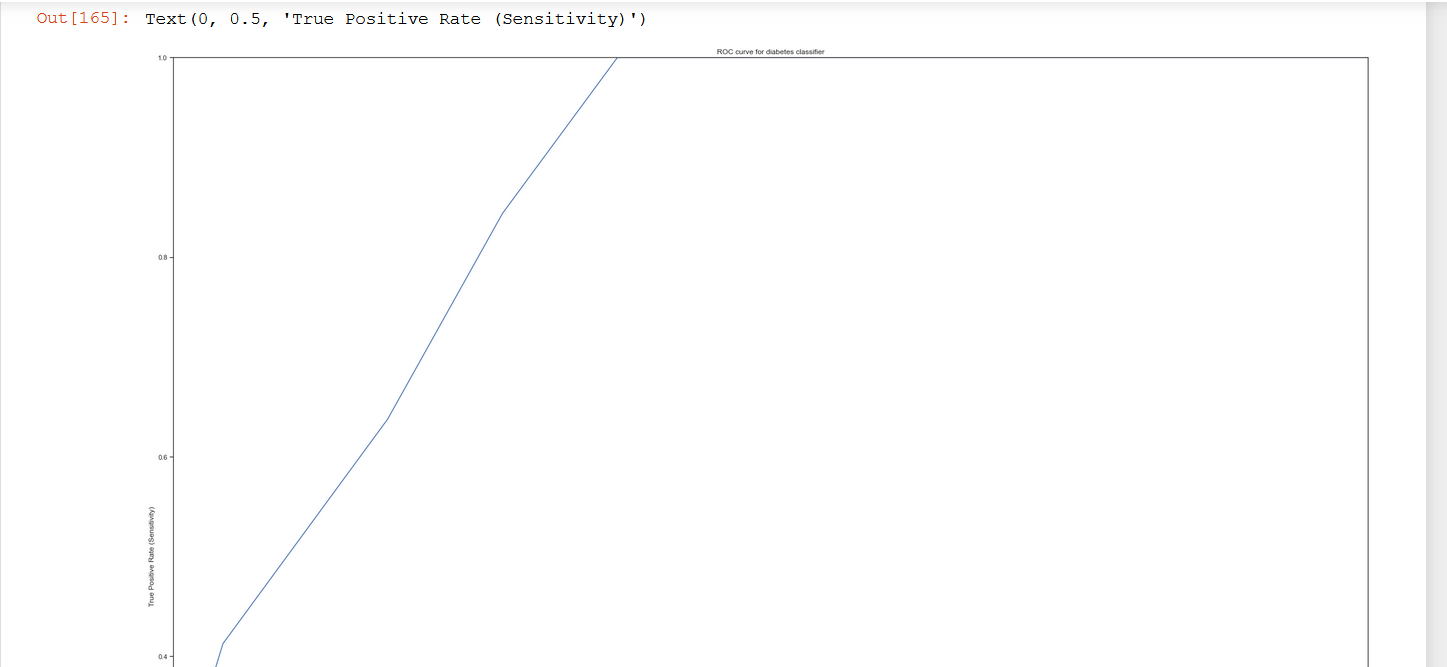


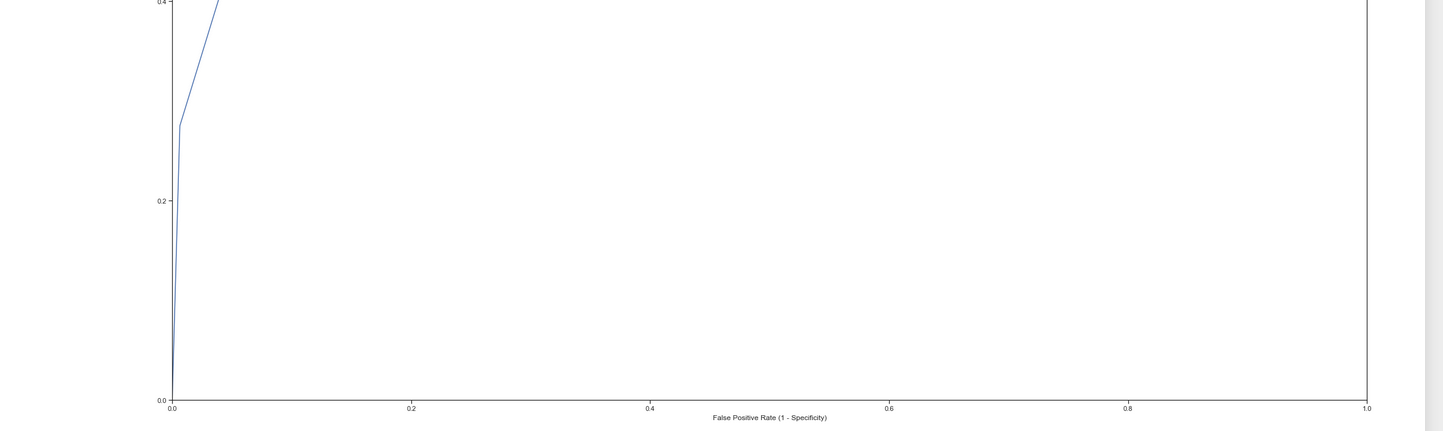


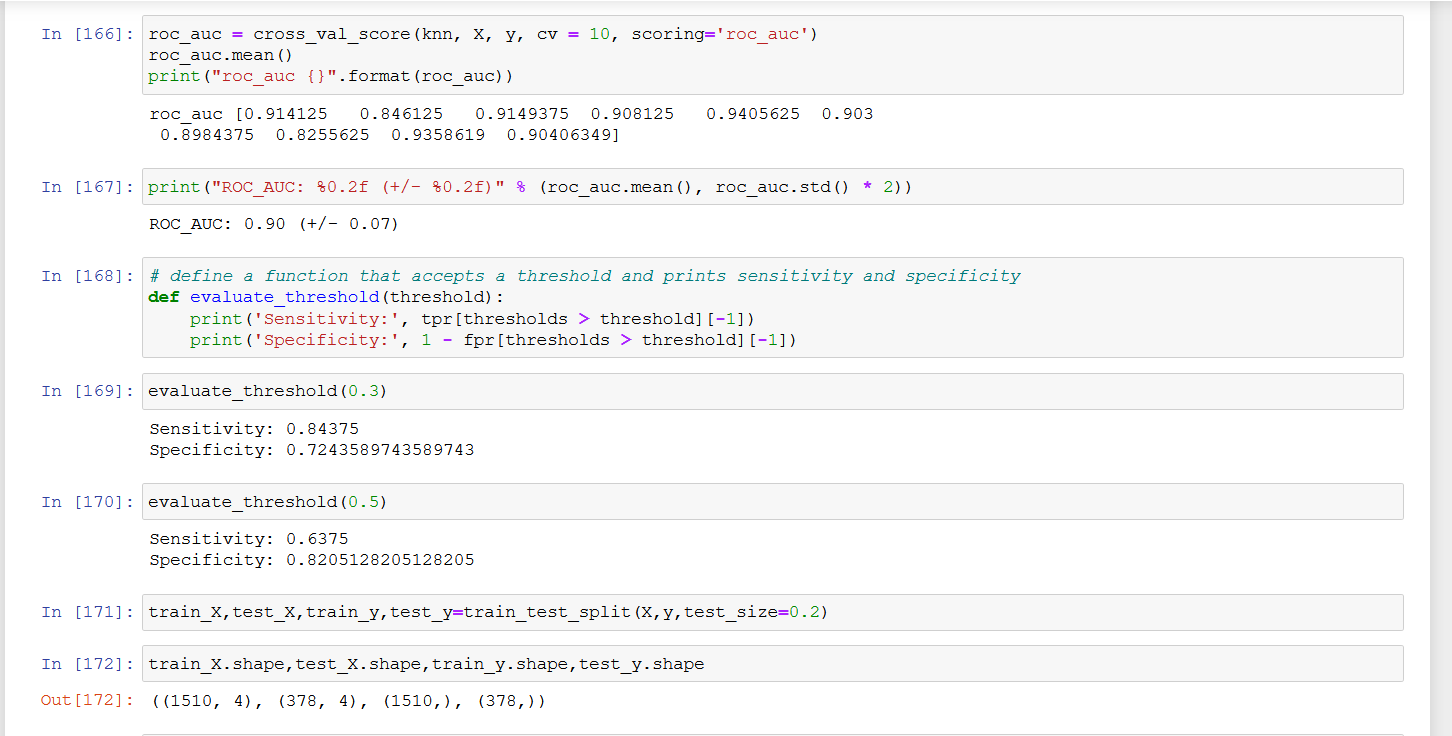


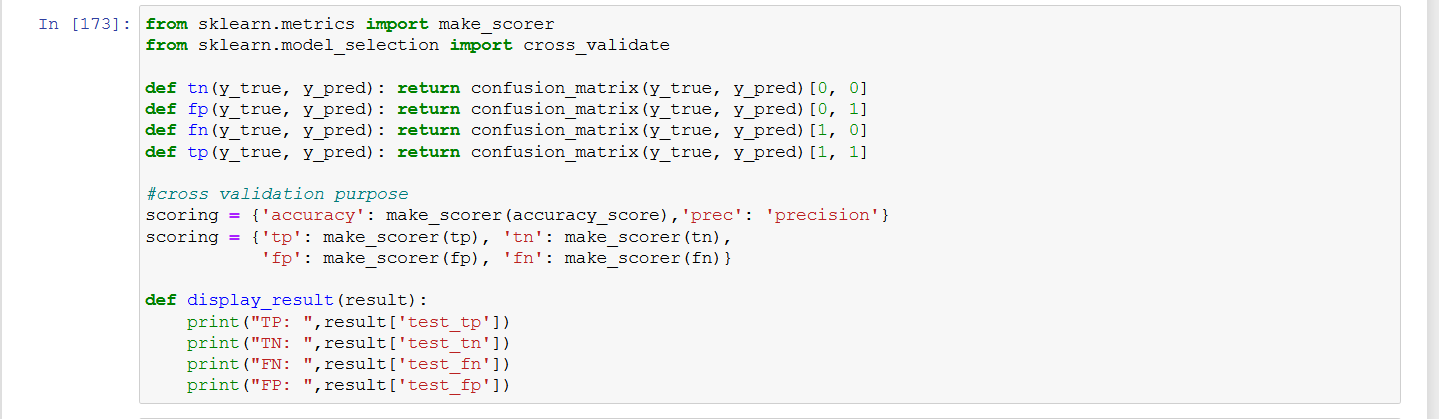


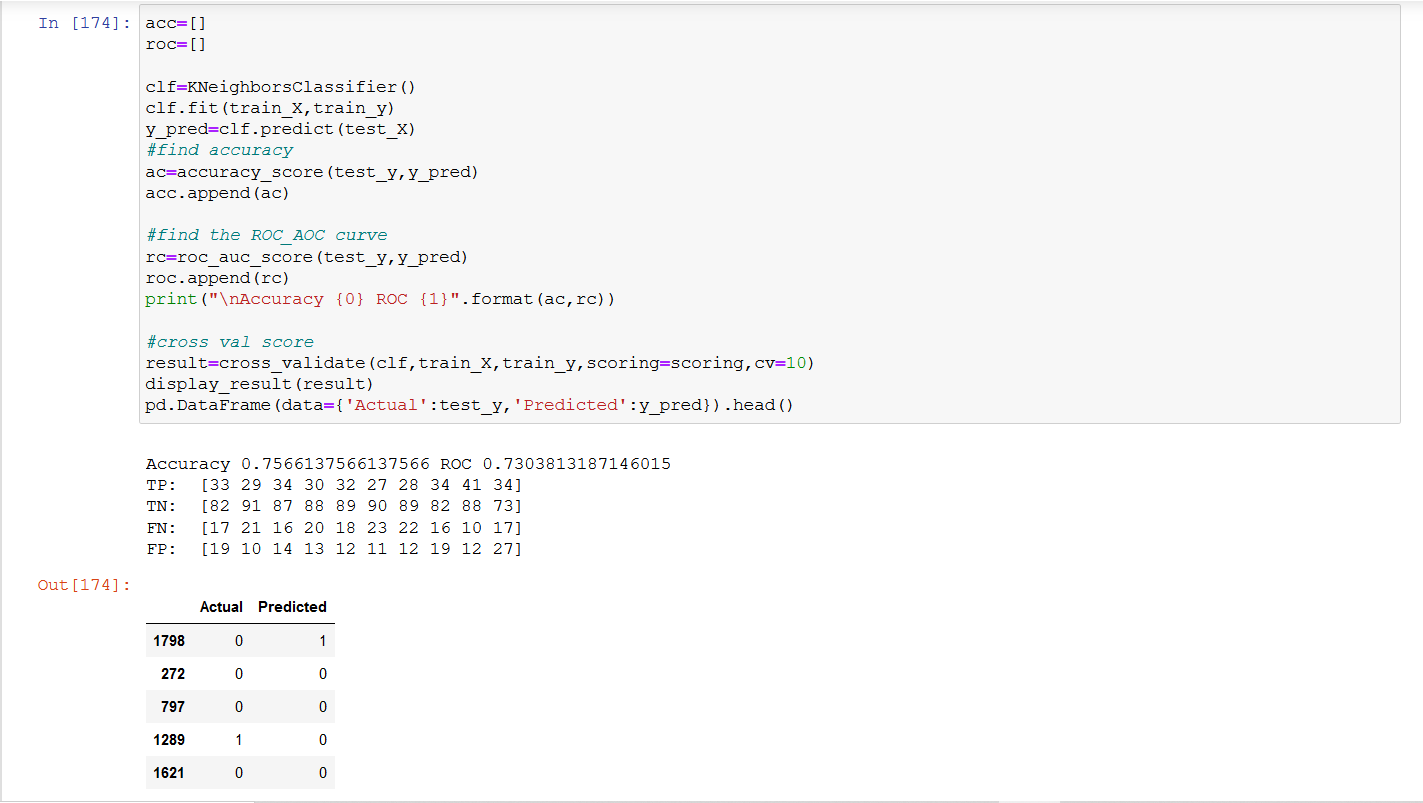


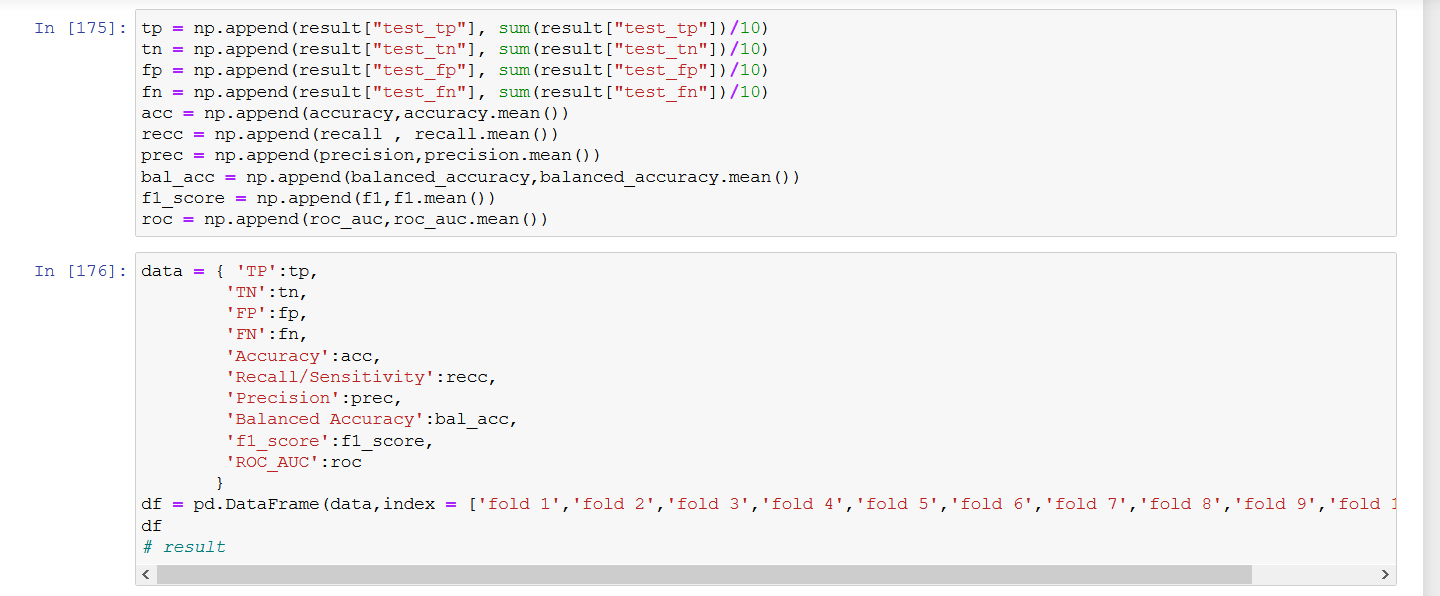


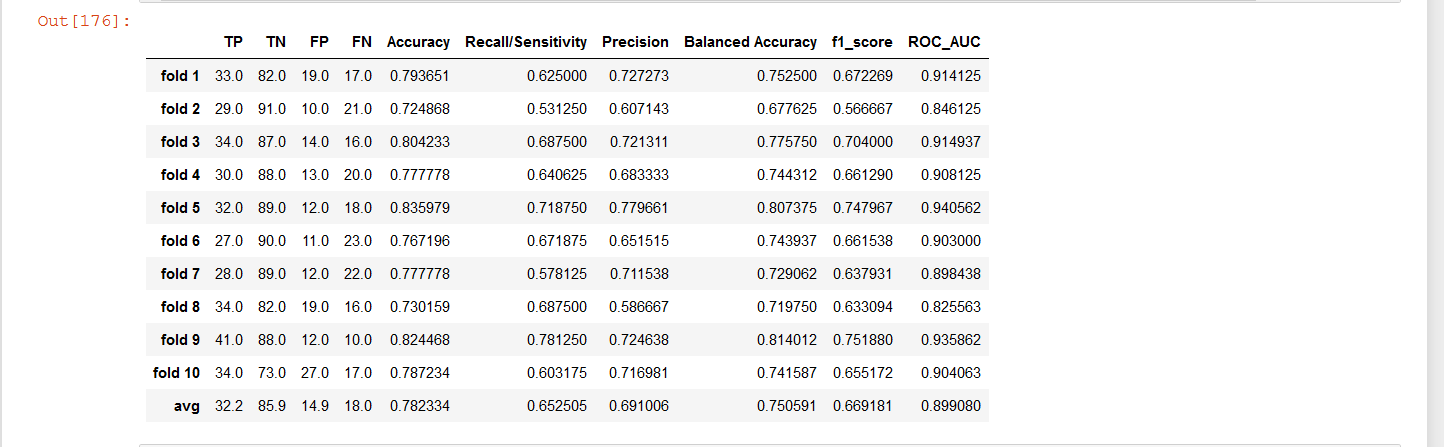












We have seen 3 different classifiers in this project:

* Random Forest
* Naïve Bayes
* KNN

We have also calculated various evaluating metrics that was asked for 10 folds in the cross validation.

According to the project Random Forest Classifier proves out to be the best among all of the three classifiers as its accuracy is 98 % and the ROC\_AUC score is 99.6%.

So according to our model, 30% of the people are diabetic and 70% of the people are non-diabetic

**Source Code**

! pip3 install pandas

! pip3 install seaborn

! pip3 install sklearn

! pip3 install matplotlib

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import itertools

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, cross\_val\_score

from sklearn.metrics import accuracy\_score

# Reading the Data

diabetes=pd.read\_csv('Desktop/Diabetes-Prediction-master/Diabetes-Prediction-master/diabetes.csv') # Please update the path according to your system

diabetes.head()

# Data Cleaning process and preparation Stage

# Let's have a look at the Dimensions of the data set

print(diabetes.shape)

# We will now remove unusual rows of data

diabetes\_mod = diabetes[(diabetes.BloodPressure != 0) & (diabetes.BMI != 0) & (diabetes.Glucose != 0)]

# Dimensions of data set after cleansing are as follows using the below command

print(diabetes\_mod.shape)

# Feature Selection Stage

# Features/Response

feature\_names = ['Pregnancies', 'Glucose', 'BMI', 'DiabetesPedigreeFunction']

X = diabetes\_mod[feature\_names]

y = diabetes\_mod.Outcome

# Histogram for the Diabetes Data

diabetes.hist(bins=10,figsize=(10,10))

plt.show()

# Heat Map Generation

#correlation

sns.heatmap(diabetes.corr())

# we can see skin thickness,insulin,pregnencies and age are full independent to each other

#age and pregencies has negative correlation

#let’s count total outcome in each target 0 1

#0 means no diabetes

#1 means patient with diabtes

sns.countplot(y=diabetes['Outcome'],palette='Set1')

sns.set(style="ticks")

sns.pairplot(diabetes, hue="Outcome")

# Outlier Box Plot Visualization

sns.set(style="whitegrid")

diabetes.boxplot(figsize=(15,6))

#box plot

sns.set(style="whitegrid")

sns.set(rc={'figure.figsize':(4,2)})

sns.boxplot(x=diabetes['Insulin'])

plt.show()

sns.boxplot(x=diabetes['BloodPressure'])

plt.show()

sns.boxplot(x=diabetes['DiabetesPedigreeFunction'])

plt.show()

# Outlier Removal

Q1=diabetes.quantile(0.25)

Q3=diabetes.quantile(0.75)

IQR=Q3-Q1

print("---Q1--- \n",Q1)

print("\n---Q3--- \n",Q3)

print("\n---IQR---\n",IQR)

print((diabetes < (Q1 - 1.5 \* IQR))|(diabetes > (Q3 + 1.5 \* IQR)))

diabetes\_out = diabetes[~((diabetes < (Q1 - 1.5 \* IQR)) |(diabetes > (Q3 + 1.5 \* IQR))).any(axis=1)]

diabetes.shape,diabetes\_out.shape

#We see that many records are deleted after outlier removal

# Scatter Matrix after outlier removal

sns.set(style="ticks")

sns.pairplot(diabetes\_out, hue="Outcome")

plt.show()

# Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

randomforest = RandomForestClassifier(n\_estimators=10, criterion='gini', max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, max\_features='auto', max\_leaf\_nodes=None, bootstrap=True, oob\_score=False, n\_jobs=1, random\_state=None, verbose=0)

# Train / Test Split

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify = y, random\_state = 0)

randomforest.fit(X\_train, y\_train)

y\_pred = randomforest.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy {}".format(accuracy))

# K-Fold Cross Validation

accuracy = cross\_val\_score(randomforest, X, y, cv = 10, scoring='accuracy')

accuracy.mean()

print("Accuracy {}".format(accuracy))

print("Accuracy: %0.2f (+/- %0.2f)" % (accuracy.mean(), accuracy.std() \* 2))

# Confusion Matrix

from sklearn.metrics import confusion\_matrix

# Method to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.rcParams["figure.figsize"] = (35,30)

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.show()

confusion = confusion\_matrix(y\_test, y\_pred)

print(confusion)

plot\_confusion\_matrix(confusion, classes=['Non Diabetic', 'Diabetic'], title='Confusion matrix')

# Values of TP, TN, FP, FN based on confusion matrix

# True Positives

TP = confusion[1, 1]

print("The value of TP is :" + str(TP))

# True Negatives

TN = confusion[0, 0]

print("The value of TN is :" + str(TN))

# False Positives

FP = confusion[0, 1]

print("The value of FP is :" + str(FP))

# False Negatives

FN = confusion[1, 0]

print("The value of FN is :" + str(FN))

# Number of Positive examples

P = (TP + FN)

print(P)

# Number of Negative Examples

N = (TN + FP)

print(N)

# Evaluating various metrics based on Confusion Matrix

from sklearn.metrics import recall\_score, precision\_score

# Classification Accuracy

print((TP + TN) / float(TP + TN + FP + FN))

print(accuracy\_score(y\_test, y\_pred))

accuracy = cross\_val\_score(randomforest, X, y, cv = 10, scoring='accuracy')

accuracy.mean()

print("Accuracy {}".format(accuracy))

print("Accuracy: %0.2f (+/- %0.2f)" % (accuracy.mean(), accuracy.std() \* 2))

# Sensitivity or TPR (True Positive Rate)

print(TP / float(TP + FN))

print(recall\_score(y\_test, y\_pred))

recall = cross\_val\_score(randomforest, X, y, cv = 10, scoring='recall')

recall.mean()

print("Recall {}".format(recall))

print("Recall: %0.2f (+/- %0.2f)" % (recall.mean(), recall.std() \* 2))

# Specificity or TNR (True Negative Rate)

print(TN / float(TN + FP))

# False Positive Rate

FPR = (FP / (TN + FP))

print(FPR)

# False Negative Rate

FNR = ((FN) / (TP + FN))

print(FNR)

# Precision

print(TP / float(TP + FP))

print(precision\_score(y\_test, y\_pred))

precision = cross\_val\_score(randomforest, X, y, cv = 10, scoring='precision')

precision.mean()

print("Precision {}".format(precision))

print("Precision: %0.2f (+/- %0.2f)" % (precision.mean(), precision.std() \* 2))

# Heidke Skill Score

HSS = 2 \* (TP \* TN - FP \* FN) / ((TP + FN) \* (FN + TN) + (TP + FP) \* (FP + TN))

print (HSS)

# Balanced Accuracy (BACC)

BACC = 1/2 \* ((TP / (TP + FN)) + ((TN) / (FP + TN)))

print(BACC)

balanced\_accuracy = cross\_val\_score(randomforest, X, y, cv = 10, scoring='balanced\_accuracy')

balanced\_accuracy.mean()

print("Balanced\_accuracy {}".format(balanced\_accuracy))

print("Balanced\_Accuracy: %0.2f (+/- %0.2f)" % (balanced\_accuracy.mean(), balanced\_accuracy.std() \* 2))

# True Skill Statistics

TSS = (((TP) / (TP + FN)) - ((FP) / (FP + TN)) )

print(TSS)

# F1 measure (F1)

F1 = ((2 \* TP)/ (2 \* (TP + FP + FN)))

print(F1)

f1 = cross\_val\_score(randomforest, X, y, cv = 10, scoring='f1')

balanced\_accuracy.mean()

print("F1\_score {}".format(f1))

print("F1\_score: %0.2f (+/- %0.2f)" % (f1.mean(), f1.std() \* 2))

# Error Rate

errorrate = ((FP + FN) / (TP + FP + TN + FN))

print(errorrate)

# Negative Predicted Value

NPV = (TN / (TN + FN))

print(NPV)

# False Discovery Rate

FDR = (FP / (FP + TP))

print(FDR)

# Handling the Classification Threshold

# print the first 10 predicted responses

randomforest.predict(X\_test)[0:10]

# print the first 10 predicted probabilities of class membership

randomforest.predict\_proba(X\_test)[0:10, :]

# store the predicted probabilities for class 1 (diabetic)

y\_pred\_prob = randomforest.predict\_proba(X\_test)[:, 1]

# histogram of predicted probabilities

plt.hist(y\_pred\_prob, bins=8, linewidth=1.2)

plt.xlim(0, 1)

plt.title('Histogram of predicted probabilities')

plt.xlabel('Predicted probability of diabetes')

plt.ylabel('Frequency')

# predict diabetes if the predicted probability is greater than 0.3

from sklearn.preprocessing import binarize

y\_pred\_class = binarize([y\_pred\_prob], 0.3)[0]

# previous confusion matrix (default threshold of 0.5)

print(confusion)

# new confusion matrix (threshold of 0.3)

confusion\_new = confusion\_matrix(y\_test, y\_pred\_class)

print(confusion\_new)

# Values of TP, TN, FP, FN based on new confusion matrix

# True Positives

TP = confusion\_new[1, 1]

print("The value of TP is :" + str(TP))

# True Negatives

TN = confusion\_new[0, 0]

print("The value of TN is :" + str(TN))

# False Positives

FP = confusion\_new[0, 1]

print("The value of FP is :" + str(FP))

# False Negatives

FN = confusion\_new[1, 0]

print("The value of FN is :" + str(FN))

# We observe that the sensitivity has increased

print(TP / float(TP + FN))

print(recall\_score(y\_test, y\_pred\_class))

# specificity has decreased

print(TN / float(TN + FP))

# ROC and AUC (Area Under the Curves)

from sklearn.metrics import roc\_curve, roc\_auc\_score

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)

plt.plot(fpr, tpr)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.title('ROC curve for diabetes classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.grid(True)

roc\_auc = cross\_val\_score(randomforest, X, y, cv = 10, scoring='roc\_auc')

roc\_auc.mean()

print("roc\_auc {}".format(roc\_auc))

print("ROC\_AUC: %0.2f (+/- %0.2f)" % (roc\_auc.mean(), roc\_auc.std() \* 2))

# define a function that accepts a threshold and prints sensitivity and specificity

def evaluate\_threshold(threshold):

print('Sensitivity:', tpr[thresholds > threshold][-1])

print('Specificity:', 1 - fpr[thresholds > threshold][-1])

evaluate\_threshold(0.3)

evaluate\_threshold(0.5)

train\_X,test\_X,train\_y,test\_y=train\_test\_split(X,y,test\_size=0.2)

train\_X.shape,test\_X.shape,train\_y.shape,test\_y.shape

from sklearn.metrics import make\_scorer

from sklearn.model\_selection import cross\_validate

def tn(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[0, 0]

def fp(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[0, 1]

def fn(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[1, 0]

def tp(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[1, 1]

#cross validation purpose

scoring = {'accuracy': make\_scorer(accuracy\_score),'prec': 'precision'}

scoring = {'tp': make\_scorer(tp), 'tn': make\_scorer(tn),

'fp': make\_scorer(fp), 'fn': make\_scorer(fn)}

def display\_result(result):

print("TP: ",result['test\_tp'])

print("TN: ",result['test\_tn'])

print("FN: ",result['test\_fn'])

print("FP: ",result['test\_fp'])

acc=[]

roc=[]

clf=RandomForestClassifier()

clf.fit(train\_X,train\_y)

y\_pred=clf.predict(test\_X)

#find accuracy

ac=accuracy\_score(test\_y,y\_pred)

acc.append(ac)

#find the ROC\_AOC curve

rc=roc\_auc\_score(test\_y,y\_pred)

roc.append(rc)

print("\nAccuracy {0} ROC {1}".format(ac,rc))

#cross val score

result=cross\_validate(clf,train\_X,train\_y,scoring=scoring,cv=10)

display\_result(result)

pd.DataFrame(data={'Actual':test\_y,'Predicted':y\_pred}).head()

tp = np.append(result["test\_tp"], sum(result["test\_tp"])/10)

tn = np.append(result["test\_tn"], sum(result["test\_tn"])/10)

fp = np.append(result["test\_fp"], sum(result["test\_fp"])/10)

fn = np.append(result["test\_fn"], sum(result["test\_fn"])/10)

acc = np.append(accuracy,accuracy.mean())

recc = np.append(recall , recall.mean())

prec = np.append(precision,precision.mean())

bal\_acc = np.append(balanced\_accuracy,balanced\_accuracy.mean())

f1\_score = np.append(f1,f1.mean())

roc = np.append(roc\_auc,roc\_auc.mean())

data = { 'TP':tp,

'TN':tn,

'FP':fp,

'FN':fn,

'Accuracy':acc,

'Recall/Sensitivity':recc,

'Precision':prec,

'Balanced Accuracy':bal\_acc,

'f1\_score':f1\_score,

'ROC\_AUC':roc

}

df = pd.DataFrame(data,index = ['fold 1','fold 2','fold 3','fold 4','fold 5','fold 6','fold 7','fold 8','fold 9','fold 10','avg'])

df

# result

# Naive Bayes Classifier

from sklearn.naive\_bayes import GaussianNB

naivebayes = GaussianNB( priors=None, var\_smoothing=1e-09)

# Train / Test Split

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify = y, random\_state = 0)

naivebayes.fit(X\_train, y\_train)

y\_pred = naivebayes.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy {}".format(accuracy))

# K-Fold Cross Validation

accuracy = cross\_val\_score(naivebayes, X, y, cv = 10, scoring='accuracy')

accuracy.mean()

print("Accuracy {}".format(accuracy))

print("Accuracy: %0.2f (+/- %0.2f)" % (accuracy.mean(), accuracy.std() \* 2))

# Confusion Matrix

from sklearn.metrics import confusion\_matrix

# Method to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.rcParams["figure.figsize"] = (35,30)

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.show()

confusion = confusion\_matrix(y\_test, y\_pred)

print(confusion)

plot\_confusion\_matrix(confusion, classes=['Non Diabetic', 'Diabetic'], title='Confusion matrix')

# Values of TP, TN, FP, FN based on confusion matrix

# True Positives

TP = confusion[1, 1]

print("The value of TP is :" + str(TP))

# True Negatives

TN = confusion[0, 0]

print("The value of TN is :" + str(TN))

# False Positives

FP = confusion[0, 1]

print("The value of FP is :" + str(FP))

# False Negatives

FN = confusion[1, 0]

print("The value of FN is :" + str(FN))

# Number of Positive examples

P = (TP + FN)

print(P)

# Number of Negative Examples

N = (TN + FP)

print(N)

# Evaluating various metrics based on Confusion Matrix

from sklearn.metrics import recall\_score, precision\_score

# Classification Accuracy

print((TP + TN) / float(TP + TN + FP + FN))

print(accuracy\_score(y\_test, y\_pred))

accuracy = cross\_val\_score(naivebayes, X, y, cv = 10, scoring='accuracy')

accuracy.mean()

print("Accuracy {}".format(accuracy))

print("Accuracy: %0.2f (+/- %0.2f)" % (accuracy.mean(), accuracy.std() \* 2))

# Sensitivity or TPR (True Positive Rate)

print(TP / float(TP + FN))

print(recall\_score(y\_test, y\_pred))

recall = cross\_val\_score(naivebayes, X, y, cv = 10, scoring='recall')

recall.mean()

print("Recall {}".format(recall))

print("Recall: %0.2f (+/- %0.2f)" % (recall.mean(), recall.std() \* 2))

# Specificity or TNR (True Negative Rate)

print(TN / float(TN + FP))

# False Positive Rate

FPR = (FP / (TN + FP))

print(FPR)

# False Negative Rate

FNR = ((FN) / (TP + FN))

print(FNR)

# Precision

print(TP / float(TP + FP))

print(precision\_score(y\_test, y\_pred))

precision = cross\_val\_score(naivebayes, X, y, cv = 10, scoring='precision')

precision.mean()

print("Precision {}".format(precision))

print("Precision: %0.2f (+/- %0.2f)" % (precision.mean(), precision.std() \* 2))

# Heidke Skill Score

HSS = 2 \* (TP \* TN - FP \* FN) / ((TP + FN) \* (FN + TN) + (TP + FP) \* (FP + TN))

print (HSS)

# Balanced Accuracy (BACC)

BACC = 1/2 \* ((TP / (TP + FN)) + ((TN) / (FP + TN)))

print(BACC)

balanced\_accuracy = cross\_val\_score(naivebayes, X, y, cv = 10, scoring='balanced\_accuracy')

balanced\_accuracy.mean()

print("Balanced\_accuracy {}".format(balanced\_accuracy))

print("Balanced\_Accuracy: %0.2f (+/- %0.2f)" % (balanced\_accuracy.mean(), balanced\_accuracy.std() \* 2))

# True Skill Statistics

TSS = (((TP) / (TP + FN)) - ((FP) / (FP + TN)) )

print(TSS)

# F1 measure (F1)

F1 = ((2 \* TP)/ (2 \* (TP + FP + FN)))

print(F1)

f1 = cross\_val\_score(naivebayes, X, y, cv = 10, scoring='f1')

balanced\_accuracy.mean()

print("F1\_score {}".format(f1))

print("F1\_score: %0.2f (+/- %0.2f)" % (f1.mean(), f1.std() \* 2))

# Error Rate

errorrate = ((FP + FN) / (TP + FP + TN + FN))

print(errorrate)

# Negative Predicted Value

NPV = (TN / (TN + FN))

print(NPV)

# False Discovery Rate

FDR = (FP / (FP + TP))

print(FDR)

# Handling the Classification Threshold

# print the first 10 predicted responses

naivebayes.predict(X\_test)[0:10]

# print the first 10 predicted probabilities of class membership

naivebayes.predict\_proba(X\_test)[0:10, :]

# store the predicted probabilities for class 1 (diabetic)

y\_pred\_prob = naivebayes.predict\_proba(X\_test)[:, 1]

# histogram of predicted probabilities

plt.hist(y\_pred\_prob, bins=8, linewidth=1.2)

plt.xlim(0, 1)

plt.title('Histogram of predicted probabilities')

plt.xlabel('Predicted probability of diabetes')

plt.ylabel('Frequency')

# predict diabetes if the predicted probability is greater than 0.3

from sklearn.preprocessing import binarize

y\_pred\_class = binarize([y\_pred\_prob], 0.3)[0]

# previous confusion matrix (default threshold of 0.5)

print(confusion)

# new confusion matrix (threshold of 0.3)

confusion\_new = confusion\_matrix(y\_test, y\_pred\_class)

print(confusion\_new)

# Values of TP, TN, FP, FN based on new confusion matrix

# True Positives

TP = confusion\_new[1, 1]

print("The value of TP is :" + str(TP))

# True Negatives

TN = confusion\_new[0, 0]

print("The value of TN is :" + str(TN))

# False Positives

FP = confusion\_new[0, 1]

print("The value of FP is :" + str(FP))

# False Negatives

FN = confusion\_new[1, 0]

print("The value of FN is :" + str(FN))

# We observe that the sensitivity has increased

print(TP / float(TP + FN))

print(recall\_score(y\_test, y\_pred\_class))

# specificity has decreased

print(TN / float(TN + FP))

# ROC and AUC (Area Under the Curves)

from sklearn.metrics import roc\_curve, roc\_auc\_score

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)

plt.plot(fpr, tpr)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.title('ROC curve for diabetes classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.grid(True)

roc\_auc = cross\_val\_score(naivebayes, X, y, cv = 10, scoring='roc\_auc')

roc\_auc.mean()

print("roc\_auc {}".format(roc\_auc))

print("ROC\_AUC: %0.2f (+/- %0.2f)" % (roc\_auc.mean(), roc\_auc.std() \* 2))

# define a function that accepts a threshold and prints sensitivity and specificity

def evaluate\_threshold(threshold):

print('Sensitivity:', tpr[thresholds > threshold][-1])

print('Specificity:', 1 - fpr[thresholds > threshold][-1])

evaluate\_threshold(0.3)

evaluate\_threshold(0.5)

train\_X,test\_X,train\_y,test\_y=train\_test\_split(X,y,test\_size=0.2)

train\_X.shape,test\_X.shape,train\_y.shape,test\_y.shape

from sklearn.metrics import make\_scorer

from sklearn.model\_selection import cross\_validate

def tn(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[0, 0]

def fp(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[0, 1]

def fn(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[1, 0]

def tp(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[1, 1]

#cross validation purpose

scoring = {'accuracy': make\_scorer(accuracy\_score),'prec': 'precision'}

scoring = {'tp': make\_scorer(tp), 'tn': make\_scorer(tn),

'fp': make\_scorer(fp), 'fn': make\_scorer(fn)}

def display\_result(result):

print("TP: ",result['test\_tp'])

print("TN: ",result['test\_tn'])

print("FN: ",result['test\_fn'])

print("FP: ",result['test\_fp'])

acc=[]

roc=[]

clf=GaussianNB()

clf.fit(train\_X,train\_y)

y\_pred=clf.predict(test\_X)

#find accuracy

ac=accuracy\_score(test\_y,y\_pred)

acc.append(ac)

#find the ROC\_AOC curve

rc=roc\_auc\_score(test\_y,y\_pred)

roc.append(rc)

print("\nAccuracy {0} ROC {1}".format(ac,rc))

#cross val score

result=cross\_validate(clf,train\_X,train\_y,scoring=scoring,cv=10)

display\_result(result)

pd.DataFrame(data={'Actual':test\_y,'Predicted':y\_pred}).head()

tp = np.append(result["test\_tp"], sum(result["test\_tp"])/10)

tn = np.append(result["test\_tn"], sum(result["test\_tn"])/10)

fp = np.append(result["test\_fp"], sum(result["test\_fp"])/10)

fn = np.append(result["test\_fn"], sum(result["test\_fn"])/10)

acc = np.append(accuracy,accuracy.mean())

recc = np.append(recall , recall.mean())

prec = np.append(precision,precision.mean())

bal\_acc = np.append(balanced\_accuracy,balanced\_accuracy.mean())

f1\_score = np.append(f1,f1.mean())

roc = np.append(roc\_auc,roc\_auc.mean())

data = { 'TP':tp,

'TN':tn,

'FP':fp,

'FN':fn,

'Accuracy':acc,

'Recall/Sensitivity':recc,

'Precision':prec,

'Balanced Accuracy':bal\_acc,

'f1\_score':f1\_score,

'ROC\_AUC':roc

}

df = pd.DataFrame(data,index = ['fold 1','fold 2','fold 3','fold 4','fold 5','fold 6','fold 7','fold 8','fold 9','fold 10','avg'])

df

# result

# K- Nearest Neighbour Classification

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=5, weights='uniform', algorithm='auto', leaf\_size=30, p=2, metric='minkowski', metric\_params=None, n\_jobs=None)

# Train / Test Split

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify = y, random\_state = 0)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy {}".format(accuracy))

# K-Fold Cross Validation

accuracy = cross\_val\_score(knn, X, y, cv = 10, scoring='accuracy')

accuracy.mean()

print("Accuracy {}".format(accuracy))

print("Accuracy: %0.2f (+/- %0.2f)" % (accuracy.mean(), accuracy.std() \* 2))

# Confusion Matrix

from sklearn.metrics import confusion\_matrix

# Method to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.rcParams["figure.figsize"] = (35,30)

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.show()

confusion = confusion\_matrix(y\_test, y\_pred)

print(confusion)

plot\_confusion\_matrix(confusion, classes=['Non Diabetic', 'Diabetic'], title='Confusion matrix')

# Values of TP, TN, FP, FN based on confusion matrix

# True Positives

TP = confusion[1, 1]

print("The value of TP is :" + str(TP))

# True Negatives

TN = confusion[0, 0]

print("The value of TN is :" + str(TN))

# False Positives

FP = confusion[0, 1]

print("The value of FP is :" + str(FP))

# False Negatives

FN = confusion[1, 0]

print("The value of FN is :" + str(FN))

# Number of Positive examples

P = (TP + FN)

print(P)

# Number of Negative Examples

N = (TN + FP)

print(N)

# Evaluating various metrics based on Confusion Matrix

from sklearn.metrics import recall\_score, precision\_score

# Classification Accuracy

print((TP + TN) / float(TP + TN + FP + FN))

print(accuracy\_score(y\_test, y\_pred))

accuracy = cross\_val\_score(knn, X, y, cv = 10, scoring='accuracy')

accuracy.mean()

print("Accuracy {}".format(accuracy))

print("Accuracy: %0.2f (+/- %0.2f)" % (accuracy.mean(), accuracy.std() \* 2))

# Sensitivity or TPR (True Positive Rate)

print(TP / float(TP + FN))

print(recall\_score(y\_test, y\_pred))

recall = cross\_val\_score(knn, X, y, cv = 10, scoring='recall')

recall.mean()

print("Recall {}".format(recall))

print("Recall: %0.2f (+/- %0.2f)" % (recall.mean(), recall.std() \* 2))

# Specificity or TNR (True Negative Rate)

print(TN / float(TN + FP))

# False Positive Rate

FPR = (FP / (TN + FP))

print(FPR)

# False Negative Rate

FNR = ((FN) / (TP + FN))

print(FNR)

# Precision

print(TP / float(TP + FP))

print(precision\_score(y\_test, y\_pred))

precision = cross\_val\_score(knn, X, y, cv = 10, scoring='precision')

precision.mean()

print("Precision {}".format(precision))

print("Precision: %0.2f (+/- %0.2f)" % (precision.mean(), precision.std() \* 2))

# Heidke Skill Score

HSS = 2 \* (TP \* TN - FP \* FN) / ((TP + FN) \* (FN + TN) + (TP + FP) \* (FP + TN))

print (HSS)

# Balanced Accuracy (BACC)

BACC = 1/2 \* ((TP / (TP + FN)) + ((TN) / (FP + TN)))

print(BACC)

balanced\_accuracy = cross\_val\_score(knn, X, y, cv = 10, scoring='balanced\_accuracy')

balanced\_accuracy.mean()

print("Balanced\_accuracy {}".format(balanced\_accuracy))

print("Balanced\_Accuracy: %0.2f (+/- %0.2f)" % (balanced\_accuracy.mean(), balanced\_accuracy.std() \* 2))

# True Skill Statistics

TSS = (((TP) / (TP + FN)) - ((FP) / (FP + TN)) )

print(TSS)

# F1 measure (F1)

F1 = ((2 \* TP)/ (2 \* (TP + FP + FN)))

print(F1)

f1 = cross\_val\_score(knn, X, y, cv = 10, scoring='f1')

balanced\_accuracy.mean()

print("F1\_score {}".format(f1))

print("F1\_score: %0.2f (+/- %0.2f)" % (f1.mean(), f1.std() \* 2))

# Error Rate

errorrate = ((FP + FN) / (TP + FP + TN + FN))

print(errorrate)

# Negative Predicted Value

NPV = (TN / (TN + FN))

print(NPV)

# False Discovery Rate

FDR = (FP / (FP + TP))

print(FDR)

# Handling the Classification Threshold

# print the first 10 predicted responses

knn.predict(X\_test)[0:10]

# print the first 10 predicted probabilities of class membership

knn.predict\_proba(X\_test)[0:10, :]

# store the predicted probabilities for class 1 (diabetic)

y\_pred\_prob = knn.predict\_proba(X\_test)[:, 1]

# histogram of predicted probabilities

plt.hist(y\_pred\_prob, bins=8, linewidth=1.2)

plt.xlim(0, 1)

plt.title('Histogram of predicted probabilities')

plt.xlabel('Predicted probability of diabetes')

plt.ylabel('Frequency')

# predict diabetes if the predicted probability is greater than 0.3

from sklearn.preprocessing import binarize

y\_pred\_class = binarize([y\_pred\_prob], 0.3)[0]

# previous confusion matrix (default threshold of 0.5)

print(confusion)

# new confusion matrix (threshold of 0.3)

confusion\_new = confusion\_matrix(y\_test, y\_pred\_class)

print(confusion\_new)

# Values of TP, TN, FP, FN based on new confusion matrix

# True Positives

TP = confusion\_new[1, 1]

print("The value of TP is :" + str(TP))

# True Negatives

TN = confusion\_new[0, 0]

print("The value of TN is :" + str(TN))

# False Positives

FP = confusion\_new[0, 1]

print("The value of FP is :" + str(FP))

# False Negatives

FN = confusion\_new[1, 0]

print("The value of FN is :" + str(FN))

# We observe that the sensitivity has increased

print(TP / float(TP + FN))

print(recall\_score(y\_test, y\_pred\_class))

# specificity has decreased

print(TN / float(TN + FP))

# ROC and AUC (Area Under the Curves)

from sklearn.metrics import roc\_curve, roc\_auc\_score

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)

plt.plot(fpr, tpr)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.title('ROC curve for diabetes classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.grid(True)

roc\_auc = cross\_val\_score(knn, X, y, cv = 10, scoring='roc\_auc')

roc\_auc.mean()

print("roc\_auc {}".format(roc\_auc))

print("ROC\_AUC: %0.2f (+/- %0.2f)" % (roc\_auc.mean(), roc\_auc.std() \* 2))

# define a function that accepts a threshold and prints sensitivity and specificity

def evaluate\_threshold(threshold):

print('Sensitivity:', tpr[thresholds > threshold][-1])

print('Specificity:', 1 - fpr[thresholds > threshold][-1])

evaluate\_threshold(0.3)

evaluate\_threshold(0.5)

train\_X,test\_X,train\_y,test\_y=train\_test\_split(X,y,test\_size=0.2)

train\_X.shape,test\_X.shape,train\_y.shape,test\_y.shape

from sklearn.metrics import make\_scorer

from sklearn.model\_selection import cross\_validate

def tn(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[0, 0]

def fp(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[0, 1]

def fn(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[1, 0]

def tp(y\_true, y\_pred): return confusion\_matrix(y\_true, y\_pred)[1, 1]

#cross validation purpose

scoring = {'accuracy': make\_scorer(accuracy\_score),'prec': 'precision'}

scoring = {'tp': make\_scorer(tp), 'tn': make\_scorer(tn),

'fp': make\_scorer(fp), 'fn': make\_scorer(fn)}

def display\_result(result):

print("TP: ",result['test\_tp'])

print("TN: ",result['test\_tn'])

print("FN: ",result['test\_fn'])

print("FP: ",result['test\_fp'])

acc=[]

roc=[]

clf=KNeighborsClassifier ()

clf.fit(train\_X,train\_y)

y\_pred=clf.predict(test\_X)

#find accuracy

ac=accuracy\_score(test\_y,y\_pred)

acc.append(ac)

#find the ROC\_AOC curve

rc=roc\_auc\_score(test\_y,y\_pred)

roc.append(rc)

print("\nAccuracy {0} ROC {1}".format(ac,rc))

#cross val score

result=cross\_validate(clf,train\_X,train\_y,scoring=scoring,cv=10)

display\_result(result)

pd.DataFrame(data={'Actual':test\_y,'Predicted':y\_pred}).head()

tp = np.append(result["test\_tp"], sum(result["test\_tp"])/10)

tn = np.append(result["test\_tn"], sum(result["test\_tn"])/10)

fp = np.append(result["test\_fp"], sum(result["test\_fp"])/10)

fn = np.append(result["test\_fn"], sum(result["test\_fn"])/10)

acc = np.append(accuracy,accuracy.mean())

recc = np.append(recall , recall.mean())

prec = np.append(precision,precision.mean())

bal\_acc = np.append(balanced\_accuracy,balanced\_accuracy.mean())

f1\_score = np.append(f1,f1.mean())

roc = np.append(roc\_auc,roc\_auc.mean())

data = { 'TP':tp,

'TN':tn,

'FP':fp,

'FN':fn,

'Accuracy':acc,

'Recall/Sensitivity':recc,

'Precision':prec,

'Balanced Accuracy':bal\_acc,

'f1\_score':f1\_score,

'ROC\_AUC':roc

}

df = pd.DataFrame(data,index = ['fold 1','fold 2','fold 3','fold 4','fold 5','fold 6','fold 7','fold 8','fold 9','fold 10','avg'])

df

# result

The following is the link to my GitHub Page where one can find full implementation of my project along with the desired outputs:

<https://github.com/TanishBugnait29/Supervised-Data-Mining-Classification>