ILLINOIS INSTITUTE OF TECHNOLOGY

Credit Card Defaulters Detection Using Machine Learning

Harshini Kurupati
A20480932
(hkurupati@hawk.iit.edu)

Vismaya Mohanram
A20519405
(vm@hawk.iit.edu)

Yasha Srinivas
A20516915
(ysrinivasa@hawk.iit.edu)

May 1st 2023

Prof. Yan Yan
Machine Learning - CS 584

INTRODUCTION

Credit card businesses must forecast the possibility of clients failing on their payments. The repercussions of being unable to make correct forecasts can be financially devastating, with possible losses up to millions of dollars. Machine learning models may be trained on past data to forecast if a consumer is going to fail on their credit card payments.

In this project, we will create a machine learning model that can reliably predict if a consumer will default on their payments using a dataset of credit card users and their payment habits. Each card holder's demographics, credit history, and payment status are all included in the dataset. To discover the best model for predicting credit card defaults, we will utilize this data to train and evaluate several machine learning methods, such as logistic regression, decision trees, and random forests.

This research will result in a machine learning model that can forecast the possibility of a consumer failing on credit card payments. Credit card firms can use this model to proactively identify high-risk consumers and take the appropriate steps to prevent losses.

PROBLEM DESCRIPTION

The identification of credit card holders who are likely to default on their payments is the challenge we are attempting to tackle in this project. Credit card defaults can cause considerable financial losses for credit card businesses, and identifying high-risk consumers ahead of time is crucial to minimizing these losses.

To solve this issue, we will use a dataset comprising information on credit card holders' demographic traits, credit history, and payment activity. This information will be used to train and test several machine learning models that will forecast whether a consumer is likely to miss a credit card payment.

The goal of this project is to discover the main parameters that are most predictive of credit card defaults and to create a machine learning model that can forecast the chance of default accurately. Furthermore, we must guarantee that the model is resilient and dependable, as well as that it can generalize effectively to new data.

This research will result in a machine learning model that can properly forecast the likelihood of credit card defaults, which credit card firms may use to proactively identify high-risk consumers and take the required steps to avert losses. This might involve reaching out to clients in a targeted manner, altering credit limits or interest rates, or even suspending credit card accounts to avoid more losses.

When calculating credit card defaults, false positives should be minimised because if a transaction is misclassified as fraud when it is not, the bank's reputation suffers and clients lose faith in that financial institution. False negatives, likewise, should not be utilized to misclassify a fraudulent transaction as non-fraudulent. For the outcome analysis, we will compute many evaluation metrics such as accuracy, F1-score, precision, sensitivity, and specificity, and their percentage values will be compared.

The proposed model will then be compared against Decision Tree (DT), Logistic Regression (LR), K-Nearest Neighbor (KNN), Random Forest (RF), and Adaboost models, and it will be noticed that the predicted accuracy value is higher when compared to the individual models.

PREVIOUS WORK

Several research and analysis have been undertaken on the supplied credit card default dataset. Here are some past works done on this dataset:

- The credit card default dataset was used in a 2011 study published in the Journal of customer Affairs to evaluate the influence of disclosure requirements on customer behavior. They discovered that consumers who were given more extensive information about credit card conditions and fees were more likely to make educated credit card selections and avoid default.
- A 2015 research published in the Journal of Financial Planning examined the impact of credit counseling in lowering credit card debt and default using the credit card default dataset. Credit counseling, the authors discovered, may be a useful intervention for those battling with credit card debt, but long-term success required regular assistance and financial education.
- The credit card default dataset was utilized in a 2018 study published in the Journal of Financial Counseling and Planning to evaluate the influence of financial stress on credit card use and default among low-income families. Financial stress was discovered to be a strong predictor of credit card default, and financial education and counseling can assist individuals in managing their debt and avoiding default.

DATASET

Link: https://raw.githubusercontent.com/MLWave/Black-Boxxy/master/credit-card-default.csv

From April 2005 to September 2005, this dataset comprises information on default payments, demographic variables, credit data, payment history, and bill statements for credit card clients in Taiwan. The dataset utilized in this research provides information on credit card users' credit histories and payment behavior, and it may be used to train a machine learning model to predict if a consumer is going to fail on credit card payments.

The dataset has 30,000 rows and 24 columns in total. Each row represents a distinct credit card customer, while the columns include their demographics, credit history, and payment habits.

The columns of the dataset are described as follows:

1. **ID**: Unique identifier for each credit card holder
2. **LIMIT_BAL**: The credit limit given to each card holder
3. **SEX**: Gender of the card holder (1 = male, 2 = female)
4. **EDUCATION**: Education level of the card holder (1 = graduate school, 2 = university, 3 = high school, 4 = others)
5. **MARRIAGE**: Marital status of the card holder (1 = married, 2 = single, 3 = others)
6. **AGE**: Age of the card holder in years
7. **PAY_0** to **PAY_6**: Repayment status of the card holder for the past 6 months. Each column represents a month, and the values range from -1 to 9, where -1 = pay duly, 1 = payment delay for one month, 2 = payment delay for two months, and so on.
8. **BILL_AMT1** to **BILL_AMT6**: Amount of the bill statement for the past 6 months. Each column represents a month.
9. **PAY_AMT1** to **PAY_AMT6**: Amount of previous payment made for the past 6 months. Each column represents a month.
10. **DEFAULT PAYMENT NEXT MONTH**: Indicates whether the card holder defaulted on their payment or not. 1 = Yes (defaulted), 0 = No (did not default).

To manage any missing or erroneous data, the dataset will be preprocessed, and feature engineering techniques will be used to extract the most important features. To determine the optimal model for predicting credit card defaults, the preprocessed data will be used to train and evaluate several machine learning techniques, such as logistic regression, decision trees, and random forests.

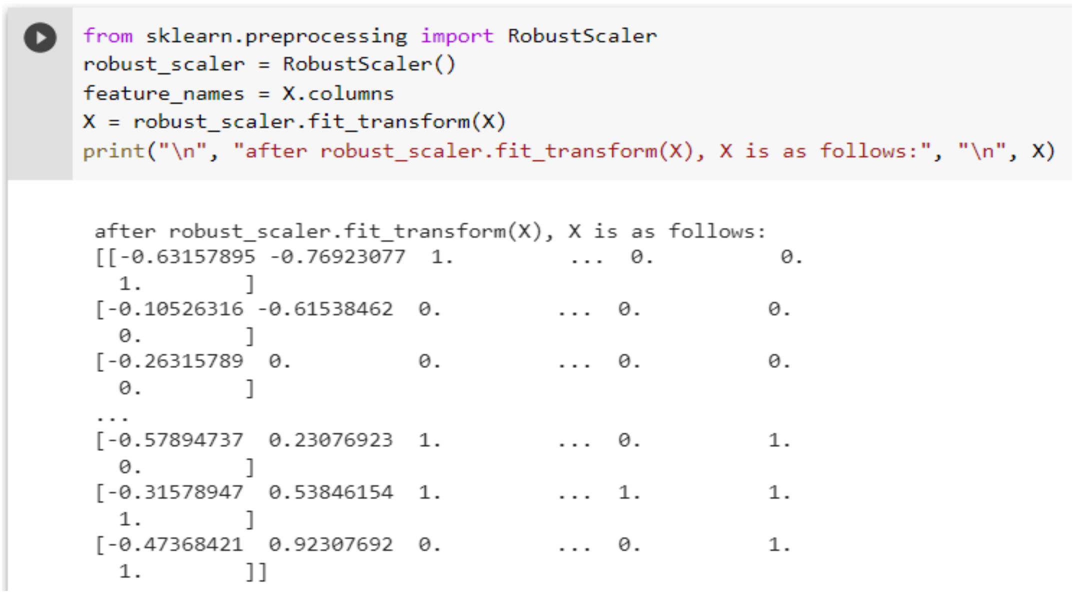
APPLICATION: DETAILED DESCRIPTION OF METHODS USED

The following steps were followed to create a prediction model for the credit card default dataset:

1. **Loading the dataset:** We began by loading the dataset into our programming environment, Google Collab. To read the CSV file into a pandas DataFrame, we utilized Python's pandas package.

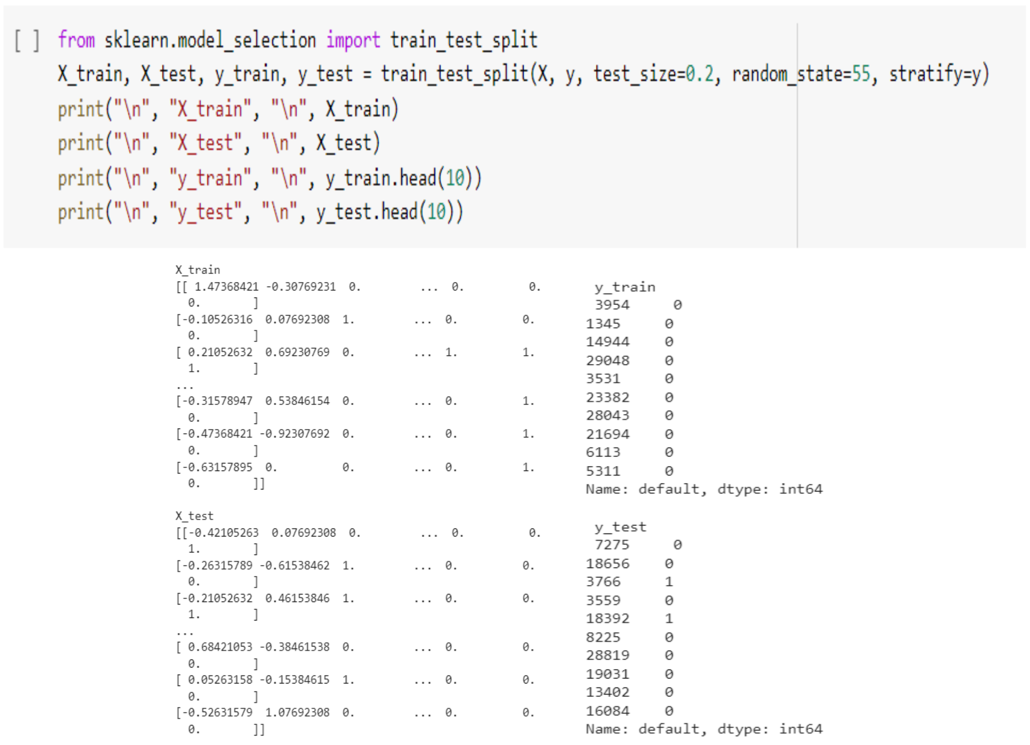
2. **Exploring the dataset:** Before analyzing the data, we explored the dataset to gain a sense of what it included. To get a summary of the data, we utilized pandas functions like head(), tail(), info(), describe(), value counts(), and so on.

3. **Preprocessing the data:** After exploring the data, we preprocessed it to prepare it for analysis. Missing values were handled, categorical variables were encoded, and numeric characteristics were scaled. To obtain the robust estimates of the data's center and scale, we developed a RobustScaler object and fitted it to the feature matrix X. Lastly, we use the fitted scaler to modify the feature matrix X to produce scaled copies of the features. By dividing the dataset into training and testing sets, we can evaluate the model's performance on previously unknown data. We guarantee that the model is trained and evaluated on representative samples of each class by stratifying the split based on the target variable.



```
from sklearn.preprocessing import RobustScaler
robust_scaler = RobustScaler()
feature_names = X.columns
X = robust_scaler.fit_transform(X)
print("\n", "after robust_scaler.fit_transform(X), X is as follows:", "\n", X)
```

```
after robust_scaler.fit_transform(X), X is as follows:
[[-0.63157895 -0.76923077  1.         ...  0.          0.
   1.        ]
 [-0.10526316 -0.61538462  0.         ...  0.          0.
   0.        ]
 [-0.26315789  0.          0.         ...  0.          0.
   0.        ]
 ...
 [-0.57894737  0.23076923  1.         ...  0.          1.
   0.        ]
 [-0.31578947  0.53846154  1.         ...  1.          1.
   1.        ]
 [-0.47368421  0.92307692  0.         ...  0.          1.
   1.        ]]
```

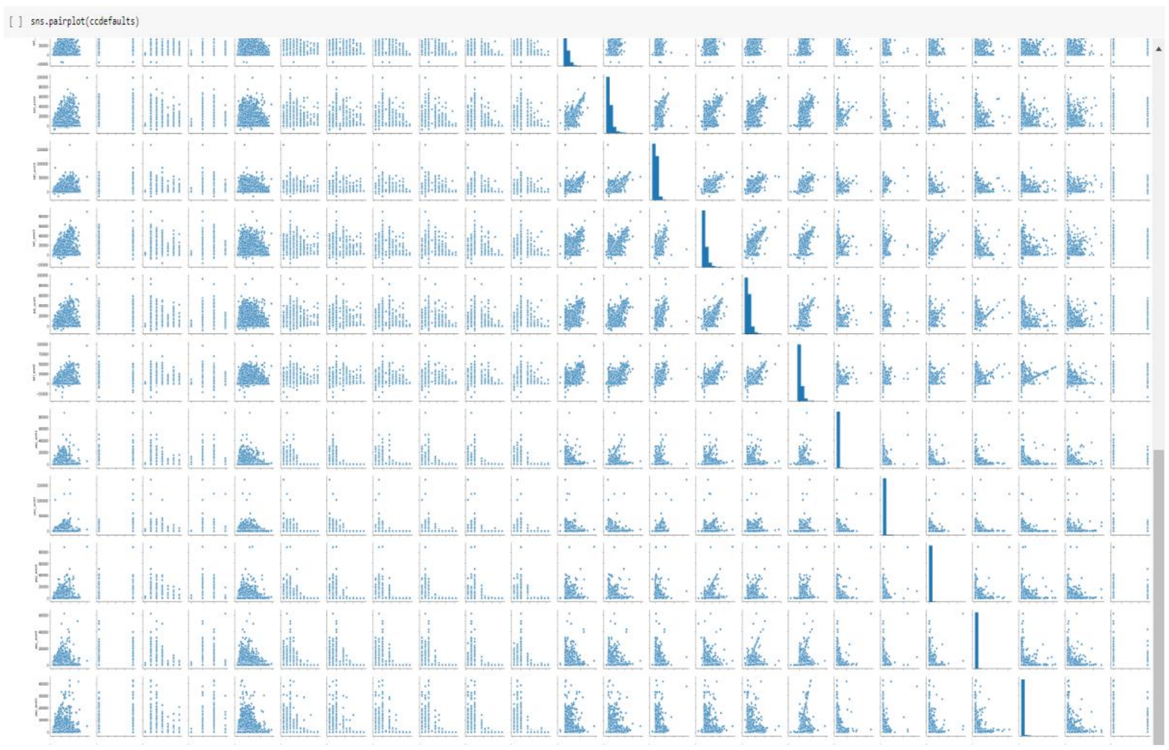


```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=55, stratify=y)
    print("\n", "X_train", "\n", X_train)
    print("\n", "X_test", "\n", X_test)
    print("\n", "y_train", "\n", y_train.head(10))
    print("\n", "y_test", "\n", y_test.head(10))
```

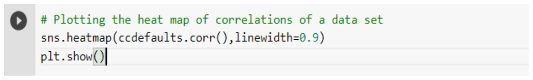
```
X_train
[[ 1.47368421 -0.30769231  0.         ...  0.          0.
   0.        ]
 [-0.10526316  0.07692308  1.         ...  0.          0.
   0.        ]
 [ 0.21052632  0.69230769  0.         ...  1.          1.
   1.        ]
 ...
 [-0.31578947  0.53846154  0.         ...  0.          1.
   0.        ]
 [-0.47368421 -0.92307692  0.         ...  0.          1.
   0.        ]
 [-0.63157895  0.          0.         ...  0.          1.
   0.        ]]

X_test
[[-0.42105263  0.07692308  0.         ...  0.          0.
   1.        ]
 [-0.26315789 -0.61538462  1.         ...  0.          0.
   0.        ]
 [-0.21052632  0.46153846  1.         ...  0.          0.
   1.        ]
 ...
 [ 0.68421053 -0.38461538  0.         ...  0.          0.
   0.        ]
 [ 0.05263158 -0.15384615  1.         ...  0.          0.
   0.        ]
 [-0.52631579  1.07692308  0.         ...  0.          0.
   0.        ]]
```

```
 y_train
 3954     0
1345     0
14944    0
29048    0
3531     0
23382    0
28043    0
21694    0
6113     0
5311     0
Name: default, dtype: int64

 y_test
 7275     0
18656    0
3766     1
3559     0
18392    1
8225     0
28819    0
19031    0
13402    0
16084    0
Name: default, dtype: int64
```

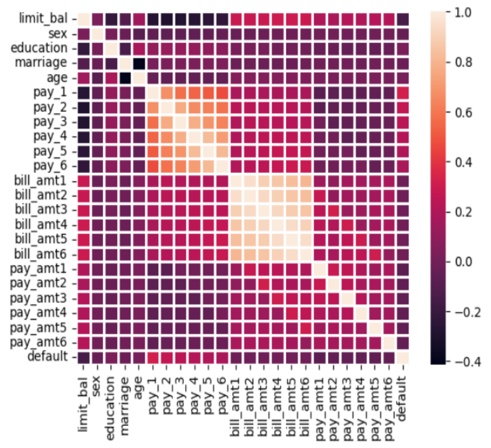
4. **Visualizing the data:** Data visualization is a strong tool for extracting insights from data. We created data visualizations such as histograms, scatterplots, and heatmaps using tools such as Matplotlib or Seaborn.



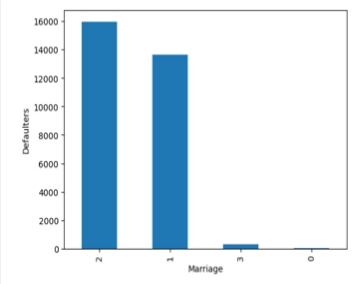
```
[ ] sns.pairplot(ccdefaults)
```



```
# Plotting the heat map of correlations of a data set
sns.heatmap(ccdefaults.corr(),linewidth=0.9)
plt.show()
```

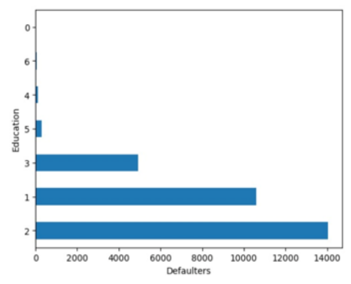


This heatmap shows the correlation coefficients between all pairs of variables in the credit card defaulters' dataset.



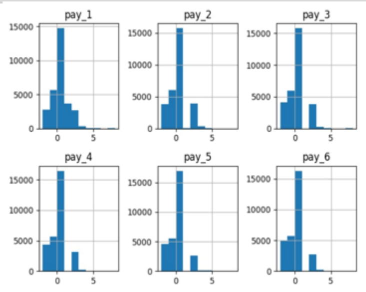
By looking at the plot, we can see that the majority of the defaulters belong to the 'Single' category, followed by the 'Married' category.

The 'Others' category has the least number of defaulters.



By looking at the plot, we can see that the majority of the defaulters belong to the 'university' category, followed by the 'graduate school' category.

The 'high school' category has a lower number of defaulters, while the 'others' category has the least number of defaulters



By looking at the plot, we can see that the majority of borrowers had a payment status of 0, which means they made a full payment on time.

However, there is also a significant number of borrowers who had a payment status of -1, -2, or higher, which means they made a late payment or missed a payment.

5. **Building a predictive model and evaluating the model:** After preprocessing and visualizing the data, we are creating a predictive model to predict if an individual would fail on their credit card payments. To develop the model, we use various Machine Learning methods such as Decision Tree (DT), Logistic Regression (LR), K-Nearest Neighbor (KNN), Random Forest (RF), and Adaboost models. We analyze the model's performance once it has been built to determine how well it predicts credit card defaults. To assess the model's performance, we employ measures such as accuracy, precision, sensitivity, specificity, and F1-score.

6. **Ensemble Learning using Voting Classifier:** We have used an ensemble learning voting technique to combine the predictions of the various Machine Learning models (Logistic Regression, Decision Tree Classifier, K-Nearest Neighbors Classifier, Adaboost Classifier, and Random Forest Classifier) and created a final prediction for credit card defaults, and have evaluated the combined model's performance using various metrics.

7. **Interpret the results:** Lastly, we have evaluated the findings of the study to acquire a better understanding of the factors that lead to credit card defaults. This entails us to assess the prediction model's feature importance’s and for developing data visualizations to discover patterns and trends.

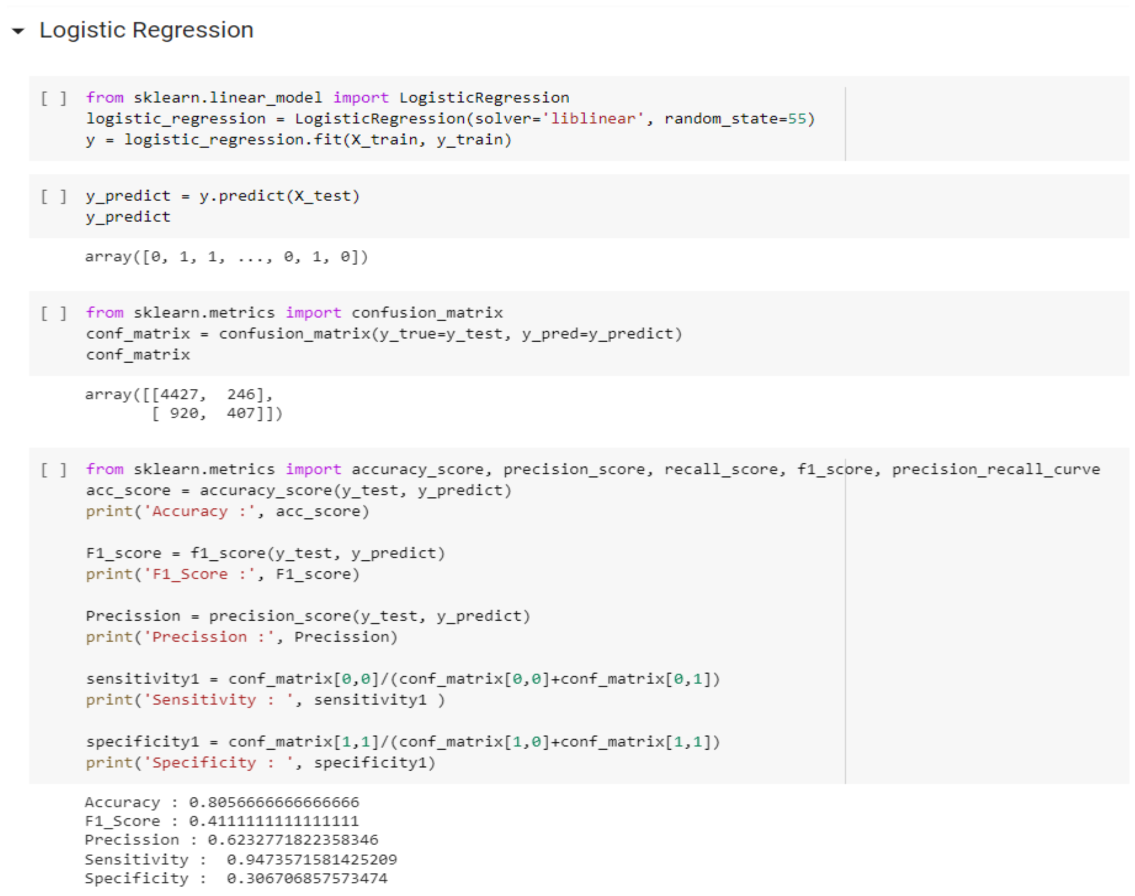
MODEL IMPLEMENTATION AND ANALYSIS

To start off with the model training, we must first split the data into training and testing. For this, we have used the 80-20 train-test split.

Based on the selected features, we have applied various Machine Learning Classification algorithms to check the model's accuracy. The algorithms implemented and their performance are as below:

➢ LOGISTIC REGRESSION

Logistic Regression is a statistical algorithm used for binary classification problems where the outcome variable is binary (i.e., has only two possible values) and the algorithm models the probability of the outcome variable taking one of the two values based on one or more predictor variables.



▾ Logistic Regression

```
[ ] from sklearn.linear_model import LogisticRegression
    logistic_regression = LogisticRegression(solver='liblinear', random_state=55)
    y = logistic_regression.fit(X_train, y_train)
```

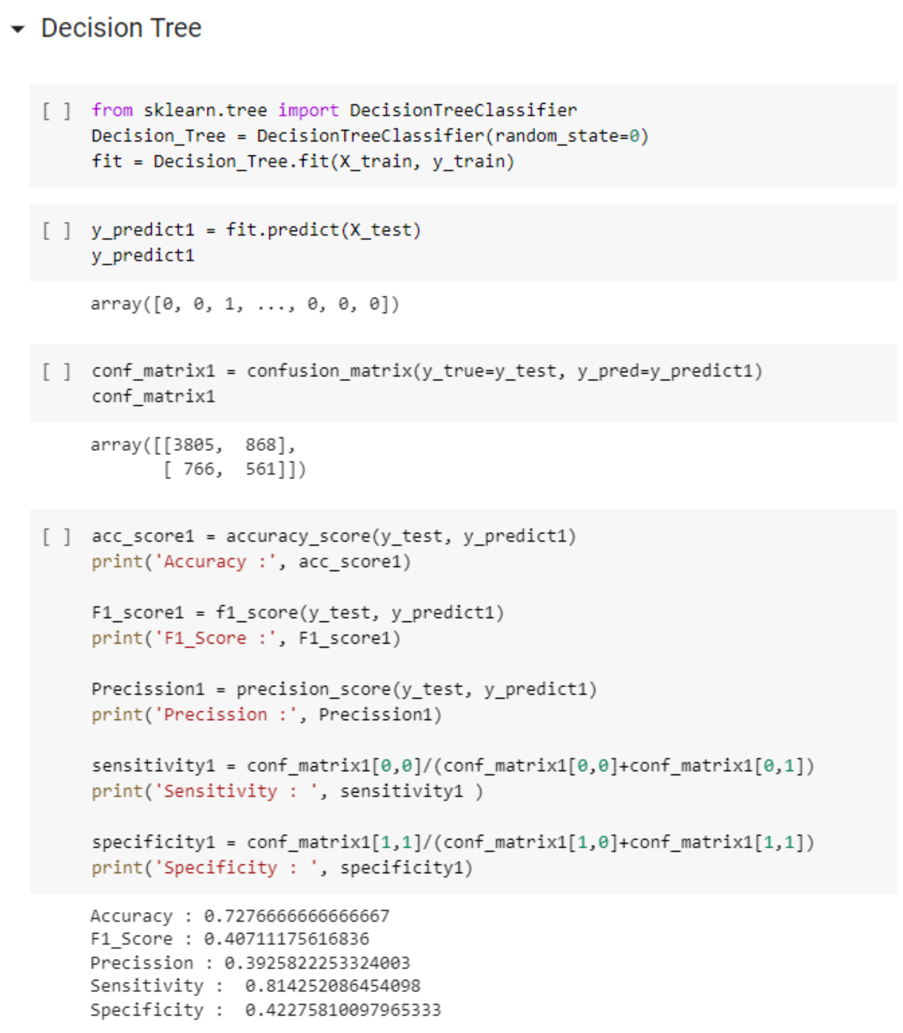
```
[ ] y_predict = y.predict(X_test)
    y_predict

    array([0, 1, 1, ..., 0, 1, 0])
```

```
[ ] from sklearn.metrics import confusion_matrix
    conf_matrix = confusion_matrix(y_true=y_test, y_pred=y_predict)
    conf_matrix

    array([[4427,  246],
           [ 920,  407]])
```

```
[ ] from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, precision_recall_curve
    acc_score = accuracy_score(y_test, y_predict)
    print('Accuracy :', acc_score)

    F1_score = f1_score(y_test, y_predict)
    print('F1_Score :', F1_score)

    Precission = precision_score(y_test, y_predict)
    print('Precission :', Precission)

    sensitivity1 = conf_matrix[0,0]/(conf_matrix[0,0]+conf_matrix[0,1])
    print('Sensitivity : ', sensitivity1 )

    specificity1 = conf_matrix[1,1]/(conf_matrix[1,0]+conf_matrix[1,1])
    print('Specificity : ', specificity1)

    Accuracy : 0.8056666666666666
    F1_Score : 0.4111111111111111
    Precission : 0.6232771822358346
    Sensitivity :  0.9473571581425209
    Specificity :  0.306706857573474
```

➢ DECISION TREE

Decision Tree is a non-parametric algorithm used for both regression and classification problems that creates a model in the form of a tree structure, where each internal node represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label or a numeric value.



Decision Tree

```
[ ] from sklearn.tree import DecisionTreeClassifier
    Decision_Tree = DecisionTreeClassifier(random_state=0)
    fit = Decision_Tree.fit(X_train, y_train)
```

```
[ ] y_predict1 = fit.predict(X_test)
    y_predict1
```

```
array([0, 0, 1, ..., 0, 0, 0])
```

```
[ ] conf_matrix1 = confusion_matrix(y_true=y_test, y_pred=y_predict1)
    conf_matrix1
```

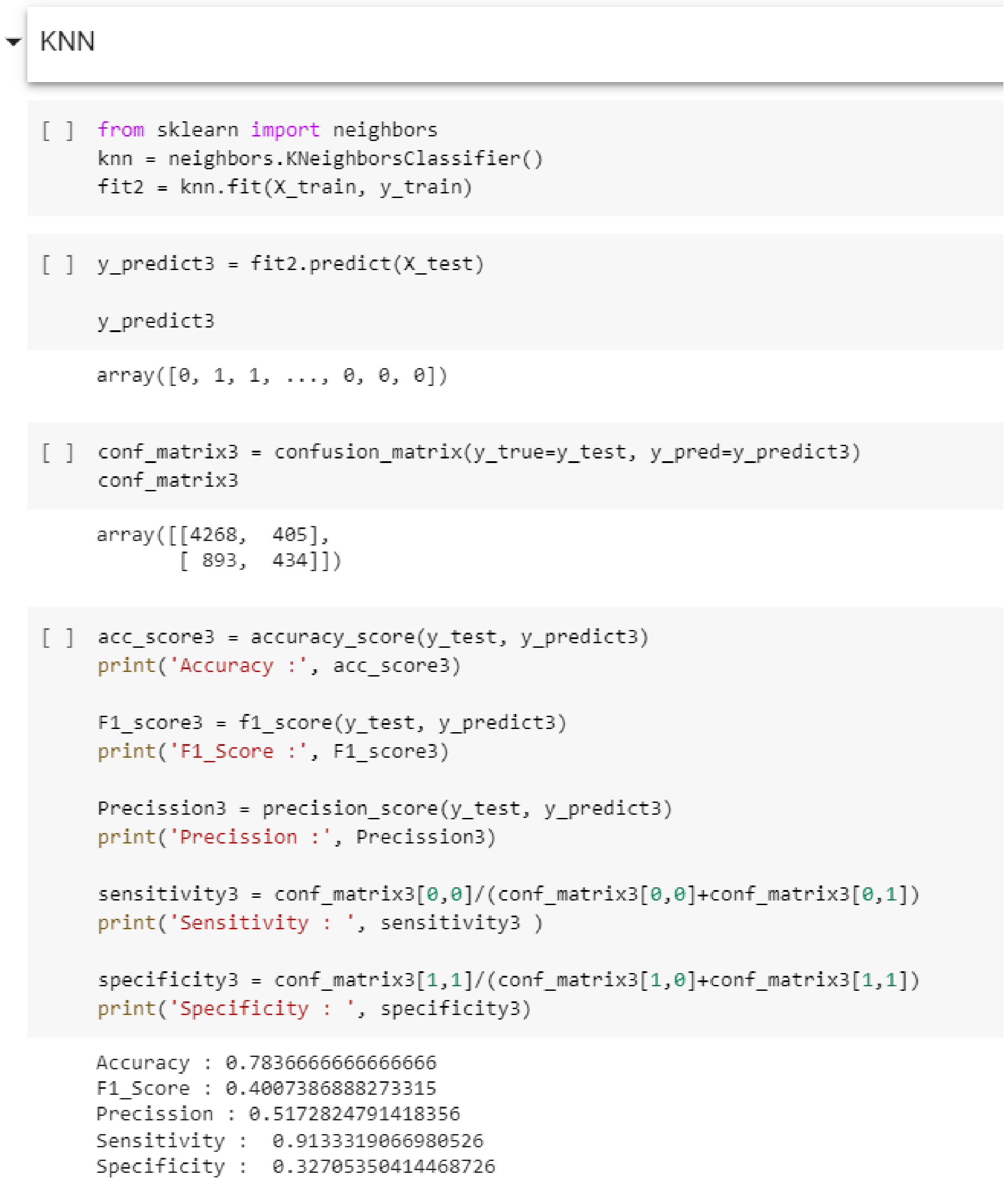
```
array([[3805,  868],
       [ 766,  561]])
```

```
[ ] acc_score1 = accuracy_score(y_test, y_predict1)
    print('Accuracy :', acc_score1)

    F1_score1 = f1_score(y_test, y_predict1)
    print('F1_Score :', F1_score1)

    Precission1 = precision_score(y_test, y_predict1)
    print('Precission :', Precission1)

    sensitivity1 = conf_matrix1[0,0]/(conf_matrix1[0,0]+conf_matrix1[0,1])
    print('Sensitivity : ', sensitivity1 )

    specificity1 = conf_matrix1[1,1]/(conf_matrix1[1,0]+conf_matrix1[1,1])
    print('Specificity : ', specificity1)
```

```
Accuracy : 0.7276666666666667
F1_Score : 0.40711175616836
Precission : 0.3925822253324003
Sensitivity :  0.814252086454098
Specificity :  0.42275810097965333
```

➢ K-NEAREST NEIGHBOR

K-Nearest Neighbors (KNN) is a supervised learning algorithm used for classification and regression problems, which finds the k closest training examples in the feature space to the new input data and predicts the output variable based on the majority vote of the k neighbors or the mean value of their output variables.



KNN

```
[ ] from sklearn import neighbors
    knn = neighbors.KNeighborsClassifier()
    fit2 = knn.fit(X_train, y_train)
```

```
[ ] y_predict3 = fit2.predict(X_test)

    y_predict3
```

```
array([0, 1, 1, ..., 0, 0, 0])
```

```
[ ] conf_matrix3 = confusion_matrix(y_true=y_test, y_pred=y_predict3)
    conf_matrix3
```

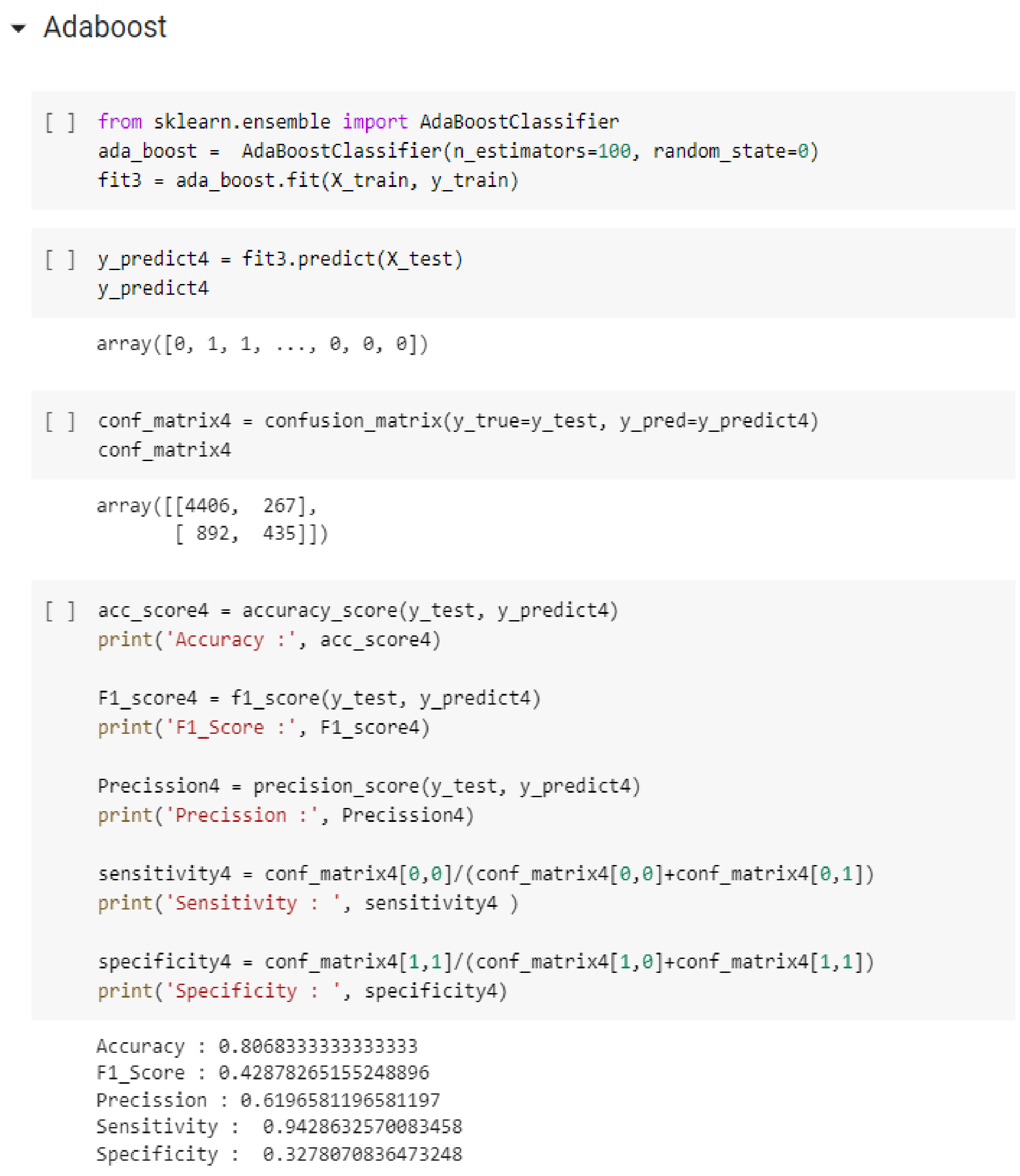
```
array([[4268,  405],
       [ 893,  434]])
```

```
[ ] acc_score3 = accuracy_score(y_test, y_predict3)
    print('Accuracy :', acc_score3)

    F1_score3 = f1_score(y_test, y_predict3)
    print('F1_Score :', F1_score3)

    Precission3 = precision_score(y_test, y_predict3)
    print('Precission :', Precission3)

    sensitivity3 = conf_matrix3[0,0]/(conf_matrix3[0,0]+conf_matrix3[0,1])
    print('Sensitivity : ', sensitivity3 )

    specificity3 = conf_matrix3[1,1]/(conf_matrix3[1,0]+conf_matrix3[1,1])
    print('Specificity : ', specificity3)
```

```
Accuracy : 0.7836666666666666
F1_Score : 0.4007386888273315
Precission : 0.5172824791418356
Sensitivity :  0.9133319066980526
Specificity :  0.32705350414468726
```

➢ <u>ADABOOST</u>

Adaboost (Adaptive Boosting) is a machine learning algorithm that combines multiple weak classifiers to form a strong classifier, where each weak classifier is trained on the misclassified examples from the previous classifier, and the final prediction is made based on a weighted sum of the weak classifiers' predictions.



Adaboost

```
[ ] from sklearn.ensemble import AdaBoostClassifier
    ada_boost =  AdaBoostClassifier(n_estimators=100, random_state=0)
    fit3 = ada_boost.fit(X_train, y_train)
```

```
[ ] y_predict4 = fit3.predict(X_test)
    y_predict4
```

```
array([0, 1, 1, ..., 0, 0, 0])
```

```
[ ] conf_matrix4 = confusion_matrix(y_true=y_test, y_pred=y_predict4)
    conf_matrix4
```

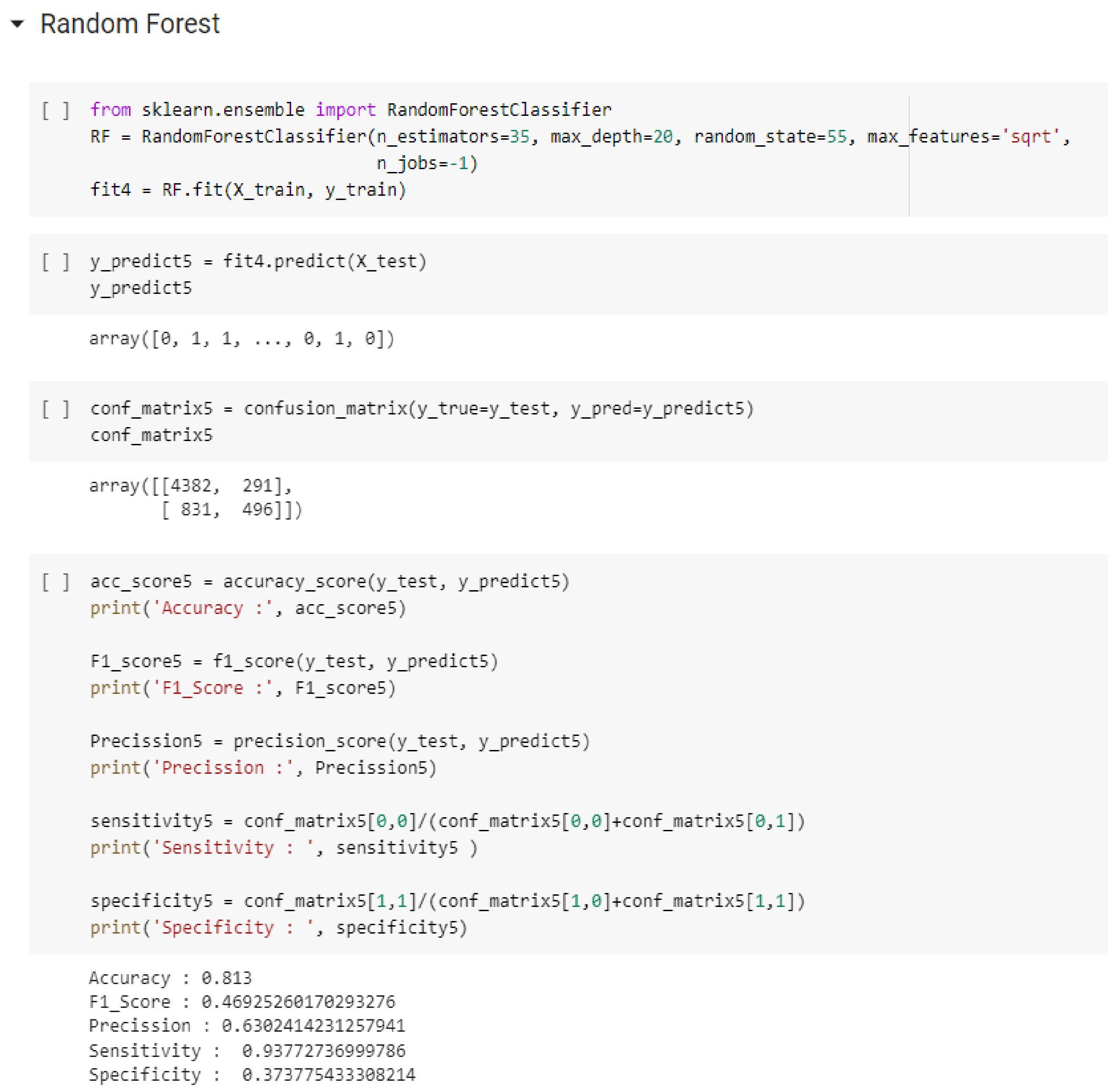
```
array([[4406,  267],
       [ 892,  435]])
```

```
[ ] acc_score4 = accuracy_score(y_test, y_predict4)
    print('Accuracy :', acc_score4)

    F1_score4 = f1_score(y_test, y_predict4)
    print('F1_Score :', F1_score4)

    Precission4 = precision_score(y_test, y_predict4)
    print('Precission :', Precission4)

    sensitivity4 = conf_matrix4[0,0]/(conf_matrix4[0,0]+conf_matrix4[0,1])
    print('Sensitivity : ', sensitivity4 )

    specificity4 = conf_matrix4[1,1]/(conf_matrix4[1,0]+conf_matrix4[1,1])
    print('Specificity : ', specificity4)
```

```
Accuracy : 0.8068333333333333
F1_Score : 0.42878265155248896
Precission : 0.6196581196581197
Sensitivity :  0.9428632570083458
Specificity :  0.3278070836473248
```

➢ RANDOM FOREST

Random Forest is a supervised learning algorithm used for classification and regression problems that creates an ensemble of decision trees and predicts the output variable based on the majority vote (classification) or the average (regression) of the individual trees' predictions.



▾ Random Forest

```
[ ]  from sklearn.ensemble import RandomForestClassifier
     RF = RandomForestClassifier(n_estimators=35, max_depth=20, random_state=55, max_features='sqrt',
                                 n_jobs=-1)
     fit4 = RF.fit(X_train, y_train)
```

```
[ ]  y_predict5 = fit4.predict(X_test)
     y_predict5
```

```
array([0, 1, 1, ..., 0, 1, 0])
```

```
[ ]  conf_matrix5 = confusion_matrix(y_true=y_test, y_pred=y_predict5)
     conf_matrix5
```

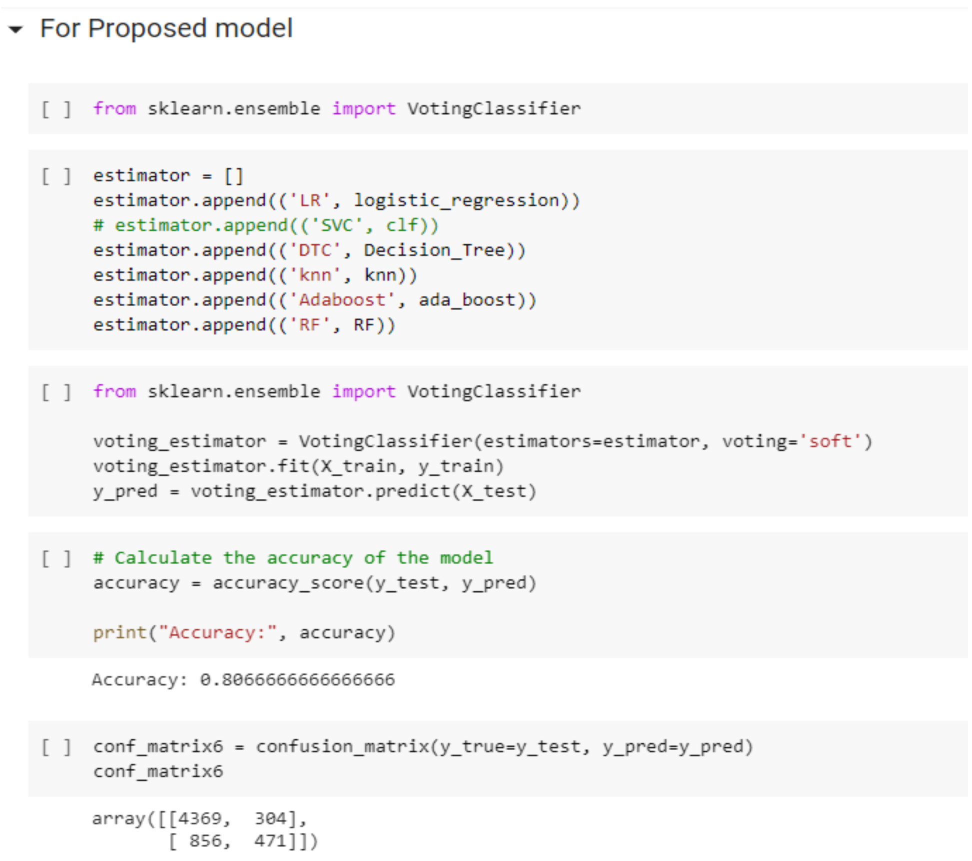
```
array([[4382,  291],
       [ 831,  496]])
```

```
[ ]  acc_score5 = accuracy_score(y_test, y_predict5)
     print('Accuracy :', acc_score5)

     F1_score5 = f1_score(y_test, y_predict5)
     print('F1_Score :', F1_score5)

     Precission5 = precision_score(y_test, y_predict5)
     print('Precission :', Precission5)

     sensitivity5 = conf_matrix5[0,0]/(conf_matrix5[0,0]+conf_matrix5[0,1])
     print('Sensitivity : ', sensitivity5 )

     specificity5 = conf_matrix5[1,1]/(conf_matrix5[1,0]+conf_matrix5[1,1])
     print('Specificity : ', specificity5)
```

```
Accuracy : 0.813
F1_Score : 0.46925260170293276
Precission : 0.6302414231257941
Sensitivity :  0.93772736999786
Specificity :  0.373775433308214
```

➢ **<u>PROPOSED MODEL USING VOTING CLASSIFIER</u>**

A voting classifier model in machine learning is an ensemble learning technique that combines the predictions of multiple base models to make a final prediction. It works by creating several base models using different algorithms or hyperparameters, and then aggregating their predictions through a majority vote or weighted average. There are two types of voting classifier models: hard voting and soft voting. In soft voting, the final prediction is based on the weighted average of the predicted probabilities of all individual models. This technique can be used to improve the overall performance of a single model, especially in cases where no single model performs well on its own, and can also help reduce overfitting.



For Proposed model

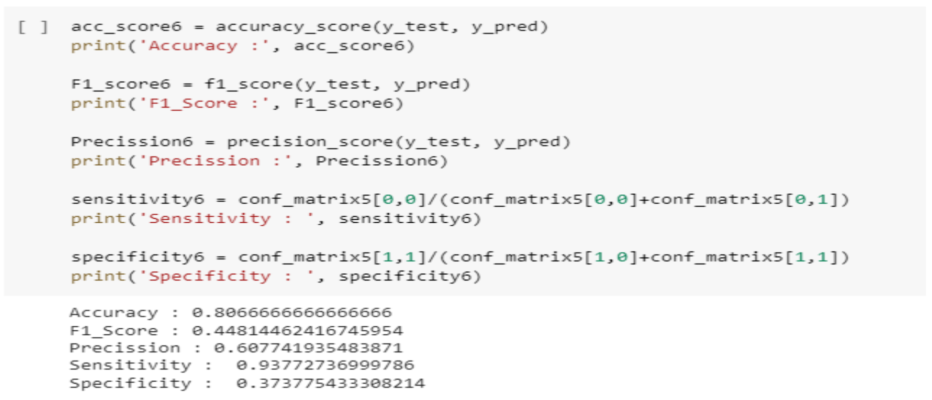
```
[ ]  from sklearn.ensemble import VotingClassifier
```

```
[ ]  estimator = []
     estimator.append(('LR', logistic_regression))
     # estimator.append(('SVC', clf))
     estimator.append(('DTC', Decision_Tree))
     estimator.append(('knn', knn))
     estimator.append(('Adaboost', ada_boost))
     estimator.append(('RF', RF))
```

```
[ ]  from sklearn.ensemble import VotingClassifier

     voting_estimator = VotingClassifier(estimators=estimator, voting='soft')
     voting_estimator.fit(X_train, y_train)
     y_pred = voting_estimator.predict(X_test)
```

```
[ ]  # Calculate the accuracy of the model
     accuracy = accuracy_score(y_test, y_pred)

     print("Accuracy:", accuracy)

     Accuracy: 0.8066666666666666
```

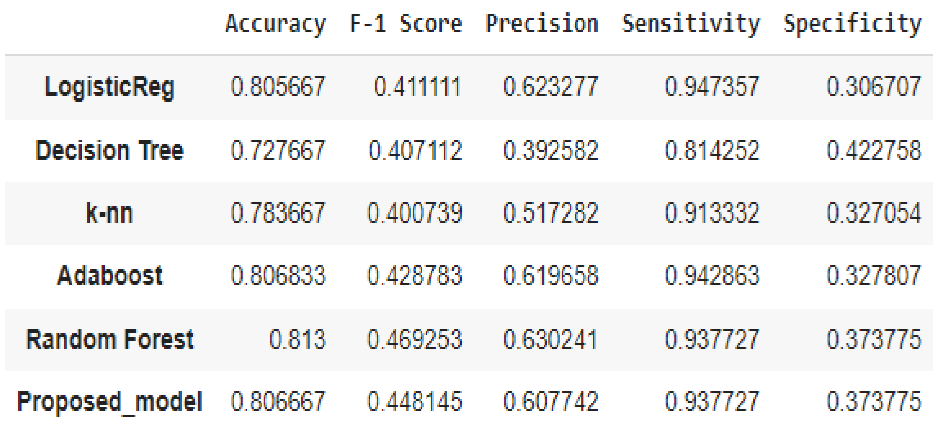
```
[ ]  conf_matrix6 = confusion_matrix(y_true=y_test, y_pred=y_pred)
     conf_matrix6

     array([[4369,  304],
            [ 856,  471]])
```



```
[ ]  acc_score6 = accuracy_score(y_test, y_pred)
     print('Accuracy :', acc_score6)

     F1_score6 = f1_score(y_test, y_pred)
     print('F1_Score :', F1_score6)

     Precission6 = precision_score(y_test, y_pred)
     print('Precission :', Precission6)

     sensitivity6 = conf_matrix5[0,0]/(conf_matrix5[0,0]+conf_matrix5[0,1])
     print('Sensitivity : ', sensitivity6)

     specificity6 = conf_matrix5[1,1]/(conf_matrix5[1,0]+conf_matrix5[1,1])
     print('Specificity : ', specificity6)

     Accuracy : 0.8066666666666666
     F1_Score : 0.44814462416745954
     Precission : 0.607741935483871
     Sensitivity :  0.93772736999786
     Specificity :  0.373775433308214
```

CONCLUSION

The below table shows the performance of six different machine learning models on a specific problem. The models are evaluated based on five performance metrics: accuracy, F-1 score, precision, sensitivity, and specificity. The models include Logistic Regression, Decision Tree, k-Nearest Neighbors, AdaBoost, Random Forest, and a Proposed Model.



	Accuracy	F-1 Score	Precision	Sensitivity	Specificity
LogisticReg	0.805667	0.411111	0.623277	0.947357	0.306707
Decision Tree	0.727667	0.407112	0.392582	0.814252	0.422758
k-nn	0.783667	0.400739	0.517282	0.913332	0.327054
Adaboost	0.806833	0.428783	0.619658	0.942863	0.327807
Random Forest	0.813	0.469253	0.630241	0.937727	0.373775
Proposed_model	0.806667	0.448145	0.607742	0.937727	0.373775

Based on the table, the Random Forest model has the highest accuracy of 0.813 and the highest F-1 score of 0.469. However, the Proposed Model has a slightly higher precision of 0.607742 and the same sensitivity and specificity as the Random Forest model.

Overall, the results indicate that the proposed model has performed well in predicting credit card defaults and Random Forest is the best-performing individual model among the ones tested.

FUTURE WORK

Future work on the credit card default dataset could include:

- Adding more demographic and financial variables that may contribute to credit card default.
- Investigating novel machine learning algorithms and strategies to increase the model's performance.
- Examining the influence of external factors on credit card default, such as economic circumstances.
- Creating more robust models that can manage the dataset's imbalance.
- Integrating the predictive model into a real-world application for credit card firms to utilize in credit risk management and default reduction.

REFERENCES

[1] S. S. H. Padmanabhuni, A. S. Kandukuri, D. Prusti and S. K. Rath, "Detecting Default Payment Fraud in Credit Cards," *2019 IEEE International Conference on Intelligent Systems and Green Technology (ICISGT)*, Visakhapatnam, India, 2019, pp. 15-153, doi: 10.1109/ICISGT44072.2019.00018.

[2] Y. Sayjadah, I. A. T. Hashem, F. Alotaibi and K. A. Kasmiran, "Credit Card Default Prediction using Machine Learning Techniques," 2018 Fourth International Conference on Advances in Computing, Communication & Automation (ICACCA), Subang Jaya, Malaysia, 2018, pp. 1-4, doi: 10.1109/ICACCAF.2018.8776802.

[3] Ying Chen, Ruirui Zhang, "Research on Credit Card Default Prediction Based on *k*-Means SMOTE and BP Neural Network", *Complexity*, vol. 2021, Article ID 6618841, 13 pages, 2021. https://doi.org/10.1155/2021/6618841

[4] M. Zan, G. Yanrong, and F. Guanlong, “Credit card fraud classification based on GAN-AdaBoost-DT imbalance classification algorithm,” *Journal of Computer Applications*, vol. 39, no. 2, pp. 314–318, 2019.

[5] R. Mei, Y. Xu, and G. Wang, “Study on analysis and influence factors of credit card default prediction model,” *Statistics and Applications*, vol. 5, no. 3, pp. 263–275, 2016.

[6] Akshay A. Bardiya, “Credit Card Defaulters Using Machine Learning”, Volume 10, Issue V, May 2022. DOI Link: https://doi.org/10.22214/ijraset.2022.43255