DATA SCIENCE PROJECT REPORT

ON

Detecting Credit Card Fraud with Machine Learning in python

CONTENTS

INTRODUCTION

'Fraud' in credit card transactions are unauthorized and unwanted usage of an account by someone other than the owner of that account. In simple words, Credit Card Fraud can be defined as a case where a person uses someone else's credit card for personal reasons while the owner and the card issuing authorities are unaware of the fact that the card is being used. Necessary prevention measures can be taken to stop this abuse. The behaviour of such fraudulent practices should be studied to minimize such fraud transactions and by doing this we can protect credit card holders from similar occurrences in the future. The challenge is to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase. At the time of Solving this problem, machine learning and data science techniques attracted our attention most as we know machine learning algorithms can help machines learn from given data and apply that knowledge to new data. So it is possible that using machine learning and data science techniques we can study the past data of credit card frauds and predict any such frauds which are going to happen in near future.

AIM OF THIS PROJECT

This project intends to illustrate the modelling of a data set using machine learning with Credit Card Fraud Detection. The Credit Card Fraud Detection Problem includes modelling past credit card transactions with the data of the ones that turned out to be fraud. This model is then used to recognize whether a new transaction is fraud or not. Our objective here is to detect the maximum number of the fraud transactions while minimizing the incorrect fraud classifications. Credit Card Fraud Detection is a typical sample of classification and various algorithms have been applied here to find out which algorithm is most efficient for detecting credit card frauds.

THEORY

In the previous section, we have seen the work-flow of the entire work. In this section, we are going to gather all the theoretical information required for better understanding of the whole process.

Data Analysis: For any data-science problem at first we need to study the given data for getting the type of data so that we can decide which machine learning algorithm should be applied. Here the data comes out as categorical data which contains binary output (1 or 0). So classification algorithms are used for this dataset. Sometimes for complex datasets, we need advanced data analytic knowledge for predicting which algorithm should be applied and preparing the data for further work.

Dataset description: Here for this project The data we are going to use is the Kaggle Credit Card Fraud Detection dataset (click here for dataset). The dataset contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. It contains only numeric input variables which are the result of a PCA transformation. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Finding the description of data and feature scaling: We need to find the description of the dataset for getting knowledge about its feature so that we can scale the features accordingly. This feature scaling is a very important step and our data will be preprocessed properly after this step and we can apply ML algorithms on it.

Splitting the data: The next step is splitting our given data in train set and test set. The train data machine learns about the given data set behaviour and by using the test set, it is possible to examine how much the machine has learned from the train set and accurately work on new data. This splitting is very necessary for the proper work of any machine learning algorithms.

Applying machine learning algorithms: In this project, I have worked on four different algorithms so let's discuss them one by one.

i) Decision Tree: Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too. The goal of using a Decision Tree is to create a training model that can be used to predict the class or value of the target variable by learning simple decision rules inferred from prior data (training data). In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

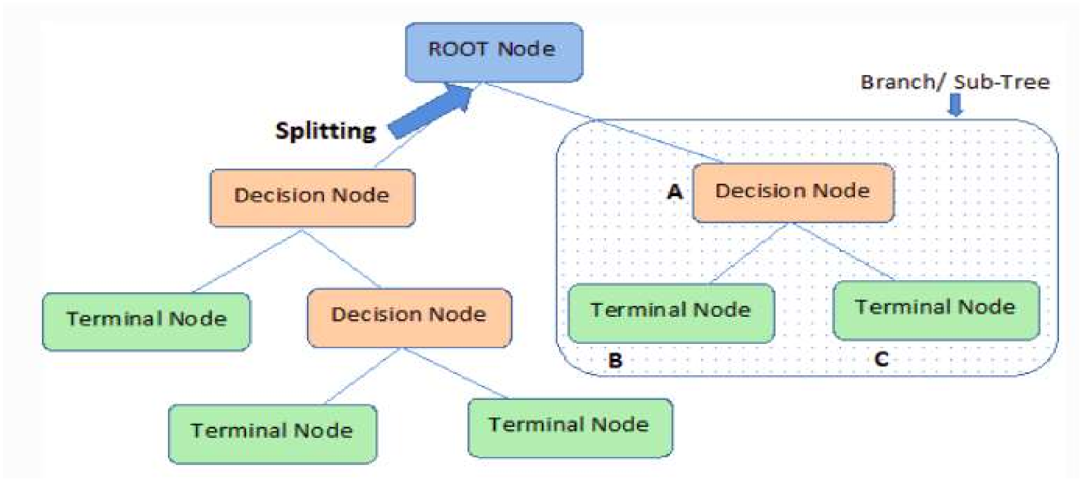


Fig:The work-flow of Decision Tree model

ii) K-Nearest Neighbour: *k*-NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, normalizing the training data can improve its accuracy dramatically. Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/*d*, where *d* is the distance to the neighbor. The neighbors are taken from a set of objects for which the class (for *k*-NN classification) or the object property value (for *k*-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A peculiarity of the *k*-NN algorithm is that it is sensitive to the local structure of the data.

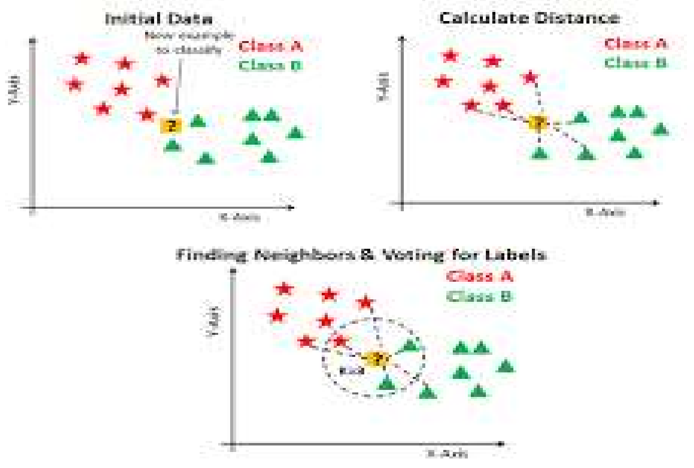


Fig : The work-flow of K-Nearest Neighbour model

iii) Logistic Regression: Logistic Regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. Logistic Regression is much similar to Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems. In this technique, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1). Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.

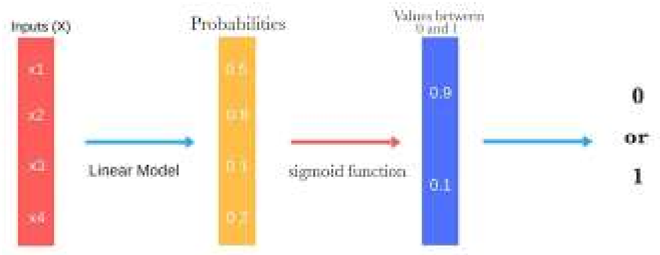


Fig: The work-flow of Logistic Regression model

iv) Random Forest: Random Forest is a popular machine learning (ML) algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

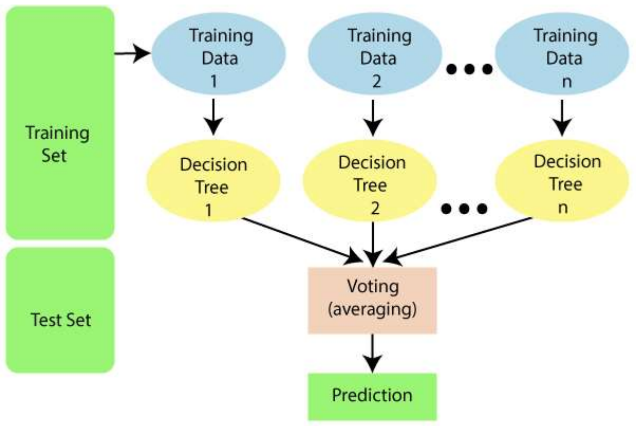


Fig: The work-flow of Random Forest model

Evolution of models: The last step of this work is evaluating all the applied models to find out the best model for our given case. The evaluation metrics we are going to use are the accuracy score metric, f1 score metric, and finally the confusion matrix.

i) Accuracy score metric: Accuracy score is one of the most basic evaluation metrics which is widely used to evaluate classification models. It can generally be expressed as:

Accuracy score = No.of correct predictions / Total no.of predictions.

ii) f1 score metric: The F1 score or F-score is one of the most popular evaluation metrics used for evaluating classification models. It can be simply defined as the harmonic mean of the model's precision and recall. It can be expressed as:

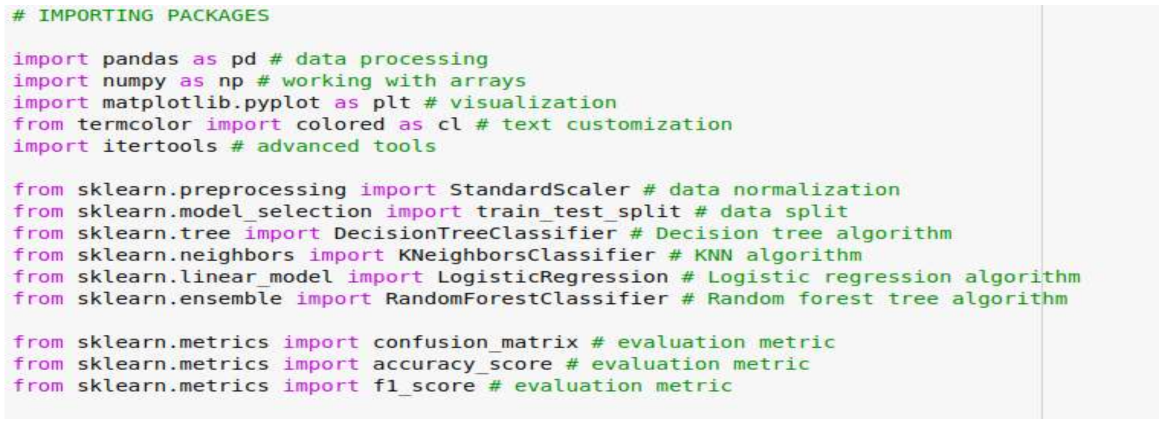
F1 score = 2((precision * recall) / (precision + recall)).

iii) Confusion matrix: Typically, a confusion matrix is a visualization of a classification model that shows how well the model has predicted the outcomes when compared to the original ones. Usually, the predicted outcomes are stored in a variable that is then converted into a correlation table.

METHODOLOGY

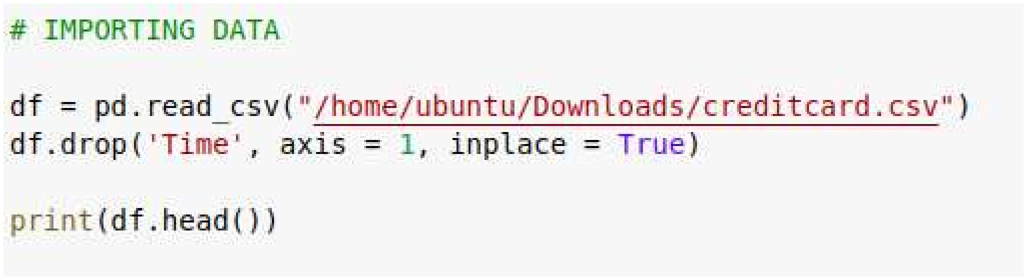
In this part, I will discuss all the methods followed for this project with coding details.

Importing libraries: For this project I have worked on python, so first it is necessary to import all the libraries needed for our work. Here I have used Pandas to work with data, NumPy to work with arrays, scikit-learn for data split, building and evaluating the classification models.



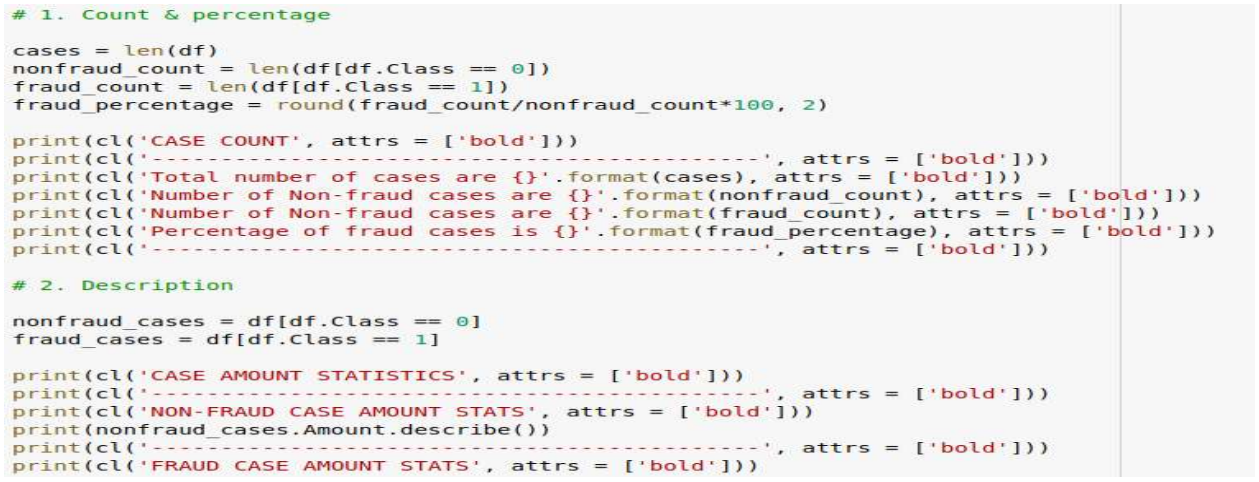
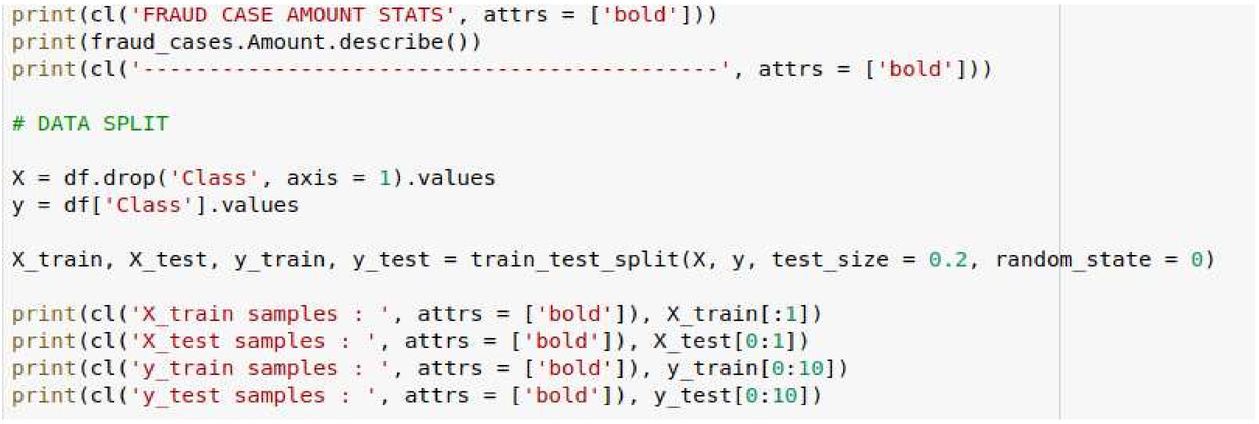
```
# IMPORTING PACKAGES

import pandas as pd # data processing
import numpy as np # working with arrays
import matplotlib.pyplot as plt # visualization
from termcolor import colored as cl # text customization
import itertools # advanced tools

from sklearn.preprocessing import StandardScaler # data normalization
from sklearn.model_selection import train_test_split # data split
from sklearn.tree import DecisionTreeClassifier # Decision tree algorithm
from sklearn.neighbors import KNeighborsClassifier # KNN algorithm
from sklearn.linear_model import LogisticRegression # Logistic regression algorithm
from sklearn.ensemble import RandomForestClassifier # Random forest tree algorithm

from sklearn.metrics import confusion_matrix # evaluation metric
from sklearn.metrics import accuracy_score # evaluation metric
from sklearn.metrics import f1_score # evaluation metric
```

Importing Dataset: By using following lines of codes a I have imported the dataset on which I have worked (description given in theory section).

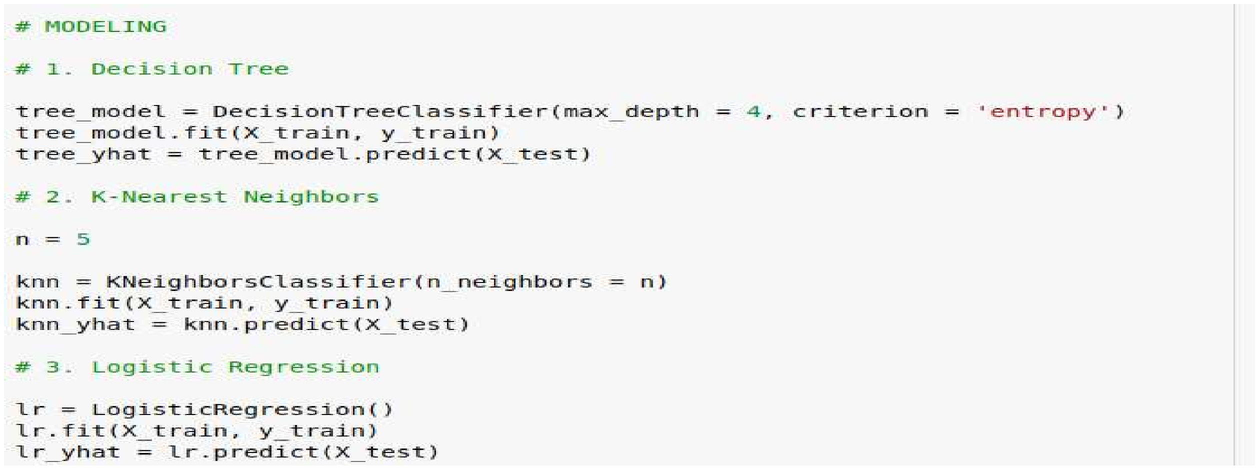


```
# IMPORTING DATA

df = pd.read_csv("/home/ubuntu/Downloads/creditcard.csv")
df.drop('Time', axis = 1, inplace = True)

print(df.head())
```

Some analysis on data and splitting of data: Let's have a look at how many fraud cases and non-fraud cases are there in our dataset. Along with that, let's also compute the percentage of fraud cases in the overall recorded transactions.


```
# 1. Count & percentage

cases = len(df)
nonfraud_count = len(df[df.Class == 0])
fraud_count = len(df[df.Class == 1])
fraud_percentage = round(fraud_count/nonfraud_count*100, 2)

print(cl('CASE COUNT', attrs = ['bold']))
print(cl('--------------------------------------------', attrs = ['bold']))
print(cl('Total number of cases are {}'.format(cases), attrs = ['bold']))
print(cl('Number of Non-fraud cases are {}'.format(nonfraud_count), attrs = ['bold']))
print(cl('Number of Non-fraud cases are {}'.format(fraud_count), attrs = ['bold']))
print(cl('Percentage of fraud cases is {}'.format(fraud_percentage), attrs = ['bold']))
print(cl('--------------------------------------------', attrs = ['bold']))

# 2. Description

nonfraud_cases = df[df.Class == 0]
fraud_cases = df[df.Class == 1]

print(cl('CASE AMOUNT STATISTICS', attrs = ['bold']))
print(cl('--------------------------------------------', attrs = ['bold']))
print(cl('NON-FRAUD CASE AMOUNT STATS', attrs = ['bold']))
print(nonfraud_cases.Amount.describe())
print(cl('--------------------------------------------', attrs = ['bold']))
print(cl('FRAUD CASE AMOUNT STATS', attrs = ['bold']))
```

```
print(cl('FRAUD CASE AMOUNT STATS', attrs = ['bold']))
print(fraud_cases.Amount.describe())
print(cl('--------------------------------------------', attrs = ['bold']))

# DATA SPLIT

X = df.drop('Class', axis = 1).values
y = df['Class'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

print(cl('X_train samples : ', attrs = ['bold']), X_train[:1])
print(cl('X_test samples : ', attrs = ['bold']), X_test[0:1])
print(cl('y_train samples : ', attrs = ['bold']), y_train[0:10])
print(cl('y_test samples : ', attrs = ['bold']), y_test[0:10])
```
By using above lines of code the whole dataset has been splitted on two different data sets (train and test data set) for the future work.

Modeling: Now it is time to model the given data in different ML algorithms using scikit-learn libraries of python. All these models can be built feasibly using the algorithms provided by the scikit-learn package.


```
# MODELING

# 1. Decision Tree

tree_model = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)

# 2. K-Nearest Neighbors

n = 5

knn = KNeighborsClassifier(n_neighbors = n)
knn.fit(X_train, y_train)
knn_yhat = knn.predict(X_test)

# 3. Logistic Regression

lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)
```

```
# 4. Random Forest Tree

rf = RandomForestClassifier(max_depth = 4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
```

Evolution Of models:

By using the following lines of code I have evaluated the built models using the evaluation metrics (accuracy score, f1 score and confusion matrix) provided by the scikit-learn package.

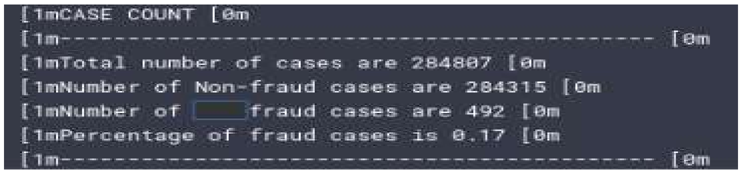
```
# EVALUATION

# 1. Accuracy score

# 2. F1 score

print(cl('F1 SCORE', attrs = ['bold']))
```

```
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix for the models

tree_matrix = confusion_matrix(y_test, tree_yhat, labels = [0, 1]) # Decision Tree
knn_matrix = confusion_matrix(y_test, knn_yhat, labels = [0, 1]) # K-Nearest Neighbors
lr_matrix = confusion_matrix(y_test, lr_yhat, labels = [0, 1]) # Logistic Regression
rf_matrix = confusion_matrix(y_test, rf_yhat, labels = [0, 1]) # Random Forest Tree

# Plot the confusion matrix

plt.rcParams['figure.figsize'] = (6, 6)

# 1. Decision tree

tree_cm_plot = plot_confusion_matrix(tree_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'Decision Tree')
plt.savefig('tree_cm_plot.png')
plt.show()
```

```
# 2. K-Nearest Neighbors

knn_cm_plot = plot_confusion_matrix(knn_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'KNN')
plt.savefig('knn_cm_plot.png')
plt.show()

# 3. Logistic regression

lr_cm_plot = plot_confusion_matrix(lr_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'Logistic Regression')
plt.savefig('lr_cm_plot.png')
plt.show()


# 4. Random forest tree

rf_cm_plot = plot_confusion_matrix(rf_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'Random Forest Tree')
plt.savefig('rf_cm_plot.png')
plt.show()
```

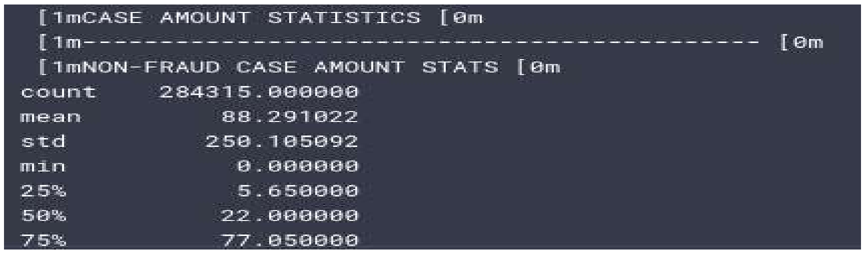
The model evolution step is very important to gain information about the performance of different ML algorithms on a given dataset and this helps to decide the best algorithm or model for our case so that we can achieve our aim or detect the maximum number of the fraud transactions while minimizing the incorrect fraud classifications.

RESULTS AND DISCUSSIONS

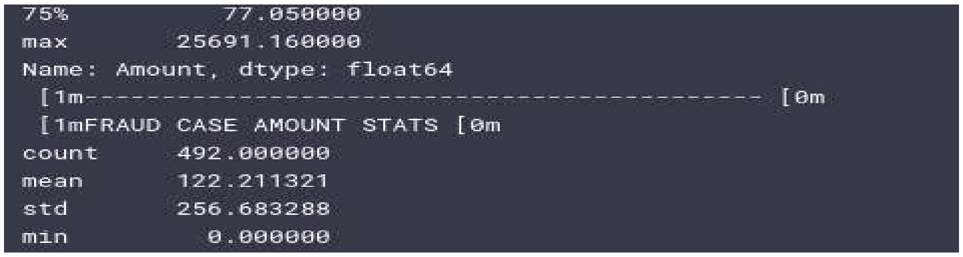


```
[1mCASE COUNT [0m
[1m-------------------------------------------- [0m
[1mTotal number of cases are 284807 [0m
[1mNumber of Non-fraud cases are 284315 [0m
[1mNumber of     fraud cases are 492 [0m
[1mPercentage of fraud cases is 0.17 [0m
[1m-------------------------------------------- [0m
```

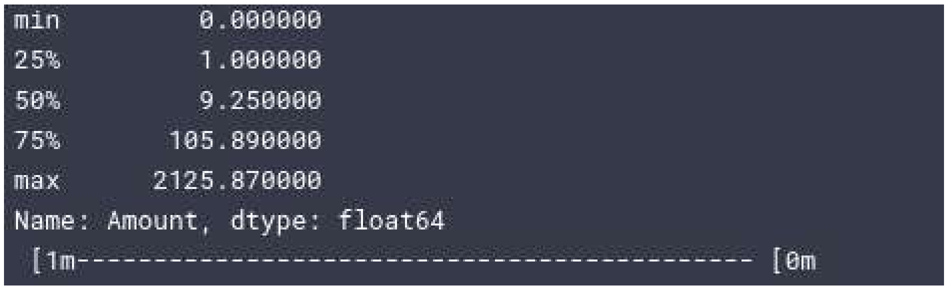
The above result we got after preliminary data analysis in the given data. From this result it is clear that the number of fraud cases are much less than the number of non-fraud cases and the percentage of fraud cases is just 0.17. So, it can be stated that the data we are dealing with is highly imbalanced data and needs to be handled carefully when modeling and evaluating. Below here I have presented the statistical results of both fraud and non-fraud transaction amount data.



```
  [1mCASE AMOUNT STATISTICS [0m
  [1m-------------------------------------------- [0m
  [1mNON-FRAUD CASE AMOUNT STATS [0m
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
```



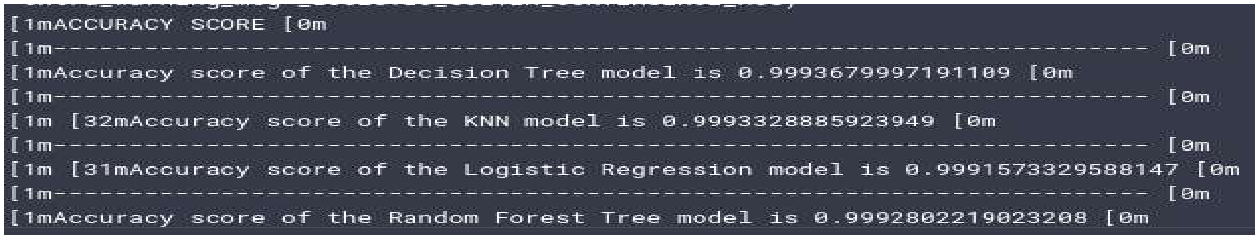
```
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
 [1m-------------------------------------------- [0m
 [1mFRAUD CASE AMOUNT STATS [0m
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
```



```
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
 [1m-------------------------------------------- [0m
```

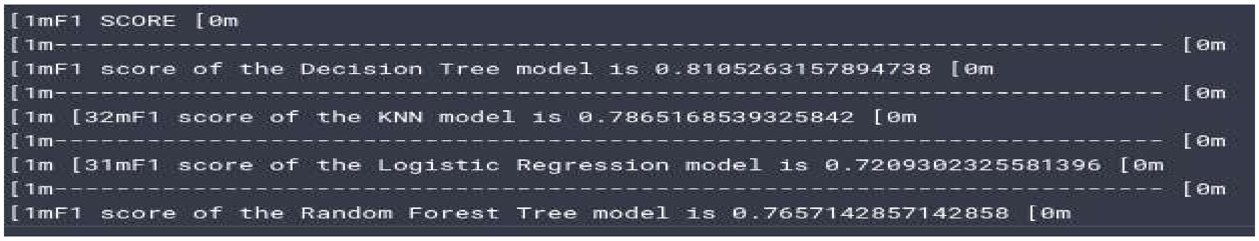
While seeing the statistics, it is clear that the values in the ‘Amount’ variable are varying enormously when compared to the rest of the variables. To reduce its wide range of values, it can be normalized using the ‘StandardScaler’ method in python.

After the modeling, the models are evaluated using the evaluation metrics and below I have presented all the results obtained after evolution of models.



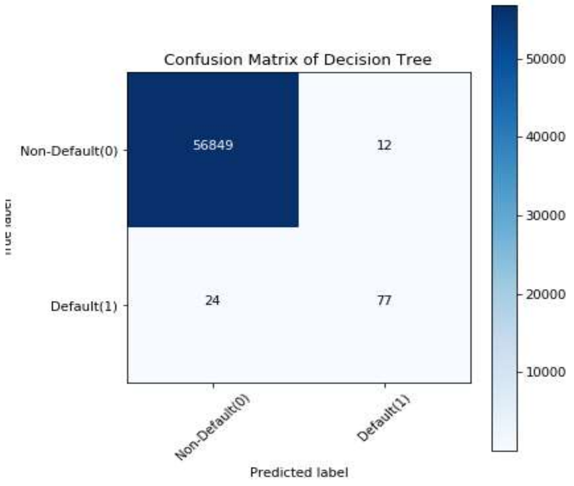
```
[1mACCURACY SCORE [0m
[1m---------------------------------------------------------------------- [0m
[1mAccuracy score of the Decision Tree model is 0.9993679997191109 [0m
[1m---------------------------------------------------------------------- [0m
[1m [32mAccuracy score of the KNN model is 0.9993328885923949 [0m
[1m---------------------------------------------------------------------- [0m
[1m [31mAccuracy score of the Logistic Regression model is 0.9991573329588147 [0m
[1m---------------------------------------------------------------------- [0m
[1mAccuracy score of the Random Forest Tree model is 0.9992802219023208 [0m
```

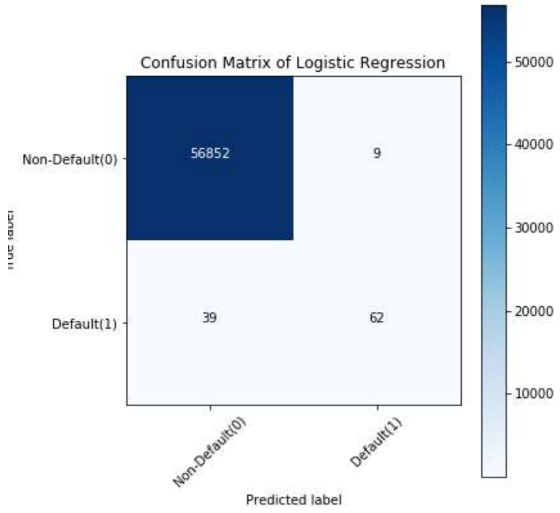
According to the accuracy score evaluation metric, the Decision Tree model reveals to be the most accurate model and the Logistic regression model to be the least accurate model. However, when we round up the results of each model, it shows 0.99 (99% accurate) which is a very good score.

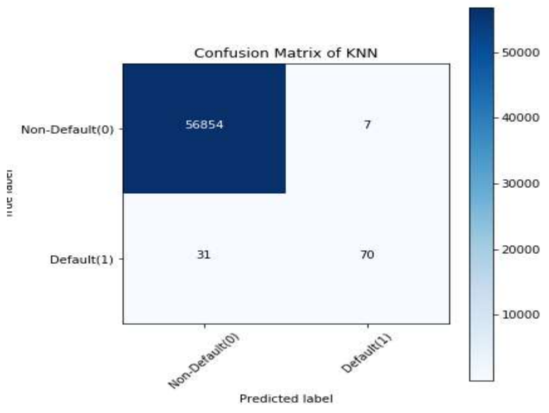


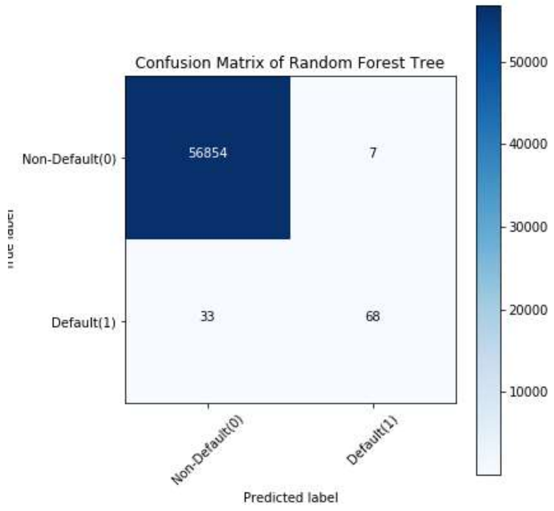
```
[1mF1 SCORE [0m
[1m----------------------------------------------------------------------- [0m
[1mF1 score of the Decision Tree model is 0.8105263157894738 [0m
[1m----------------------------------------------------------------------- [0m
[1m [32mF1 score of the KNN model is 0.7865168539325842 [0m
[1m----------------------------------------------------------------------- [0m
[1m [31mF1 score of the Logistic Regression model is 0.7209302325581396 [0m
[1m----------------------------------------------------------------------- [0m
[1mF1 score of the Random Forest Tree model is 0.7657142857142858 [0m
```

 On the basis of the F1 score evaluation metric, the Decision Tree model snatches the first place again and the Logistic regression model remains to be the least accurate model.









The results of this confusion matrices are very interesting and let's discuss this explicitly by making a table.

Name of algorithm	Non-fraud case and predicted as non-fraud case	Non-fraud case but predicted as fraud case	Fraud case but predicted as non-fraud case	Fraud case and predicted and predicted as fraud case
Decision Tree	56849	12	24	77
K-NN	56854	7	31	70
Linear Regression	56852	9	39	62
Random Forest	56854	7	33	67

While comparing the confusion matrix of all the models, it can be seen that the Decision Tree model has performed a very good job of classifying the fraud transactions from the non-fraud transactions. The Decision tree model has predicted most number of fraud cases as fraud cases and least number of fraud cases as non-fraud cases. So it is concluded that the most appropriate model which can be used for this case is the Decision Tree model and the model which can be neglected is the Logistic regression model.

CONCLUSIONS

Detecting the fraud credit card transactions are very important for protecting card holders from paying the extra amount of money that they have not used and the machine learning algorithm gave us hope to solve this problem. In this project, after working on four different ML algorithms, it is possible to conclude that the Decision Tree algorithm is most useful for this case as it is able to maximize the detected number of fraud cases and minimize the wrong detection (detection of fraud cases as non fraud cases). The Decision Tree algorithm gave more than 99% accuracy. This high percentage of accuracy is to be expected due to the huge imbalance between the number of non-fraud and fraud transactions which has been detected earlier at the time of data analysis. Since the entire dataset consists of only two days’ transaction records, it's only a fraction of data that can be made available if this project were to be used on a commercial scale. Being based on machine learning algorithms, the program will only increase its efficiency over time as more data is put into it.

REFERENCES

1. **https://medium.com/datazen/credit-card-fraud-detection-with-machine-learning-in-python-ac7281991d87**
2. https://data-flair.training/blogs/data-science-machine-learning-project-credit-card-fraud-detection/
3. https://www.geeksforgeeks.org/ml-credit-card-fraud-detection/
4. S P Maniraj, Aditya Saini, Swarna Deep Sarkar Shadab Ahmed, Credit Card Fraud Detection using Machine Learning and Data Science. **IJERT V8I 090031.** **https://www.ijert.org/**
5. Hands-On Machine Learning with Scikit learning and Tensor flow by Aurelien Geron.
6. http://www.wikipedia.org/