Prediction of Credit Card fraud

Introduction & Problem Statement:

A credit card is one of the most used financial products to make online purchases and payments. Though the Credit cards can be a convenient way to manage your finances, they can also be risky. Credit card fraud is the unauthorized use of someone else's credit card or credit card information to make purchases or withdraw cash.

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

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. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

We have to build a classification model to predict whether a transaction is fraudulent or not.

Exploratory Data Exploration:

To protect the user's identity and the security of their confidential information, the dataset provider has applied Principal Component Analysis transformation on the original numerical features and compressed it into 28 principal components.
Only two features have not been transformed i.e. 1) Time and 2) Amount.
The feature class will be the target column with user labels.
0 : non-fraudulent 1 : fraudulent.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	V25	V26	V27	V28	Class	Scaled_Amount
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0	0.244964
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	0	-0.342475
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	0	1.160686
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0	0.140534
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	0	-0.073403
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0	-0.350151
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	0	-0.254117
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	0	-0.081839
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	0	-0.313249
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	0	0.514355

284807 rows × 30 columns

The dataset exclusively comprises numerical features as evident from the datatypes of all features and since there are no instances of missing values there is no need for null-value handling in this dataset.

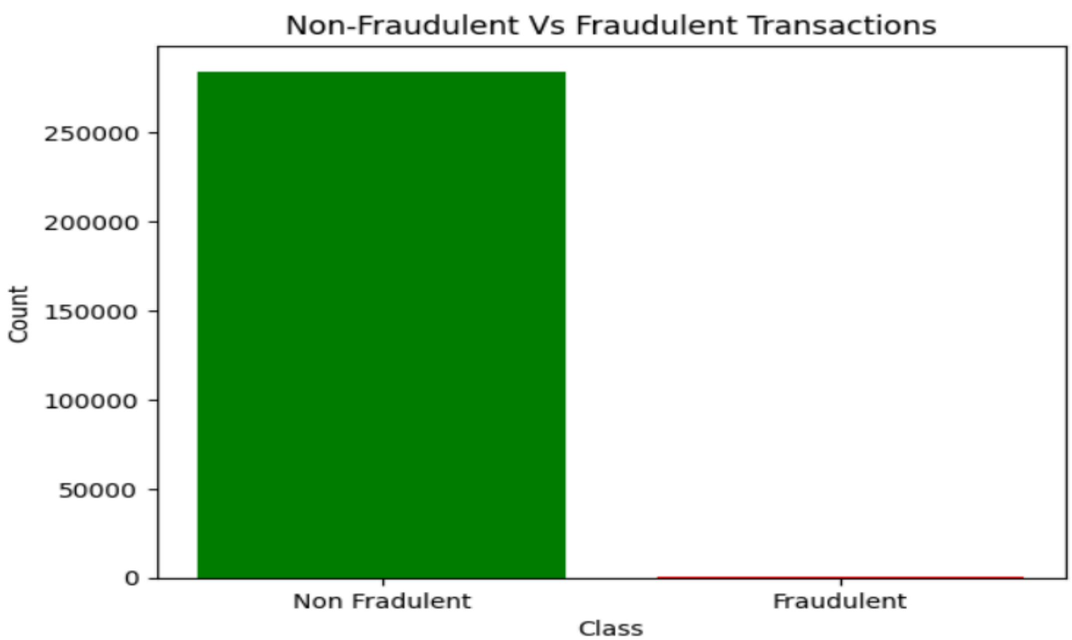
Data types

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Null values for each variable

	0
Time	0
V1	0
V2	0
V3	0
V4	0
V5	0
V6	0
V7	0
V8	0
V9	0
V10	0
V11	0
V12	0
V13	0
V14	0
V15	0
V16	0
V17	0
V18	0
V19	0
V20	0
V21	0
V22	0
V23	0
V24	0
V25	0
V26	0
V27	0
V28	0
Amount	0
Class	0

The barplot below reveals a significant imbalance between classes (0-Non Fraudulent) and (1-Fraudulent).



Majority of features are in PCA form except Time and Amount, a descriptive analysis of these 2 fields are shown below.

```
df['Time'].describe()
```

	Time
count	284807.000000
mean	94813.859575
std	47488.145955
min	0.000000
25%	54201.500000
50%	84692.000000
75%	139320.500000
max	172792.000000

dtype: float64

```
df['Amount'].describe()
```

	Amount
count	284807.000000
mean	88.349619
std	250.120109
min	0.000000
25%	5.600000
50%	22.000000
75%	77.165000
max	25691.160000

dtype: float64

From the counts of Fraudulent and Non Fraudulent Transactions in the dataset we can see that we have a class imbalance.

```
Number of Non-Fraudulent Transactions:  284315
Number of Fraudulent Transactions:  492
Percentage of Fraudulent Transactions:  0.17
```

- We can observe that the genuine transactions are over 99%.
- We will apply scaling techniques on the "Amount" feature to transform the range of values.
- We will drop the original "Amount" column and add a new column with the scaled values. We will also drop the "Time" columns as it is irrelevant.
- Now, we will split the credit card data with a split of 70-30 using train_test_split().
- train_test_split() function in scikit-learn is a useful utility for splitting a dataset into training and testing sets.
- Parameters
  X: Feature matrix
  Y: Target variable
- Test_size: Proportion of the dataset to include in the test split.
  Here we have set the test_size as 0.3 means 30% of the data we take as a testing data set.
- Random_state: We have set the seed for random number generation, to ensure the reproducibility.
- Shapes of Training and Testing dataset are shown below.

```
Shape of the training dataset train_X:  (199364, 29)
Shape of the testing dataset test_X:  (85443, 29)
```

Model Selection for this Fraud detection project

We will explore various machine learning algorithms to determine the most effective model for our binary classification problem. The task involves predicting one of the two class labels. We plan to access the performance of different algorithms, such as Random Forest and

- Decision Tree Algorithm The Decision Tree Algorithm is a supervised machine learning technique employed for both classification and regression tasks. Its objective is to create a training model capables of predicting the value of a target class variable.This is achieved by learning straightforward if-then-else decision rules derived from the patterns present in the training data.

- Random Forest Algorithm Random Forest is a supervised Machine Learning algorithm. It creates a "forest" out of an ensemble of "decision trees", which are normally trained using the "bagging" technique. The bagging method's basic principle is that combining different learning models improves the outcome.
  To get a more precise and reliable forecast, random forest creates several decision trees and merges them.

```
# Decision Tree
decision_tree = DecisionTreeClassifier()

# Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
# Here we are providing the RandomForestClassifier to create 100 trees in the forest. The large number of trees
generally leads to better performance, but it may also increase the training time.
```

```
# Printing the scores of the both classifiers
print("Decision Tree: ", round((decision_tree_score),4))
print("Random Forest: ", round((random_forest_score),4))

Decision Tree:  99.9239
Random Forest:  99.9532
```

- Decision Tree Score is 99.92392589211521
- Random Forest Score is 99.9531851643786

The Random Forest classifier has slightly an edge over the Decision Tree Classifier.

Evaluation Metrics -

We will calculate the following parameters to evaluate our models.

- Accuracy_score
- Precision_score
- Confusion_matrix
- Recall_score
- F-1 score

```
Evaluation of Decision Tree Model:
Accuracy: 0.9992
Precision: 0.7887
recall_score: 0.7619
F1-Score: 0.7751
```

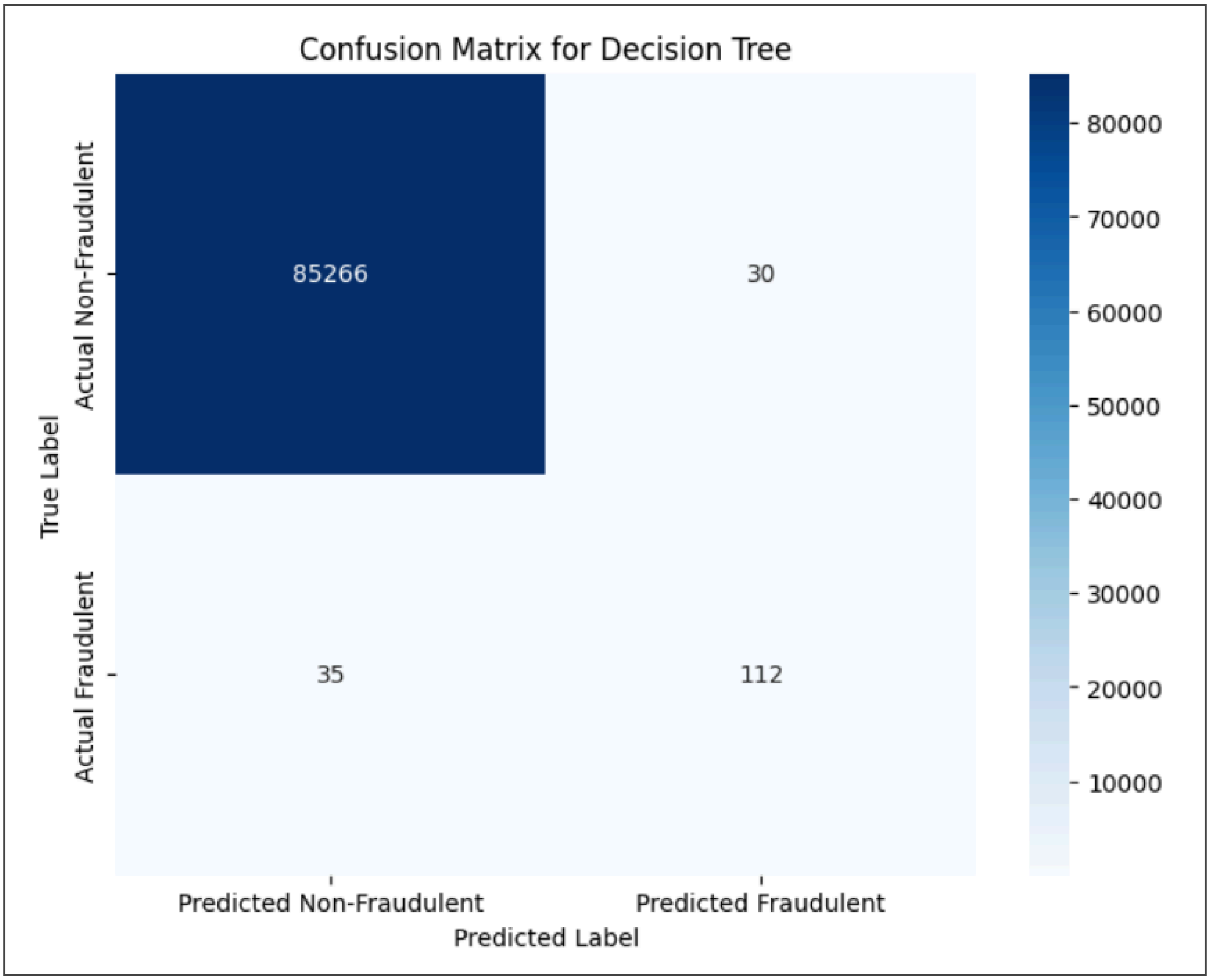
```
Evaluation of Random Forest Model:
Accuracy: 0.9995
Precision: 0.935
recall_score: 0.7823
F1-Score: 0.8519
```

Confusion matrix

A confusion matrix is a simple table that shows how well a classification model is performing by comparing its predictions to the actual results. It breaks down the predictions into four categories: correct predictions for both classes (true positives and true negatives) and incorrect predictions (false positives and false negatives). This helps us understand where the model is making mistakes, so you can improve it.

- True Positive (TP): The model correctly predicted a positive outcome (the actual outcome was positive).
- True Negative (TN): The model correctly predicted a negative outcome (the actual outcome was negative).
- False Positive (FP): The model incorrectly predicted a positive outcome (the actual outcome was negative). Also known as a Type I error.
- False Negative (FN): The model incorrectly predicted a negative outcome (the actual outcome was positive). Also known as a Type II error.

We will create confusion matrices for both models and compare the results.



- ***Non-Fraudulent transactions :***

Correctly predicted as non-fraudulent (True Negative) 85266 transactions.

Incorrectly predicted as fraudulent (False Positive) 30 transactions.

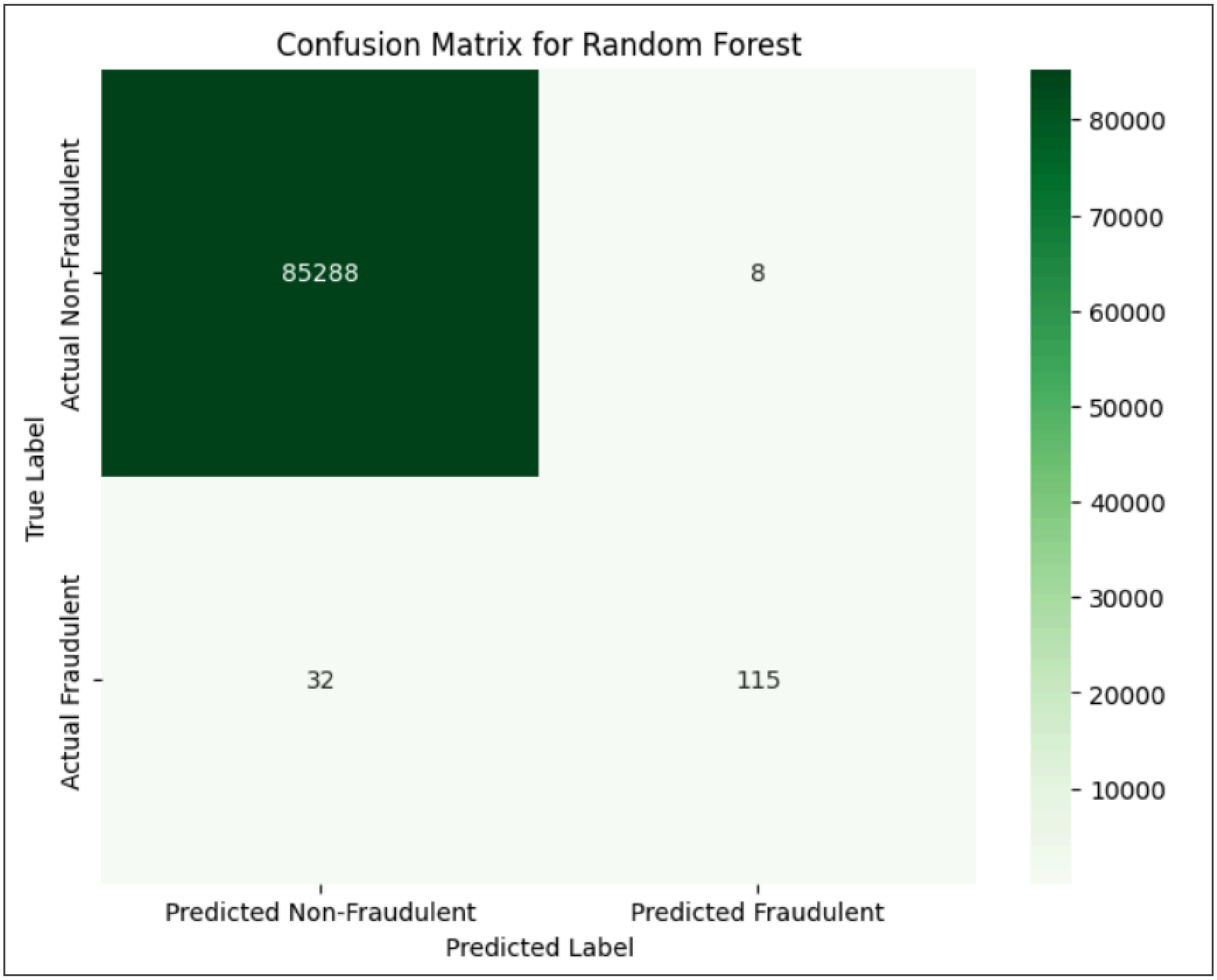
- ***Fraudulent Transactions :***

Incorrectly predicted as non-fraudulent(False Negative) : 35 transactions

Correctly predicted as fraudulent(True Positive) : 112 transactions

Conclusions

- The Decision Tree model correctly identified 112 fraudulent transactions.
- Incorrectly identified 35 transactions as non-fraudulent.
- Correctly identified 85266 non-fraudulent transactions.
- Incorrectly identified 30 non-fraudulent transactions as fraudulent.



- ***Non-Fraudulent transactions:***

Correctly predicted as non-fraudulent(True Negative) 85288 transactions.

Incorrectly predicted as fraudulent(False Positive) 8 transactions.

- ***Fraudulent transactions:***

Incorrectly predicted as non-fraudulent(False Negative) : 32 transactions

Correctly predicted as fraudulent(True Positive) : 115 transactions

Conclusions

- The Random Forest model correctly identified 115 fraudulent transactions.
- Incorrectly identified 32 transactions as non-fraudulent.
- Correctly identified 85288 non-fraudulent transactions.
- Incorrectly identified only 8 non-fraudulent transactions as fraudulent.

Class Imbalance

- The Random Forest model works better than Decision Trees. In the presence of a class-Imbalance issue, where genuine transactions account for over 99% of the dataset and credit card fraud transactions constitute only 0.17%.
- Training the model without addressing the imbalance can lead to biased predictions.
- Despite the apparent accuracy, such a model may not effectively capture the nuances of the minority class (fraud transactions) and may not generalize well to real-world situations.
- The class imbalance problem can be solved by various techniques, Oversampling is one of them.

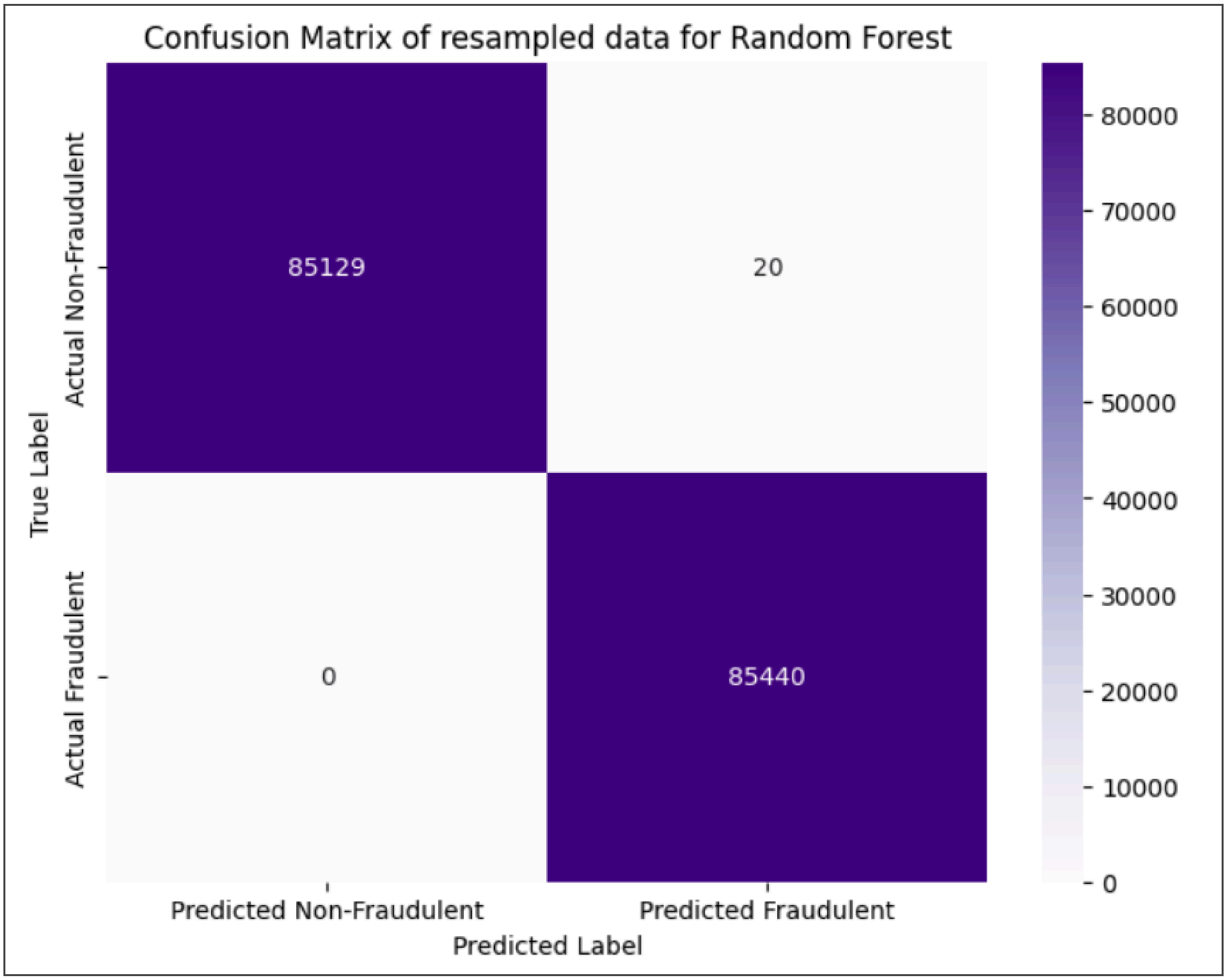
```
# We will use the SMOT (Synthetic Minority Oversampling Technique)
from imblearn.over_sampling import SMOTE

X_resampled, Y_resampled = SMOTE().fit_resample(X,Y)
print("Resampled shape of X: ",X_resampled.shape)
print("Resampled shape of Y: ",Y_resampled.shape)
```

```
Resampled shape of X:  (568630, 29)
Resampled shape of Y:  (568630,)
```

```
predictions_resampled = rf_resampled.predict(test_X)
random_forest_score_resampled = rf_resampled.score(test_X, test_Y) * 100
```

```
print(random_forest_score_resampled)
```

```
99.98710350608772
```



- ***Non-Fraudulent transactions:***

Correctly predicted as non-fraudulent(True Negative) 85128 transactions.

Incorrectly predicted as fraudulent(False Positive) 20 transactions.

- ***Fraudulent transactions:***

Incorrectly predicted as non-fraudulent(False Negative) : 0 transactions

Correctly predicted as fraudulent(True Positive) : 85436 transactions

Conclusions

- The Random Forest model correctly identified 85440 fraudulent transactions.
- Incorrectly identified 0 transactions as non-fraudulent.
- Correctly identified 85129 non-fraudulent transactions.
- Incorrectly identified only 20 non-fraudulent transactions as fraudulent.

Links for Dataset and Github repository link -
https://github.com/AnandSharma-22/Credit-Card-Fraud-Transaction-Detection

Thank you.