# Credit Card Fraud Transaction Detection

Ravindra Neralla
DSC 680
Milestone 3 – Project 3

# Abstract

Most of the people around the world are restricted to home due to COVID-19 pandemic, resulted in more online activities like buying regular groceries also shifted to online. Fraudsters taking this as an opportunity to steal credit card information and do more crime, resulted in online credit card fraud to soar to a whopping $32 billion in year 2020. Surprisingly, this amount is superior to profits posted by some of the blue chip companies in 2017, such as **Coca-Cola** ($2 billions), Warren Buffet's **Berkshire Hathaway** ($24 billions) and **JP Morgan Chase** ($23.5 billions) [3].

Large volume of transactions happens daily, criminals took this as an opportunity to create series of fake transactions through legitimate marketplaces to launder their money digitally. Online payment processing involves a number of different financial service providers and go-between companies that money launderers can often hide in plain sight.

Fraudulent transactions resulting from stolen information and the fraudulent transactions created to launder money both have the potential to negatively affect online businesses. Most major credit card networks like AmEx, Visa, or Mastercard will cut off payment processing if a business shows a fraud rate that is too high. Each company has its own policy, but generally speaking, if a business's fraud rate is greater than one percent, they risk losing the ability to process credit card payments entirely.


**Problem Statement/Hypothesis:**

The main research objective in the dataset is identifying whether a credit card transaction is fraudulent or legitimate. It will be helpful for financial organizations to block the card and take appropriate actions. The machine learning algorithms I have used in this work are Logistic Regression and Random Forest.

**Data Source:**

The dataset is source from the Kaggle Credit Card Fraud Detection dataset [3]. It contains two-day transactions made on 09/2013 by European cardholders. The dataset contains 492 fradulant transactions out of 284,807 transactions. Thus, it is highly unbalanced, with the positive (frauds) accounting for only 0.17%. Due to privacy reasons, we don't know the names of the other features. All we know is that all of them (except time and amount) went through PCA transformation, which means that they were previously scaled.

Dataset contains features V1, V2, … V28 are the principal components obtained with PCA transformation. The only features which have not been transformed are *'Time'* and *'Amount'*. *'Time'* is the seconds elapsed between each transaction and the first. '**Amount'** is the transaction amount. **'Class'** is the response variable with 1 as fraud and 0 otherwise.

Sample Data:

Out[3]:

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | ... | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0 |
| 1 | 0.0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0.255425 | ... | -0.225775 | -0.638672 | 0.101288 | -0.339846 | 0 |
| 2 | 1.0 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | ... | 0.247998 | 0.771679 | 0.909412 | -0.689281 | -0 |
| 3 | 1.0 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1.387024 | ... | -0.108300 | 0.005274 | -0.190321 | -1.175575 | 0 |
| 4 | 2.0 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0.817739 | ... | -0.009431 | 0.798278 | -0.137458 | 0.141267 | -0 |

5 rows × 31 columns

**Technical Approach:**

Data Analysis:

```
[4]: df.describe()
```

Out[4]:

| ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|
| ... | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 284807.000000 | 284807.000000 |
| ... | 1.537294e-16 | 7.959909e-16 | 5.367590e-16 | 4.458112e-15 | 1.453003e-15 | 1.699104e-15 | -3.660161e-16 | -1.206049e-16 | 88.349619 | 0.001727 |
| ... | 7.345240e-01 | 7.257016e-01 | 6.244603e-01 | 6.056471e-01 | 5.212781e-01 | 4.822270e-01 | 4.036325e-01 | 3.300833e-01 | 250.120109 | 0.041527 |
| ... | -3.483038e+01 | -1.093314e+01 | -4.480774e+01 | -2.836627e+00 | -1.029540e+01 | -2.604551e+00 | -2.256568e+01 | -1.543008e+01 | 0.000000 | 0.000000 |
| ... | -2.283949e-01 | -5.423504e-01 | -1.618463e-01 | -3.545861e-01 | -3.171451e-01 | -3.269839e-01 | -7.083953e-02 | -5.295979e-02 | 5.600000 | 0.000000 |
| ... | -2.945017e-02 | 6.781943e-03 | -1.119293e-02 | 4.097606e-02 | 1.659350e-02 | -5.213911e-02 | 1.342146e-03 | 1.124383e-02 | 22.000000 | 0.000000 |
| ... | 1.863772e-01 | 5.285536e-01 | 1.476421e-01 | 4.395266e-01 | 3.507156e-01 | 2.409522e-01 | 9.104512e-02 | 7.827995e-02 | 77.165000 | 0.000000 |
| ... | 2.720284e+01 | 1.050309e+01 | 2.252841e+01 | 4.584549e+00 | 7.519589e+00 | 3.517346e+00 | 3.161220e+01 | 3.384781e+01 | 25691.160000 | 1.000000 |

I have loaded the data into a python data frame and looked at sample data to understand the variables. After looking at the stats in the data, I observed that Amount value ranges from $0 to $25,691.16. The mean value of all transactions is $88.35.

a) <u>Verify for null values in dataset:</u>

Verified data loaded into data frame for any null values present, found that there are no NULL values in the dataset.

```
In [5]: df.isnull().sum()
```

```
Out[5]: 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
```

b) <u>Verify data types in the dataset:</u>

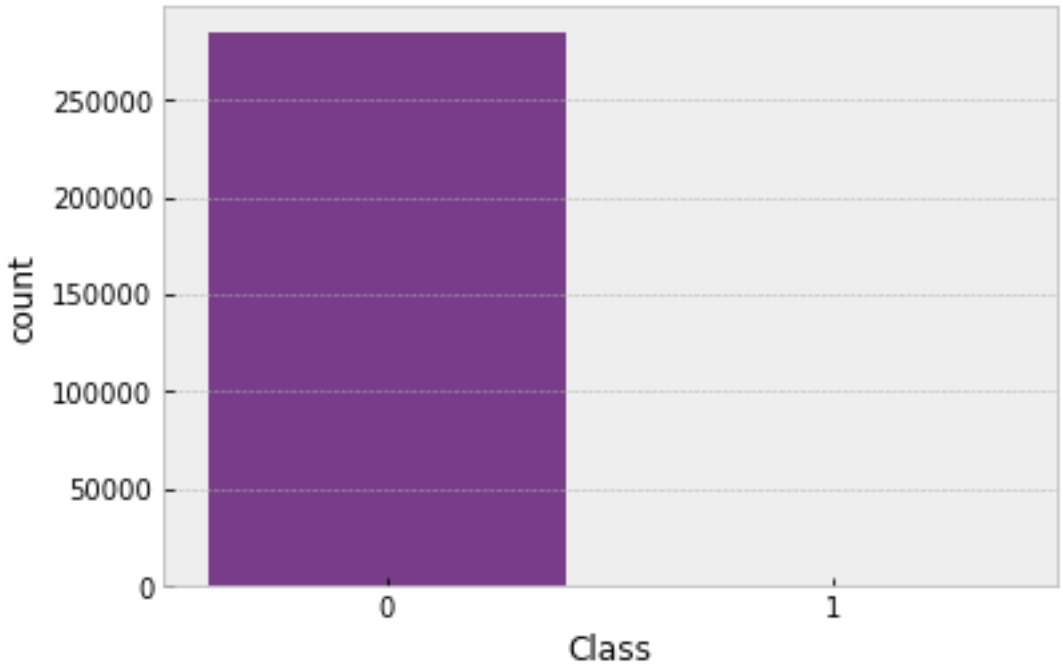
Vefiy data types of all the variables in the dataset, observed that all the variables V1..V28 and time and amount also float type.

```
In [6]: df.dtypes

Out[6]: Time      float64
        V1        float64
        V2        float64
        V3        float64
        V4        float64
        V5        float64
        V6        float64
        V7        float64
        V8        float64
        V9        float64
        V10       float64
        V11       float64
        V12       float64
        V13       float64
        V14       float64
        V15       float64
        V16       float64
        V17       float64
        V18       float64
        V19       float64
        V20       float64
        V21       float64
        V22       float64
        V23       float64
        V24       float64
```

c) Verify counts and unique values in the dataset:

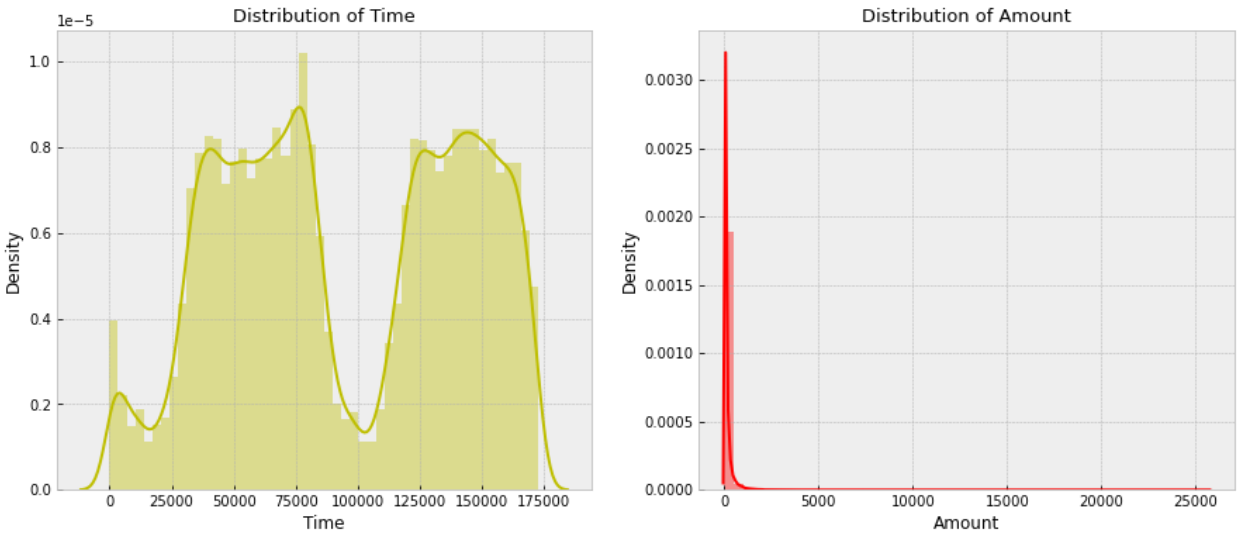
```
In [7]: df.Class.unique()

Out[7]: array([0, 1], dtype=int64)

In [8]: df.Class.value_counts()

Out[8]: 0    284315
        1       492
        Name: Class, dtype: int64
```

d) Fraud vs Genuine Transactions:




There are only 492 fraudulent transactions and 284315 transactions are legitimate. This indicates that the dataset is highly imbalanced, algorithm that classifies everything as non-fraudulent would give 99.83% accuracy. To address the problem of imbalanced dataset we can use undersampling and oversampling data approach techniques. Oversampling increases the number of minority class members (i.e. fraud transactions) in the training set. The advantage of oversampling is that no information from the original training set is lost unlike in undersampling, as all observations from the minority and majority classes are kept. On the other hand, it is prone to overfitting. In this project, I used SMOTE (Synthetic Minority Oversampling Technique) to make the dataset balanced. It creates synthetic points from the minority class.

e) Time and Amount distribution:




The distribution of the monetary value of all transactions is heavily right-skewed. The vast majority of transactions are relatively small and only a tiny fraction of transactions comes even close to the maximum.

f) Scale Time and Amount variables:

I used Feature Scaling to normalize the distribution, after applying scaling, amount value ranges from -0.2 to 1.78, which is in the same order of magnitude as other features. Before applying the scaling, amount was ranging from 0 to 25,000.

Out[12]:

| | scaled_time | scaled_amount | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | ... | V20 | V21 | V22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.994983 | 1.783274 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | ... | 0.251412 | -0.018307 | 0.277838 |
| 1 | -0.994983 | -0.269825 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | ... | -0.069083 | -0.225775 | -0.638672 |
| 2 | -0.994972 | 4.983721 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | ... | 0.524980 | 0.247998 | 0.771679 |
| 3 | -0.994972 | 1.418291 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | ... | -0.208038 | -0.108300 | 0.005274 |
| 4 | -0.994960 | 0.670579 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | ... | 0.408542 | -0.009431 | 0.798278 |

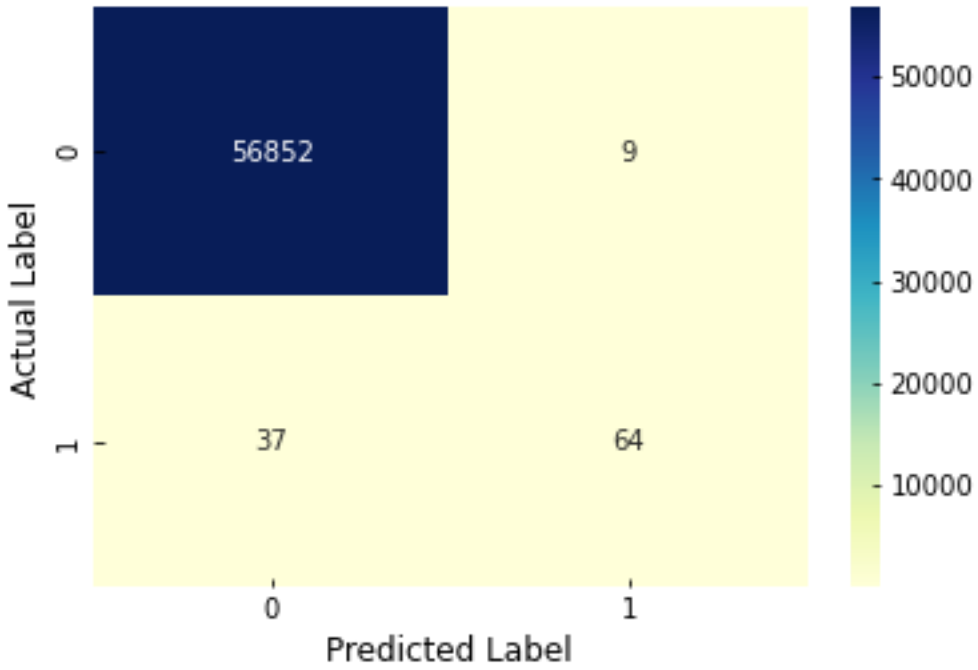
5 rows × 31 columns

# Model Building:

## I) Logistic Regression without SMOTE

As the problem is of classification type, I started building model using Logistic Regression without applying SMOTE on the original data, below are Precision, Recall and f1-score.

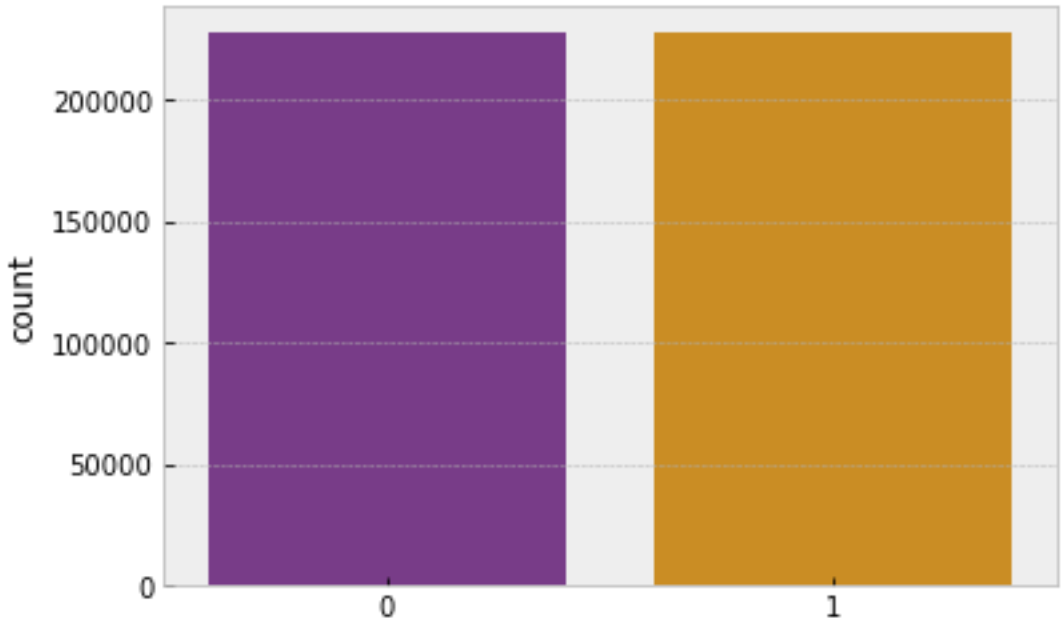
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **Non-fraud** | 1.00 | 1.00 | 1.00 | 56861 |
| **Fraud** | 0.88 | 0.63 | 0.74 | 101 |
| **accuracy** | | | 1.00 | 56962 |
| **macro avg** | 0.94 | 0.82 | 0.87 | 56962 |
| **weighted avg** | 1.00 | 1.00 | 1.00 | 56962 |



Based on the above Precision, Recall and F1 scores and confusion matrix, model is not a good model, it is biased towards majority class and the recall in minority class is not as high as desired.

# Balance dataset using SMOTE:

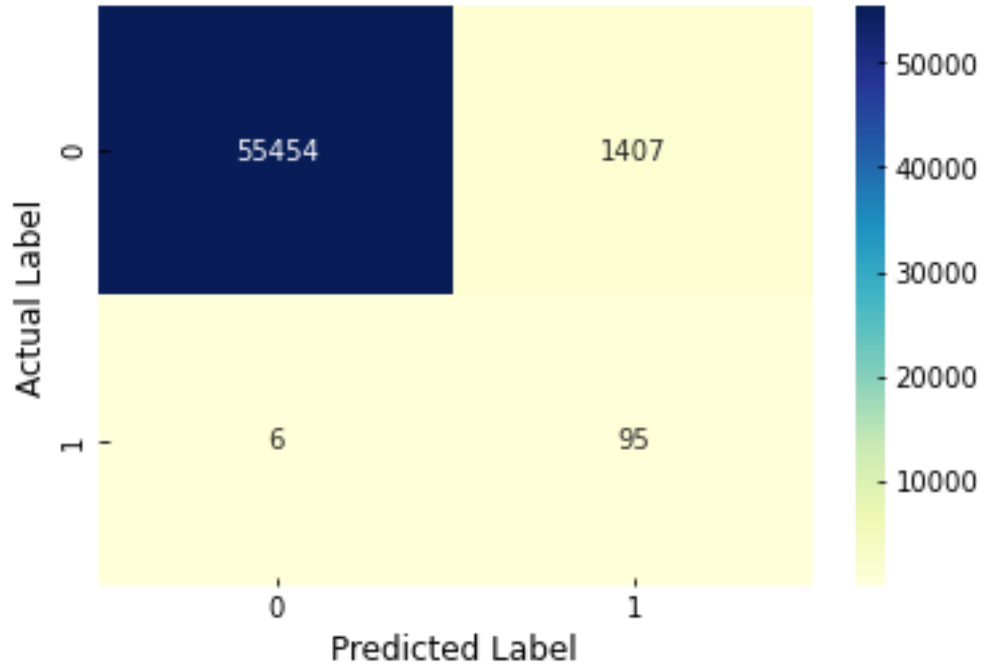
```
Transaction Number x_train dataset:  (227845, 30)
Transaction Number y_train dataset:  (227845, 1)
Transaction Number x_test dataset:  (56962, 30)
Transaction Number y_test dataset:  (56962, 1)
Before OverSampling, counts of label '1': [391]
Before OverSampling, counts of label '0': [227454]

After OverSampling, the shape of train_x: (454908, 30)
After OverSampling, the shape of train_y: (454908,)

After OverSampling, counts of label '1', %: 50.0
After OverSampling, counts of label '0', %: 50.0
```

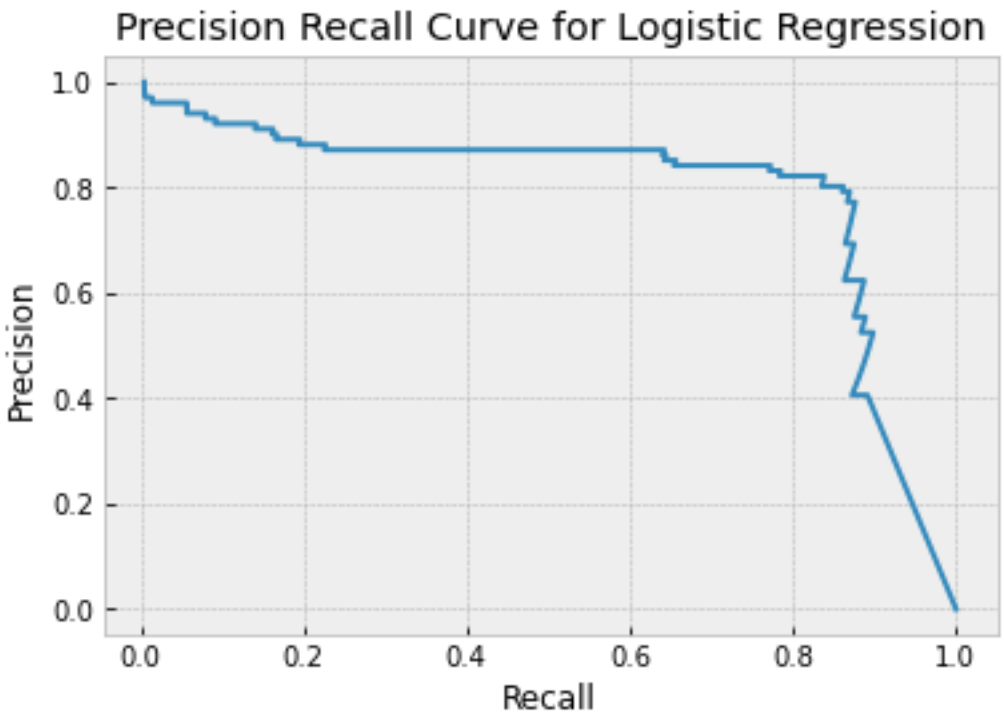



Dataset is balanced after applying SMOTE, in the next step I have built the logistic regression. One important thing to point out here is that I used SMOTE after cross validation in order to avoid data leakage problem and hence overfitting.

# II) Logistic Regression with SMOTE

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.98 | 0.99 | 56861 |
| 1 | 0.06 | 0.94 | 0.12 | 101 |
| **accuracy** | | | 0.98 | 56962 |
| **macro avg** | 0.53 | 0.96 | 0.55 | 56962 |
| **weighted avg** | 1.00 | 0.98 | 0.99 | 56962 |

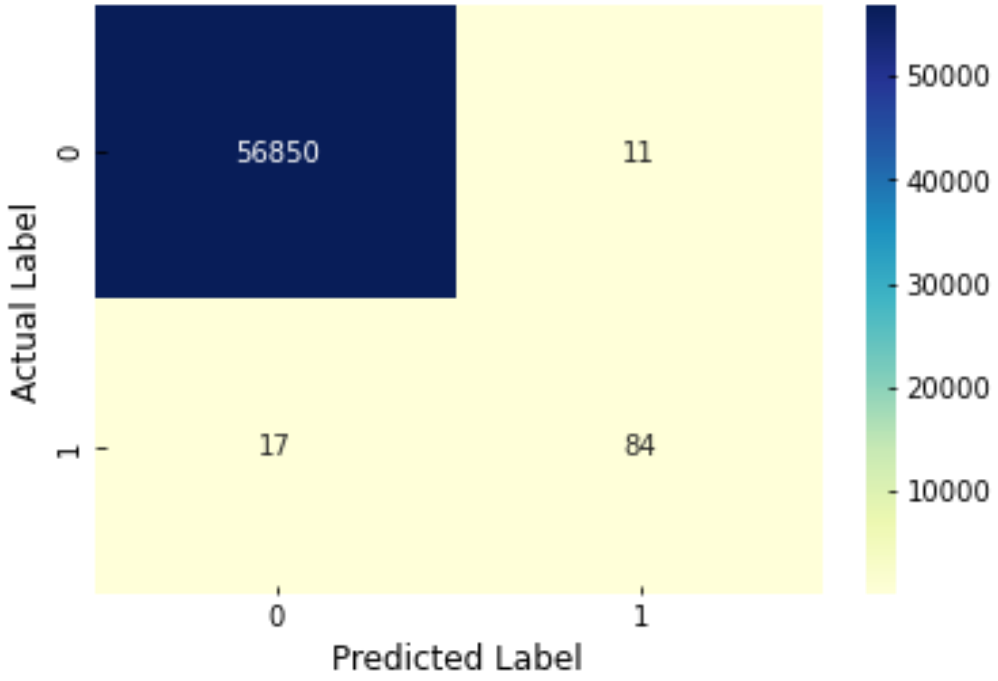


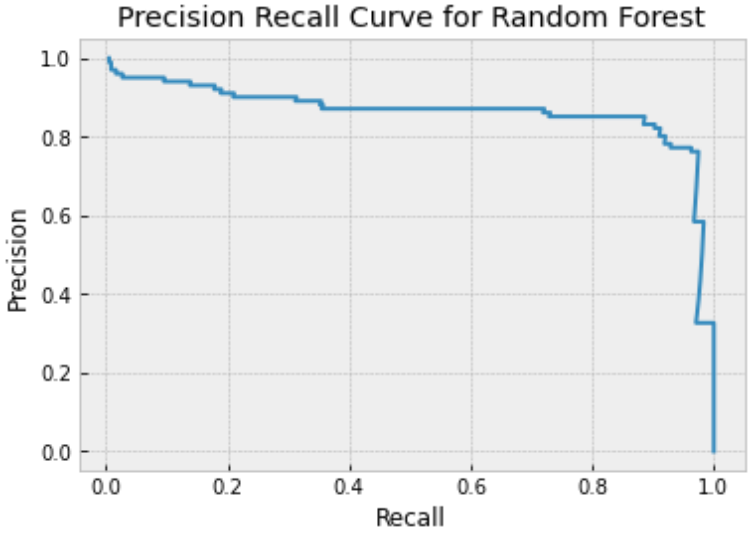



After applying SMOTE, Logistic Regression Model produced better results. Recall value is high (94% with SMOTE vs 63% without SMOTE), which means model is able to detect the highest number of fraud transactions. Low precision value (6%) indicates that model classified a lot of non-fraud transactions as fraud. The customers of a financial institution are not going to be satisfied with that fact and may even stop using the service of that financial institution. In the next step, I have built Random Forest model to achieve good precision.

# III) Random Forest Classifier with SMOTE

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56861 |
| 1 | 0.88 | 0.83 | 0.86 | 101 |
| **accuracy** | | | 1.00 | 56962 |
| **macro avg** | 0.94 | 0.92 | 0.93 | 56962 |
| **weighted avg** | 1.00 | 1.00 | 1.00 | 56962 |


56850
11
17
84
Actual Label
Predicted Label
0
1
0
1
50000
40000
30000
20000
10000


Precision Recall Curve for Random Forest
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Random Forest with SMOTE gave better results than Logistic Regression, model resulted in high recall and high precision. Even though the recall has decreased a little bit, Random Forest model increased the precision significantly.

**Conclusion**:

Random Forest Classifier gave us the best results, able to detect more than 80% fraud transactions correctly, at the same time not classifying a lot of non-fraud transactions as fraud.

| | Class | Precision% | Recall% | F1% |
|---|---|---|---|---|
| **Logistic Regression Without SMOTE** | Fraud | 88 | 63 | 74 |
| | Non- Fraud | 100 | 100 | 100 |
| **Logistic Regression With SMOTE** | Fraud | 6 | 94 | 12 |
| | Non- Fraud | 100 | 98 | 99 |
| **Random Forest With SMOTE** | Fraud | 88 | 83 | 86 |
| | Non- Fraud | 100 | 100 | 100 |

- **Recall of fraud cases (sensitivity)** summarizes true positive rate (True positive/True positive + False Negative) - how many cases we got correct out of all the positive ones
- **Recall of non-fraud (specificity)** summarizes true negative rate (True negative/True negative + False positive) - how many cases we got correct out of all the negative ones
- **Precision of fraud cases** (True positive/True positive + False positive) summarizes the accuracy of fraud cases detected - out of all predicted as fraud, how many are correct
- **Precision of non-fraud cases** (True negative/True negative + False negative) summarizes the accuracy of non-fraud cases detected - out of all predicted as non-fraud, how many are correct
- **F1-score** is the harmonic mean of recall and precision

**References:**


1. Jason Brownlee –

   **Imbalanced Classification with the Fraudulent Credit Card Transactions Dataset**

2. Credit Card Fraud Detection

3. Rafael Pierre - Detecting Financial Fraud Using Machine Learning: Winning the War Against Imbalanced Data

4. Jason Brownlee - Imbalanced Classification With Python (7-Day Mini-Course)

5. Kaggle Data Set