

# Credit Card Fraud Detection

Data Science With Python Lab Project Report

Bachelor  
in  
Computer Science

By  
**M.Jahnavi,J.Sunitha**

S190310

S190269



Rajiv Gandhi University Of Knowledge And Technologies

S.M. Puram , Srikakulam -532410

Andhra Pradesh, India

# Abstract

Today we are living in digital world where most of the activities performed are online .Fraud transactions are ever growing since the growth of e-commerce applications.Credit card fraud activities is presently the most frequently occurring problem in the current world.Billions of transactions are happening around every second.Fraud transactions are allowing users to misuse the money of genuine users causing them financial loss.Credit card fraud generally happens when card was stolen for any unauthorized purposes.According to the Nilson report, by 2025 due to credit card fraud the United States has been projected to suffer losses upto 12.5 billion dollars.To avoid all these fraudulent activities the credit card fraud detection system was introduced.The project aims to focus mainly on machine learning algorithms such as Random forest and Decision tree are algorithms used to know accuracy ,precision ,recall and F1-score.The ROC curve plotted based on the confusion matrix.The above two algorithms are compared and the algorithm that has greatest accuracy ,precision,recall,F1-score is considered as the best algorithm used to detect fraud.

# Contents

<b>Abstract</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Introduction to Your Project . . . . .	3
1.2 Application . . . . .	4
1.3 Motivation Towards Your Project . . . . .	4
1.4 Problem Statement . . . . .	4
<b>2 Approach To Your Project</b>	<b>5</b>
2.1 Explain About Your Project . . . . .	5
2.2 Data Set . . . . .	5
2.3 Prediction technique . . . . .	6
2.4 Graphs . . . . .	7
<b>3 Code</b>	<b>13</b>
3.1 Importing required modules . . . . .	13
3.2 Reading Data Sample . . . . .	13
3.3 Data Preprocessing and Preparing Datasets . . . . .	14
3.4 Exploratory Data Analysis(EDA) Visualization . . . . .	17
3.5 Handling Imbalanced Datasets . . . . .	19
3.6 Machine Learning Model Selection . . . . .	22
<b>4 Conclusion and Future Work</b>	<b>40</b>

# Chapter 1

## Introduction

### 1.1 Introduction to Your Project

Credit card is considered as a “nice target of fraud” since in a very short time attackers can get lots of money without much risk and most of the time the fraud is discovered after few days. To commit the credit card fraud either offline or online, fraudsters are looking for sensitive information such as credit card number, bank account and social security numbers. Thus, with this massive problem in transaction system, banks take credit card fraud very seriously, and have highly sophisticated security systems to monitor transactions and detect the frauds as quickly as possible once it is committed. A secured and trusted banking payment system requires high speed verification and authentication mechanisms that let legitimate users easily conduct their business, while flagging and detecting suspicious transaction attempts by others. Fraud detection has become a vital activity in order to decrease the impact of fraudulent transactions on service delivery, costs, and reputation of the company. According to Kount, one of the top five fraud detection consultants revealed by [topcreditcardprocessors.com](http://topcreditcardprocessors.com) in the month of August 2013, 40% of the total financial fraud is related to credit card and the loss of amount due to credit card fraud worldwide is \$5.55 billion. There are various methods used for fraud detection each of them tries to increase the detection rate while keeping false alarm rate at minimum. Different methods have been used for fraud detection including such as, Logistic Regression, Decision Tree, Random Forest etc.

## 1.2 Application

To commit the credit card fraud either offline or online, fraudsters are looking for sensitive information such as credit card number, bank account and social security numbers. In case of offline payment to perform the fraudulent transactions, an attacker has to steal the credit card itself, while in the case of online payment, the fraudsters should be steal costumer's identity.

A credit card company uses a dataset of past transactions labeled as fraudulent or legitimate. They train a logistic regression or a random forest classifier on this data and then deploy the trained model in their payment processing system to assess the risk of each incoming transaction. If the model predicts a high probability of fraud, the transaction is flagged for further investigation or declined.

## 1.3 Motivation Towards Your Project

Fraud detection is a set of activities undertaken to prevent money or property from being obtained through false pretenses. Fraud detection is applied to many industries such as banking or insurance. In banking, fraud may include forging checks or using stolen credit cards. A credit card fraud detection algorithm consists in identifying those transactions with a high probability of being fraud, based on historical fraud patterns. Machine learning, having three types, from that also the supervised and hybrid approach is more suitable for fraud detection. Preventing known and unknown fraud in real time is not easy but it is feasible. The proposed architecture was originally designed to detect credit card fraud in online payments and emphasize on providing a fraud prevention mechanism to verify the transaction is fraudulent or legitimate.

## 1.4 Problem Statement

The Credit Card Fraud Detection Problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be a fraud. This model is then used to identify whether a new transaction is fraudulent or not.

# Chapter 2

## Approach To Your Project

### 2.1 Explain About Your Project

Credit card fraud detection is the process of identifying purchase attempts that are fraudulent and rejecting them rather than processing the order. There are a variety of tools and techniques available for detecting fraud, with most merchants employing a combination of several of them.

The credit card fraud detection features uses user behavior and location scanning to check for unusual patterns. These patterns include user characteristics such as user spending patterns as well as usual user geographic locations to verify his identity. If any unusual pattern is detected, the system requires revivification.

The system analyses user credit card data for various characteristics. These characteristics include user country, usual spending procedures. Based upon previous data of that user the system recognizes unusual patterns in the payment procedure. So now the system may require the user to login again or even block the user for more than 3 invalid attempts.

### 2.2 Data Set

This datasets have 492 frauds out of 284,807 transactions. It is highly unbalanced, the positive class-1 (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation.

Due to confidentiality issues, the original features are not provided and more background information about the data. Features  $V_1, V_2, \dots, V_{28}$  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. But, we did not consider Time for training purpose as it is of no use to build the models and may not impact our target variable.

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.

## 2.3 Prediction technique

We are using already predefined machine learning techniques from the scikit learn library based on the Accuracy and sensitivity measures we pick up the best one. Actually Random Forest Classifier works well on our data.

RandomForest works even efficiently for this imbalanced datasets. RandomForest takes around 10-15 minutes for training. Maximum Accuracy of 99.996483% and macro-average of F1-Score of 1.00 achieved with Oversampling technique.

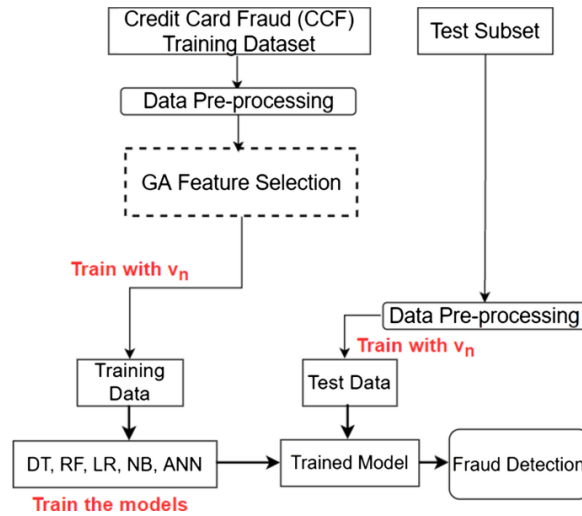


Figure 2.1

## 2.4 Graphs

Let us plot a bar graph

```
fig = plt.figure()
fig.set_figheight(8)
fig.set_figwidth(10)
ax = sns.barplot(x=class_count_df['Class'],
                 y=class_count_df['Counts'])
ax.bar_label(ax.containers[0], color='red') ## Showing Values at top of
↪ Each Bar.
ax.set_xticklabels(labels=list(class_count_df['Class']), c='blue',
↪ rotation=0, fontsize=10, fontweight='bold')
labels, location = plt.yticks()
ax.set_yticklabels(labels=labels.astype(int), c='blue', fontsize=8,
↪ fontweight='bold')
plt.xlabel(xlabel='Type of Transactions', fontsize=14,
↪ fontweight='bold').set_color('purple')
plt.ylabel(ylabel='Frequency', fontsize=14,
↪ fontweight='bold').set_color('purple')
plt.title(label='Count Values of Normal vs Fraud Class', fontsize=24,
↪ fontweight='bold').set_color('purple')
```



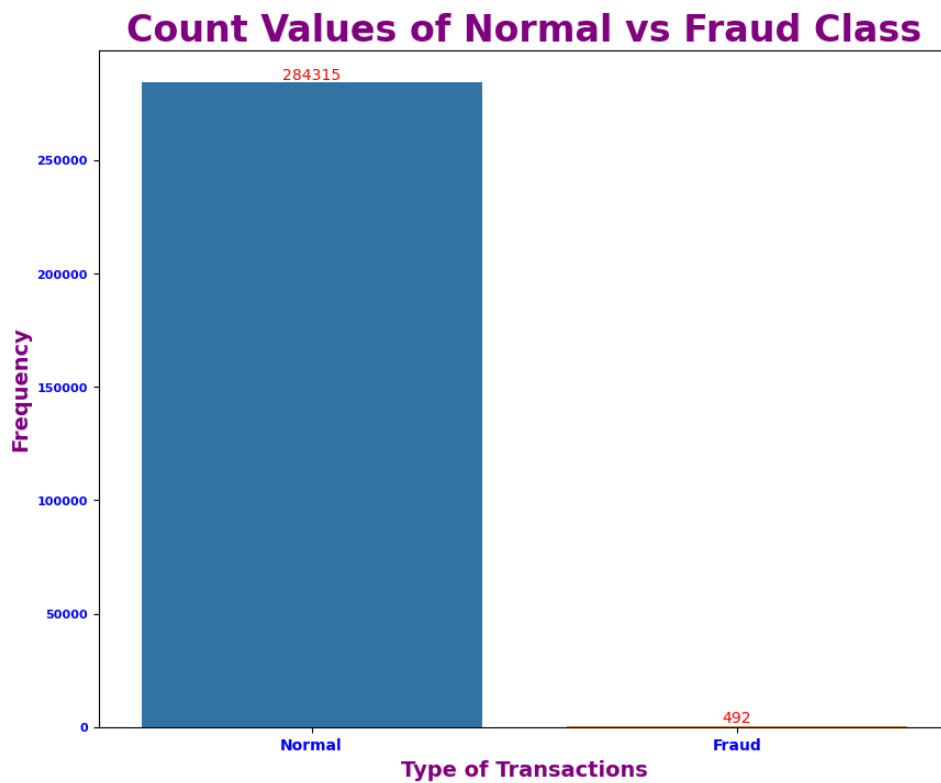


Figure 2.2

Let us plot a Histogram

```
fig, (ax0, ax1) = plt.subplots(nrows=2,
                                ncols=1,
                                sharex=True)

fig.suptitle("Variation of Amount per Class", color='green')
bins=50

ax0.hist(fraud['Amount'], bins=bins, color='red')
ax0.set_title('Fraud')
ax0.set_ylim(0, 100)
ax0.set_ylabel('No. of Transactions')

ax1.hist(normal['Amount'], bins=bins, color='red')
ax1.set_title('Normal')
ax1.set_ylabel('No. of Transactions')
```

```
plt.xlim(0, 20000)

plt.xlabel('Amount ($)')

plt.yscale('log')
```

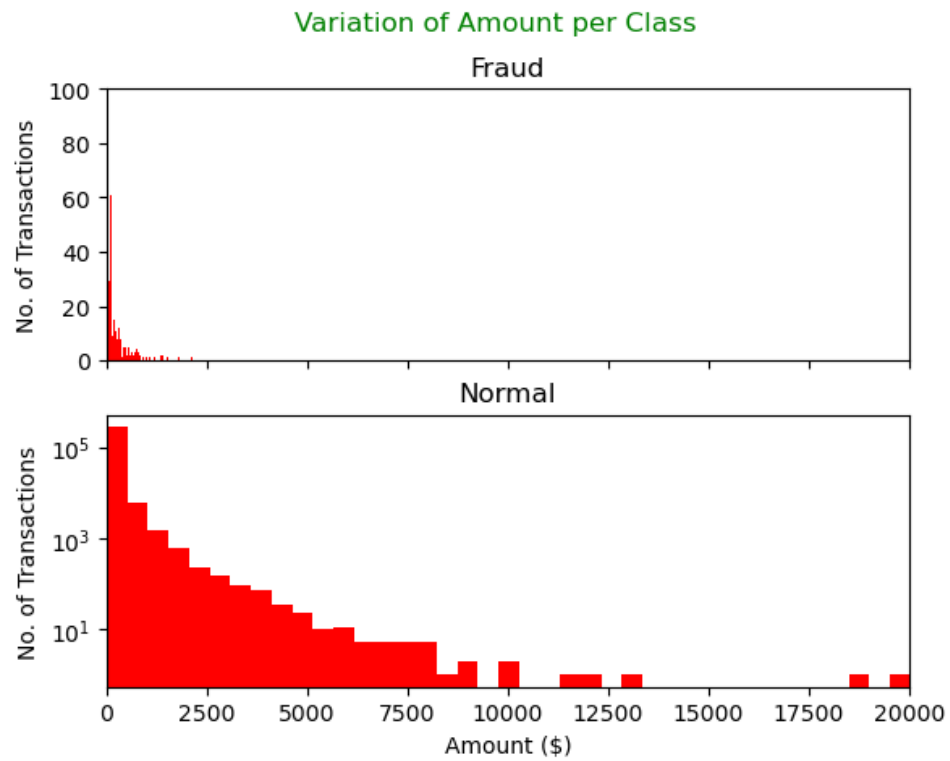


Figure 2.3

Let us plot a scatter plot

Let us plot a scatter plot between Amount and Class

```
main_df.plot.scatter(x="Class",y="Amount")
```

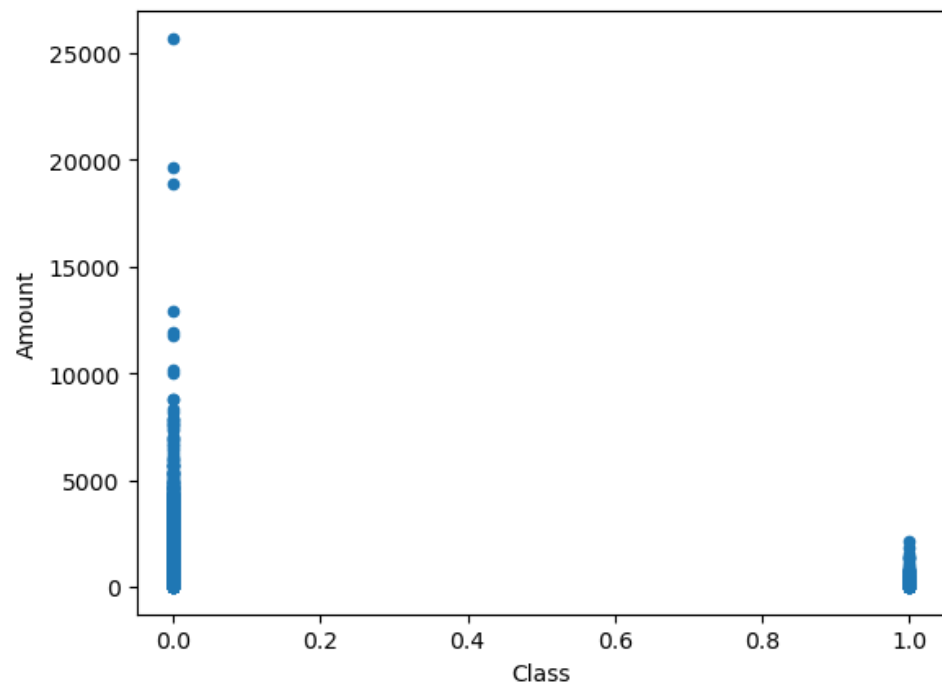


Figure 2.4

Let us plot a Line plot

Let us plot a Line plots for Amount and Class

```
main_df.plot.scatter(x="Amount")
```

```
main_df.plot.scatter(x="Class")
```

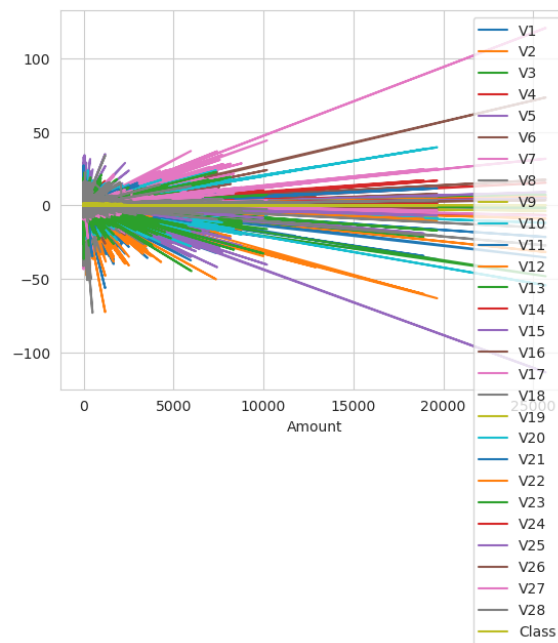


Figure 2.5: Line plot for Amount

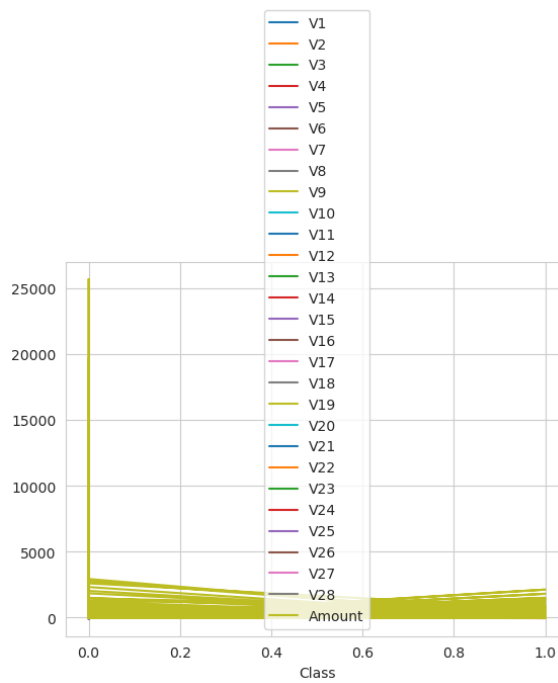


Figure 2.6: Line plot for Class

Let us plot a pie chart

Let us plot a Line plot between fraud and normal data

```
classes= ["fraud","normal"]  
data = [492,284315]  
fig = plt.figure(figsize =(10, 7))  
plt.pie(data, labels = classes)  
plt.show()
```

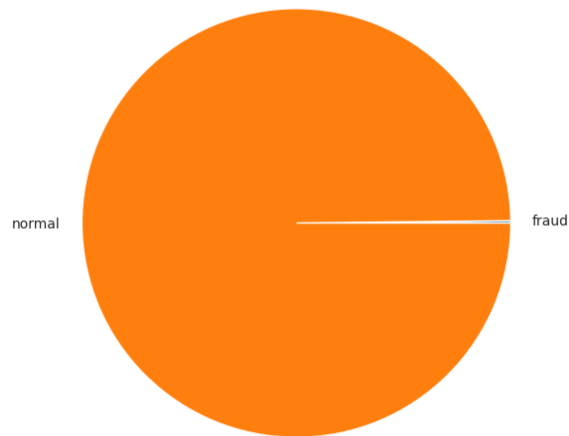


Figure 2.7

# Chapter 3

## Code

### 3.1 Importing required modules

we are importing pandas for data manipulating numpy for calculations , pyplot for graphs , seaborn for effective plots

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

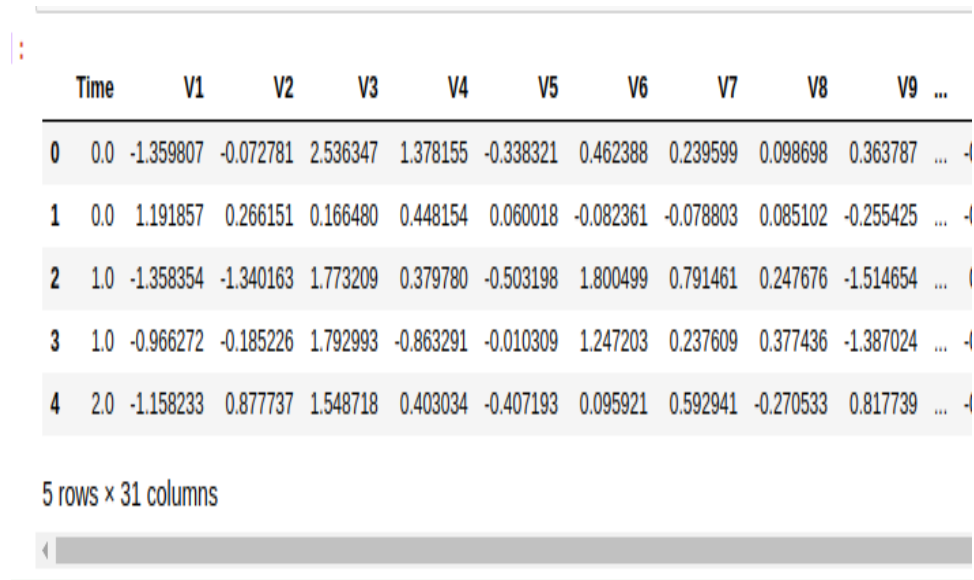
### 3.2 Reading Data Sample

Now I want to see some rows and columns

To see the first 5 rows, we use df.head(5) function

```
#code

df = pd.read_csv("creditcard.csv")
df.head(5)
```



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...

5 rows x 31 columns

Figure 3.1

### 3.3 Data Preprocessing and Preparing Datasets

Checking for any null values..

*#code*

```
main_df.isna().sum()
```

```
main_df.isnull().values.any()
```

Output

```
Time      0
```

```
V1        0
```

```
V2        0
```

```
.         .
```

```
.         .
```

```
V27       0
```

```
V28       0
```

```
Amount    0
```

```
Class     0
```

```
dtype: int64
```

False

## Checking info of our datasets...

*#code*

```
main_df.info()
```

Output

```
<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
.	.	.	..
.	.	.	..
29	Amount	284807 non-null	float64
30	Class	284807 non-null	int64

```
dtypes: float64(30), int64(1)
```

```
memory usage: 67.4 MB
```

## Observing Statistical Distribution of Datasets

*#code*

```
main_df.describe()
```

Output



	Time	V1	V2	V3	V4
<b>count</b>	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
<b>mean</b>	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15
<b>std</b>	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00
<b>min</b>	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+01
<b>25%</b>	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01
<b>50%</b>	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-01
<b>75%</b>	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01
<b>max</b>	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01

8 rows × 6 columns

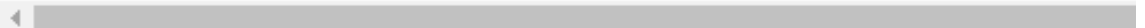


Figure 3.2

Dropping Time Attribute

It is of no use to build the models and may not impact our target variable.

*#code*

```
main_df.drop('Time', axis=1, inplace=True)
```

Now, X and y are our normal datasets

*#code*

```
X = main_df.drop('Class', axis=1)
```

```
y = main_df['Class']
```

```
X.shape, y.shape
```

Output ((284807, 29), (284807,))

Normalizing Amount attribute values using StandardScaler

*#code*

```
temp_df = main_df.copy()
```

```
from sklearn.preprocessing import StandardScaler
```

```

standard = StandardScaler()

amount = temp_df['Amount'].values

temp_df['Amount'] = standard.fit_transform(amount.reshape(-1,1))

X_scaled = temp_df.drop('Class', axis=1)

y_scaled = temp_df['Class']

X_scaled.shape, y_scaled.shape

```

Output ((284807, 29), (284807,))

### 3.4 Exploratory Data Analysis(EDA) Visualization

Making data ready for Plotting of Count values of Different Classes

```

#code

class_count_df =

↪ pd.DataFrame(main_df['Class'].value_counts().rename_axis('Class').reset_index())

class_count_df['Class'].replace({0: 'Normal', 1: 'Fraud'}, inplace=True)

class_count_df.head()

```

Output

	Class	Counts
0	Normal	284315
1	Fraud	492

Analyzing before plotting graph of Variation of Amount per Class

```

#code

fraud = main_df[main_df['Class'] == 1]

normal = main_df[main_df['Class'] == 0]

fraud.shape, normal.shape

#Output

```

```

((492, 30), (284315, 30))

#code
fraud_frc = len(fraud)/float(len(main_df))

fraud_frc

#Output
0.001727485630620034

#code
print(f"Percentage of Fraud Tnx in datasets :{fraud_frc*100:.2f}%")

#output
Percentage of Fraud Tnx in datasets : 0.17%

#code
fraud.Amount.describe()

#output
count      492.000000
mean       122.211321
std        256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%        105.890000
max        2125.870000
Name: Amount, dtype: float64

#code
normal.Amount.describe()

#output
count      284315.000000
mean       88.291022

```

```
std          250.105092
min          0.000000
25%          5.650000
50%          22.000000
75%          77.050000
max          25691.160000
Name: Amount, dtype: float64
```

## 3.5 Handling Imbalanced Datasets

### 1) Choose Proper Evaluation Metrics

Accuracy may be good enough for a well-balanced class but not ideal for the imbalanced class problem. The other metrics like precision( measure of how accurate the classifier's prediction of a specific class ) and recall ( measure of the classifier's ability to identify a class ) are also considered.

For an imbalanced class dataset, F1 score is a more appropriate metric. F1 score is defined as the harmonic mean between precision and recall. It is used as a statistical measure to rate performance. F1-score ranges between 0 and 1. The closer it is to 1, the better the model.

### 2) Resampling(Undersampling and Oversampling)

#### OverSampling

```
#code

main_df.Class.value_counts()

#Output

0      284315
1         492

Name: Class, dtype: int64

#code

from sklearn.utils import resample
```

```

#create two different dataframe of majority and minority class

df_majority = main_df[(main_df['Class']==0)]
df_minority = main_df[(main_df['Class']==1)]

# upsample minority class

df_minority_oversampled = resample(df_minority,
                                   replace=True,
                                   n_samples=284315,
                                   random_state=42)

# Combine majority class with upsampled minority class

df_oversampled = pd.concat([df_minority_oversampled, df_majority])
df_oversampled.Class.value_counts()

#Output

1      284315
0      284315

Name: Class, dtype: int64

\par Final Sampled Dataset:

#code

X_oversampled = df_oversampled.drop('Class', axis=1)
y_oversampled = df_oversampled['Class']
X_oversampled.shape, y_oversampled.shape

#Output

((568630, 29), (568630,))

```

## Undersampling

```

#code

from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

```

```

rus = RandomUnderSampler(random_state=42)

X_undersampled, y_undersampled = rus.fit_resample(X, y)

print(f"The number of Classes before the fit {Counter(y)}")

print(f"The number of Classes after the fit
↪ {Counter(y_undersampled)}")

#Output

The number of Classes before the fit Counter({0: 284315, 1: 492})

The number of Classes after the fit Counter({0: 492, 1: 492})

```

### 3) SMOTE(Synthetic Minority Oversampling Technique)

Simply adding duplicate records of minority class often don't add any new information to the model. In SMOTE new instances are synthesized from the existing data. If we explain it in simple words, SMOTE looks into minority class instances and use k nearest neighbor to select a random nearest neighbor, and a synthetic instance is created randomly in feature space.

```

#code

from imblearn.over_sampling import SMOTE

# Resampling the minority class. The strategy can be changed as
↪ required.

sm = SMOTE(sampling_strategy='minority', random_state=42)

# Fit the model to generate the data.

X_smote, y_smote = sm.fit_resample(main_df.drop('Class', axis=1),
↪ main_df['Class'])

smote_df = pd.concat([pd.DataFrame(X_smote), pd.DataFrame(y_smote)],
↪ axis=1)

X_smote.shape

#output

(568630, 29)

```

```

#code
X_smote.shape

#output
(568630, 29)

#code
smote_df.Class.value_counts()

#output
0      284315
1      284315

Name: Class, dtype: int64

```

## 3.6 Machine Learning Model Selection

This project mainly focuses on handling imbalanced datasets and detecting credit-card frauds using Following Machine Learning Algorithms:

**Models used :**

- Logistic Regression
- RandomForestClassifier
- DecisionTreeClassifier
- KNeighbors Classifier

These models are fitted to different datasets acquired after StandardScaler, Oversampling, Undersampling and SMOTE techniques. Thus, separate files are created for each Machine Learning Models so that every datasets acquired after above mentioned techniques are fitted separately to our model using single function.

### Importing required modules

First, let us import necessary libraries

```

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix,
↪ accuracy_score

```

## Decision Tree Classifier

### Function For Model Fitting, Model Evaluation and Visualization

```

def DT_model(X,y):

    from sklearn.model_selection import train_test_split

    np.random.seed(42)

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

    dt = DecisionTreeClassifier()

    dt.fit(X_train, y_train)

    print("-----Training Prediction-----")

    y_preds = dt.predict(X_train)

    print(f"Classification Report:\n\n{classification_report(y_train,
↪ y_preds)}\n\n")

    cf_matrix = confusion_matrix(y_train, y_preds)

    fig, ax = plt.subplots(figsize=(6,4))

    sns.heatmap(cf_matrix, annot=True, cmap='coolwarm', fmt='g')

    fig.suptitle(t="Confusion Matrix",

                color="orange",

                fontsize=16);

    ax.set(xlabel="Predicted Label",

```



```

        ylabel="Actual Label");

print(f"Accuracy Score:\n\n{accuracy_score(y_train,
↪ y_preds)*100:2f}%\n")

print("-----Test Prediction-----")

y_preds = dt.predict(X_test)

print(f"Classfication Report:\n\n{classification_report(y_test,
↪ y_preds)}\n\n")

cf_matrix = confusion_matrix(y_test, y_preds)

fig, ax = plt.subplots(figsize=(6,4))

sns.heatmap(cf_matrix, annot=True, cmap='coolwarm', fmt='g')

fig.suptitle(t="Confusion Matrix",

            color="orange",

            fontsize=16);

ax.set(xlabel="Predicted Label",

       ylabel="Actual Label");

print(f"Accuracy Score:\n\n{accuracy_score(y_test,
↪ y_preds)*100:2f}%\n")

```

## Decision Tree On Normal Datasets

*#code*

```
DT_model(X,y)
```

*#ouput*

```
-----Training Prediction-----
```

Classfication Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	227451
1	1.00	1.00	1.00	394
accuracy			1.00	227845

macro avg	1.00	1.00	1.00	227845
weighted avg	1.00	1.00	1.00	227845

Accuracy Score: 100.000000%

-----Test Prediction-----

Classfifcation Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.74	0.80	0.76	98
accuracy			1.00	56962
macro avg	0.87	0.90	0.88	56962
weighted avg	1.00	1.00	1.00	56962

Accuracy Score: 99.915733%

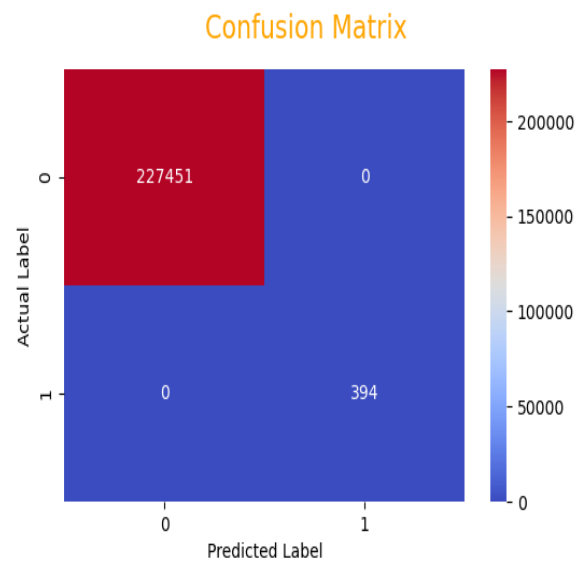


Figure 3.3

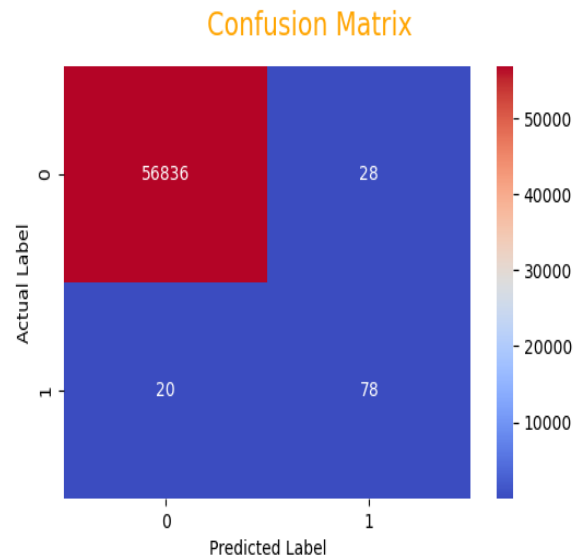


Figure 3.4

### Decision Tree On UnderSampled Dataset

*#code*

```
DT_model(X_undersampled,y_undersampled)
```

*#output*

-----Training Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	393
1	1.00	1.00	1.00	394
accuracy			1.00	787
macro avg	1.00	1.00	1.00	787
weighted avg	1.00	1.00	1.00	787

Accuracy Score:100.000000%

-----Test Prediction-----

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.87	0.91	0.89	99
1	0.90	0.86	0.88	98
accuracy			0.88	197
macro avg	0.88	0.88	0.88	197
weighted avg	0.88	0.88	0.88	197

Accuracy Score:88.324873%

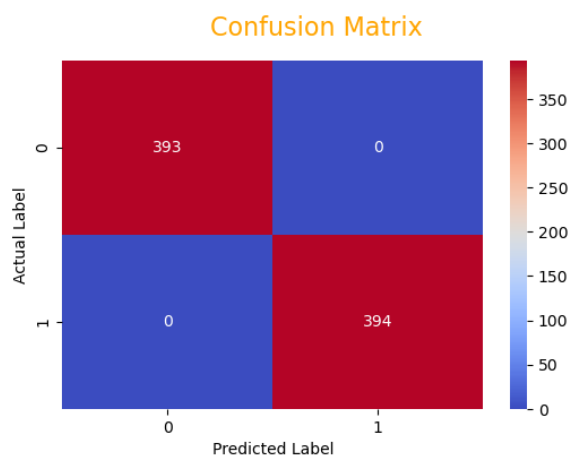


Figure 3.5

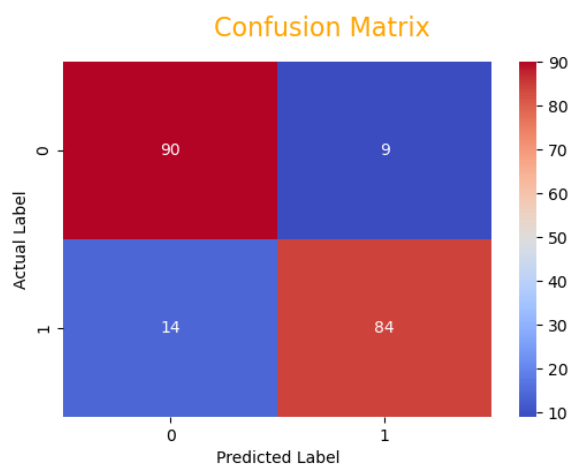


Figure 3.6

## Decision Tree On SMOTE Dataset

*#code*

```
DT_model(X_smote,y_smote)
```

*#Output*

-----Training Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	227565
1	1.00	1.00	1.00	227339
accuracy			1.00	454904
macro avg	1.00	1.00	1.00	454904
weighted avg	1.00	1.00	1.00	454904

Accuracy Score:100.000000%

-----Test Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56750
1	1.00	1.00	1.00	56976
accuracy			1.00	113726
macro avg	1.00	1.00	1.00	113726
weighted avg	1.00	1.00	1.00	113726

Accuracy Score:99.819742%

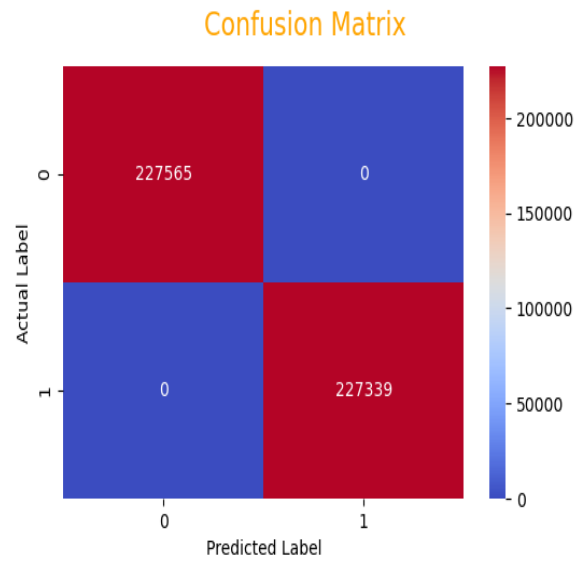


Figure 3.7

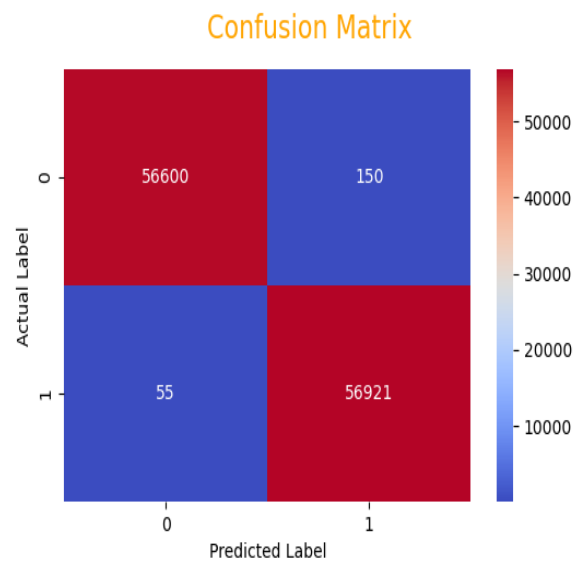


Figure 3.8

## Random Forest Classifier

### Function For Model Fitting, Model Evaluation and Visualization

```
#code

def RF_model(X,y):

    from sklearn.model_selection import train_test_split
```

```

np.random.seed(42)

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

rf = RandomForestClassifier()

rf.fit(X_train, y_train)

from sklearn.metrics import classification_report, confusion_matrix,
    ↪ accuracy_score

print("-----Training Prediction-----")

y_preds = rf.predict(X_train)

print(f"Classfifcation Report:\n\n{classification_report(y_train,
    ↪ y_preds)}\n\n")

cf_matrix = confusion_matrix(y_train, y_preds)

fig, ax = plt.subplots(figsize=(6,4))

sns.heatmap(cf_matrix, annot=True, cmap='coolwarm', fmt='g')

fig.suptitle(t="Confusion Matrix of Training Datasets",
    color="orange",
    fontsize=16);

ax.set(xlabel="Predicted Label",
    ylabel="Actual Label");

print(f"Accuracy Score:\n\n{accuracy_score(y_train,
    ↪ y_preds)*100:2f}%\n\n")

print("-----Test Prediction-----")

y_preds = rf.predict(X_test)

print(f"Classfifcation Report:\n\n{classification_report(y_test,
    ↪ y_preds)}\n\n")

cf_matrix = confusion_matrix(y_test, y_preds)

fig, ax = plt.subplots(figsize=(6,4))

```

```

sns.heatmap(cf_matrix, annot=True, cmap='coolwarm', fmt='g')
fig.suptitle(t="Confusion Matrix of Testing Datasets",
             color="orange",
             fontsize=16);
ax.set(xlabel="Predicted Label",
      ylabel="Actual Label");
print(f"Accuracy Score:\n\n{accuracy_score(y_test,
↪ y_preds)*100:2f}%\n")

```

### RandomForestClassifier on Normal Datasets.

*#code*

```
RF_model(main.X, main.y)
```

*#Output*

-----Training Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	227451
1	1.00	1.00	1.00	394
accuracy			1.00	227845
macro avg	1.00	1.00	1.00	227845
weighted avg	1.00	1.00	1.00	227845

Accuracy Score: 100.000000%

-----Test Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.97	0.78	0.86	98
accuracy			1.00	56962



macro avg	0.99	0.89	0.93	56962
weighted avg	1.00	1.00	1.00	56962

Accuracy Score: 99.957867%

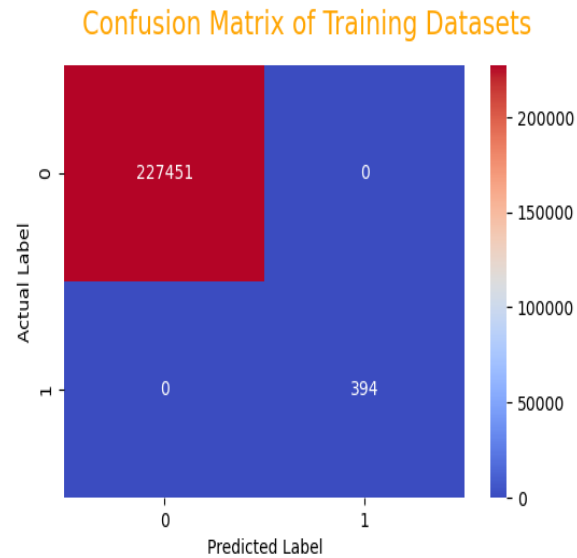


Figure 3.9

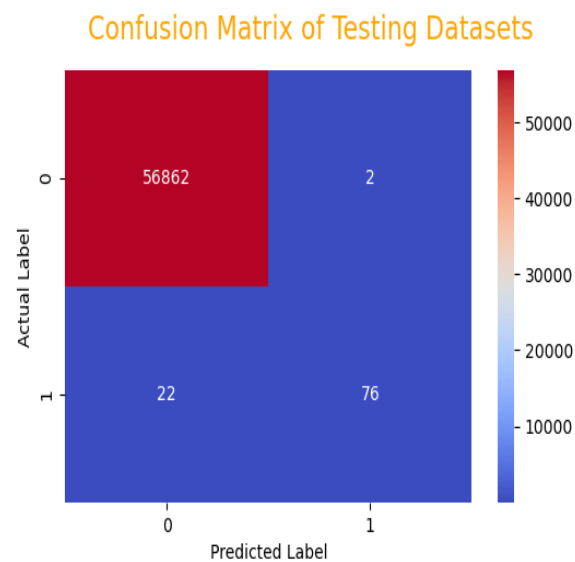


Figure 3.10

### RandomForestClassifier on StandardScaled Dataset

*#code*

```
RF_model(main.X_scaled, main.y_scaled)
```

*#Output*

-----Training Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	227451
1	1.00	1.00	1.00	394
accuracy			1.00	227845
macro avg	1.00	1.00	1.00	227845
weighted avg	1.00	1.00	1.00	227845

Accuracy Score: 100.000000%

-----Test Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.97	0.78	0.86	98
accuracy			1.00	56962
macro avg	0.99	0.89	0.93	56962
weighted avg	1.00	1.00	1.00	56962

Accuracy Score: 99.957867%

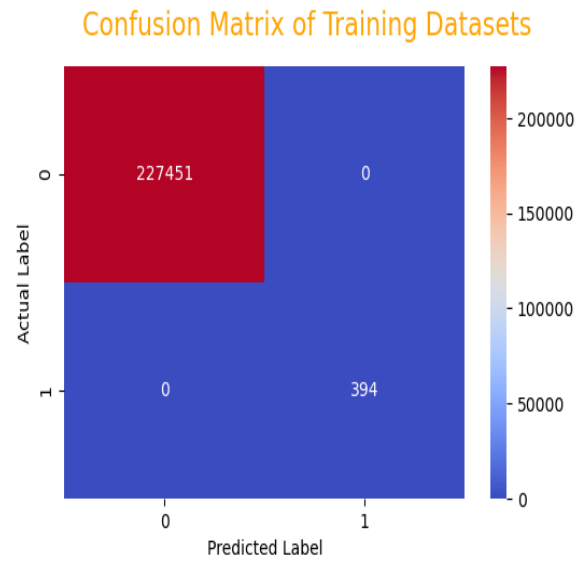


Figure 3.11

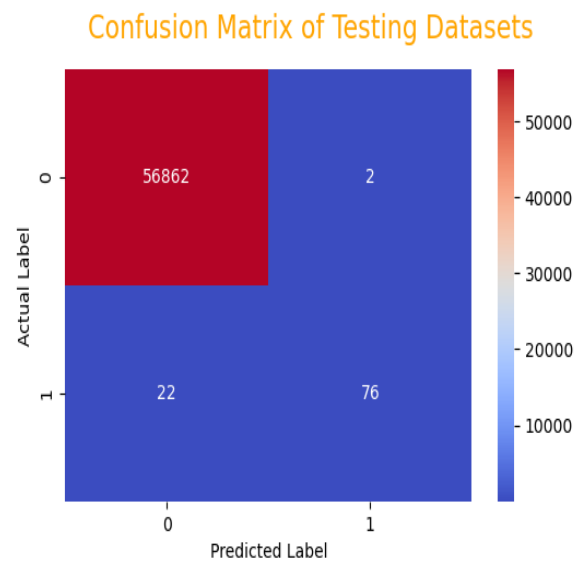


Figure 3.12

### RandomForestClassifier on Undersampled Dataset

*#code*

```
RF_model(main.X_undersampled, main.y_undersampled)
```

*#Output*

-----Training Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	393
1	1.00	1.00	1.00	394
accuracy			1.00	787
macro avg	1.00	1.00	1.00	787
weighted avg	1.00	1.00	1.00	787

Accuracy Score: 100.000000%

-----Test Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.96	0.93	99
1	0.96	0.89	0.92	98
accuracy			0.92	197
macro avg	0.93	0.92	0.92	197
weighted avg	0.93	0.92	0.92	197

Accuracy Score: 92.385787%

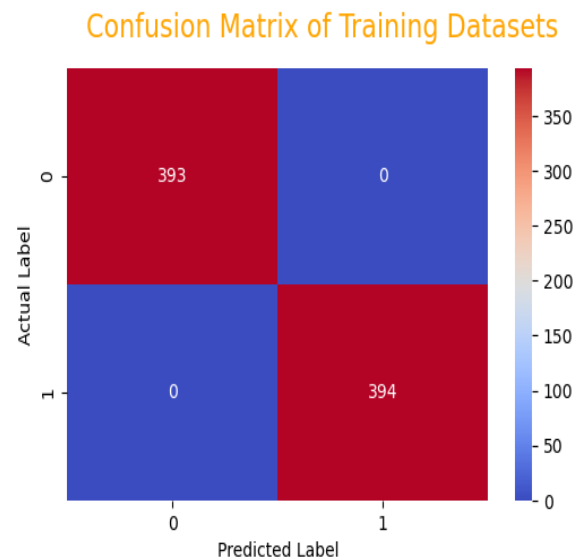


Figure 3.13

Confusion Matrix of Testing Datasets

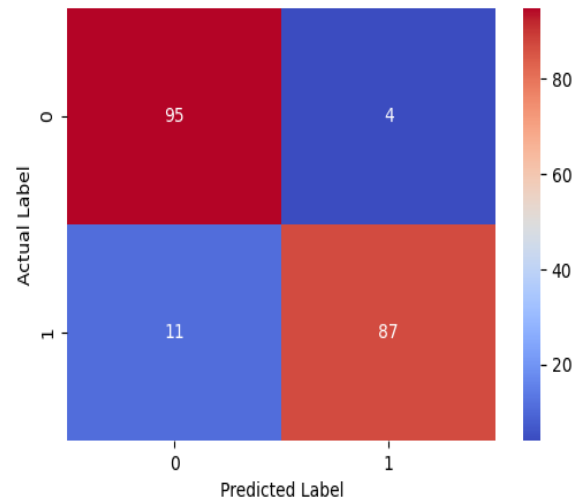


Figure 3.14

## RandomForestClassifier on Oversampled Dataset

*#code*

```
RF_model(X_oversampled,y_oversampled)
```

*#output*

-----Training Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	227335
1	1.00	1.00	1.00	227569
accuracy			1.00	454904
macro avg	1.00	1.00	1.00	454904
weighted avg	1.00	1.00	1.00	454904

Accuracy Score: 100.000000%

-----Test Prediction-----

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56980

1	1.00	1.00	1.00	56746
accuracy			1.00	113726
macro avg	1.00	1.00	1.00	113726
weighted avg	1.00	1.00	1.00	113726

Accuracy Score: 99.996483%

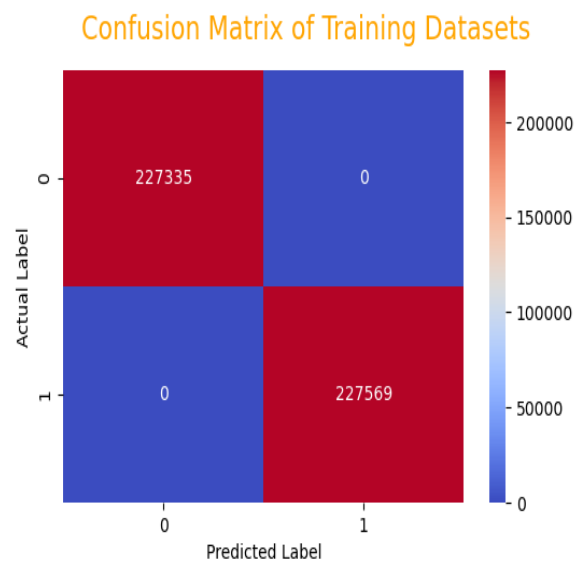


Figure 3.15

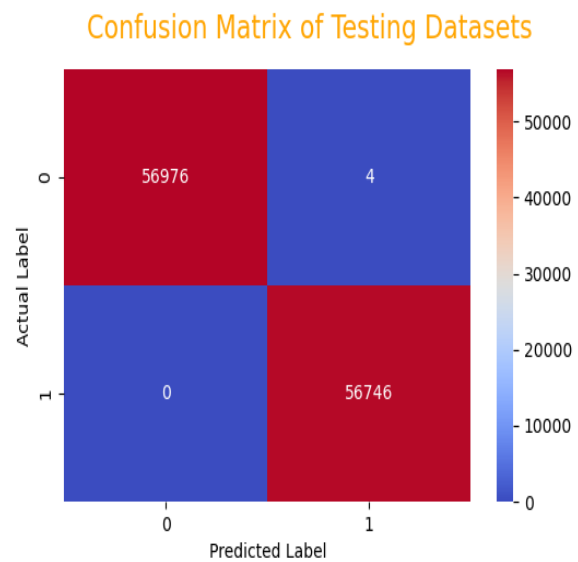


Figure 3.16

RandomForestClassifier on SMOTE Dataset

```

#code

RF_model(main.X_smote, main.y_smote)

#output

-----Training Prediction-----

Classfifcation Report:

              precision    recall  f1-score   support

     0           1.00       1.00       1.00     227565
     1           1.00       1.00       1.00     227339

 accuracy              1.00       1.00       1.00     454904
 macro avg           1.00       1.00       1.00     454904
weighted avg           1.00       1.00       1.00     454904

Accuracy Score: 100.000000%

-----Test Prediction-----

Classfifcation Report:

              precision    recall  f1-score   support

     0           1.00       1.00       1.00      56750
     1           1.00       1.00       1.00      56976

 accuracy              1.00       1.00       1.00     113726
 macro avg           1.00       1.00       1.00     113726
weighted avg           1.00       1.00       1.00     113726

Accuracy Score: 99.990328%

```

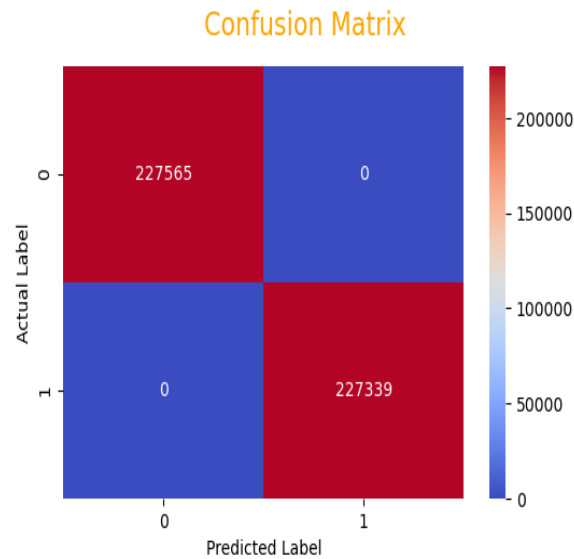


Figure 3.17

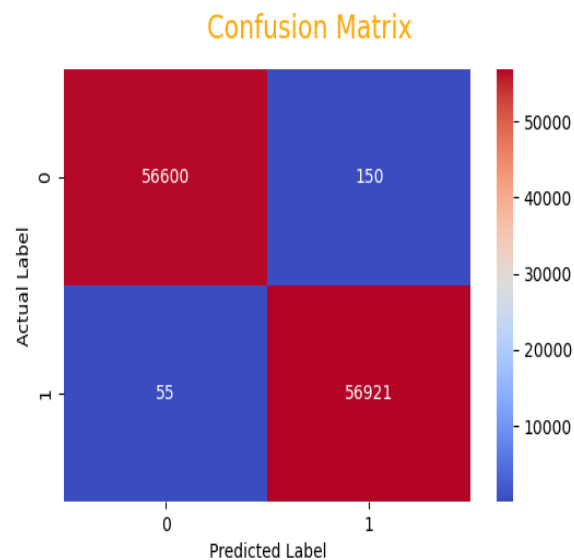


Figure 3.18

### Observations:

- a) Undersampling doesn't work efficiently for Large majority class datasets as it ignore many valuable tuples. But, can be efficient for small majority class datasets
- b) RandomForest works even efficiently for this imbalanced datasets.
- c) RandomForest takes around 10-15 minutes for training.
- d) Maximum Accuracy of 99.996483% and macro-average of F1-Score of 1.00 acheived with Oversampling technique



# Chapter 4

## Conclusion and Future Work

We summarize all the models and their calculations in the tabular format

Models	Accuracy
Logistic Regression	94.443839
Decision Tree	99.781969
Random Forest	99.991824

a) Out of all 3 Machine Learning Models used, Random Forest Classifier works efficiently with Maximum Accuracy of 99.996483% and macro-average of F1-Score of 1.00 achieved with Oversampling technique.

b) Oversampling Techniques proved to be efficient for handling Imbalanced Datasets.

c) RandomForest, XGBoost, DecisionTree, K-Neighbors work efficiently even for this Imbalanced Datasets.

d) RandomForest takes lots of Training Time among all of Six models used.

### Future Work

a) I've used default hyperparameters during model instantiation. You can play with hyperparameters for improving its efficiency.

b) You can apply other machine learning models as well.