Configuration Manual

MSc Research Project

Fintech

Jeevan Kumar A

X21225940

School of Computing

National College of Ireland

Supervisor: Brian Byrne

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……………………Jeevan Kumar A…………………………………………………………………

Student ID: ………………………X21225940…………………………………………………………………………………

Programme: …MSc Fintech…………………………………………… **Year:**2022/2023

Module: …Msc Research Project…………………………………………………

Supervisor: …………Brain Byrne………………………………………………

Submission Due Date: ……………14/08/2023…………………………………………

Project Title: **A Comprehensive Study of SMOTE-Enhanced Machine Learning Models on Learning Models on Credit Card Fraud Dataset**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary.

Signature: ……Jeevan Kumar A…………………………………………………………………

Date: ……………14/08/2023………………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Introduction

The objective of this document is to give a machine learning-based project overview. The explanation provides a brief description of the dataset, pre-processing procedures, modeling algorithms, and outcome analysis.

System Requirements

The next sections go into detail about the exact hardware and software requirements for the project.

12.1 Hardware Requirements

The required hardware specifications are displayed in Figure 1 below. 64-bit Windows 10 with 8GB of RAM, 256GB of storage, and a 24" display.

22.1 Software Requirements

- Google Colab Notebook
- Python (Version 3.9.6)

32.1 Code Execution

Go to your respective Gmail account and select the drive. The given figure.1 shows the process to open a Google Colab notebook. The web browser displays the system's folder structure and navigates to the folder containing the code file. Run each cell by launching the relevant code file from the folder.

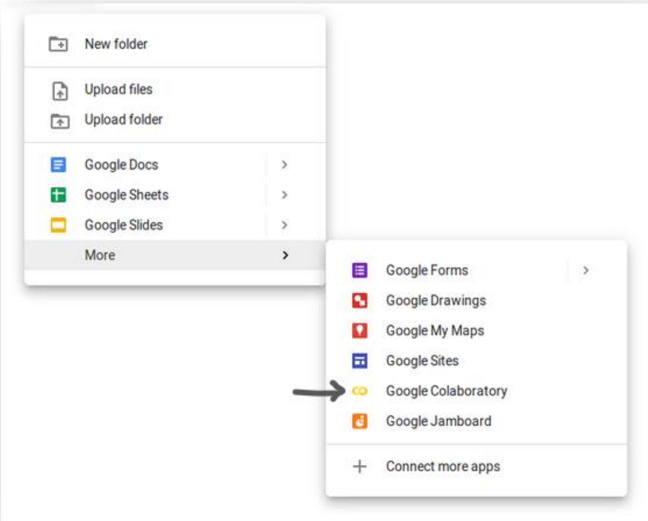

Figure: 1

5 Data Collection

The bellow given link is the source to the Kaggle data repository.

https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

6 Data Preprocessing

A list of all the Python libraries required to complete the project is shown in Figure 2.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
import matplotlib.pylab as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
import collections
from collections import Counter
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accuracy_score, classification_report
from imblearn.over_sampling import SMOTE
import sklearn.metrics as metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import LinearSVC
```

Figure: 2 Necessary Python Libraries

Figure 3 and figure 4 show to read a CSV file using the Pandas library and data types of 31 feature columns.

```
df = pd.read_csv('/content/drive/MyDrive/creditcard.csv')
df.info()
```

Figure:3 Read a csv file

```
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
```

Figure: 4 Details about Datatypes

Figure.5 shows 10 records with 31 features.

```
df[:10]
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	0.128539	-0.189115	0.133558	-0.021053	149.62	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	0.167170	0.125895	-0.008983	0.014724	2.69	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	-0.327642	-0.139097	-0.055353	-0.059752	378.66	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	0.647376	-0.221929	0.062723	0.061458	123.50	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	-0.206010	0.502292	0.219422	0.215153	69.99	0
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794	0.105915	0.253844	0.081080	3.67	0
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137	-0.257237	0.034507	0.005168	4.99	0
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	40.80	0
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205	-0.384157	0.011747	0.142404	93.20	0
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733	0.094199	0.246219	0.083076	3.68	0

10 rows × 31 columns

Figure: 5 Number of Features

To check is there any Null values exists in the given dataset!

```
df.isna().sum()

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
dtype: int64
```

Figure: 6 Check Null values

The dataset has two classes as Genuine and Fraud labels. Figure 7 shows the number of class counts.

```
df['Class'].value_counts()

0    284315
1       492
Name: Class, dtype: int64
```

Figure.8 shows the visualization of target class labels. It shows 99.8% are Genuine cases and 0.2% is Fraud cases.

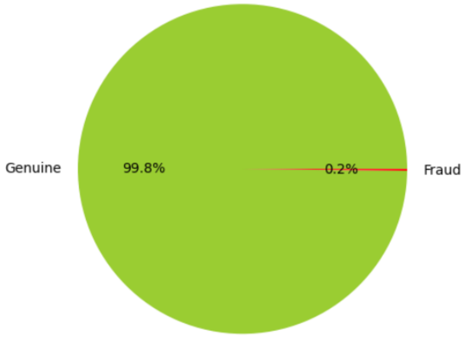

Figure: 8 Pie chart

5 Feature Selection

The discussion below shows the process to remove most of the correlated features.

5.1 Heat Map

The given figure 9 and Figure 10 show the use of Heatmap and the Correlation matrix to select features that are relevant to create a Machine learning model.

```
import matplotlib.pylab as plt
import seaborn as sns
plt.figure(figsize=(12,8))
sns.heatmap(df.corr())
plt.show()
```

Figure: 9 Use of seaborn

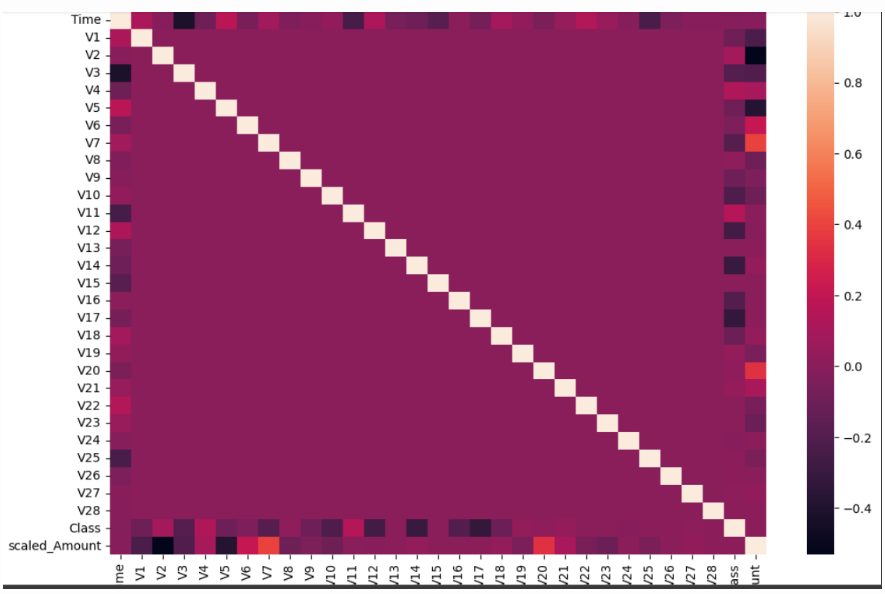

Figure:10 Heat Map

Figure 11 shows the number of features deleted from the dataset out of 31 features.

```
drop_list1 = ['Time','V28','V27','V26','V25','V24','V23','V22','V20','V15','V13','V8']
```

Figure:11 Dropped Feature

6 Feature Scaling

Figure 12 shows the result of Standardizing the feature "Amount".

```
from sklearn.preprocessing import StandardScaler
df['scaled_Amount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1,1))
df = df.drop(['Amount'],axis=1)
```

```
df['scaled_Amount']
```

```
0         0.244964
1        -0.342475
2         1.160686
3         0.140534
4        -0.073403
            ...
284802   -0.350151
284803   -0.254117
284804   -0.081839
284805   -0.313249
284806    0.514355
Name: scaled_Amount, Length: 284807, dtype: float64
```

Figure.12 Scaling feature

7 Machine Learning models with the unbalanced dataset

7.1 GaussianNB

Figure 13 shows the GaussianNB model the Figure 14 shows the output of Recall, Precision, and Accuracy scores.

```
from sklearn.naive_bayes import GaussianNB

drop_list1 = ['Time','V28','V27','V26','V25','V24','V23','V22','V20','V15','V13','V8']

df_drop_data = df.drop(drop_list1,axis=1)

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

X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2)

gaussian = GaussianNB()
gaussian.fit(X_train1, y_train1)

y_pred1 = gaussian.predict(X_test1)

from sklearn import metrics
print(metrics.classification_report(y_test1, y_pred1))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred1 , y_test1)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , y_pred1)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , y_pred1)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred1)))
# print('Confusion Matrix : \n', cnf_matrix)
print("\n")

pd.Series(y_pred1).value_counts()

pd.Series(y_test1).value_counts()
```

Figure: 13 GaussianNB model

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56874
           1       0.08      0.84      0.14        88

    accuracy                           0.98     56962
   macro avg       0.54      0.91      0.57     56962
weighted avg       1.00      0.98      0.99     56962

Accuracy :0.98460
Precision : 0.07898
Recall : 0.84091
F1 : 0.14439


0    56874
1       88
dtype: int64
```

Figure: 14 Precision, Accuracy, Recall scores

7.2 <u>Logistic regression</u>

The results of the logistic regression model are displayed in Figures 15 and 16.

```python
from sklearn.linear_model import LogisticRegression

logistic = LogisticRegression(C = 0.01, penalty = 'l2')
logistic.fit(X_train1, y_train1)

y_pred1 = logistic.predict(X_test1)

from sklearn import metrics
print(metrics.classification_report(y_test1, y_pred1))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred1 , y_test1)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , y_pred1)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , y_pred1)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred1)))
# print('Confusion Matrix : \n', cnf_matrix)
print("\n")

pd.Series(y_pred1).value_counts()

pd.Series(y_test1).value_counts()
```

Figure.15 Logistic Regression

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56874
           1       0.80      0.56      0.66        88

    accuracy                           1.00     56962
   macro avg       0.90      0.78      0.83     56962
weighted avg       1.00      1.00      1.00     56962

Accuracy :0.99910
Precision : 0.80328
Recall : 0.55682
F1 : 0.65772


0    56874
1       88
dtype: int64
```

Figure: 16 Precision, Accuracy, Recall scores

7.3 RandomForest Classifier

The results of the Random Forest model are displayed in Figures 17 and 18.

```python
from sklearn.ensemble import RandomForestClassifier

rfor = RandomForestClassifier(max_depth=5, random_state=0)

rfor.fit(X_train1, y_train1)
y_pred_rf = rfor.predict(X_test1)

from sklearn import metrics
print(metrics.classification_report(y_test1, y_pred_rf))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred_rf , y_test1)))
print('AUC : {0:0.5f}'.format(metrics.roc_auc_score(y_test1 , y_pred_rf)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , y_pred_rf)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , y_pred_rf)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred_rf)))
# print('Confusion Matrix : \n', cnf_matrix)
print("\n")

pd.Series(y_pred_rf).value_counts()

pd.Series(y_test1).value_counts()
```

Figure: 17 Precision, Accuracy and Recall

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56874
           1       0.84      0.74      0.79        88

    accuracy                           1.00     56962
   macro avg       0.92      0.87      0.89     56962
weighted avg       1.00      1.00      1.00     56962

Accuracy :0.99939
AUC : 0.86921
Precision : 0.84416
Recall : 0.73864
F1 : 0.78788


0    56874
1       88
dtype: int64
```

Figure: 18 Precision, Accuracy and Recall scores

7.4 SVM

The SVM model and its accuracy scores are shown in Figures 19 and 20.

```python
from sklearn.svm import LinearSVC

sklearn_svm = LinearSVC(class_weight='balanced', random_state=31, loss="hinge", fit_intercept=False)

sklearn_svm.fit(X_train1, y_train1)

import sklearn.metrics as metrics

y_pred_sklearn_svm = sklearn_svm.predict(X_test1)

print(metrics.classification_report(y_test1, y_pred_sklearn_svm))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred_sklearn_svm , y_test1)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , y_pred_sklearn_svm)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , y_pred_sklearn_svm)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred_sklearn_svm)))
print("\n")
```

Figure:19 SVM Model

```
              precision    recall  f1-score   support

           0       1.00      0.90      0.95     56874
           1       0.01      0.88      0.03        88

    accuracy                           0.90     56962
   macro avg       0.51      0.89      0.49     56962
weighted avg       1.00      0.90      0.95     56962

Accuracy :0.90337
Precision : 0.01382
Recall : 0.87500
F1 : 0.02722
```

Figure: 20 Precision, Accuracy and Recall

7.5 <u>XG Boost Classifier</u>

The results of the XGBoost classifier model are displayed in Figures 21 and 22.

```python
from xgboost import XGBClassifier
from sklearn.metrics import mean_squared_error
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accuracy_score, classification_report

xgb_clf = XGBClassifier(max_depth = 5, learning_rate = 0.08, objective = 'binary:logistic')

xgb_clf.fit(X_train1, y_train1)

y_pred_xgb = xgb_clf.predict(X_test1)

print('XG Boost Classifier:')
print('--------------------')
print('The accuracy score is: %.3f' % accuracy_score(y_test1, y_pred_xgb))
print('The precision score is: %.3f' % precision_score(y_test1, y_pred_xgb))
print('The recall score is: %.3f' % recall_score(y_test1, y_pred_xgb))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred_xgb)))
print("\n")

print(metrics.classification_report(y_test1, y_pred_xgb))

pd.Series(y_pred_xgb).value_counts()

pd.Series(y_test1).value_counts()
```

Figure: 21 XGBoost classifier

```
XG Boost Classifier:
--------------------
The accuracy score is: 0.999
The precision score is: 0.893
The recall score is: 0.761
F1 : 0.82209


              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56874
           1       0.89      0.76      0.82        88

    accuracy                           1.00     56962
   macro avg       0.95      0.88      0.91     56962
weighted avg       1.00      1.00      1.00     56962

0    56874
1       88
dtype: int64
```

Figure: 22 Precision, Accuracy and Recall

7.6 AdaBoost Classifier

AdaBoost classifier model and scores are shown in Figures 23 and 24.

```
from sklearn.ensemble import AdaBoostClassifier
abc = AdaBoostClassifier(n_estimators=50, learning_rate=1, random_state=0)
AdaBoost = abc.fit(X_train1, y_train1)

y_pred_adaboost = AdaBoost.predict(X_test1)

print('AdaBoostClassifier:')
print('--------------------')

from sklearn import metrics
print(metrics.classification_report(y_test1, y_pred_adaboost))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred_adaboost , y_test1)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , y_pred_adaboost)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , y_pred_adaboost)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred_adaboost)))
# print('Confusion Matrix : \n', cnf_matrix)
print("\n")

pd.Series(y_pred_adaboost).value_counts()
pd.Series(y_test1).value_counts()
```

Figure:23 AdaBoost classifier

```
AdaBoostClassifier:
--------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56874
           1       0.70      0.70      0.70        88

    accuracy                           1.00     56962
   macro avg       0.85      0.85      0.85     56962
weighted avg       1.00      1.00      1.00     56962

Accuracy :0.99907
Precision : 0.69663
Recall : 0.70455
F1 : 0.70056


0    56874
1       88
dtype: int64
```

Figure: 24 Precision, Accuracy and Recall

8 Class imbalance issue with SMOTE method

Figure 25 shows the application of SMOTE method.

```
from imblearn.over_sampling import SMOTE
from collections import Counter

# drop_list1 = ['Time','V28','V27','V26','V25','V24','V23','V22','V20','V15','V13','V8']

# df_drop_data = df.drop(drop_list1,axis=1)
# y = df_drop_data['Class'].values #target
# X = df_drop_data.drop(['Class'],axis=1).values #features
# X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2)

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train1, y_train1)
```

Figure:25 SMOTE method to train and test

8.1 Logistic Regression

The Regression model with a balanced dataset is shown in Figures 26 and 27, along with the model's results.

```
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train1, y_train1)

print('Resampled dataset shape %s' % Counter(y_train_smote))

# SMOTE Sampling with Logistic Regression
logreg_2 = LogisticRegression(max_iter=1000)
logreg_2.fit(X_train_smote, y_train_smote)

y_pred_smote = logreg_2.predict(X_test1)

print(metrics.classification_report(y_test1, y_pred_smote))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_test1 , y_pred_smote)))
print('AUC : {0:0.5f}'.format(metrics.roc_auc_score(y_test1 , y_pred_smote)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , y_pred_smote)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , y_pred_smote)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred_smote)))

# Predicted values counts for fraud and genuine of test dataset
pd.Series(y_pred_smote).value_counts()

# Actual values counts for fraud and genuine of test dataset
pd.Series(y_test1).value_counts()
```

Figure:26 Logistic Regression model

```
Resampled dataset shape Counter({0: 227441, 1: 227441})
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56874
           1       0.05      0.88      0.09        88

    accuracy                           0.97     56962
   macro avg       0.52      0.92      0.54     56962
weighted avg       1.00      0.97      0.99     56962

Accuracy :0.97407
AUC : 0.92461
Precision : 0.04990
Recall : 0.87500
F1 : 0.09442
0    56874
1       88
dtype: int64
```

Figure:27 Precision, Accuracy and Recall

8.2 <u>GaussianNB Classifier</u>

GaussianNB classifier and its scores are displayed in Figures 28 and 29.

```python
gaussian_2 = GaussianNB()
gaussian_2.fit(X_train_smote, y_train_smote)

# Predict from Test set
gaussian_smote_pred = gaussian_2.predict(X_test1)

# Model Evolution
print(metrics.classification_report(y_test1, gaussian_smote_pred))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(gaussian_smote_pred , y_test1)))
print('AUC : {0:0.5f}'.format(metrics.roc_auc_score(y_test1 , gaussian_smote_pred)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , gaussian_smote_pred)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , gaussian_smote_pred)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , gaussian_smote_pred)))
# print('Confusion Matrix : \n', cnf_matrix)
print("\n")

# Predicted values counts for fraud and genuine of test dataset
pd.Series(gaussian_smote_pred).value_counts()

# Actual values counts for fraud and genuine of test dataset
pd.Series(y_test1).value_counts()
```

Figure: 28 GaussianNB classifier

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56874
           1       0.07      0.85      0.12        88

    accuracy                           0.98     56962
   macro avg       0.53      0.92      0.56     56962
weighted avg       1.00      0.98      0.99     56962

Accuracy :0.98151
AUC : 0.91699
Precision : 0.06726
Recall : 0.85227
F1 : 0.12469


0    56874
1       88
dtype: int64
```

Figure:29 Precision, Accuracy and Recall

8.3 Random Forest Classifier

Figure 30 and Figure 31 show the Random Forest classifier and its scores.

```python
rfor_2 = RandomForestClassifier(max_depth=5, random_state=0)

rfor_2.fit(X_train_smote, y_train_smote)
y_pred_rf_smote = rfor_2.predict(X_test1)

from sklearn import metrics
print(metrics.classification_report(y_test1, y_pred_rf_smote))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred_rf_smote , y_test1)))
print('AUC : {0:0.5f}'.format(metrics.roc_auc_score(y_test1 , y_pred_rf_smote)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , y_pred_rf_smote)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , y_pred_rf_smote)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred_rf_smote)))
# print('Confusion Matrix : \n', cnf_matrix)
print("\n")

pd.Series(y_pred_rf_smote).value_counts()

pd.Series(y_test1).value_counts()
```

Figure: 30 GaussianNB classifier

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     56874
           1       0.20      0.85      0.32        88

    accuracy                           0.99     56962
   macro avg       0.60      0.92      0.66     56962
weighted avg       1.00      0.99      1.00     56962

Accuracy :0.99435
AUC : 0.92342
Precision : 0.19531
Recall : 0.85227
F1 : 0.31780


0    56874
1       88
dtype: int64
```

Figure: 31 Precision, Accuracy and Recall scores

8.4 Support vector Machine classifier

Figure: 32 and Figure 33 show the SVM classifier model and its scores.

```python
from sklearn.svm import LinearSVC

sklearn_svm_smt = LinearSVC(class_weight='balanced', random_state=31, loss="hinge", fit_intercept=False)

sklearn_svm_smt.fit(X_train_smote, y_train_smote)

# Predict labels for the test data
y_pred_svm = sklearn_svm_smt.predict(X_test1)

print(metrics.classification_report(y_test1, y_pred_svm))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred_svm , y_test1)))
print('AUC : {0:0.5f}'.format(metrics.roc_auc_score(y_test1 , y_pred_svm)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , y_pred_svm)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , y_pred_svm)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred_svm)))
print("\n")
```

Figure:32 SVM classifier

```
              precision    recall  f1-score   support

           0       1.00      0.85      0.92     56874
           1       0.01      0.93      0.02        88

    accuracy                           0.85     56962
   macro avg       0.50      0.89      0.47     56962
weighted avg       1.00      0.85      0.92     56962

Accuracy :0.84665
AUC : 0.88917
Precision : 0.00931
Recall : 0.93182
F1 : 0.01843
```

Figure:33 Precision, Accuracy and Recall

8.5 XG Boost Classifier

The results of the XGBoost classifier are shown in Figures 34 and 35.

```
from xgboost import XGBClassifier

xgb_clf_smt = XGBClassifier(max_depth = 5, learning_rate = 0.08, objective = 'binary:logistic')

xgb_clf_smt.fit(X_train_smote, y_train_smote)
y_pred_xgb_smt = xgb_clf_smt.predict(X_test1)

print('XG Boost Classifier:')
print('--------------------')
print('The accuracy score is: %.3f' % accuracy_score(y_test1, y_pred_xgb_smt))
print('The precision score is: %.3f' % precision_score(y_test1, y_pred_xgb_smt))
print('The recall score is: %.3f' % recall_score(y_test1, y_pred_xgb_smt))
print('AUC : {0:0.5f}'.format(metrics.roc_auc_score(y_test1 , y_pred_xgb_smt)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred_xgb_smt)))
print("\n")

print(metrics.classification_report(y_test1, y_pred_xgb_smt))

pd.Series(y_pred_xgb_smt).value_counts()
pd.Series(y_test1).value_counts()
```

Figure:34 XGBoost classifier

```
XG Boost Classifier:
--------------------
The accuracy score is: 0.992
The precision score is: 0.146
The recall score is: 0.852
AUC : 0.92229
F1 : 0.25000


              precision    recall  f1-score   support

           0       1.00      0.99      1.00     56874
           1       0.15      0.85      0.25        88

    accuracy                           0.99     56962
   macro avg       0.57      0.92      0.62     56962
weighted avg       1.00      0.99      0.99     56962

0    56874
1       88
dtype: int64
```

Figure:35 Precision, Accuracy and Recall scores

8.6 AdaBoostClassifier

AdaBoost classifier is shown in Figures 36 and 37 together with its results.

```
# Train Adaboost Classifer
AdaBoost_smt = abc_smt.fit(X_train_smote, y_train_smote)

#Predict the response for test dataset
y_pred_adaboost_smt = AdaBoost_smt.predict(X_test1)

print('AdaBoostClassifier:')
print('--------------------')

from sklearn import metrics
print(metrics.classification_report(y_test1, y_pred_adaboost_smt))

print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred_adaboost_smt , y_test1)))
print('AUC : {0:0.5f}'.format(metrics.roc_auc_score(y_test1 , y_pred_adaboost_smt)))
print('Precision : {0:0.5f}'.format(metrics.precision_score(y_test1 , y_pred_adaboost_smt)))
print('Recall : {0:0.5f}'.format(metrics.recall_score(y_test1 , y_pred_adaboost_smt)))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test1 , y_pred_adaboost_smt)))
# print('Confusion Matrix : \n', cnf_matrix)
print("\n")

# Predicted values counts for fraud and genuine of test dataset
pd.Series(y_pred_adaboost_smt).value_counts()

# Actual values counts for fraud and genuine of test dataset
pd.Series(y_test1).value_counts()
```

Figure:36 AdaBoost classifier

```
AdaBoostClassifier:
--------------------
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56874
           1       0.05      0.88      0.10        88

    accuracy                           0.98     56962
   macro avg       0.53      0.93      0.54     56962
weighted avg       1.00      0.98      0.99     56962

Accuracy :0.97574
AUC : 0.92545
Precision : 0.05318
Recall : 0.87500
F1 : 0.10026


0    56874
1       88
dtype: int64
```

Figure:37 Precision, Accuracy and Recall scores

9 Model Results

9.1 Scores for Imbalanced Dataset

Model	Precision		Recall		Accuracy
	Class 0	Class 1	Class 0	Class 1	
LogisticRegression	1	0.80	1	0.56	0.99
GaussianNB	1	0.08	0.98	0.84	0.98
RandomForest	1	0.84	1	0.74	0.99
SVM	1	0.01	0.90	0.88	0.91
XGBoost	1	0.89	1	0.76	0.99
AdaBoost	1	0.70	1	0.70	0.99

9.2 Scores for Balanced Dataset

Model	Precision		Recall		Accuracy
	Class 0	Class 1	Class 0	Class 1	
LogisticRegression	1	0.05	0.97	0.88	0.97
GaussianNB	1	0.07	0.98	0.85	0.98
RandomForest	1	0.20	0.99	0.85	0.99
SVM	1	0.01	0.85	0.93	0.85
XGBoost	1	0.15	0.99	0.85	0.99
AdaBoost	1	0.05	0.98	0.88	0.98

9.3 Comparison of scores for balanced and imbalanced datasets

Comparing results for both cases, the result shows RandomForest and Boosting algorithms always give a good accuracy score. **Class 0** is the label for Genuine cases and **class 1** is the label for Fraud cases. Comparing result RandomForest: for the Imbalanced dataset the recall value of Fraud cases is 0.74 while for the balanced dataset, the recall value became 0.85 and the accuracy score remains the same as 0.99. It shows the Recall value has increased as compared to 0.74 after applying SMOTE method. Through this, we can compare the result for the rest of the methods.

**********************END****************************