# FindDefault (Prediction of Credit Card fraud)

**Name: Bathalapalli Sarang**

**Email_id: bathalapalli9920@gmail.com**

**Deployed prototype model: https://creditfrauddetector.streamlit.app/**

**Sample input for above prototype: finddefault\input_test_UI.csv**

**Github: https://github.com/SarrangTech/creditFraud**

**Model training: click here for all trained models**

**Saved models: click here for saved joblib files**

## I. Description of design choices and Performance evaluation of the model

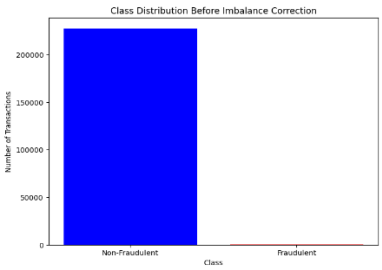
### A. Preprocessing

1. The task is a binary classification task, requiring a supervised machine learning algorithm to classify between fraud and non fraud classes.
2. There were no missing_values and all the variables were pre encoded, in addition there were no significant problems with multicollinearity with VIF not exceeding 11, which was confirmed by heatmap which indicated no strong positive or negative correlation between features.
3. However there were lots of outliers, eg with 13% of data in feature V27 being outliers.
4. Due the prevalent presence of outliers, I decided not to perform standardization right away, because it uses mean and standard deviation, which in turn are very sensitive to outliers.
5. The Robust Scaler uses statistics that are robust to outliers, such as the median and the interquartile range (IQR), to scale features. I also experimented with quantile transformer, power transformer, standardization (on top of Robust scaler), but got the best results with robust scaler.

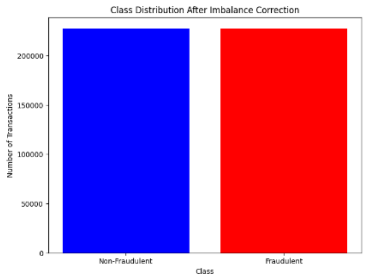
**B. Class imbalance**

1. The given dataset was heavily imbalanced, where we have 492 frauds out of 284,807 transactions the positive class (frauds) account for only 0.172% of all transactions.
2. I used the below class imbalance correction techniques

i. **SMOTE** (Synthetic Minority Over-sampling Technique): I chose this due to its simplicity and effectiveness in addressing class imbalance by generating synthetic samples along the line segments connecting minority class instances.

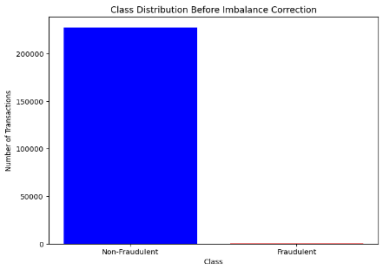



**Original**

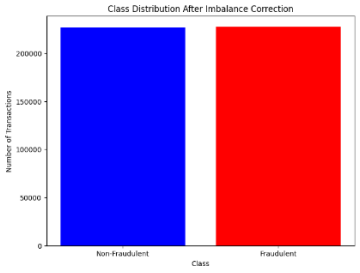



**Smote**

j. **SMOTE_ENN** (SMOTE combined with Edited Nearest Neighbors): Selected for its ability to combine oversampling and undersampling techniques to address class imbalance effectively while mitigating potential drawbacks such as overfitting caused by SMOTE. SMOTEENN provides a balanced approach by first oversampling minority class instances with SMOTE and then removing noisy examples using ENN, resulting in improved model performance.



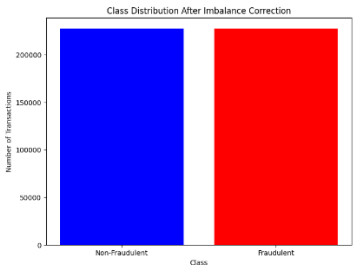

**Original**




**Smote_ENN**

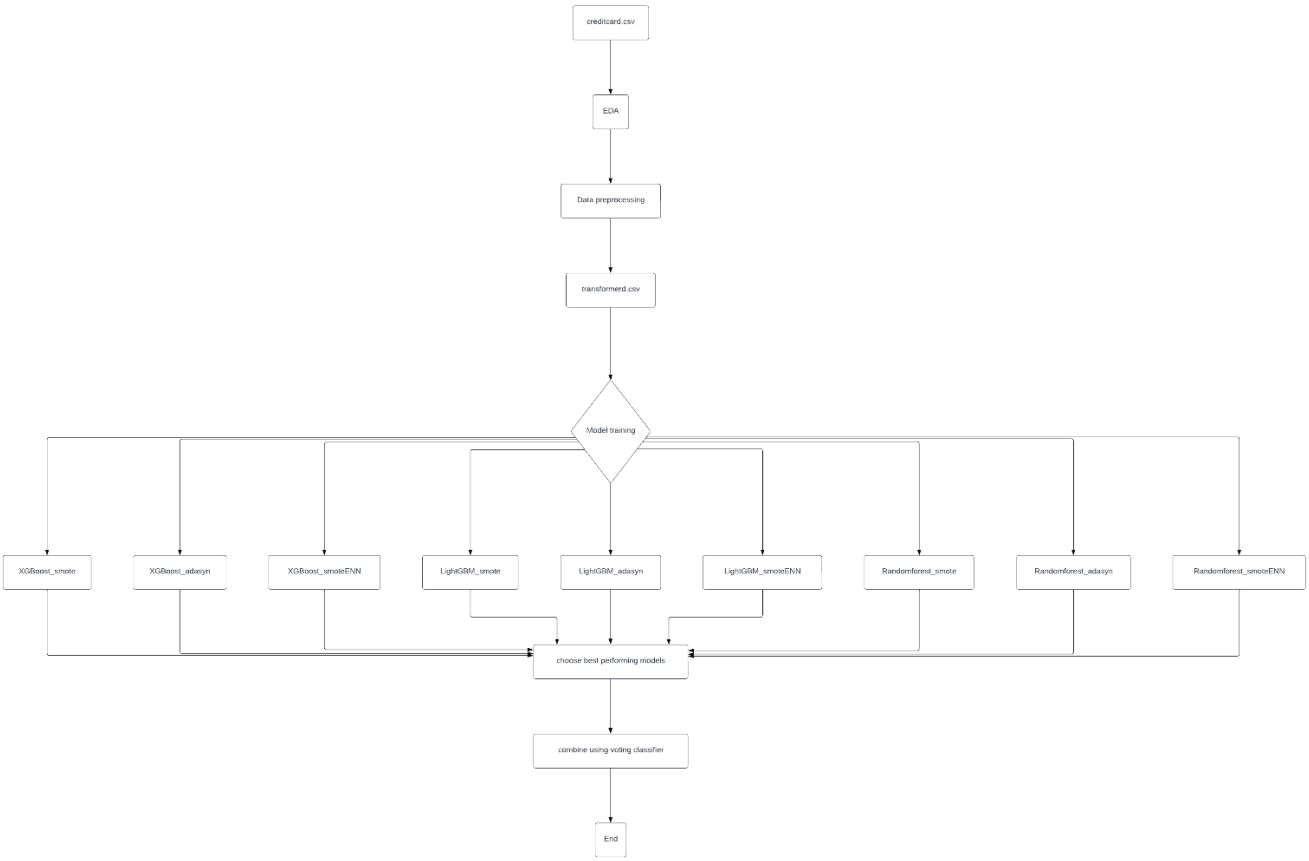
3. **ADASYN** (Adaptive Synthetic Sampling):
   Opted for its adaptive synthetic sample generation, which focuses on regions with severe class imbalance to provide a more nuanced oversampling approach. ADASYN dynamically adjusts the density of synthetic samples based on the local distribution of minority class instances, making it suitable for datasets with varying levels of class imbalance. This adaptability can lead to better model generalization compared to static oversampling techniques like SMOTE.


**Original**




**Adasyn**

4. I didn't use undersampling and ensemble methods for correcting class imbalance because undersampling can create more bias and since minority class was too small, and ensemble(BalancedRandomForest)often may not always generalize well to new data.




## C. Model training

1. There are 30 features & 2,84,807 rows. i.e. dataset is large & high dimensional.

2. **Random Forest (RF):**
   - i. RF is chosen for its robustness to noise and outliers, making it suitable for datasets with many features and potential outliers.
   - ii. Capable of handling high-dimensional data efficiently.
   - iii. Provides built-in feature importance measures, aiding in interpretability.

3. **XGBoost (Extreme Gradient Boosting):**

**iv.** Selected for its scalability, efficiency, and high performance, especially on structured/tabular data.

v. Well-suited for large datasets due to its speed and memory efficiency.

vi. Employs a gradient boosting framework, which iteratively improves the model's performance by focusing on the most challenging examples.

4. **LightGBM (Light Gradient Boosting Machine):**

vii. Chosen for its speed, efficiency, and ability to handle large datasets, making it suitable for our dataset with a large number of rows and features.

viii. Provides better performance than traditional gradient boosting algorithms while requiring less computational resources.

ix. Utilizes a leaf-wise tree growth strategy and histogram-based binning, resulting in faster training times.

5. **Voting Classifier:**

The Voting Classifier was chosen to combine the results of Random Forest, XGBoost, and LightGBM models due to its ability to aggregate predictions from multiple models and produce a final prediction based on a majority or weighted vote. This ensemble technique helps mitigate biases inherent in individual models and leverages the strengths of each model. By considering the collective decision of multiple models, the Voting Classifier aims to improve overall prediction accuracy and generalization performance on the imbalanced dataset.

6. Logistic Regression is a linear model and may not capture the complex non-linear relationships present in the data. Naive Bayes assumes independence among features, which might not hold true for this dataset with potentially correlated features. KNN can be computationally expensive and less efficient with high-dimensional data, leading to slower predictions and scalability issues compared to tree-based ensemble methods.

**D. Hyperparameter tuning: -**

The hyperparameter tuning process involved different strategies tailored to each model: Grid Search for LightGBM, Randomized Search for Random Forest, and initially Grid Search for XGBoost, which resulted in performance degradation due to suboptimal parameter combinations. However, upon reverting to default parameters for XGBoost, improved performance was observed. This highlights the importance of model-specific optimization techniques and the need for careful parameter selection to avoid overfitting and achieve optimal model performance.

**Gridsearch:-**

```
LightGBM Smote: Best parameters found:  {'colsample_bytree': 1.0, 'learn
ing_rate': 0.2, 'n_estimators': 100, 'num_leaves': 40, 'subsample': 0.8
}
```

```
LightGBM SmoteENN: {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'n_es
timators': 150, 'num_leaves': 40, 'subsample': 0.8}
```

```
LightGBM Adasyn: {'colsample_bytree': 0.8, 'learning_rate': 0.2, 'n_esti
mators': 150, 'num_leaves': 30, 'subsample': 0.8}
```

**Randomized search:-**

```
Randomforest smote: {'n_estimators': 200, 'min_samples_split': 5, 'min_
samples_leaf': 2, 'max_features': 'auto', 'max_depth': None, 'bootstrap
': True}
```

```
Randomforest smoteENN: Best Parameters: {'n_estimators': 100, 'min_samp
les_split': 2, 'min_samples_leaf': 4, 'max_features': 'auto', 'max_dept
h': None, 'bootstrap': False}
```

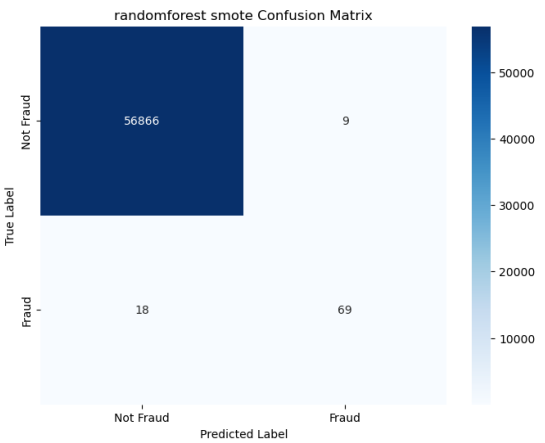
```
Randomforest Adasyn: Best Parameters: {'n_estimators': 50, 'min_samples
_split': 10, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth'
: None, 'bootstrap': False}
```

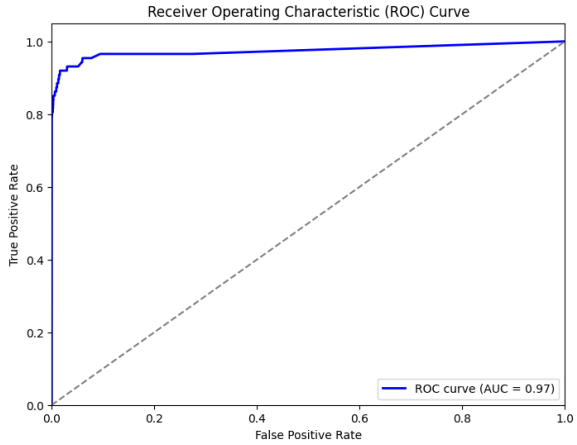
### E. Model metrics: -

Accuracy score may not provide a complete picture of model performance in case of imbalanced data, hence metrics like precision and recall provide a better picture. And especially precision, recall in a task like predicting credit card fraud is imperative, because a low precision may result in lots of false positive(predicting fraud when there is no fraud), and low recall may result in lots of false negatives(missing fraudulent transactions), both of the scenarios result in significant monetary loss. Below are metrics on test set.

#### 1. RandomForest smote:

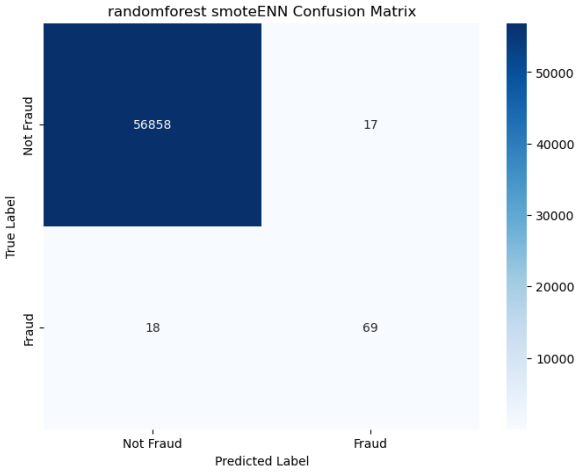
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56875 |
| 1 | 0.88 | 0.79 | 0.84 | 87 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.94 | 0.90 | 0.92 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |

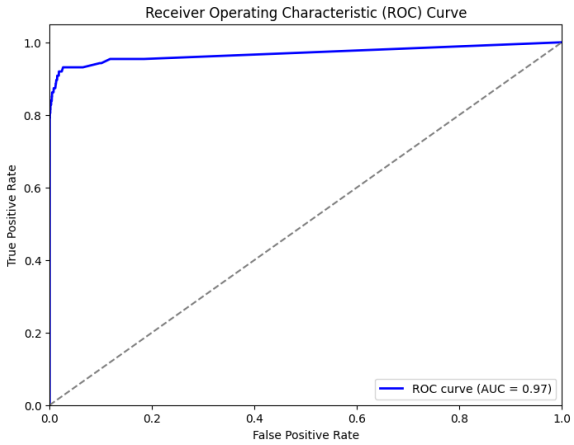


## 2. RandomForest smoteENN:

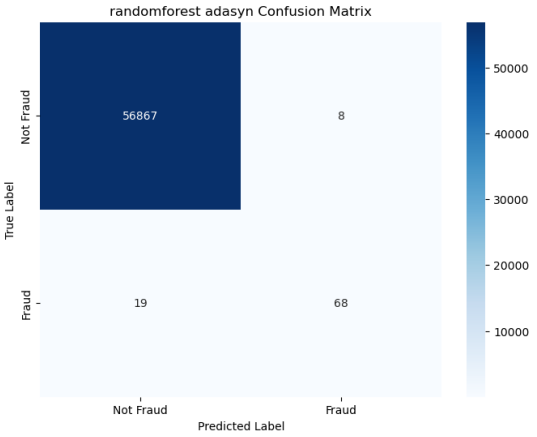
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56875 |
| 1 | 0.80 | 0.79 | 0.80 | 87 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.90 | 0.90 | 0.90 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |

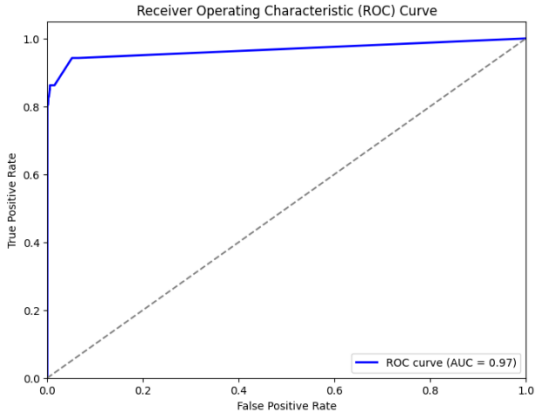



## 3. RandomForest Adasyn:

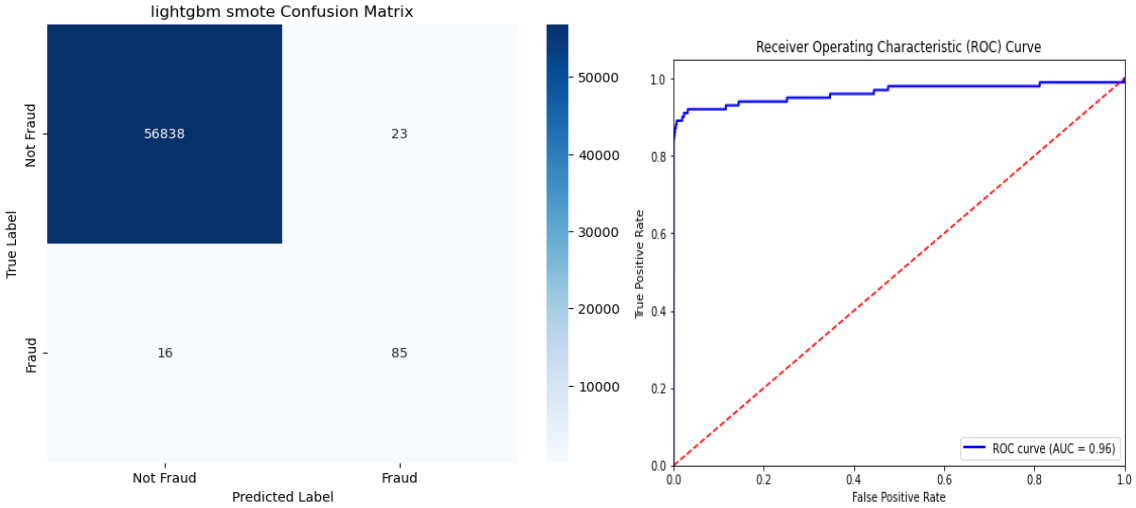
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56875 |
| 1 | 0.89 | 0.78 | 0.83 | 87 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.95 | 0.89 | 0.92 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |



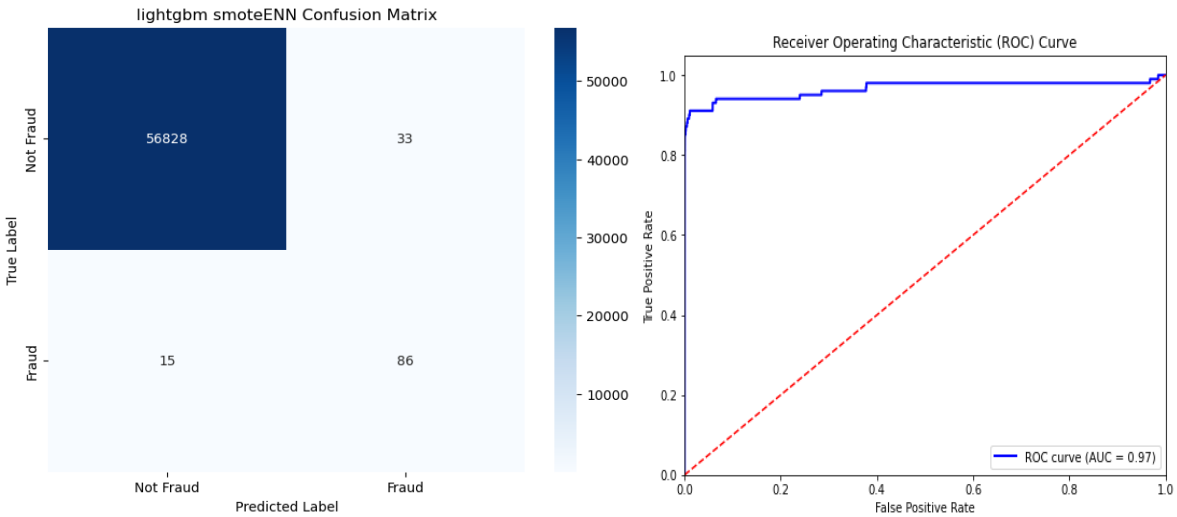
### 4. LightGBM smote:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56854 |
| 1 | 0.84 | 0.79 | 0.81 | 108 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.92 | 0.89 | 0.91 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |



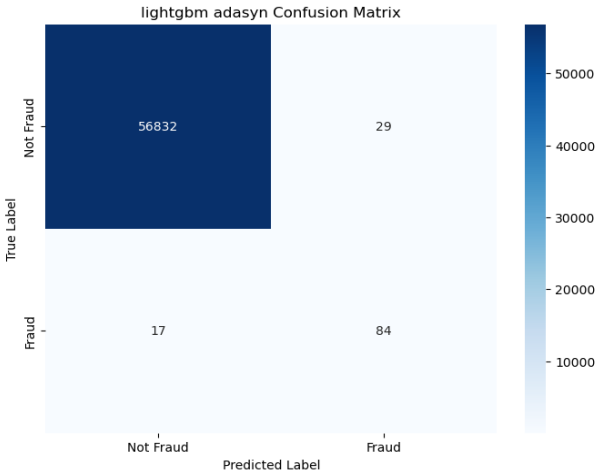

### 5. LightGBM smote_ENN:

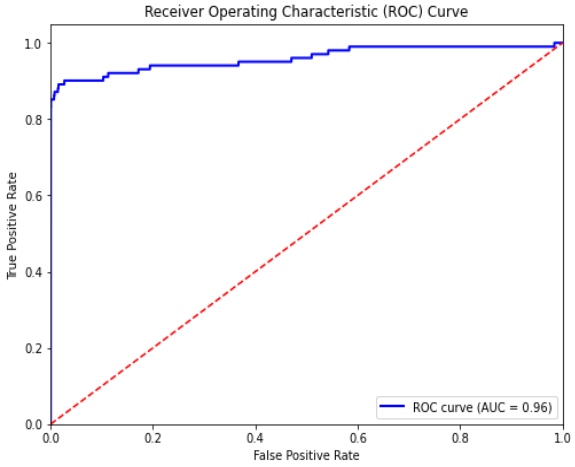
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56843 |
| 1 | 0.85 | 0.72 | 0.78 | 119 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.93 | 0.86 | 0.89 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |



## 6. LightGBM adasyn:

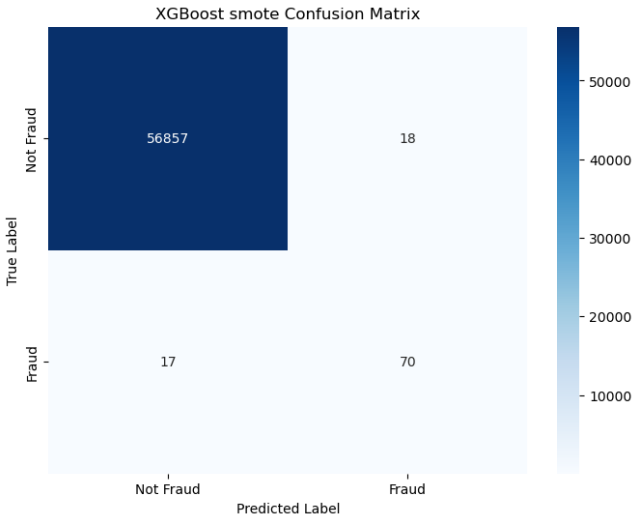
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56849 |
| 1 | 0.83 | 0.74 | 0.79 | 113 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.92 | 0.87 | 0.89 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |

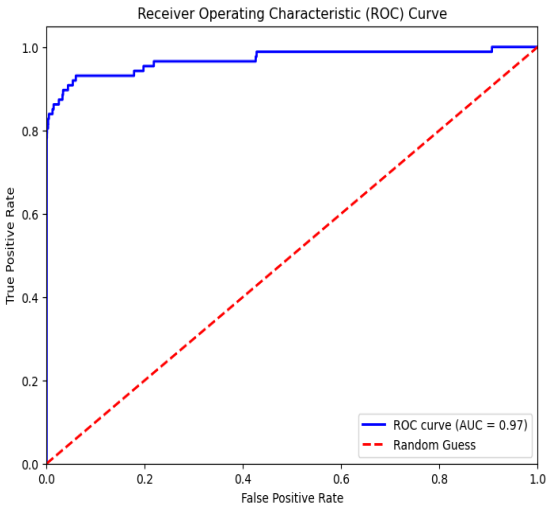



## 7. XGBoost smote:

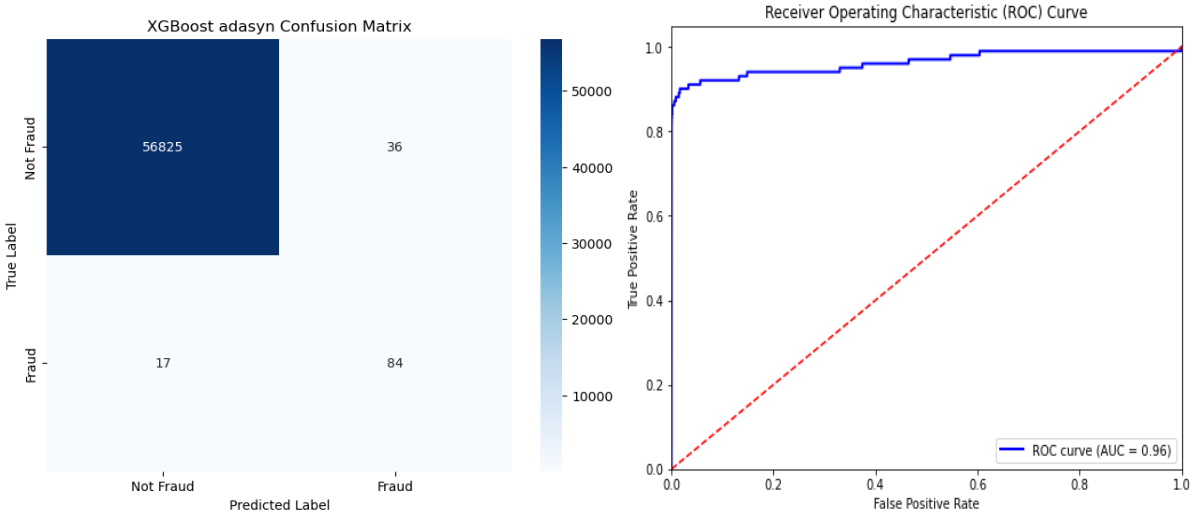
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56875 |
| 1 | 0.80 | 0.80 | 0.80 | 87 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.90 | 0.90 | 0.90 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |



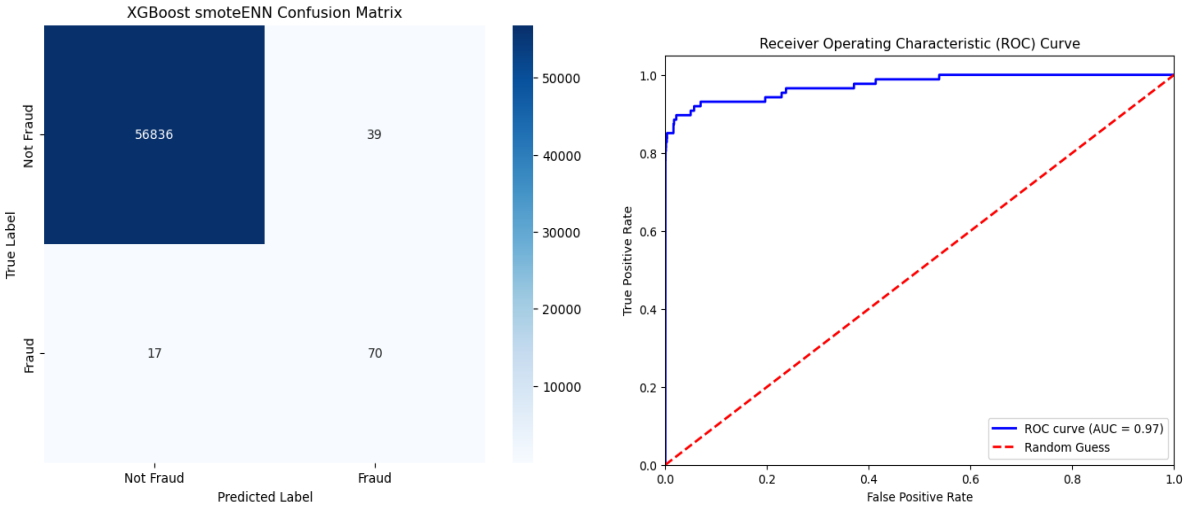
## 8. XGBoost adasyn:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56842 |
| 1 | 0.83 | 0.70 | 0.76 | 120 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.92 | 0.85 | 0.88 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |




## 9. XGBoost smoteENN:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56875 |
| 1 | 0.64 | 0.80 | 0.71 | 87 |
| accuracy | | | 1.00 | 56962 |
| macro avg | 0.82 | 0.90 | 0.86 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |



**F. Future work:**

1. Collect a more extensive dataset spanning a more extended period to capture a broader range of fraudulent activities and improve the model's ability to generalize.
2. Experiment with additional techniques for handling imbalanced data, such as ensemble methods like EasyEnsemble and BalanceCascade.
3. Explore more sophisticated anomaly detection algorithms like isolation forests or autoencoders to enhance fraud detection performance.
4. Incorporate real-time data streaming and advanced feature engineering techniques to improve the model's ability to detect emerging patterns of fraudulent activity.
5. Deploy the model in a production environment and continually monitor its performance to maintain effective fraud detection capabilities over time.

In [41]:
```
import warnings

warnings.filterwarnings("ignore")
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
```

In [252]:
```
df=pd.read_csv("D:\\DS Bootcamp\\datascience\\datasets\\creditcard.csv")
```

In [3]:
```
df.head()
```

Out[3]:

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

5 rows × 31 columns

In [4]:
```
df.shape
```

Out[4]:
```
(284807, 31)
```

In [5]:
```
df.info()
```

```
<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
```

## EDA

In [6]:
```
from ydata_profiling import ProfileReport
```

In [7]:
```
pr_df=ProfileReport(df)
```

In [8]:
```
pr_df
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Out[8]:

In [9]:
```
pr_df.to_file("findDefault_raw_report.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

## handling missing values

In [20]:
```
df.isnull().sum()
```

Out[20]:
```
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
```

**there is no evidence of missing values**

## solving multicollinearity

In [3]:
```
numeric_columns = df.select_dtypes(include=['number']).columns
purely_numeric_columns = [col for col in numeric_columns if df[col].nunique
```

In [14]:
```
print(purely_numeric_columns)
```

```
['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V1
1', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount']
```

In [15]:
```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor
subset_vars = purely_numeric_columns
subset_df = df[subset_vars]
vif_data = pd.DataFrame()
vif_data["Variable"] = subset_vars
vif_data["VIF"] = [variance_inflation_factor(subset_df.values, i) for i in
print(vif_data.sort_values(by='VIF',ascending=False))
```

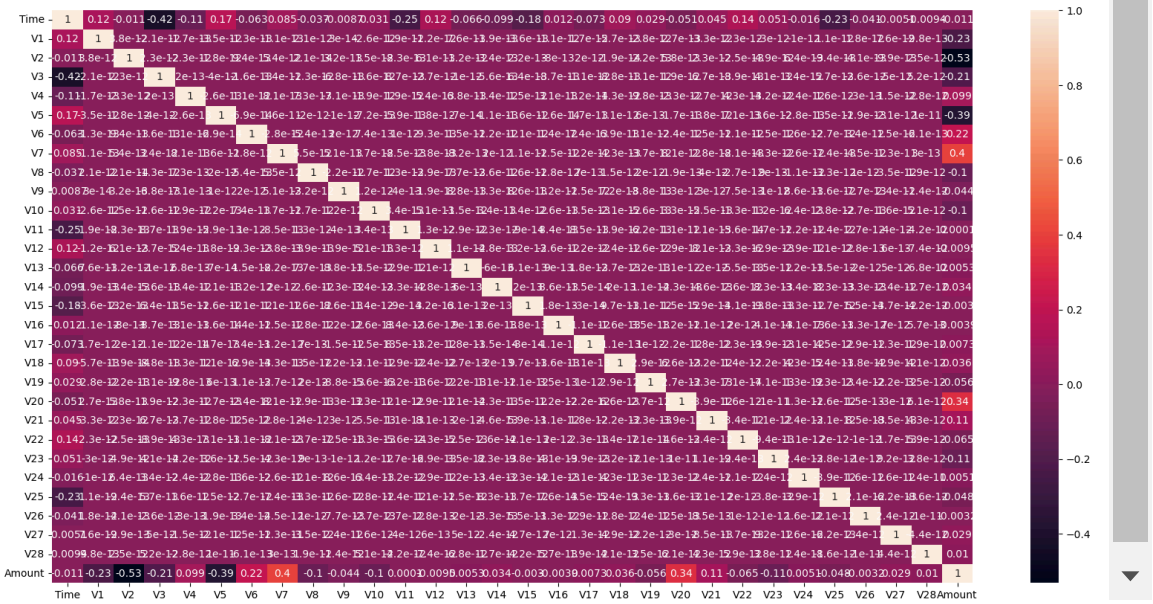
```
   Variable        VIF
29   Amount  11.499791
2        V2   3.869377
5        V5   2.753075
7        V7   2.510165
0      Time   2.339084
20      V20   2.233934
1        V1   1.621694
6        V6   1.522122
3        V3   1.255585
23      V23   1.149268
4        V4   1.137944
10      V10   1.115668
21      V21   1.100720
8        V8   1.097151
22      V22   1.082384
19      V19   1.037809
11      V11   1.028861
14      V14   1.026832
9        V9   1.018831
15      V15   1.014135
25      V25   1.013388
12      V12   1.011961
27      V27   1.008979
18      V18   1.006568
17      V17   1.004772
13      V13   1.003434
28      V28   1.001425
24      V24   1.000659
26      V26   1.000487
16      V16   1.000371
```

In [8]:
```
"""
VIF higher than 10 which means possible multicollinearity problem, there is
hence no action needs to be taken as of now, confirming this with correlati
"""
```

Out[8]:
```
'\nVIF higher than 10 which means possible multicollinearity problem, ther
e is no significant case of high multicollinearity\nhence no action needs
to be taken as of now, confirming this with correlation\n'
```

In [7]:
```
plt.figure(figsize=(20, 10))
sns.heatmap(data=df[purely_numeric_columns].corr(),annot=True)
```

Out[7]: <Axes: >




In [18]:
```
import numpy as np
correlation_matrix = df[purely_numeric_columns].corr()
threshold = 0.7
strong_correlations = {}
for i in range(len(correlation_matrix.columns)):
    for j in range(i+1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i, j]) >= threshold:
            column1 = correlation_matrix.columns[i]
            column2 = correlation_matrix.columns[j]
            correlation_strength = correlation_matrix.iloc[i, j]
            strong_correlations[(column1, column2)] = correlation_strength
for (column1, column2), correlation_strength in strong_correlations.items()
    print(f"Strong correlation between {column1} and {column2}: {correlatio
```

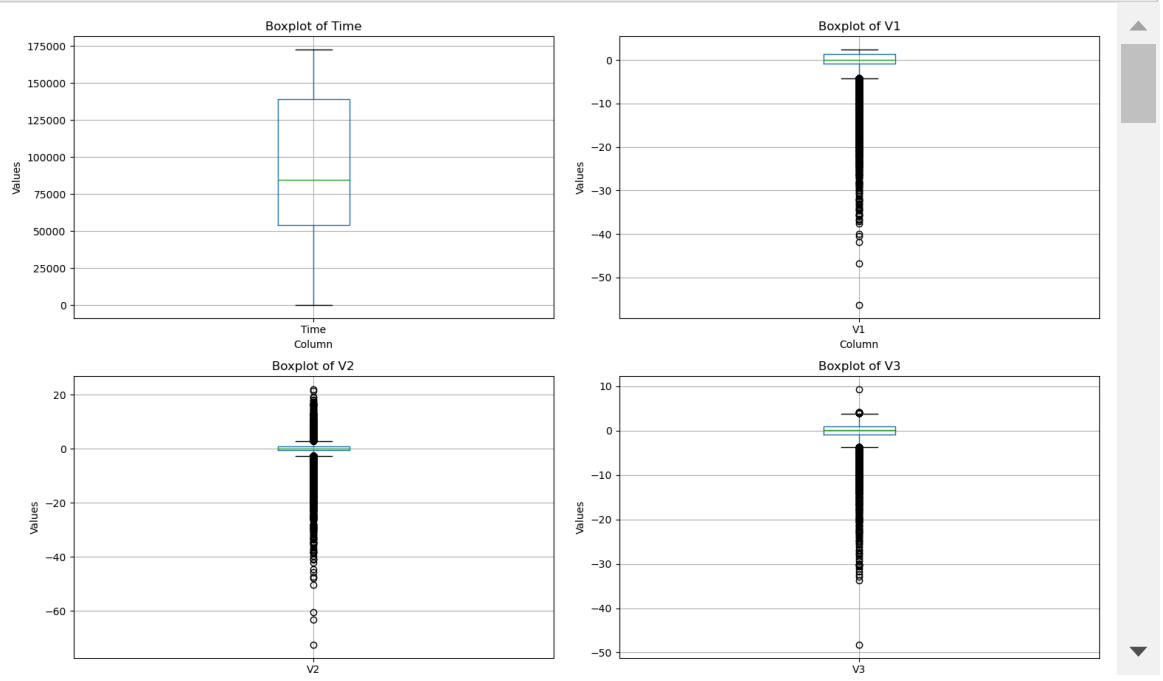
**above code confirms that there is no strong positive or negative correlation**

# Encoding categorical data

In [23]:
```
# df.info()# all features are of numerical datatype, hence no encoding is r
```

## outlier detection

In [31]:
```
import matplotlib.pyplot as plt

# Assuming your dataset is stored in a DataFrame called 'df'
# Extract all columns except the last one
columns_to_plot = df.columns[:-1]

# Calculate the number of rows and columns for subplots
num_columns = len(columns_to_plot)
num_rows = (num_columns + 1) // 2  # Use integer division to get the number

# Create subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(15, 5 * num_rows))  # Increa

# Flatten the axes array to simplify indexing
axes = axes.flatten()

# Iterate through each column except the last one
for i, column in enumerate(columns_to_plot):
    # Draw a boxplot for the current column
    df.boxplot(column=column, ax=axes[i])
    axes[i].set_title(f'Boxplot of {column}')
    axes[i].set_xlabel('Column')
    axes[i].set_ylabel('Values')

# Hide any unused subplots
for j in range(num_columns, len(axes)):
    axes[j].axis('off')

# Adjust layout to prevent overlap of titles
plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust bottom margin to provid

# Show the plot
plt.show()
```



In [33]:
```
import pandas as pd
Q1 = df.iloc[:, :-1].quantile(0.25)
Q3 = df.iloc[:, :-1].quantile(0.75)
IQR = Q3 - Q1
threshold = 1.5
outliers = ((df.iloc[:, :-1] < (Q1 - threshold * IQR)) | (df.iloc[:, :-1] >
outliers_count = outliers.sum()
percentage_outliers = outliers_count / len(df) * 100
print("Percentage of outliers in each column:")
print(percentage_outliers)
```

```
Percentage of outliers in each column:
Time       0.000000
V1         2.479574
V2         4.749181
V3         1.180800
V4         3.914230
V5         4.316959
V6         8.063355
V7         3.141777
V8         8.473809
V9         2.908285
V10        3.334188
V11        0.273870
V12        5.388912
V13        1.182555
V14        4.967926
V15        1.016127
V16        2.873525
V17        2.605273
V18        2.644949
V19        3.583128
V20        9.750463
V21        5.090114
V22        0.462418
V23        6.510023
V24        1.676223
V25        1.884434
V26        1.964839
V27       13.750715
V28       10.653530
Amount    11.201972
dtype: float64
```

### certain columns(V27,V28,amount) have significant portion of outliers, but as our goal is to detect fraudlent credit card transactions, outliers might provide valuable info, hence i am deciding to retain them, however will use feature scaling to reduce its impact

## feature scaling

### due to the presence of lots of outliers, i will not be using standard scaler, because it uses mean and standard deviation, which are very sensitive to outliers.

In [253]:
```
df.head()
```

Out[253]:

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

5 rows × 31 columns

## Robust Scaler:

The Robust Scaler uses statistics that are robust to outliers, such as the median and the interquartile range (IQR), to scale features. It subtracts the median from each feature and then scales it by dividing by the IQR (the difference between the 75th and 25th percentiles).

In [254]:
```
from sklearn.preprocessing import RobustScaler
import pandas as pd
features = df.iloc[:, :-1]
scaler = RobustScaler()
scaled_features = scaler.fit_transform(features)
scaled_df = pd.DataFrame(scaled_features, columns=features.columns)
df[features.columns] = scaled_df
df.head()
```

Out[254]:

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.994983 | -0.616237 | -0.098602 | 1.228905 | 0.878152 | -0.217859 | 0.631245 | 0.177406 | 0.14 |
| **1** | -0.994983 | 0.524929 | 0.143100 | -0.006970 | 0.293974 | 0.087726 | 0.164395 | -0.105740 | 0.11 |
| **2** | -0.994972 | -0.615587 | -1.002407 | 0.830932 | 0.251024 | -0.344345 | 1.778007 | 0.668164 | 0.42 |
| **3** | -0.994972 | -0.440239 | -0.178789 | 0.841250 | -0.529808 | 0.033775 | 1.303832 | 0.175637 | 0.66 |
| **4** | -0.994960 | -0.526089 | 0.579239 | 0.713861 | 0.265632 | -0.270695 | 0.317183 | 0.491625 | -0.54 |

5 rows × 31 columns

**Given that i've only applied RobustScaler and not standardization, ill be focusing on models that are not sensitive to the scale of features. RobustScaler is suitable for datasets with outliers as it scales the features using statistics that are robust to outliers. Therefore, models that can handle varying scales and are robust to outliers would be appropriate. These include but are not limited to tree based models(Decision Trees, Random Forest), GBM's(XGboost, LightGBM)**

In [256]:
```
df.to_csv('transformed_data.csv', index=False)
```

In [ ]:
```
```

In [ ]:
```
```

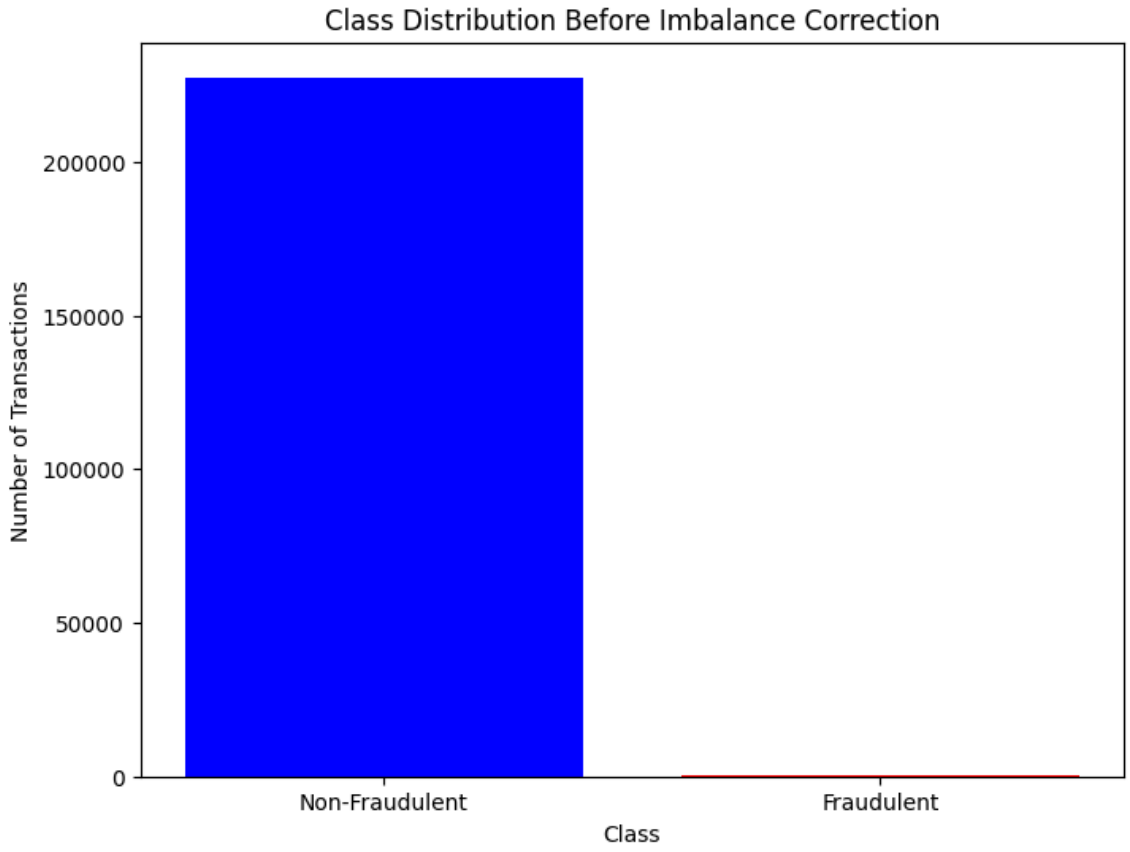
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

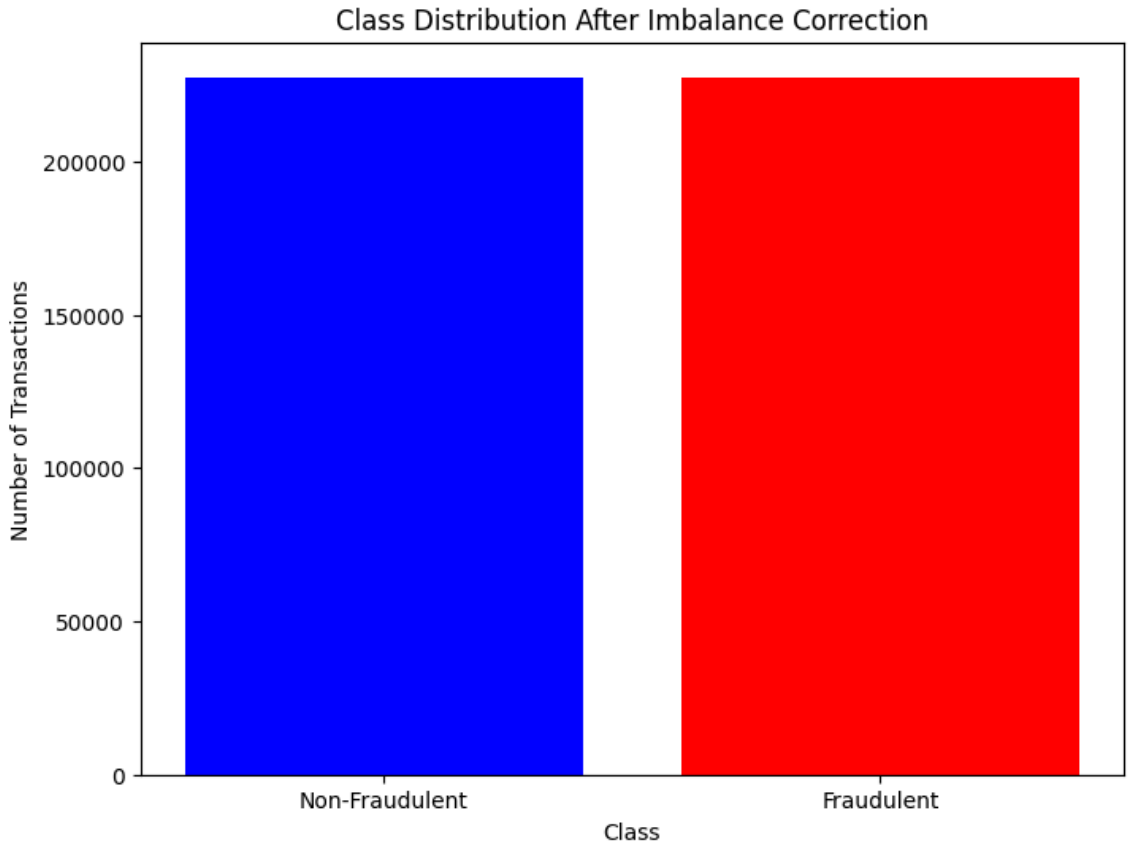
In [2]:
```
df= pd.read_csv("C:\\Users\\Sarrang\\FindDefault\\transformed_data.csv")
```

In [3]:
```
len(df)
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [ ]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
unique_classes, class_counts = np.unique(y_train, return_counts=True)
plt.figure(figsize=(8, 6))
plt.bar(unique_classes, class_counts, color=['blue', 'red'])
plt.xticks(unique_classes, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution Before Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()

# Calculate class distribution after imbalance correction
unique_classes_resampled, class_counts_resampled = np.unique(y_train_resamp

# Plot class distribution after imbalance correction
plt.figure(figsize=(8, 6))
plt.bar(unique_classes_resampled, class_counts_resampled, color=['blue', 'r
plt.xticks(unique_classes_resampled, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution After Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()
```






**Random forest hyperparameter tuning**

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report, precision_score

# Define the parameter grid to search
param_dist = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]
}

# Define the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=rf_classifier, param_distribut
                                   scoring='precision', cv=3, verbose=2, n_

# Fit the model
random_search.fit(X_train_resampled, y_train_resampled)

# Get the best parameters
best_params = random_search.best_params_
print("Best Parameters:", best_params)

# Make predictions
y_pred = random_search.predict(X_test)

# Calculate precision score
precision = precision_score(y_test, y_pred)
print("Precision Score:", precision)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424: F
utureWarning: `max_features='auto'` has been deprecated in 1.1 and will be
removed in 1.3. To keep the past behaviour, explicitly set `max_features
='sqrt'` or remove this parameter as it is also the default value for Rand
omForestClassifiers and ExtraTreesClassifiers.
  warn(
```

```
Best Parameters: {'n_estimators': 200, 'min_samples_split': 5, 'min_sample
s_leaf': 2, 'max_features': 'auto', 'max_depth': None, 'bootstrap': True}
Precision Score: 0.8625
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56875
           1       0.86      0.79      0.83        87

    accuracy                           1.00     56962
   macro avg       0.93      0.90      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

In [ ]:
```
classifier=RandomForestClassifier(n_estimators=200,
                                  min_samples_split= 5,
                                  min_samples_leaf=2,
                                  max_features='auto',
                                  max_depth=None,
                                  bootstrap=True)
classifier.fit(X_train_resampled,y_train_resampled)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424: F
utureWarning: `max_features='auto'` has been deprecated in 1.1 and will be
removed in 1.3. To keep the past behaviour, explicitly set `max_features
='sqrt'` or remove this parameter as it is also the default value for Rand
omForestClassifiers and ExtraTreesClassifiers.
  warn(
```

Out[10]:
```
RandomForestClassifier(max_features='auto', min_samples_leaf=2,
                       min_samples_split=5, n_estimators=200)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [ ]:
```
y_pred=classifier.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56875
           1       0.88      0.79      0.84        87

    accuracy                           1.00     56962
   macro avg       0.94      0.90      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

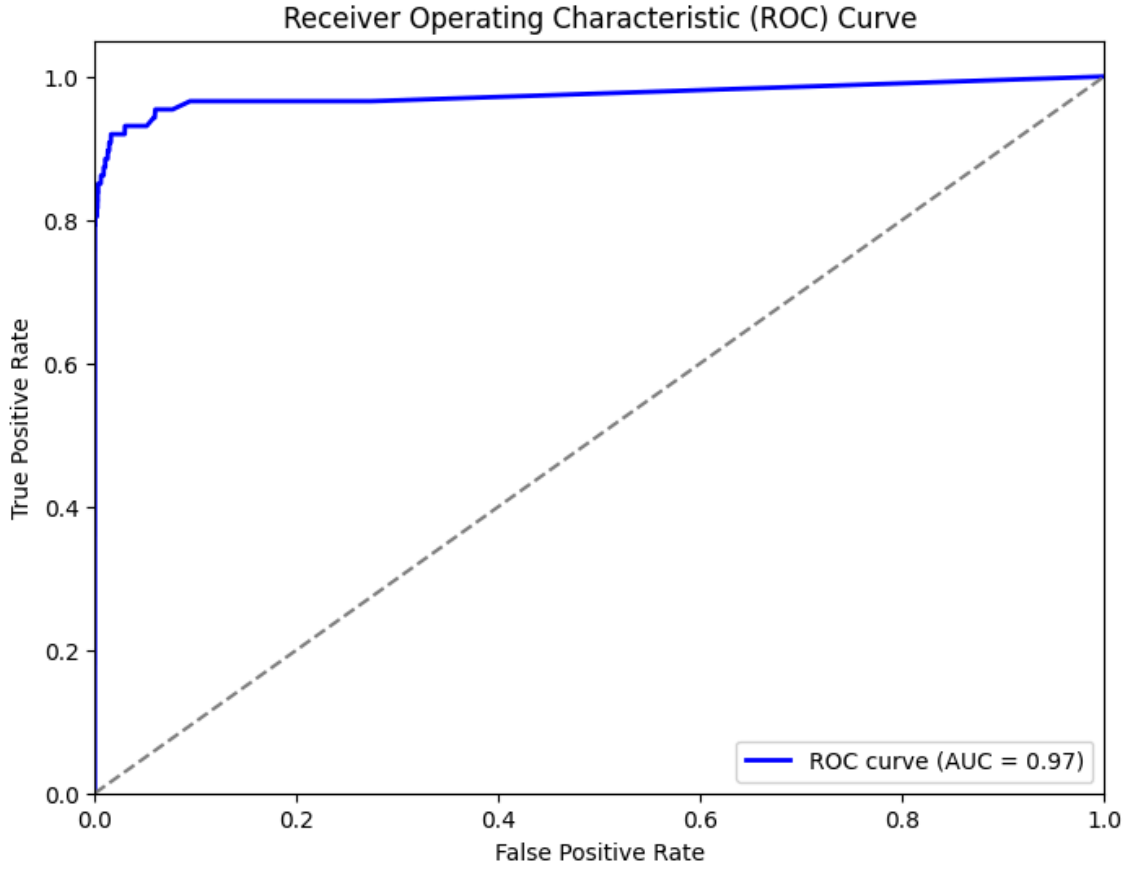
In [ ]:
```
import joblib
joblib.dump(classifier,'rf_smote_FD.joblib')
```

Out[17]:
```
['rf_smote_FD.joblib']
```

In [5]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\rf_smote_FD.joblib
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('randomforest smote Confusion Matrix')
plt.show()
```

```
[[56866    18]
 [    9    69]]
```

In [ ]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

y_pred_proba = classifier.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba[:, 1])

auc_roc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % au
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()


print(auc_roc)
```




```
0.9744640646709614
```

In [ ]:

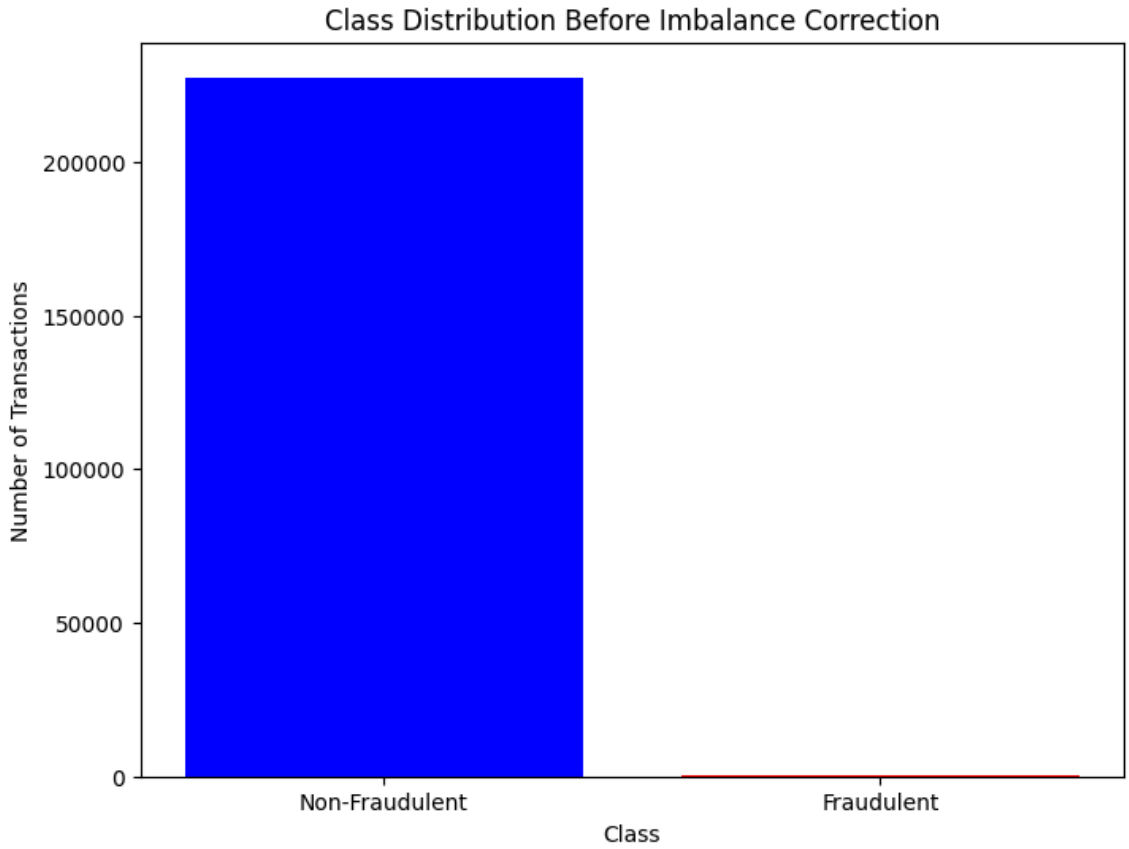
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

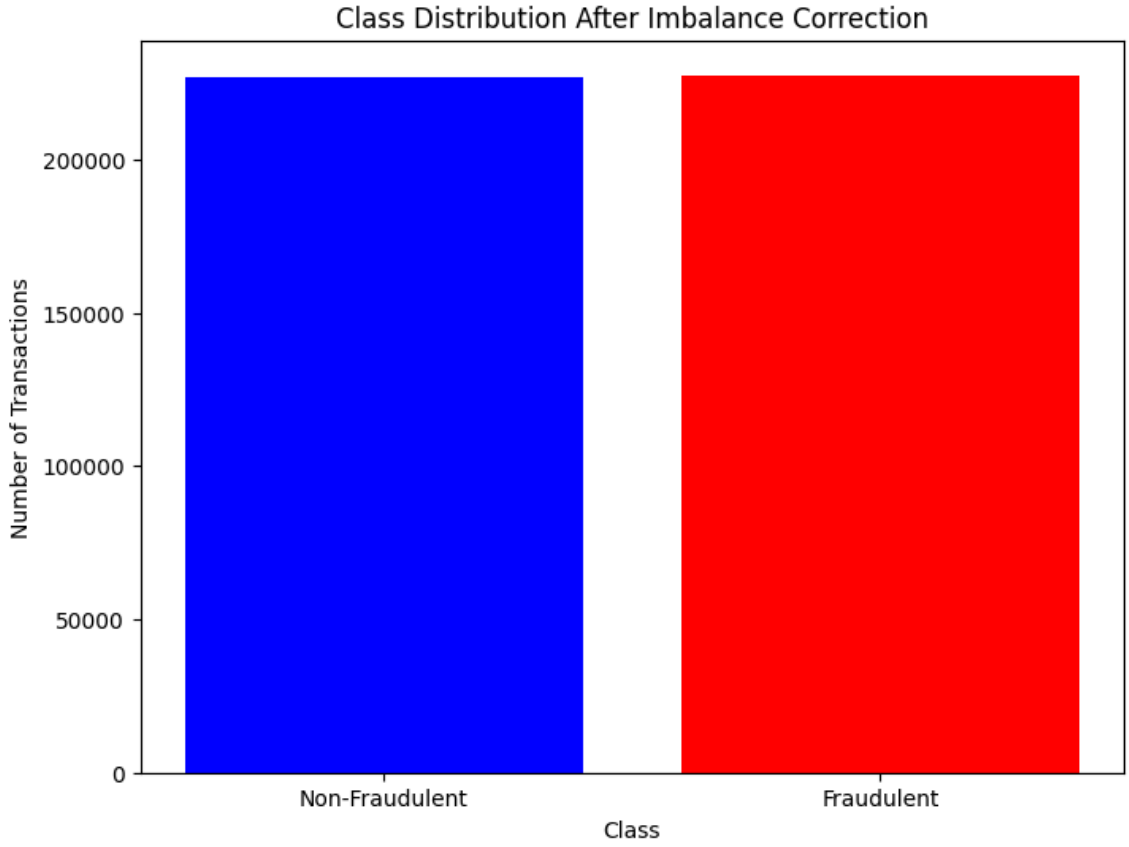
In [2]:
```
df= pd.read_csv("C:\\Users\\Sarrang\\FindDefault\\transformed_data.csv")
```

In [3]:
```
len(df)
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [6]:
```
from imblearn.combine import SMOTEENN
smoteenn = SMOTEENN(random_state=42)
X_train_resampled, y_train_resampled = smoteenn.fit_resample(X_train, y_tra

unique_classes, class_counts = np.unique(y_train, return_counts=True)
plt.figure(figsize=(8, 6))
plt.bar(unique_classes, class_counts, color=['blue', 'red'])
plt.xticks(unique_classes, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution Before Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()

# Calculate class distribution after imbalance correction
unique_classes_resampled, class_counts_resampled = np.unique(y_train_resamp

# Plot class distribution after imbalance correction
plt.figure(figsize=(8, 6))
plt.bar(unique_classes_resampled, class_counts_resampled, color=['blue', 'r
plt.xticks(unique_classes_resampled, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution After Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()
```






**Random forest hyperparameter tuning**

In [8]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report, precision_score

# Define the parameter grid to search
param_dist = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]
}

# Define the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=rf_classifier, param_distribut
                                   scoring='precision', cv=3, verbose=2, n_

# Fit the model
random_search.fit(X_train_resampled, y_train_resampled)

# Get the best parameters
best_params = random_search.best_params_
print("Best Parameters:", best_params)

# Make predictions
y_pred = random_search.predict(X_test)

# Calculate precision score
precision = precision_score(y_test, y_pred)
print("Precision Score:", precision)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424: F
utureWarning: `max_features='auto'` has been deprecated in 1.1 and will be
removed in 1.3. To keep the past behaviour, explicitly set `max_features
='sqrt'` or remove this parameter as it is also the default value for Rand
omForestClassifiers and ExtraTreesClassifiers.
  warn(
```

```
Best Parameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_sample
s_leaf': 4, 'max_features': 'auto', 'max_depth': None, 'bootstrap': False}
Precision Score: 0.7931034482758621
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56875
           1       0.79      0.79      0.79        87

    accuracy                           1.00     56962
   macro avg       0.90      0.90      0.90     56962
weighted avg       1.00      1.00      1.00     56962
```

In [9]:
```
classifier=RandomForestClassifier(n_estimators=100,
                                  min_samples_split=2,
                                  min_samples_leaf=4,
                                  max_features='auto',
                                  max_depth=None,
                                  bootstrap=False)

classifier.fit(X_train_resampled,y_train_resampled)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424: F
utureWarning: `max_features='auto'` has been deprecated in 1.1 and will be
removed in 1.3. To keep the past behaviour, explicitly set `max_features
='sqrt'` or remove this parameter as it is also the default value for Rand
omForestClassifiers and ExtraTreesClassifiers.
  warn(
```

Out[9]:
```
RandomForestClassifier(bootstrap=False, max_features='auto', min_samples_l
eaf=4)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [10]:
```
y_pred=classifier.predict(X_test)
print(classification_report(y_test,y_pred))
```

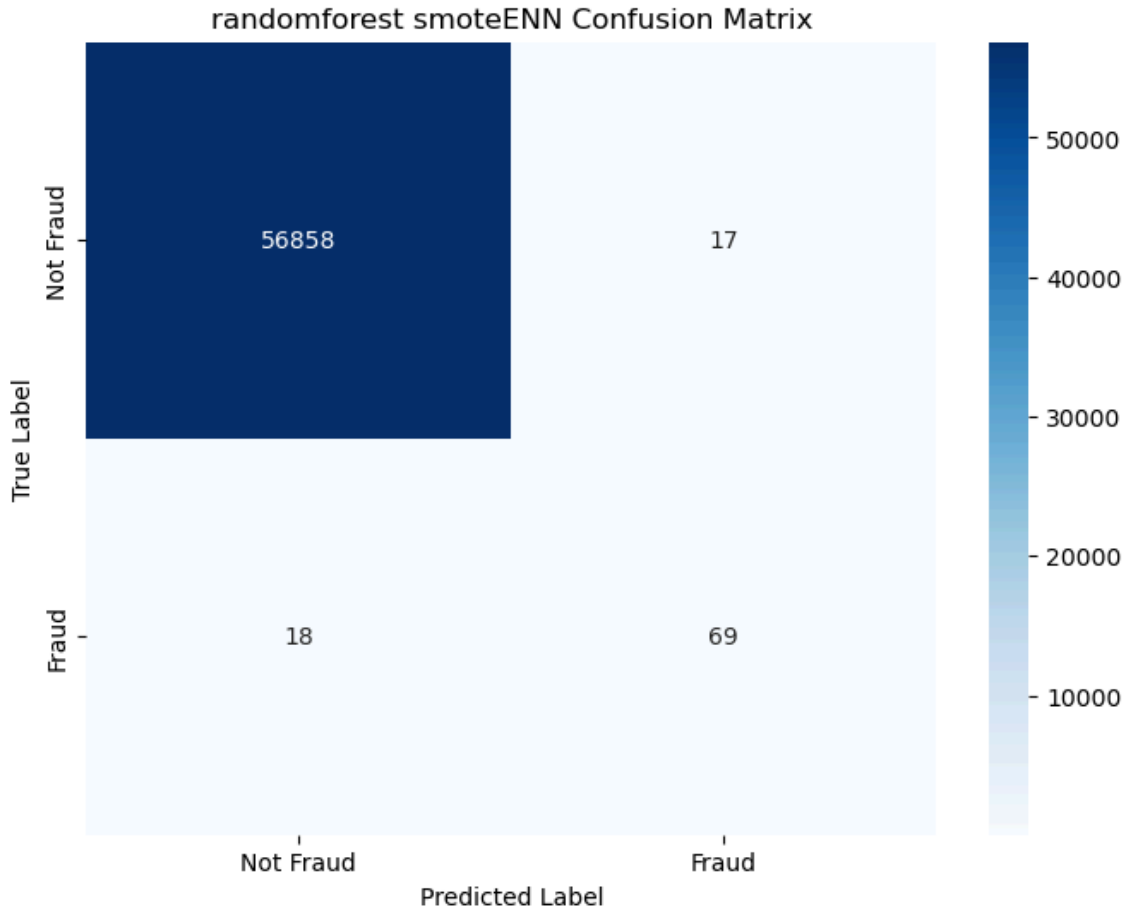
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56875
           1       0.80      0.79      0.80        87

    accuracy                           1.00     56962
   macro avg       0.90      0.90      0.90     56962
weighted avg       1.00      1.00      1.00     56962
```

In [11]:
```
import joblib
joblib.dump(classifier,'rf_smoteENN_FD.joblib')
```

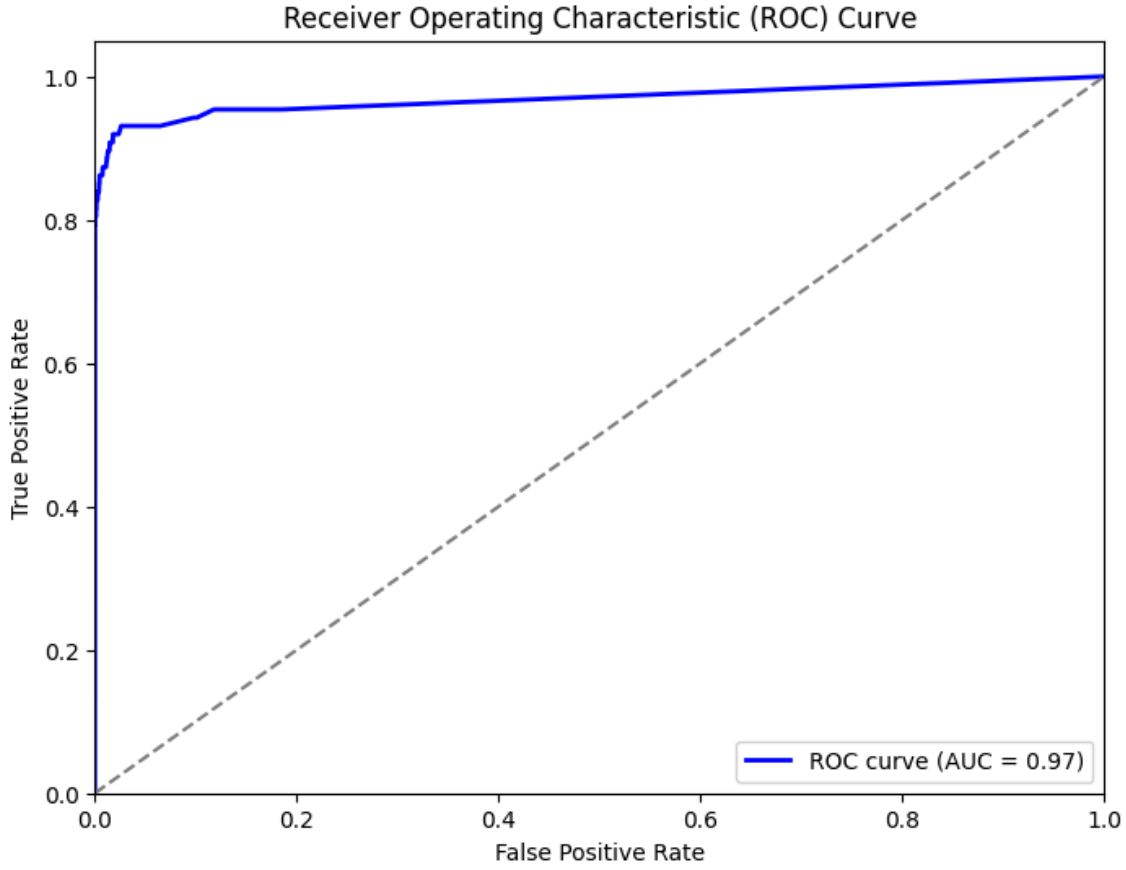
Out[11]:
```
['rf_smoteENN_FD.joblib']
```

In [5]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\rf_smoteENN_FD.job
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('randomforest smoteENN Confusion Matrix')
plt.show()
```

```
[[56858    18]
 [   17    69]]
```



In [13]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

y_pred_proba = classifier.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba[:, 1])

auc_roc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % au
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()


print(auc_roc)
```




```
0.9693589996210686
```

In [ ]:

In [ ]:
```
```

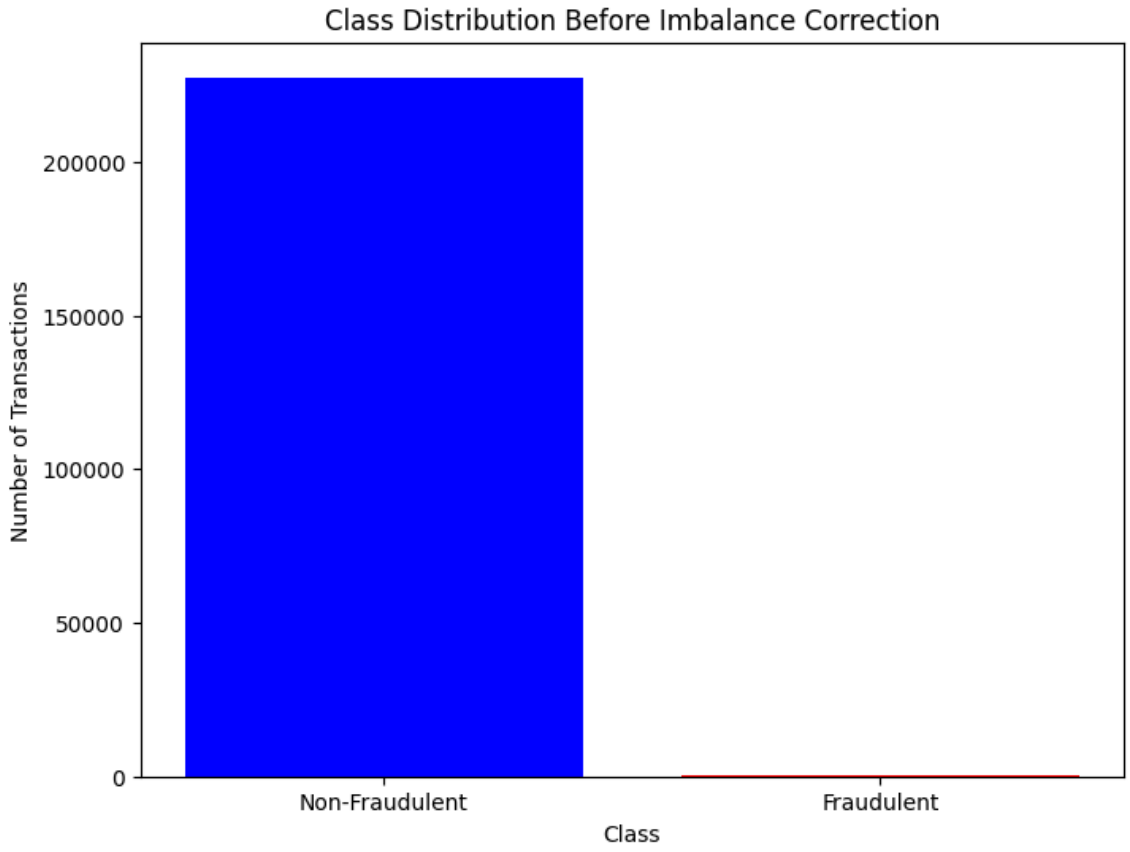
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

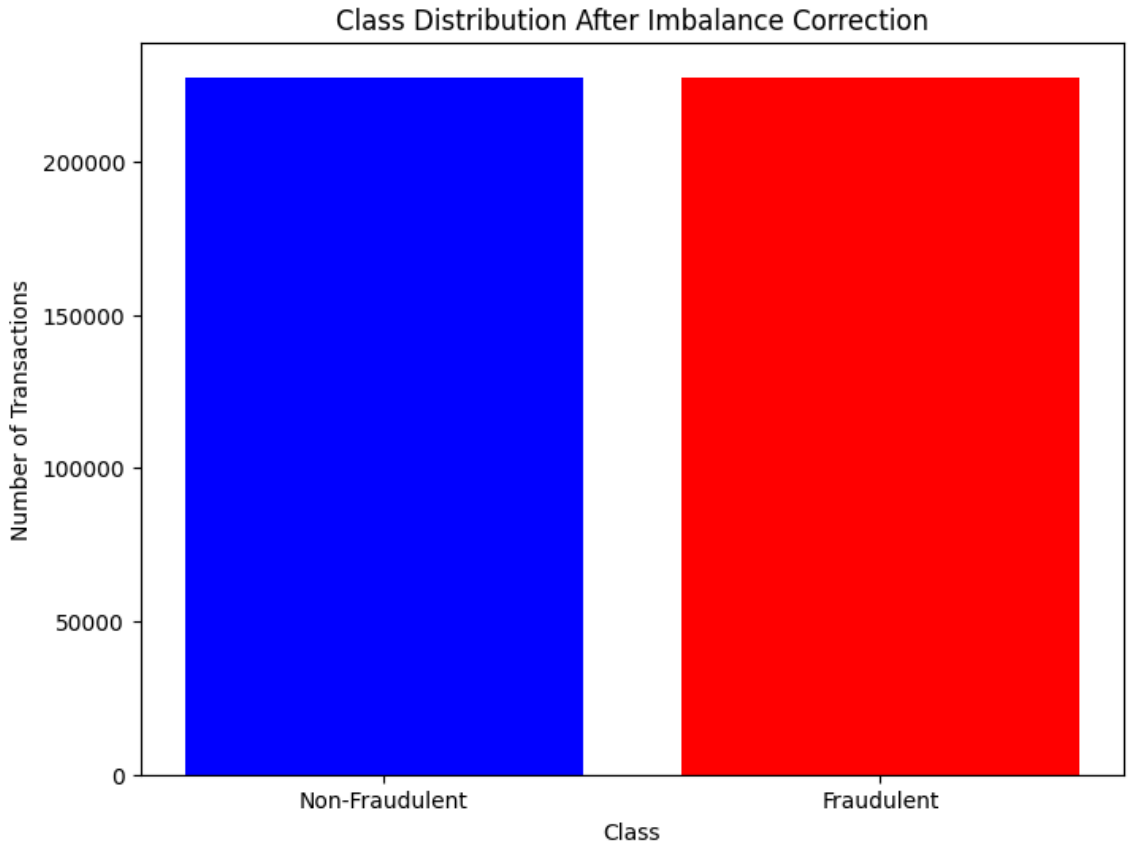
In [2]:
```
df= pd.read_csv("C:\\Users\\Sarrang\\FindDefault\\transformed_data.csv")
```

In [3]:
```
len(df)
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [5]:
```
from imblearn.over_sampling import ADASYN
adasyn = ADASYN(random_state=42)
X_train_resampled, y_train_resampled = adasyn.fit_resample(X_train, y_train
unique_classes, class_counts = np.unique(y_train, return_counts=True)
plt.figure(figsize=(8, 6))
plt.bar(unique_classes, class_counts, color=['blue', 'red'])
plt.xticks(unique_classes, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution Before Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()

# Calculate class distribution after imbalance correction
unique_classes_resampled, class_counts_resampled = np.unique(y_train_resamp

# Plot class distribution after imbalance correction
plt.figure(figsize=(8, 6))
plt.bar(unique_classes_resampled, class_counts_resampled, color=['blue', 'r
plt.xticks(unique_classes_resampled, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution After Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()
```






**Random forest hyperparameter tuning**

In [6]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report, precision_score

# Define the parameter grid to search
param_dist = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]
}

# Define the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=rf_classifier, param_distribut
                                   scoring='precision', cv=3, verbose=2, n_

# Fit the model
random_search.fit(X_train_resampled, y_train_resampled)

# Get the best parameters
best_params = random_search.best_params_
print("Best Parameters:", best_params)

# Make predictions
y_pred = random_search.predict(X_test)

# Calculate precision score
precision = precision_score(y_test, y_pred)
print("Precision Score:", precision)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Parameters: {'n_estimators': 50, 'min_samples_split': 10, 'min_sample
s_leaf': 1, 'max_features': 'log2', 'max_depth': None, 'bootstrap': False}
Precision Score: 0.9066666666666666
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56875
           1       0.91      0.78      0.84        87

    accuracy                           1.00     56962
   macro avg       0.95      0.89      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

In [7]:
```
classifier=RandomForestClassifier(n_estimators=50, min_samples_split=10, mi
classifier.fit(X_train_resampled,y_train_resampled)
```

Out[7]:
```
RandomForestClassifier(bootstrap=False, max_features='log2',
                       min_samples_split=10, n_estimators=50)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [8]:
```
y_pred=classifier.predict(X_test)
print(classification_report(y_test,y_pred))
```

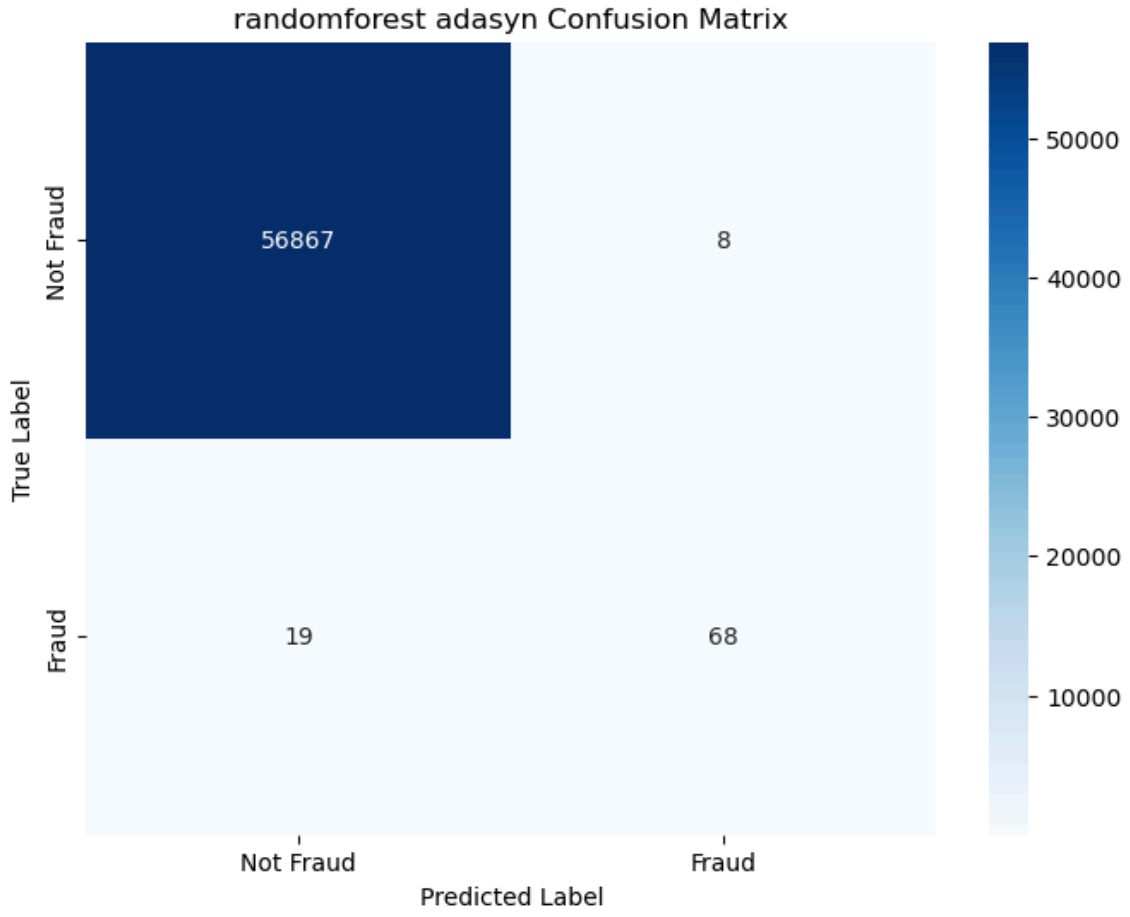
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56875
           1       0.89      0.78      0.83        87

    accuracy                           1.00     56962
   macro avg       0.95      0.89      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

In [9]:
```
import joblib
joblib.dump(classifier,'rf_adasyn_FD.joblib')
```

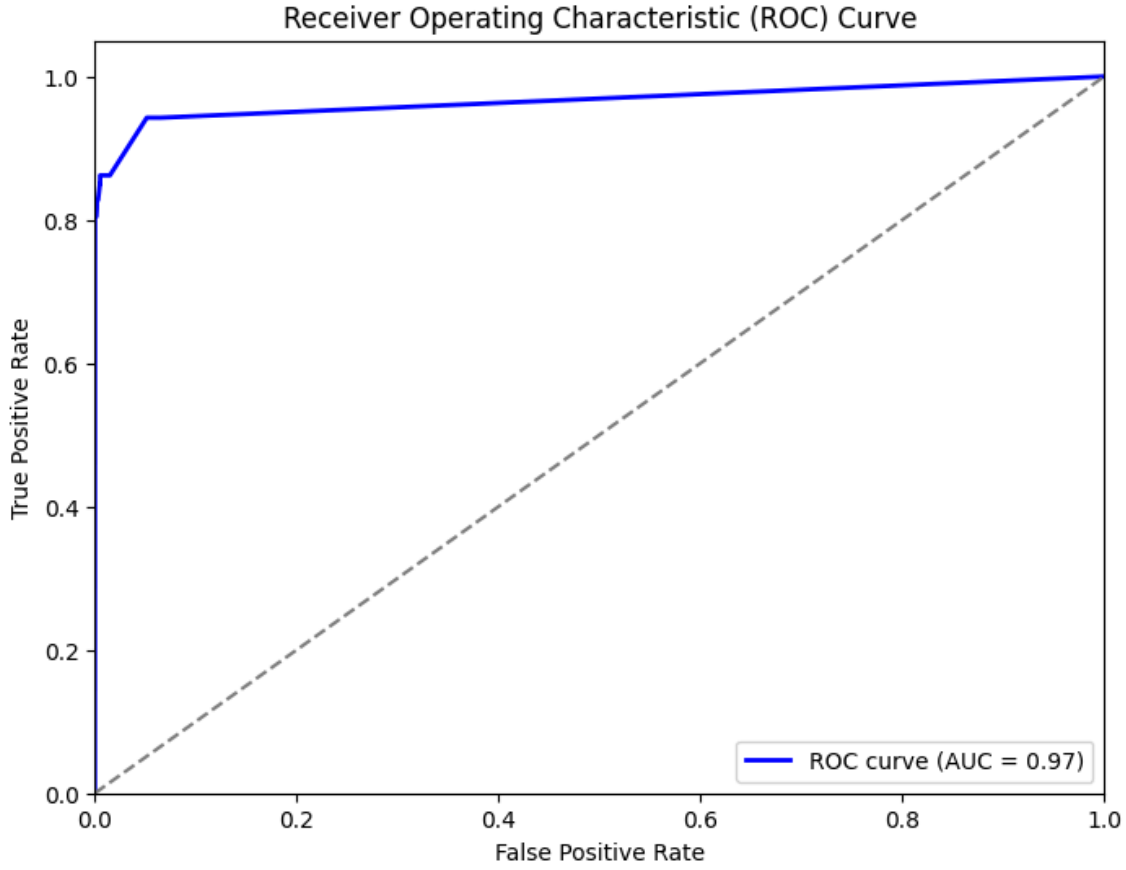
Out[9]:
```
['rf_adasyn_FD.joblib']
```

In [5]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\rf_adasyn_FD.jobli
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('randomforest adasyn Confusion Matrix')
plt.show()
```

```
[[56867    19]
 [    8    68]]
```



In [11]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

y_pred_proba = classifier.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba[:, 1])

auc_roc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % au
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()


print(auc_roc)
```




```
0.966496905393457
```

In [ ]:

# Importing the libraries

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

# Importing the dataset

In [2]:
```
df= pd.read_csv("C:\\Users\\Sarrang\\FindDefault\\transformed_data.csv")
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

# Splitting the dataset into the Training set and Test set

In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [30]:
```
len(df)
```

Out[30]: 284807

In [31]:
```
from imblearn.over_sampling import SMOTE
smote=SMOTE(random_state=42)
X_train_resampled,y_train_resampled=smote.fit_resample(X_train,y_train)
```

In [ ]:
```

```

In [32]:
```
from lightgbm import LGBMClassifier
from sklearn.model_selection import GridSearchCV

lgbm = LGBMClassifier()

param_grid = {
    'num_leaves': [20, 30, 40],
    'learning_rate': [0.05, 0.1, 0.2],
    'n_estimators': [50, 100, 150],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

grid_search = GridSearchCV(estimator=lgbm, param_grid=param_grid, cv=5, sco

grid_search.fit(X_train_resampled, y_train_resampled)

print("Best parameters found: ", grid_search.best_params_)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
[LightGBM] [Info] Number of positive: 227454, number of negative: 227454
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.116573 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 454908, number o
f used features: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.00
0000
Best parameters found:  {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'n
_estimators': 100, 'num_leaves': 40, 'subsample': 0.8}
```

In [33]:
```
classifier=LGBMClassifier(colsample_bytree=1.0,
                          learning_rate=0.2,
                          n_estimators=100,
                          num_leaves=40,
                          subsample=0.8)
classifier.fit(X_train_resampled,y_train_resampled)
```

```
[LightGBM] [Info] Number of positive: 227454, number of negative: 227454
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.117622 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 454908, number o
f used features: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.00
0000
```

Out[33]:
```
LGBMClassifier(learning_rate=0.2, num_leaves=40, subsample=0.8)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [34]:
```
y_pred=classifier.predict(X_test)
```

In [35]:
```
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

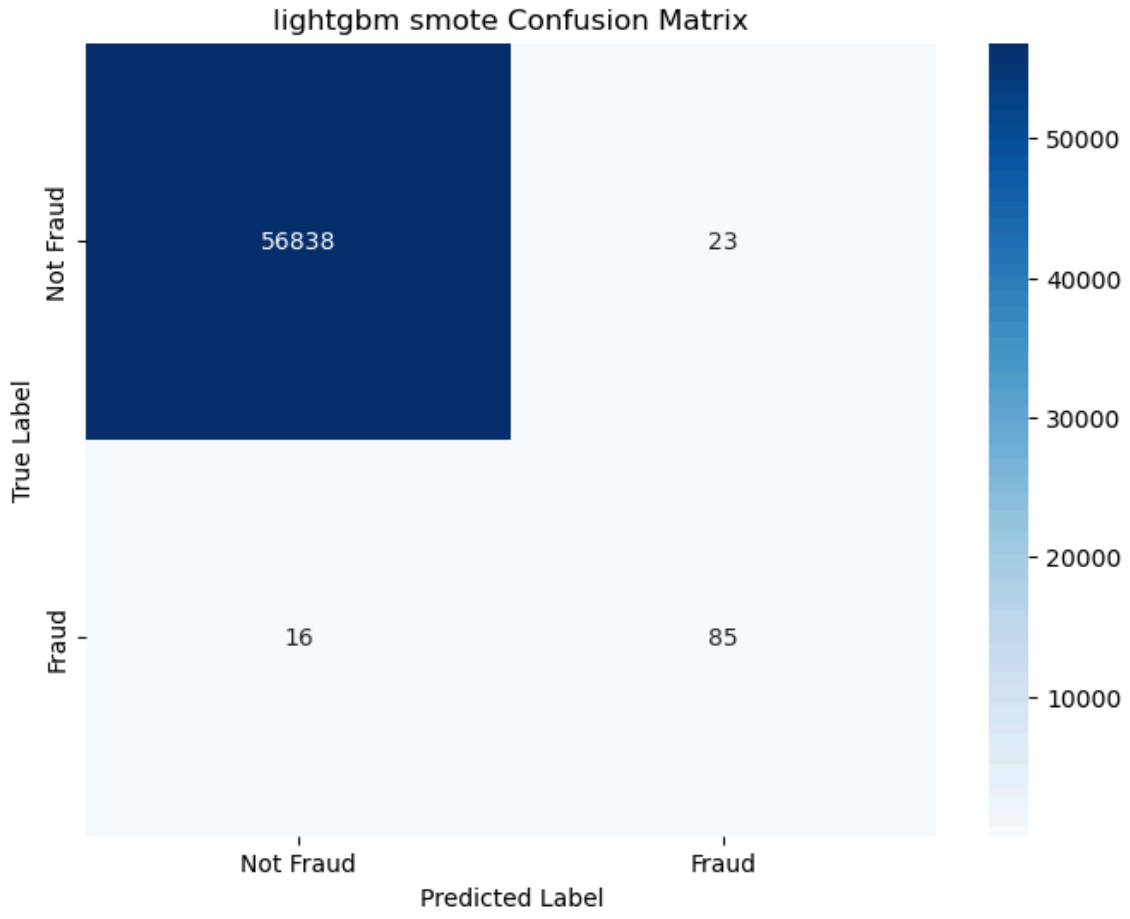
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56854
           1       0.84      0.79      0.81       108

    accuracy                           1.00     56962
   macro avg       0.92      0.89      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

In [36]:
```
import joblib
joblib.dump(classifier, 'lightgbm_smote_FD.joblib')
```

Out[36]:
```
['lightgbm_smote_FD.joblib']
```

In [6]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\lightgbm_smote_FD.
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('lightgbm smote Confusion Matrix')
plt.show()
```

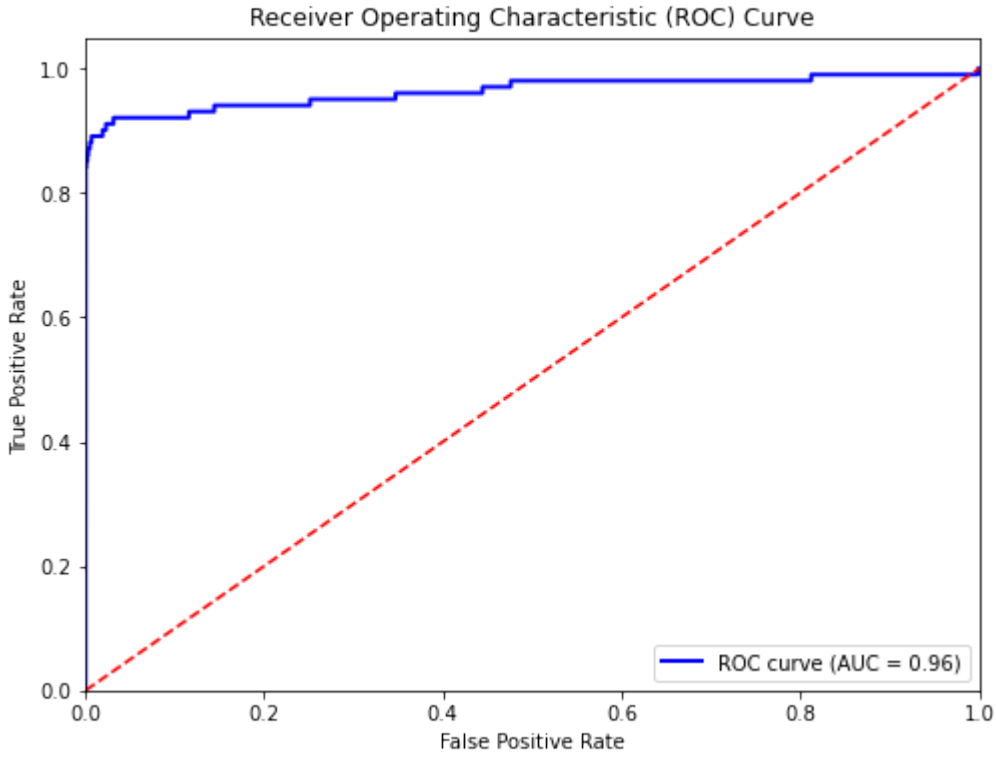
```
C:\Users\Sarrang\anaconda3\Lib\site-packages\sklearn\base.py:318: UserWarn
ing: Trying to unpickle estimator LabelEncoder from version 1.4.1.post1 wh
en using version 1.2.2. This might lead to breaking code or invalid result
s. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainab
ility-limitations (https://scikit-learn.org/stable/model_persistence.html#
security-maintainability-limitations)
  warnings.warn(

[[56838    16]
 [   23    85]]
```



In [37]:
```
from sklearn.metrics import roc_auc_score
y_pred_proba = classifier.predict_proba(X_test)
y_pred_proba_positive = y_pred_proba[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_proba_positive)
print("ROC AUC Score:", roc_auc)
```

```
ROC AUC Score: 0.963512202154951
```

In [38]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_positive)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % ro
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




feature importance

In [39]:
```
import numpy as np

# Extract feature importances
feature_importances = classifier.feature_importances_

# Get feature names
feature_names = df.columns

# Sort feature importances in descending order
indices = np.argsort(feature_importances)[::-1]

# Print feature ranking
print("Feature ranking:")
for f in range(X_train.shape[1]):
    print("%d. feature %s (%f)" % (f + 1, feature_names[indices[f]], featur
```

```
Feature ranking:
1. feature V14 (204.000000)
2. feature V1 (187.000000)
3. feature Amount (179.000000)
4. feature V4 (172.000000)
5. feature V25 (167.000000)
6. feature V24 (166.000000)
7. feature V18 (158.000000)
8. feature Time (157.000000)
9. feature V11 (155.000000)
10. feature V26 (146.000000)
11. feature V3 (145.000000)
12. feature V16 (144.000000)
13. feature V12 (141.000000)
14. feature V5 (126.000000)
15. feature V20 (119.000000)
16. feature V17 (116.000000)
17. feature V8 (114.000000)
18. feature V15 (111.000000)
19. feature V21 (111.000000)
20. feature V7 (109.000000)
21. feature V19 (108.000000)
22. feature V22 (107.000000)
23. feature V27 (105.000000)
24. feature V2 (105.000000)
25. feature V13 (104.000000)
26. feature V28 (102.000000)
27. feature V10 (99.000000)
28. feature V9 (83.000000)
29. feature V6 (81.000000)
30. feature V23 (79.000000)
```

In [ ]:

# Importing the libraries

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

# Importing the dataset

In [2]:
```
df= pd.read_csv("C:\\Users\\Sarrang\\FindDefault\\transformed_data.csv")
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

# Splitting the dataset into the Training set and Test set

In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [4]:
```
len(df)
```

Out[4]:
```
284807
```

In [6]:
```
from imblearn.combine import SMOTEENN
smote_enn = SMOTEENN(random_state=42)
X_train_resampled, y_train_resampled = smote_enn.fit_resample(X_train, y_tr
```

```
C:\Users\Sarrang9\anaconda3\lib\site-packages\joblib\externals\loky\backen
d\context.py:136: UserWarning: Could not find the number of physical cores
for the following reason:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this warnin
g by setting LOKY_MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "C:\Users\Sarrang9\anaconda3\lib\site-packages\joblib\externals\lok
y\backend\context.py", line 257, in _count_physical_cores
    cpu_info = subprocess.run(
  File "C:\Users\Sarrang9\anaconda3\lib\subprocess.py", line 505, in run
    with Popen(*popenargs, **kwargs) as process:
  File "C:\Users\Sarrang9\anaconda3\lib\subprocess.py", line 951, in __ini
t__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "C:\Users\Sarrang9\anaconda3\lib\subprocess.py", line 1420, in _exe
cute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
```

In [ ]:
```
```

In [7]:
```
from lightgbm import LGBMClassifier
from sklearn.model_selection import GridSearchCV

lgbm = LGBMClassifier()

param_grid = {
    'num_leaves': [20, 30, 40],
    'learning_rate': [0.05, 0.1, 0.2],
    'n_estimators': [50, 100, 150],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

grid_search = GridSearchCV(estimator=lgbm, param_grid=param_grid, cv=5, sco

grid_search.fit(X_train_resampled, y_train_resampled)

print("Best parameters found: ", grid_search.best_params_)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
[LightGBM] [Info] Number of positive: 227454, number of negative: 226896
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.120449 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 454350, number o
f used features: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500614 -> initscore=0.00
2456
[LightGBM] [Info] Start training from score 0.002456
Best parameters found:  {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'n
_estimators': 150, 'num_leaves': 40, 'subsample': 0.8}
```

In [8]:
```
classifier=LGBMClassifier(colsample_bytree=1.0,
                          learning_rate=0.2,
                          n_estimators=150,
                          num_leaves=40,
                          subsample=0.8)
classifier.fit(X_train_resampled,y_train_resampled)
```

```
[LightGBM] [Info] Number of positive: 227454, number of negative: 226896
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.115573 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 454350, number o
f used features: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500614 -> initscore=0.00
2456
[LightGBM] [Info] Start training from score 0.002456
```

Out[8]:
```
LGBMClassifier(learning_rate=0.2, n_estimators=150, num_leaves=40,
               subsample=0.8)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [9]:
```
y_pred=classifier.predict(X_test)
```

In [10]:
```
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

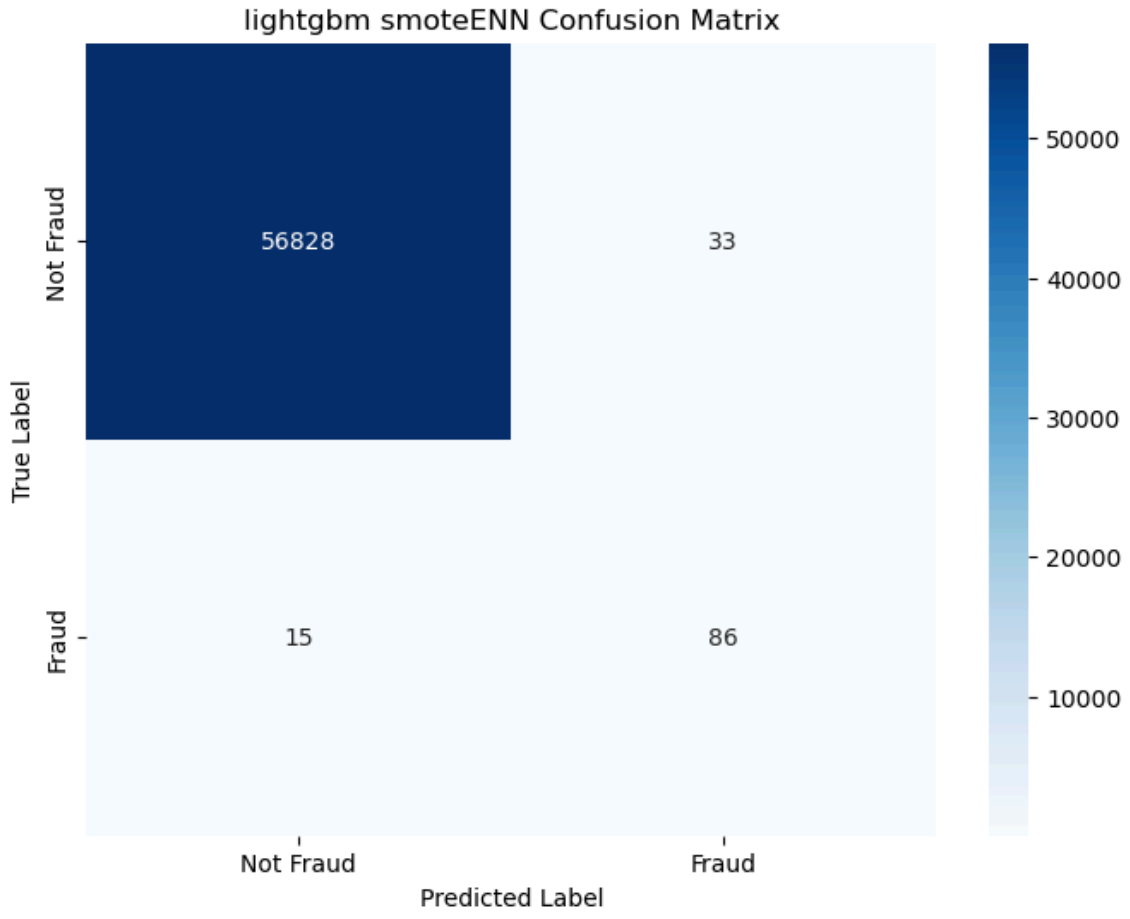
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56843
           1       0.85      0.72      0.78       119

    accuracy                           1.00     56962
   macro avg       0.93      0.86      0.89     56962
weighted avg       1.00      1.00      1.00     56962
```

In [11]:
```
import joblib
joblib.dump(classifier, 'lightgbm_smoteENN_FD.joblib')
```

Out[11]:
```
['lightgbm_smoteENN_FD.joblib']
```

In [6]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\lightgbm_smoteENN_
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('lightgbm smoteENN Confusion Matrix')
plt.show()
```

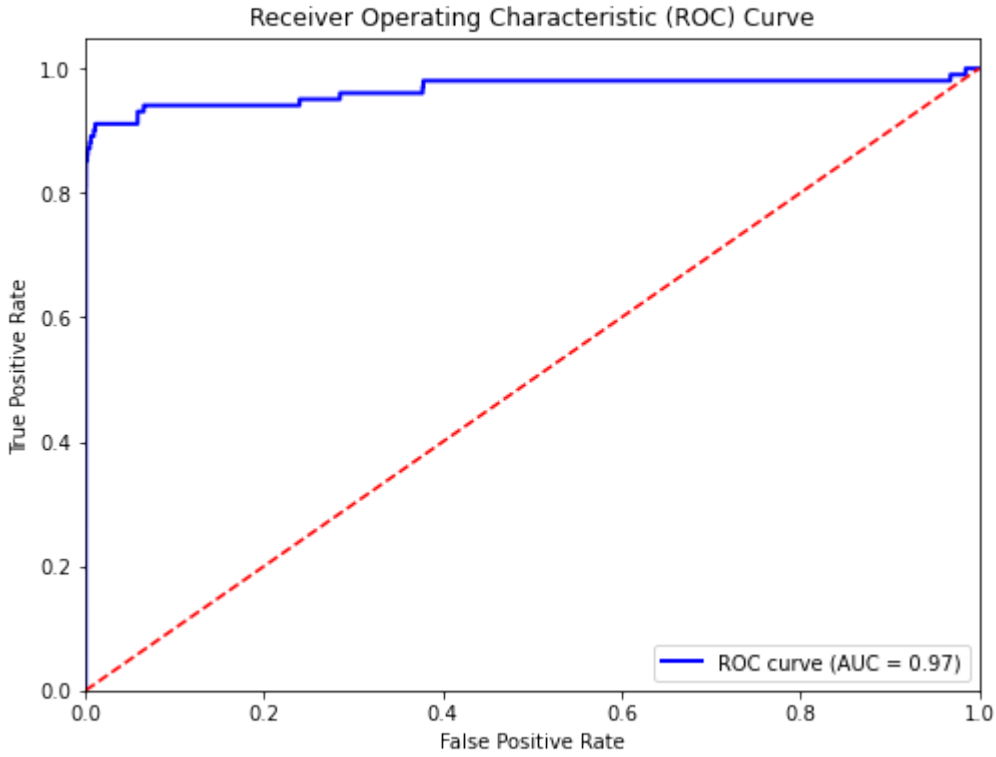
```
C:\Users\Sarrang\anaconda3\Lib\site-packages\sklearn\base.py:318: UserWarn
ing: Trying to unpickle estimator LabelEncoder from version 1.4.1.post1 wh
en using version 1.2.2. This might lead to breaking code or invalid result
s. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainab
ility-limitations (https://scikit-learn.org/stable/model_persistence.html#
security-maintainability-limitations)
  warnings.warn(

[[56828    15]
 [   33    86]]
```



In [12]:
```
from sklearn.metrics import roc_auc_score
y_pred_proba = classifier.predict_proba(X_test)
y_pred_proba_positive = y_pred_proba[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_proba_positive)
print("ROC AUC Score:", roc_auc)
```

```
ROC AUC Score: 0.9657920365470007
```

In [13]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_positive)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % ro
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



# Importing the libraries

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

# Importing the dataset

In [3]:
```
df= pd.read_csv("C:\\Users\\Sarrang\\FindDefault\\transformed_data.csv")
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

# Splitting the dataset into the Training set and Test set

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [26]:
```
len(df)
```

Out[26]: 284807

In [27]:
```
from imblearn.over_sampling import ADASYN

adasyn = ADASYN(random_state=42)
X_train_resampled, y_train_resampled = adasyn.fit_resample(X_train, y_train
```

In [ ]:
```

```

In [28]:
```
from lightgbm import LGBMClassifier
from sklearn.model_selection import GridSearchCV

lgbm = LGBMClassifier()

param_grid = {
    'num_leaves': [20, 30, 40],
    'learning_rate': [0.05, 0.1, 0.2],
    'n_estimators': [50, 100, 150],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

grid_search = GridSearchCV(estimator=lgbm, param_grid=param_grid, cv=5, sco

grid_search.fit(X_train_resampled, y_train_resampled)

print("Best parameters found: ", grid_search.best_params_)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
[LightGBM] [Info] Number of positive: 227489, number of negative: 227454
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.114935 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 454943, number o
f used features: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500038 -> initscore=0.00
0154
[LightGBM] [Info] Start training from score 0.000154
Best parameters found:  {'colsample_bytree': 0.8, 'learning_rate': 0.2, 'n
_estimators': 150, 'num_leaves': 30, 'subsample': 0.8}
```

In [29]:
```
classifier=LGBMClassifier(colsample_bytree=0.8,
                          learning_rate=0.2,
                          n_estimators=150,
                          num_leaves=30,
                          subsample=0.8)
classifier.fit(X_train_resampled,y_train_resampled)
```

```
[LightGBM] [Info] Number of positive: 227489, number of negative: 227454
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.118345 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 454943, number o
f used features: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500038 -> initscore=0.00
0154
[LightGBM] [Info] Start training from score 0.000154
```

Out[29]:
```
LGBMClassifier(colsample_bytree=0.8, learning_rate=0.2, n_estimators=150,
               num_leaves=30, subsample=0.8)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [30]:
```
y_pred=classifier.predict(X_test)
```

In [31]:
```
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

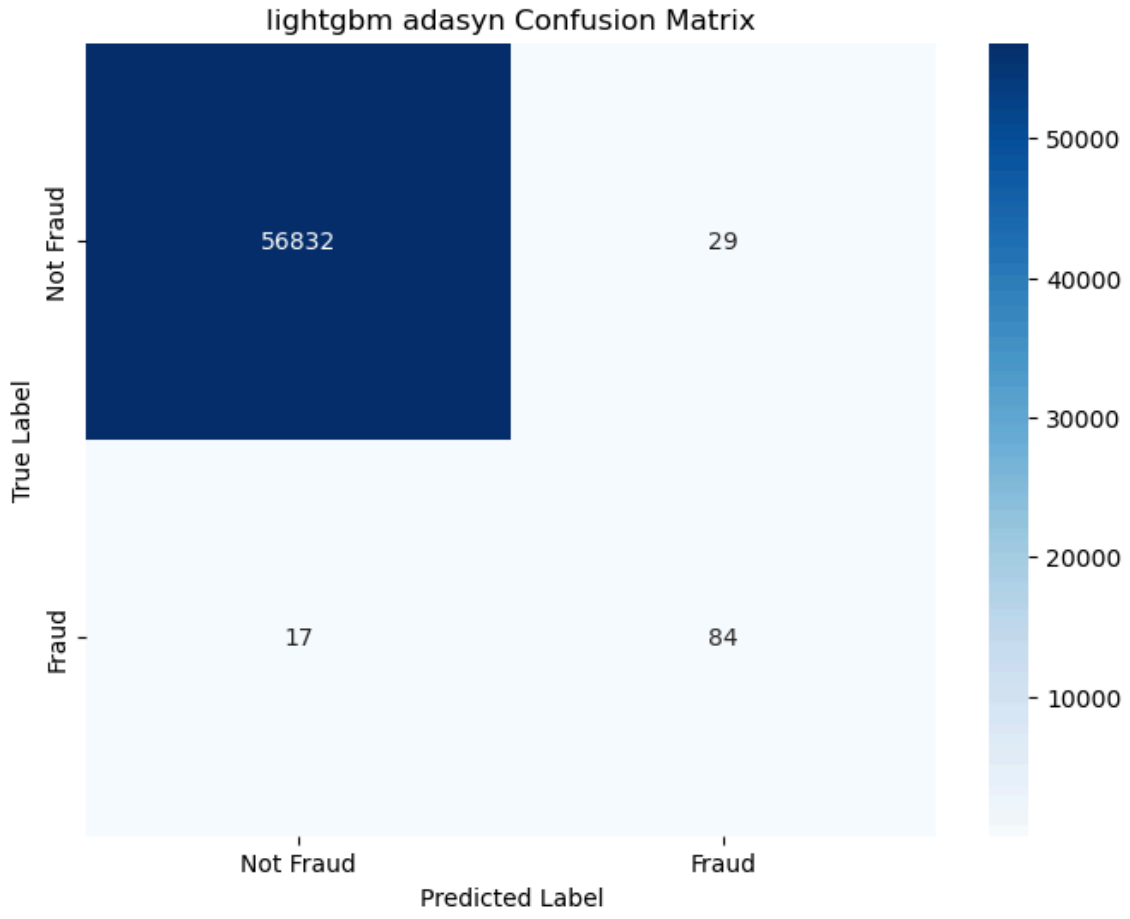
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56849
           1       0.83      0.74      0.79       113

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

In [32]:
```
import joblib
joblib.dump(classifier, 'lightgbm_adasyn_FD.joblib')
```

Out[32]:
```
['lightgbm_adasyn_FD.joblib']
```

In [5]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\lightgbm_adasyn_FD
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('lightgbm adasyn Confusion Matrix')
plt.show()
```

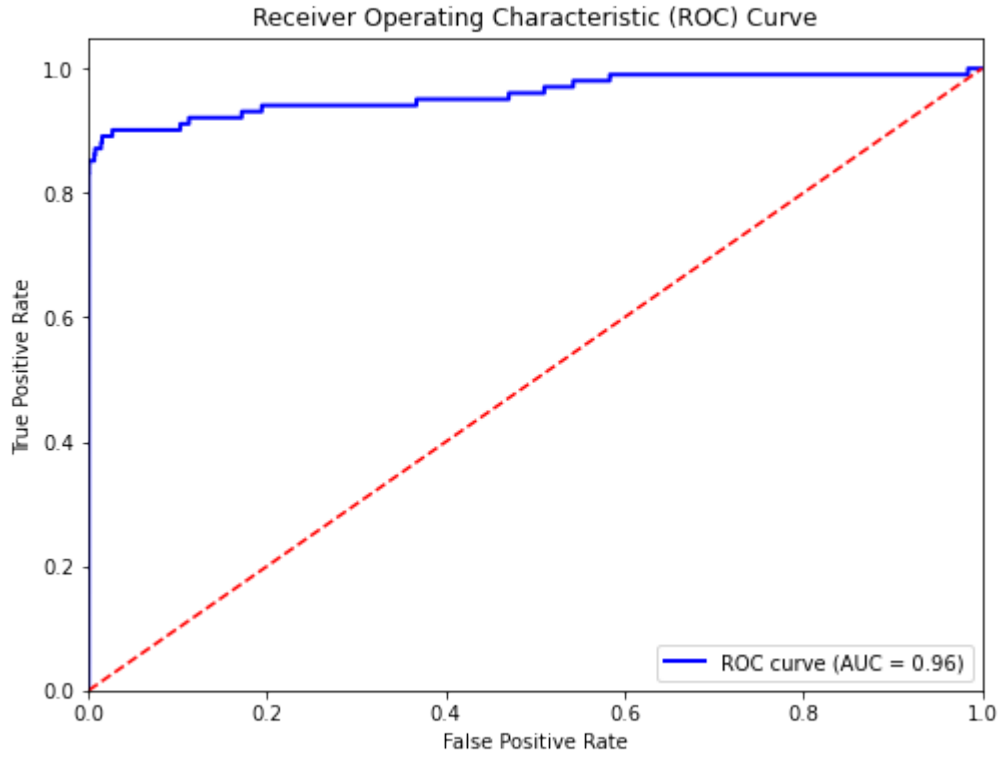
```
C:\Users\Sarrang\anaconda3\Lib\site-packages\sklearn\base.py:318: UserWarn
ing: Trying to unpickle estimator LabelEncoder from version 1.4.1.post1 wh
en using version 1.2.2. This might lead to breaking code or invalid result
s. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainab
ility-limitations (https://scikit-learn.org/stable/model_persistence.html#
security-maintainability-limitations)
  warnings.warn(

[[56832    17]
 [   29    84]]
```



In [33]:
```
from sklearn.metrics import roc_auc_score
y_pred_proba = classifier.predict_proba(X_test)
y_pred_proba_positive = y_pred_proba[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_proba_positive)
print("ROC AUC Score:", roc_auc)
```

```
ROC AUC Score: 0.959246110151192
```

In [34]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_positive)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % ro
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




feature importance

In [ ]:
```
```

# Importing the libraries

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

# Importing the dataset

In [3]:
```
df = pd.read_csv("C:\\Users\\Sarrang\\FindDefault\\transformed_data.csv")
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [85]:
```
df.value_counts('Class')
```

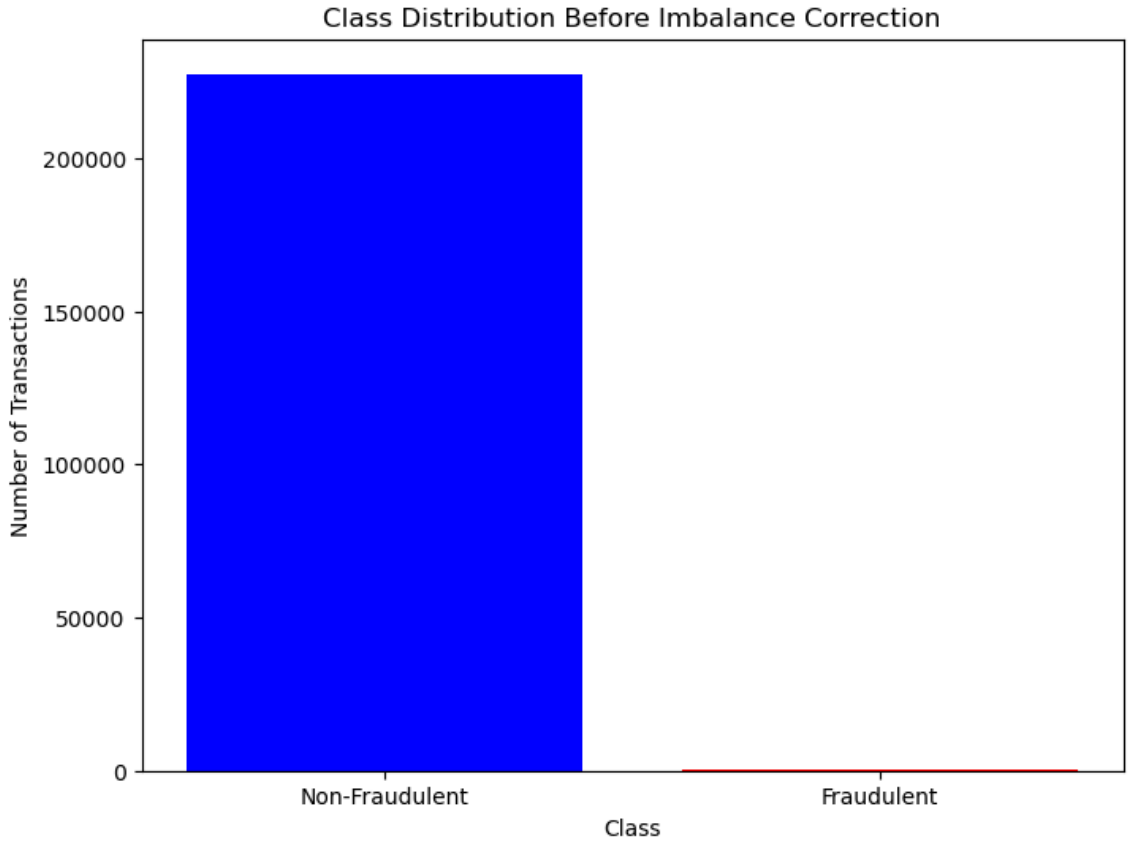
Out[85]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

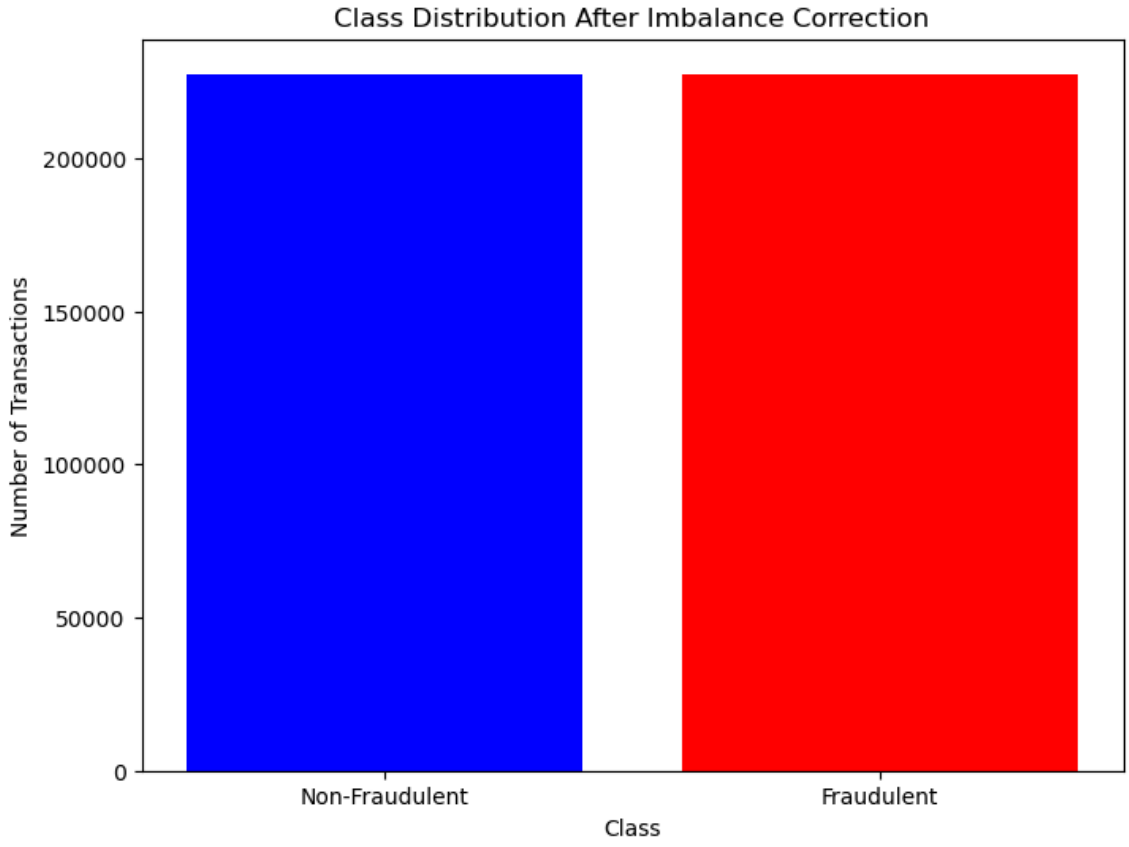
# Splitting the dataset into the Training set and Test set

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [87]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

In [71]:
```
import matplotlib.pyplot as plt
unique_classes, class_counts = np.unique(y_train, return_counts=True)
plt.figure(figsize=(8, 6))
plt.bar(unique_classes, class_counts, color=['blue', 'red'])
plt.xticks(unique_classes, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution Before Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()
unique_classes_resampled, class_counts_resampled = np.unique(y_train_resamp
plt.figure(figsize=(8, 6))
plt.bar(unique_classes_resampled, class_counts_resampled, color=['blue', 'r
plt.xticks(unique_classes_resampled, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution After Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()
```






## XGBoost hyperparameter tuning

In [76]:
```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, precision_score
param_grid = {
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'scale_pos_weight': [1, 10, 25]
}
xgb_classifier = xgb.XGBClassifier(objective='binary:logistic', random_stat
grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid,
                           scoring='precision', cv=3, verbose=2, n_jobs=-1)
grid_search.fit(X_train_resampled, y_train_resampled)
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
y_pred = grid_search.predict(X_test)
precision = precision_score(y_test, y_pred)
print("Precision Score:", precision)
```

```
Fitting 3 folds for each of 2187 candidates, totalling 6561 fits
Best Parameters: {'colsample_bytree': 1.0, 'gamma': 0.1, 'learning_rate':
0.2, 'max_depth': 5, 'min_child_weight': 1, 'scale_pos_weight': 1, 'subsam
ple': 0.8}
Precision Score: 0.45751633986928103
```

In [90]:
```
classifier=xgb.XGBClassifier()
"""
USING DEFAULT PARAMETERS IS GIVING BETTER RESULTS THAN ALTERING THEM WITH H
"""
```

In [78]:
```
# classifier=XGBClassifier(colsample_bytree=1.0,
#                          gamma=0.1,
#                          learning_rate=0.2,
#                          max_depth=5,
#                          min_child_weight=1,
#                          scale_pos_weight=1,
#                          subsample=0.8)
```

In [91]:
```
classifier.fit(X_train_resampled,y_train_resampled)
```

Out[91]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=No
ne,
              enable_categorical=False, eval_metric=None, feature_types=No
ne,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=No
ne,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [92]:
```
y_pred=classifier.predict(X_test)
```

In [93]:
```
print(classification_report(y_test,y_pred))
```

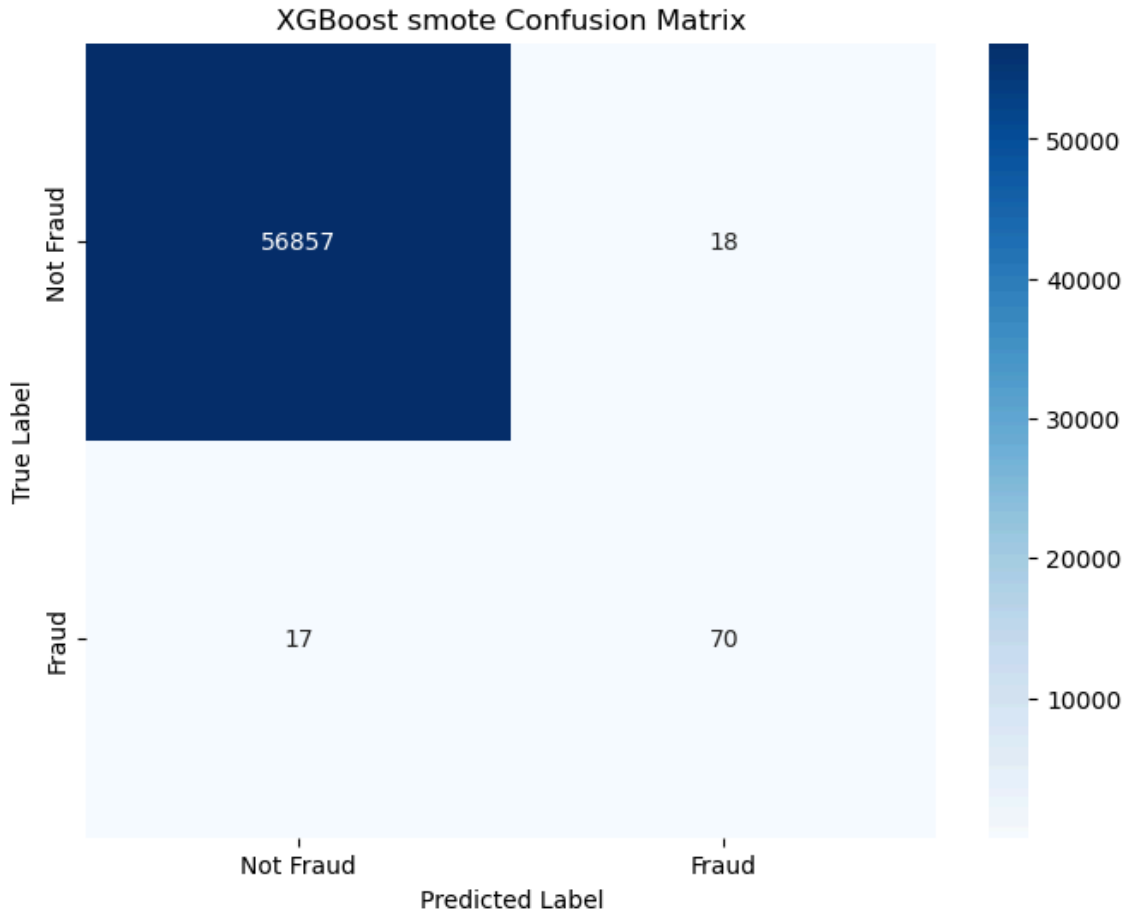
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56875
           1       0.80      0.80      0.80        87

    accuracy                           1.00     56962
   macro avg       0.90      0.90      0.90     56962
weighted avg       1.00      1.00      1.00     56962
```

In [96]:
```
import joblib
joblib.dump(classifier,'xgb_smote_FD.joblib')
```

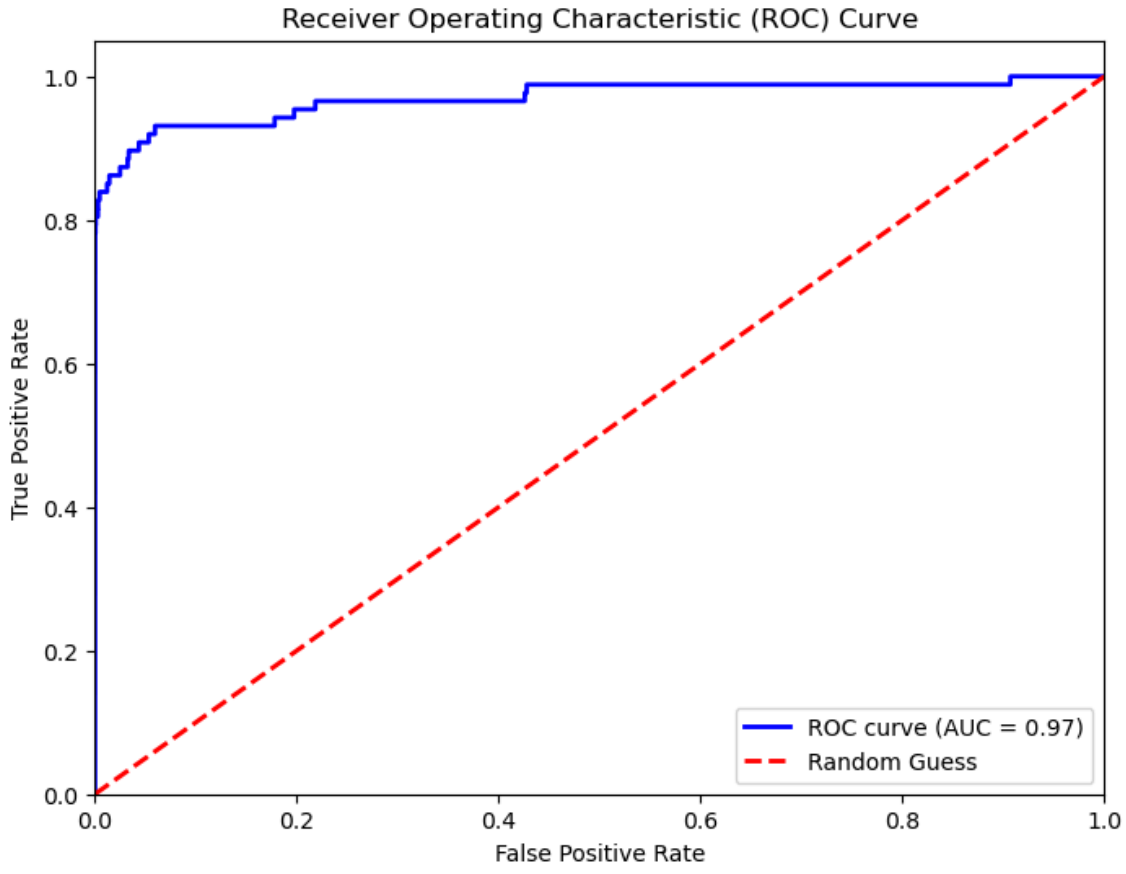
Out[96]:
```
['xgb_smote_FD.joblib']
```

In [5]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\xgb_smote_FD.jobli
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('XGBoost smote Confusion Matrix')
plt.show()
```

```
[[56857    17]
 [   18    70]]
```



In [95]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Get predicted probabilities for the positive class (fraudulent transactio
y_prob = random_search.predict_proba(X_test)[:, 1]

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_prob)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % au
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random G
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




In [ ]:

# Importing the libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

# Importing the dataset

In [2]:
```
df = pd.read_csv("C:\\Users\\Sarrang\\FindDefault\\transformed_data.csv")
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [3]:
```
df.value_counts('Class')
```

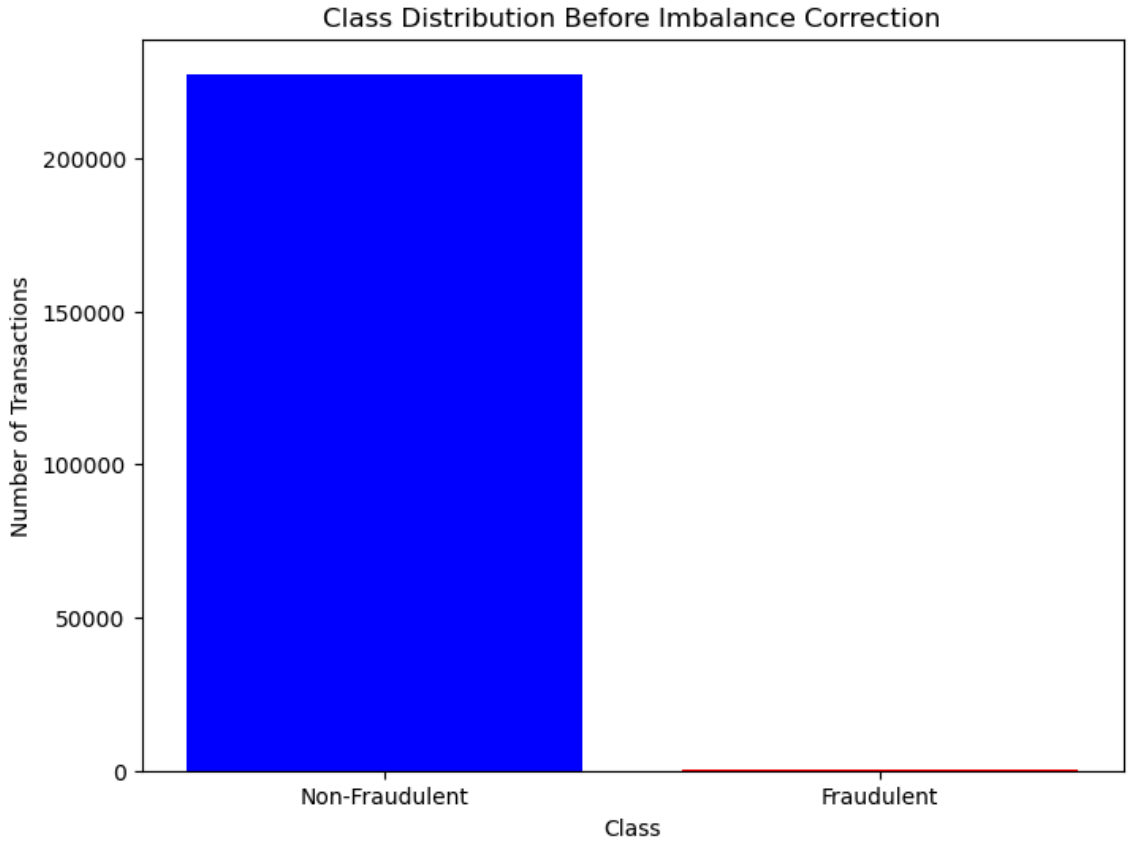
Out[3]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

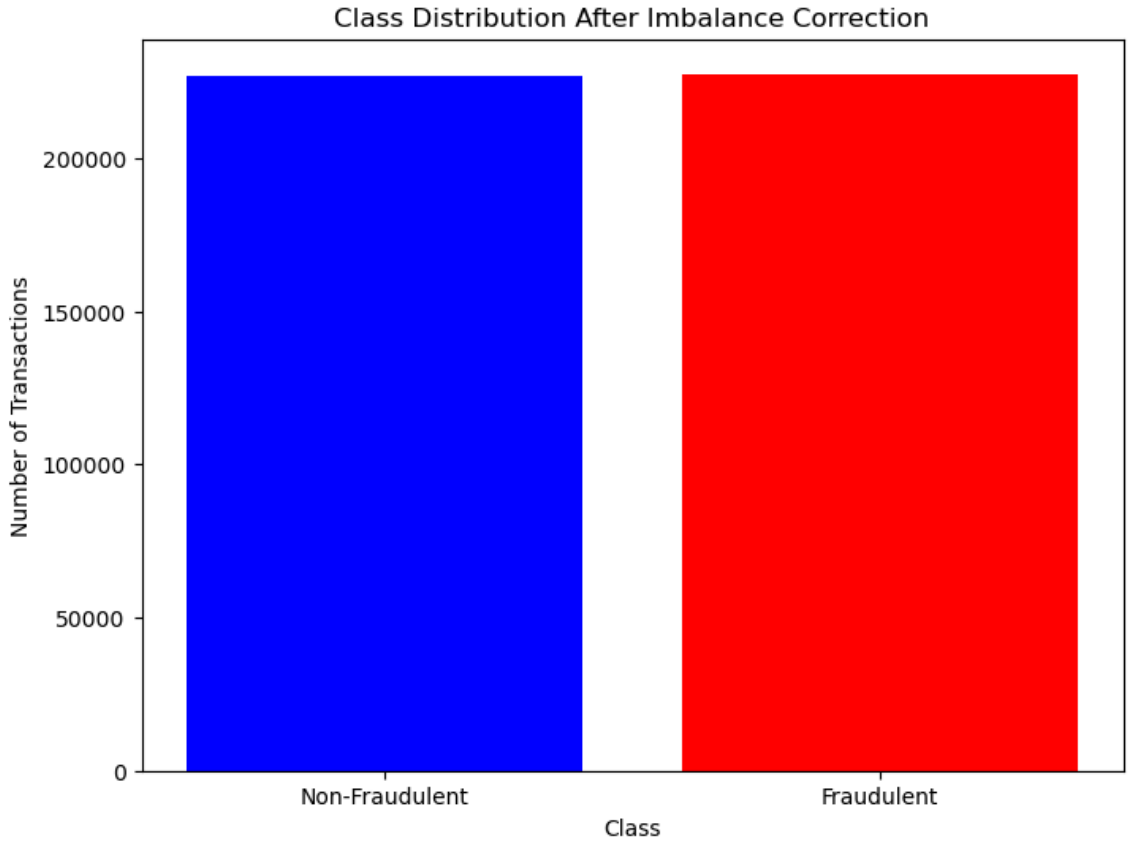
# Splitting the dataset into the Training set and Test set

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [5]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

In [6]:
```
from imblearn.combine import SMOTEENN
smoteenn = SMOTEENN(random_state=42)
X_train_resampled, y_train_resampled = smoteenn.fit_resample(X_train, y_tra

unique_classes, class_counts = np.unique(y_train, return_counts=True)
plt.figure(figsize=(8, 6))
plt.bar(unique_classes, class_counts, color=['blue', 'red'])
plt.xticks(unique_classes, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution Before Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()

# Calculate class distribution after imbalance correction
unique_classes_resampled, class_counts_resampled = np.unique(y_train_resamp

# Plot class distribution after imbalance correction
plt.figure(figsize=(8, 6))
plt.bar(unique_classes_resampled, class_counts_resampled, color=['blue', 'r
plt.xticks(unique_classes_resampled, ['Non-Fraudulent', 'Fraudulent'])
plt.title('Class Distribution After Imbalance Correction')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.show()
```






## XGBoost hyperparameter tuning

In [76]:
```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, precision_score
param_grid = {
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'scale_pos_weight': [1, 10, 25]
}
xgb_classifier = xgb.XGBClassifier(objective='binary:logistic', random_stat
grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid,
                           scoring='precision', cv=3, verbose=2, n_jobs=-1)
grid_search.fit(X_train_resampled, y_train_resampled)
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
y_pred = grid_search.predict(X_test)
precision = precision_score(y_test, y_pred)
print("Precision Score:", precision)
```

```
Fitting 3 folds for each of 2187 candidates, totalling 6561 fits
Best Parameters: {'colsample_bytree': 1.0, 'gamma': 0.1, 'learning_rate':
0.2, 'max_depth': 5, 'min_child_weight': 1, 'scale_pos_weight': 1, 'subsam
ple': 0.8}
Precision Score: 0.45751633986928103
```

In [8]:
```
import xgboost
classifier=xgboost.XGBClassifier()
"""
USING DEFAULT PARAMETERS IS GIVING BETTER RESULTS THAN ALTERING THEM WITH H
"""
```

Out[8]:
```
'\nUSING DEFAULT PARAMETERS IS GIVING BETTER RESULTS THAN ALTERING THEM WI
TH HYPERPARAMETER TUNING\n'
```

In [78]:
```
# classifier=XGBClassifier(colsample_bytree=1.0,
#                          gamma=0.1,
#                          learning_rate=0.2,
#                          max_depth=5,
#                          min_child_weight=1,
#                          scale_pos_weight=1,
#                          subsample=0.8)
```

In [9]:
```
classifier.fit(X_train_resampled,y_train_resampled)
```

Out[9]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=No
ne,
              enable_categorical=False, eval_metric=None, feature_types=No
ne,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=No
ne,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [10]:
```
y_pred=classifier.predict(X_test)
```

In [12]:
```
from sklearn. metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56875
           1       0.64      0.80      0.71        87

    accuracy                           1.00     56962
   macro avg       0.82      0.90      0.86     56962
weighted avg       1.00      1.00      1.00     56962
```

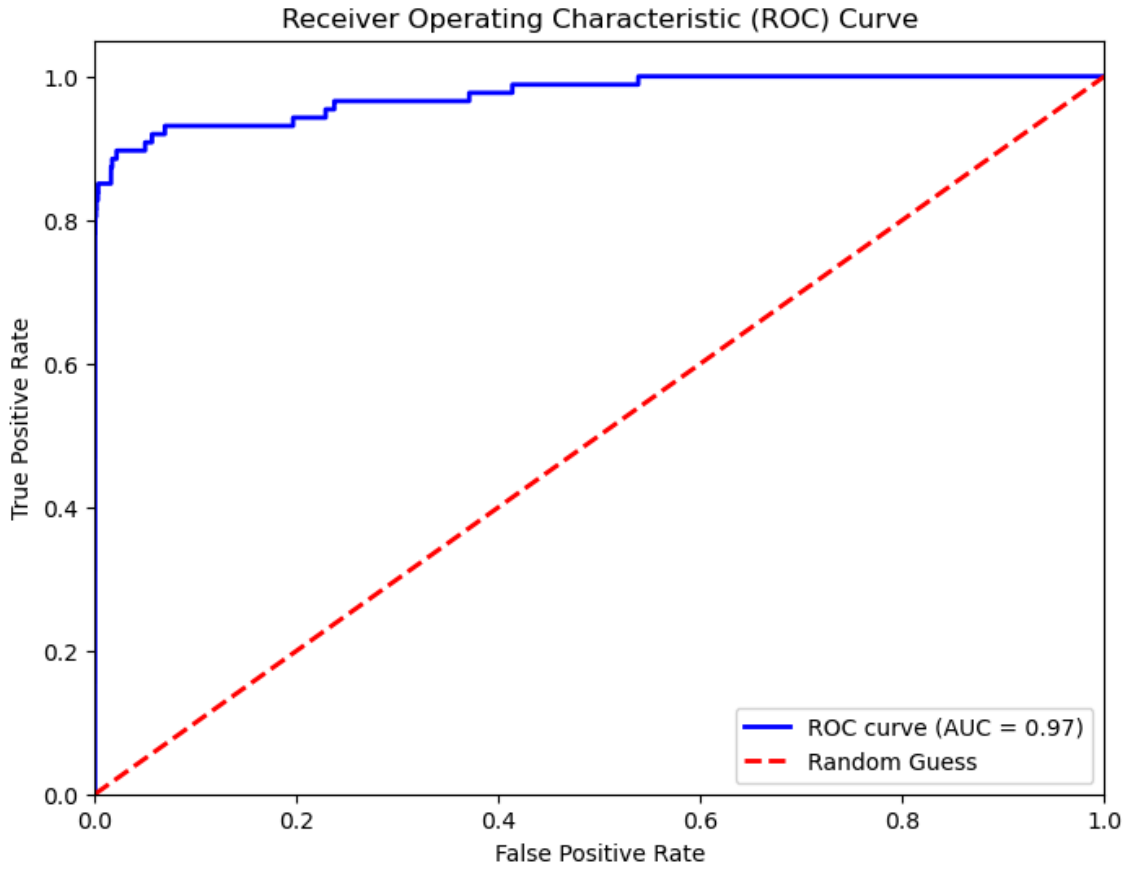
In [13]:
```
import joblib
joblib.dump(classifier,'xgb_smoteENN_FD.joblib')
```

Out[13]:
```
['xgb_smoteENN_FD.joblib']
```

In [5]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\xgb_smoteENN_FD.jo
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('XGBoost smoteENN Confusion Matrix')
plt.show()
```

```
[[56836    17]
 [   39    70]]
```

In [17]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Get predicted probabilities for the positive class (fraudulent transactio
y_prob = classifier.predict_proba(X_test)[:, 1]

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_prob)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % au
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random G
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




In [15]:

In [ ]:

# Importing the libraries

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

# Importing the dataset

In [17]:
```
df= pd.read_csv("C:\\Users\\Sarrang\\FindDefault\\transformed_data.csv")
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

# Splitting the dataset into the Training set and Test set

In [18]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [4]:
```
len(df)
```

Out[4]:
```
284807
```

In [5]:
```
from imblearn.over_sampling import ADASYN

adasyn = ADASYN(random_state=42)
X_train_resampled, y_train_resampled = adasyn.fit_resample(X_train, y_train
```

```
C:\Users\Sarrang9\anaconda3\lib\site-packages\joblib\externals\loky\backen
d\context.py:136: UserWarning: Could not find the number of physical cores
for the following reason:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this warnin
g by setting LOKY_MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "C:\Users\Sarrang9\anaconda3\lib\site-packages\joblib\externals\lok
y\backend\context.py", line 257, in _count_physical_cores
    cpu_info = subprocess.run(
  File "C:\Users\Sarrang9\anaconda3\lib\subprocess.py", line 505, in run
    with Popen(*popenargs, **kwargs) as process:
  File "C:\Users\Sarrang9\anaconda3\lib\subprocess.py", line 951, in __ini
t__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "C:\Users\Sarrang9\anaconda3\lib\subprocess.py", line 1420, in _exe
cute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
```

In [ ]:

```
```

In [6]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

xgb = XGBClassifier()

param_grid = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.05, 0.1, 0.2],
    'n_estimators': [50, 100, 150],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5, scor

grid_search.fit(X_train_resampled, y_train_resampled)

print("Best parameters found: ", grid_search.best_params_)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Best parameters found:  {'colsample_bytree': 0.9, 'learning_rate': 0.2, 'm
ax_depth': 5, 'n_estimators': 150, 'subsample': 0.9}
```

In [15]:

```
# classifier=XGBClassifier(colsample_bytree=0.9,
#                          learning_rate=0.2,
#                          max_depth=5,
#                          n_estimators=150,
#                          subsample=0.9)
classifier=XGBClassifier()
classifier.fit(X_train_resampled,y_train_resampled)
```

Out[15]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=No
ne,
              enable_categorical=False, eval_metric=None, feature_types=No
ne,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=No
ne,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=Non
e,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

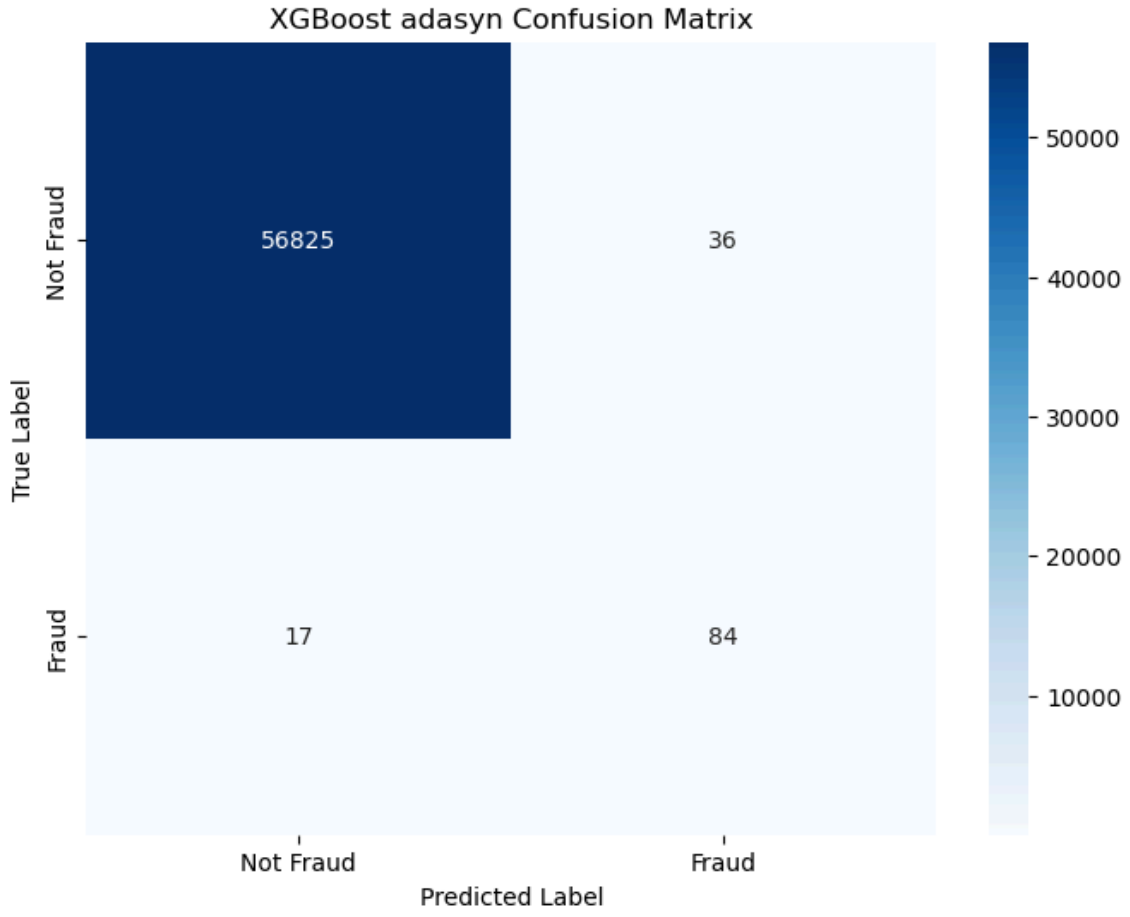
In [16]:
```
y_pred=classifier.predict(X_test)
```

In [17]:
```
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56842
           1       0.83      0.70      0.76       120

    accuracy                           1.00     56962
   macro avg       0.92      0.85      0.88     56962
weighted avg       1.00      1.00      1.00     56962
```

In [16]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\xgb_adasyn_FD.jobl
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('XGBoost adasyn Confusion Matrix')
plt.show()
```

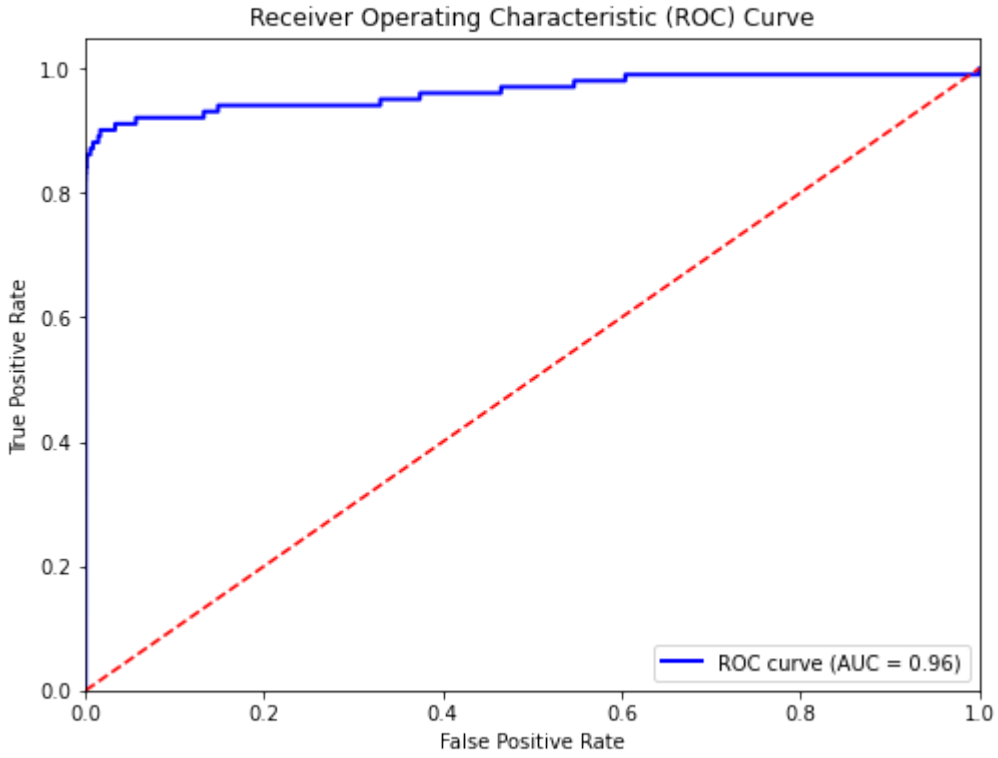
```
[[56825    17]
 [   36    84]]
```




In [19]:
```
from sklearn.metrics import roc_auc_score
y_pred_proba = classifier.predict_proba(X_test)
y_pred_proba_positive = y_pred_proba[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_proba_positive)
print("ROC AUC Score:", roc_auc)
```

```
ROC AUC Score: 0.9629689283977377
```

In [20]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_positive)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % ro
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




In [ ]:

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
df = pd.read_csv("/content/drive/MyDrive/transformed_data.csv")
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
df.value_counts('Class')
```

Out[2]:
```
Class
0    284315
1       492
dtype: int64
```

In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [4]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

In [5]:
```
from joblib import load,dump
from sklearn.ensemble import VotingClassifier
xgb1=load("/content/drive/MyDrive/saved models/xgb_smote_FD.joblib")
xgb2=load("/content/drive/MyDrive/saved models/xgb_adasyn_FD.joblib")
xgb3=load("/content/drive/MyDrive/saved models/xgb_smoteENN_FD.joblib")
lgb1=load("/content/drive/MyDrive/saved models/lightgbm_smote_FD.joblib")
lgb2=load("/content/drive/MyDrive/saved models/lightgbm_adasyn_FD.joblib")
lgb3=load("/content/drive/MyDrive/saved models/lightgbm_smoteENN_FD.joblib"
rf1=load("/content/drive/MyDrive/saved models/rf_smote_FD.joblib")
rf2=load("/content/drive/MyDrive/saved models/rf_adasyn_FD.joblib")
rf3=load("/content/drive/MyDrive/saved models/rf_smoteENN_FD.joblib")

from sklearn.metrics import precision_score
from sklearn.model_selection import cross_val_predict

voting_clf = VotingClassifier(
    estimators=[
        ('xgb1', xgb1),
        ('xgb2', xgb2),
        ('xgb3', xgb3),
        ('lgb1', lgb1),
        ('lgb2', lgb2),
        ('lgb3', lgb3),
        ('rf1', rf1),
        ('rf2', rf2),
        ('rf3', rf3)
    ],
    voting='hard'
)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:318: UserWarning:
Trying to unpickle estimator LabelEncoder from version 1.4.1.post1 when us
ing version 1.2.2. This might lead to breaking code or invalid results. Us
e at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainab
ility-limitations (https://scikit-learn.org/stable/model_persistence.html#
security-maintainability-limitations)
  warnings.warn(
```

In [6]:
```
voting_clf.fit(X_train_resampled, y_train_resampled)
```

```
[LightGBM] [Info] Number of positive: 227440, number of negative: 227440
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.264586 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 454880, number o
f used features: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.00
0000
[LightGBM] [Info] Number of positive: 227440, number of negative: 227440
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.270475 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 454880, number o
f used features: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.00
0000
[LightGBM] [Info] Number of positive: 227440, number of negative: 227440
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.160063 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 454880, number o
f used features: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.00
0000

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424: F
utureWarning: `max_features='auto'` has been deprecated in 1.1 and will be
removed in 1.3. To keep the past behaviour, explicitly set `max_features
='sqrt'` or remove this parameter as it is also the default value for Rand
omForestClassifiers and ExtraTreesClassifiers.
  warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424: F
utureWarning: `max_features='auto'` has been deprecated in 1.1 and will be
removed in 1.3. To keep the past behaviour, explicitly set `max_features
='sqrt'` or remove this parameter as it is also the default value for Rand
omForestClassifiers and ExtraTreesClassifiers.
  warn(
```

Out[6]:
```
VotingClassifier(estimators=[('xgb1',
                              XGBClassifier(base_score=None, booster=None,
                                            callbacks=None,
                                            colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=None, device=
None,
                                            early_stopping_rounds=None,
                                            enable_categorical=False,
                                            eval_metric=None,
                                            feature_types=None, gamma=Non
e,
                                            grow_policy=None,
                                            importance_type=None,
                                            interaction_constraints=None,
                                            learning_r...
                              LGBMClassifier(learning_rate=0.2,
                                             n_estimators=150, num_leaves=
40,
                                             subsample=0.8)),
                             ('rf1',
                              RandomForestClassifier(max_features='auto',
                                                     min_samples_leaf=2,
                                                     min_samples_split=5,
                                                     n_estimators=200)),
                             ('rf2',
                              RandomForestClassifier(bootstrap=False,
                                                     max_features='log2',
                                                     min_samples_split=10,
                                                     n_estimators=50)),
                             ('rf3',
                              RandomForestClassifier(bootstrap=False,
                                                     max_features='auto',
                                                     min_samples_leaf=
4))])
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [7]:
```
y_pred=voting_clf.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

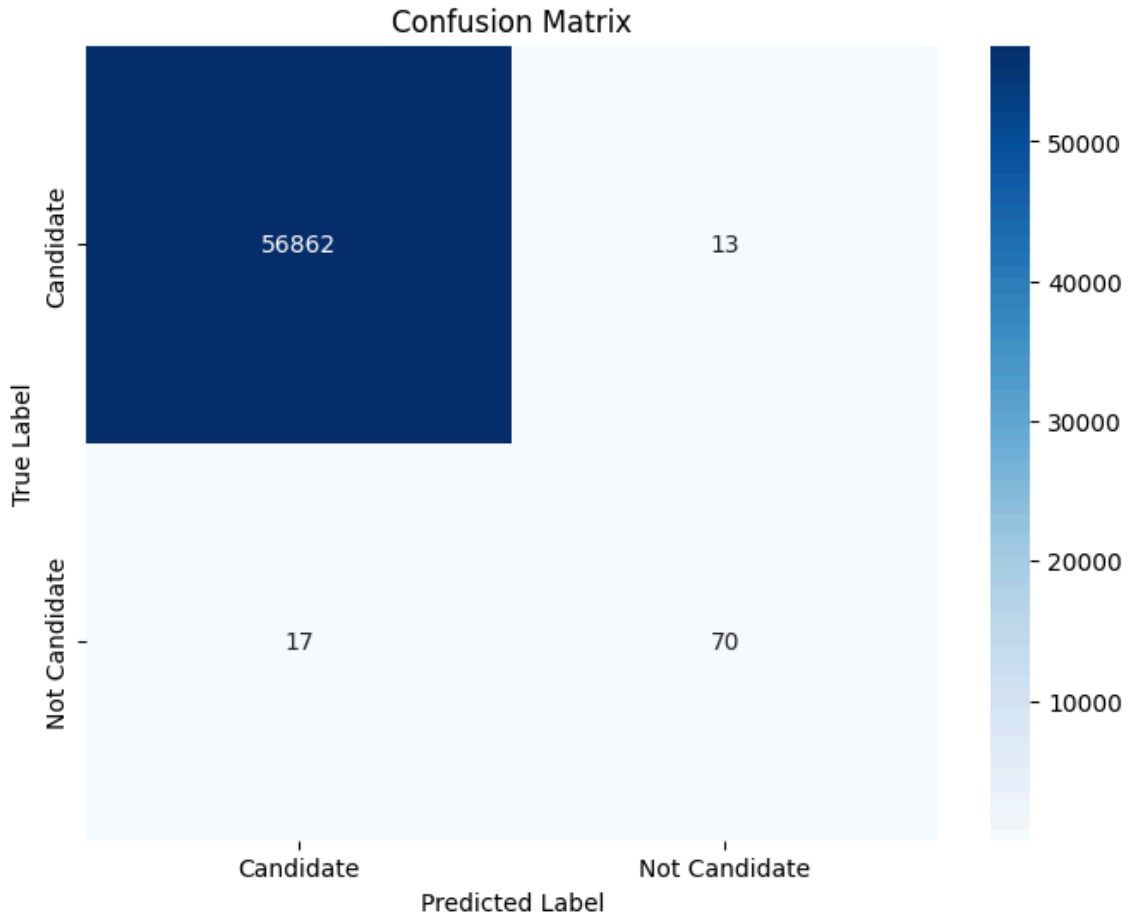
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56879
           1       0.80      0.84      0.82        83

    accuracy                           1.00     56962
   macro avg       0.90      0.92      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

In [9]:
```
# dump(voting_clf, 'voting_.FDjoblib')
```

Out[9]:
```
['voting_.FDjoblib']
```

In [13]:
```
from joblib import load
clf=load("C:\\Users\\Sarrang\\FindDefault\\saved models\\voting_cf.joblib")
y_pred=clf.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_pred,y_test))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('XGBoost adasyn Confusion Matrix')
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56879
           1       0.80      0.84      0.82        83

    accuracy                           1.00     56962
   macro avg       0.90      0.92      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```




In [ ]:
```
```

```python
import streamlit as st
import pandas as pd
import joblib
import base64
import io
import gzip

# Load the model
with gzip.open("C:\\Users\\Sarrang\\FindDefault\\saved models\\voting_clf_compressed.joblib.gz", 'rb') as f:
    # Load the model from the compressed file
    model = joblib.load(f)
st.title('Fraudulent Transaction Detection')

uploaded_file = st.file_uploader("Upload CSV file", type=["csv"])

if uploaded_file is not None:
    # Read the uploaded file
    df = pd.read_csv(uploaded_file)

    # Detect fraudulent transactions using the model
    predictions = model.predict(df)

    # Add a column to the DataFrame indicating fraudulence
    df['Fraudulent'] = predictions

    # Map numerical predictions to strings for display
    df['Fraudulent'] = df['Fraudulent'].map({0: 'No', 1: 'Yes'})

    # Apply conditional formatting to highlight fraudulent transactions
    def highlight_fraud(row):
        if row['Fraudulent'] == 'Yes':
            return ['background-color: red'] * len(row)
        else:
            return [''] * len(row)

    styled_df = df.style.apply(highlight_fraud, axis=1)

    # Display the styled DataFrame
    st.write(styled_df)
```