MercyKiragu /
SAFESWIPE

Code Issues Pull requests Actions Projects Wiki Security Insights

SAFESWIPE / notebook.ipynb

MercyKiragu updated notebook 4ab6e8c · 10 minutes ago

3.14 MB

FINAL PROJECT SUBMISSION

Please fill out:

- Student name: MERCY KIRAGU
- Student pace: FULLTIME-HYBRID
- Scheduled project review date/time:07/06/2024
- Instructor name: MARYANN MWIKALI
- Blog post URL:N/A

CREDIT CARD FRAUD DETECTION SYSTEM FOR ZEST BANK

1. BUSINESS UNDERSTANDING

a.)OVERVIEW

Zest Bank, like many financial institutions, faces the constant threat of fraudulent activities. These credit card fraudulent activities can result in significant financial losses and damage to the bank's reputation. To mitigate these risks, Zest Bank is seeking to implement a robust fraud detection system that can accurately identify and prevent fraudulent transactions in real-time.

b.) PROBLEM STATEMENT

For many banks,retainng high profitable customers is the number one goal.Banking fraud,however poses a significant threat to this goal for differrent banks including Zest Bank .In terms of substantial financial losses ,trust and credibility ,this is a concerning issue to both Zest Bank and it's customers.

Zest Bank faces a significant challenge in detecting and preventing fraudulent transactions within its extensive financial network. Conventional rule-based systems have proven insufficient against increasingly sophisticated fraud schemes, resulting in substantial financial losses and reputational harm. Therefore, there is an urgent need to develop and deploy an advanced fraud detection system capable of accurately identifying fraudulent activities in real-time, while minimizing false positives.

In this project I will create a system called SafeSwipe for detecting fraudulent credit card transactions with the help of classification machine learning models.

c.) CHALLENGES

1. **Data Volume**: Handling the vast amount of data processed daily requires a model that responds quickly to detect fraud.

2. **Imbalanced Data**: With the majority of transactions being non-fraudulent, detecting the fraudulent ones becomes challenging.

3. **Data Privacy**: Accessing data, especially private information, poses a significant challenge.

4. **Misclassification**: Some fraudulent transactions go unnoticed, leading to misclassification

misclassification.

5. **Adaptive Techniques by Scammers**: Scammers continually evolve their methods, posing a challenge to the effectiveness of fraud detection models.
6. **Accuracy Metrics** :Accuracy alone may not be the most appropriate metric to evaluate the performance of the fraud detection model.

d.) PROPOSED SOLUTIONS

Solutions:

1. **Simple and Fast Models**: Implementing models that are both simple and quick to identify anomalies and classify them as fraudulent transactions promptly.
2. **Imbalance Handling Methods**: Utilizing effective methods to address imbalance in the data,such as oversampling and undersampling methods ensuring accurate detection of fraudulent transactions.
3. **Privacy-preserving Techniques**: Reducing data dimensionality to protect user privacy while maintaining the model's effectiveness.
4. **Trustworthy Data Sources**: Using reliable data sources that verify and double-check the information, particularly during model training, to enhance accuracy.
5. **Interpretable Models**: Developing models that are simple and interpretable allows for swift adaptation to new scamming techniques with minimal adjustments, ensuring continuous deployment of effective fraud detection systems.
6. **use other metrics** :to use metrics such as roc_auc score and F1-score, along with accuracy, to evaluate the performance of a fraud detection model

e.) OBJECTIVES

1. **Developing an Efficient Model**: Create various classification machine learning models and settling on the best performing model capable of accurately identifying fraudulent credit card transactions while minimizing false positives.
2. **Handling Imbalanced Data**: Implement techniques to address the class imbalance in the dataset, ensuring that the model can effectively learn from the limited number of fraudulent transactions.
3. **Real-time Detection**: Build a system that can process transactions in real-time, providing immediate alerts for potentially fraudulent activity to prevent financial losses.
4. **Privacy Preservation**: Incorporate privacy-preserving techniques to protect sensitive customer information while still allowing for effective fraud detection.
5. **Performance Evaluation**: Conduct thorough evaluation and validation of the model's performance using appropriate metrics, such as precision, recall, F1-score, to assess its effectiveness in real-world scenarios. Also perform Cross Validation.
6. **Scalability**: Design the system to handle large volumes of transactions

6. **Scalability**: Design the system to handle large volumes of transactions efficiently, ensuring scalability to accommodate growth in transaction volume over time.
7. **Normalization**: Apply normalization techniques to preprocess the data and standardize features, ensuring that the model's performance is not affected by variations in the scale of input features.
8. **Hyperparameter Tuning**: Explore and optimize the hyperparameters of the machine learning model to enhance its performance further. Utilize techniques such as grid search, random search to systematically search the hyperparameter space and identify the configuration that maximizes the model's effectiveness in fraud detection while minimizing computational resources and training time. Conduct cross-validation to validate the chosen hyperparameters and ensure robustness of the model across different subsets of the data.

2.) DATA UNDERSTANDING

a.)DATA SOURCE

The dataset for this project was sourced from Kaggle, a platform for data science datasets.Specifically, the dataset I used is the credit card fraud detection dataset.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.The dataset consists of 31 columns/features and 284,807 rows

It contains only numerical input variables which are the result of a PCA transformation.Due to confidentiality issues original features and more background information about the data wasn't provided. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which were not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

IMPORTING THE NECESSARY LIBRARIES

In [89]:

```
# importing the necessary dependencies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import f1_score, classification_report
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.utils import resample
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
from imblearn.over_sampling import SMOTE
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score











import warnings
warnings.filterwarnings("ignore")
```

LOADING THE DATA

In [90]:

```
#loading the data
df = pd.read_csv('creditcard.csv')
```

In [91]:

```
#code for displaying all the columns
pd.options.display.max_columns = None
```

In [92]:

```
#CHECKING THE FIRST FIVE ROWS
df.head()
```

Out[92]:

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

In [93]:

```
#CHECKING THE LAST FIVE ROWS
df.tail()
```

Out[93]:

	Time	V1	V2	V3	V4	V5	V6
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.60683
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.05841
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.03126
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.62370
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.64961

3.)EXPLORATORY DATA ANALYSIS

In [94]:

```
#CHECKING THE ROWS AND COLUMNS
df.shape
```

Out[94]: (284807, 31)

In [95]:

```
#CHECKING ALL THE COLUMN NAMES
df.columns
```

Out[95]:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V1
0',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V2
0',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [96]:

```
#CHECKING THE STATSTICAL SUMMARY OF THE FEATURES
df.describe()
```

Out[96]:

	Time	V1	V2	V3	V4
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01

In [97]:

```
df.describe().transpose()
```

Out[97]:

	count	mean	std	min	25%	
Time	284807.0	9.481386e+04	47488.145955	0.000000	54201.500000	84692.(
V1	284807.0	1.168375e-15	1.958696	-56.407510	-0.920373	0.(
V2	284807.0	3.416908e-16	1.651309	-72.715728	-0.598550	0.(
V3	284807.0	-1.379537e-15	1.516255	-48.325589	-0.890365	0.'
V4	284807.0	2.074095e-15	1.415869	-5.683171	-0.848640	-0.(
V5	284807.0	9.604066e-16	1.380247	-113.743307	-0.691597	-0.(
V6	284807.0	1.487313e-15	1.332271	-26.160506	-0.768296	-0.2
V7	284807.0	-5.556467e-16	1.237094	-43.557242	-0.554076	0.(
V8	284807.0	1.213481e-16	1.194353	-73.216718	-0.208630	0.(
V9	284807.0	-2.406331e-15	1.098632	-13.434066	-0.643098	-0.(
V10	284807.0	2.239053e-15	1.088850	-24.588262	-0.535426	-0.(
V11	284807.0	1.673327e-15	1.020713	-4.797473	-0.762494	-0.(
		-1.247012e-				

V12	284807.0	[illegible] 15	0.999201	-18.683715	-0.405571	0.'
V13	284807.0	8.190001e-16	0.995274	-5.791881	-0.648539	-0.(
V14	284807.0	1.207294e-15	0.958596	-19.214325	-0.425574	0.(
V15	284807.0	4.887456e-15	0.915316	-4.498945	-0.582884	0.(
V16	284807.0	1.437716e-15	0.876253	-14.129855	-0.468037	0.(
V17	284807.0	-3.772171e-16	0.849337	-25.162799	-0.483748	-0.(
V18	284807.0	9.564149e-16	0.838176	-9.498746	-0.498850	-0.(
V19	284807.0	1.039917e-15	0.814041	-7.213527	-0.456299	0.(
V20	284807.0	6.406204e-16	0.770925	-54.497720	-0.211721	-0.(
V21	284807.0	1.654067e-16	0.734524	-34.830382	-0.228395	-0.(
V22	284807.0	-3.568593e-16	0.725702	-10.933144	-0.542350	0.(
V23	284807.0	2.578648e-16	0.624460	-44.807735	-0.161846	-0.(
V24	284807.0	4.473266e-15	0.605647	-2.836627	-0.354586	0.(
V25	284807.0	5.340915e-16	0.521278	-10.295397	-0.317145	0.(
V26	284807.0	1.683437e-15	0.482227	-2.604551	-0.326984	-0.(
V27	284807.0	-3.660091e-16	0.403632	-22.565679	-0.070840	0.(
V28	284807.0	-1.227390e-16	0.330083	-15.430084	-0.052960	0.(
Amount	284807.0	8.834962e+01	250.120109	0.000000	5.600000	22.(
Class	284807.0	1.727486e-03	0.041527	0.000000	0.000000	0.(

In [98]:

```
#CHECKING THE DATATYPES AND NON/NULL DISTRIBUTION
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
```

Handling missing values

In [99]:

```
#CHECKING IF THERE ARE ANY MISSING VALUES
df.isnull().sum()
```

Out[99]:

```
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
```

observation

There are no features with missing values.

Dealing with duplicates.

In [100...

```
df.duplicated().any()
```

Out[100...

```
True
```

In [101...

```
df.duplicated().sum()
```

Out[101...

```
1081
```

Observation

There are 1081 rows with duplicate values which we are going to drop.

In [102...

```
#DROP THE DUPLICATED ROWS
```

```
#DROP THE DUPLICATED ROWS
# Identify duplicated rows
duplicated_rows = df.duplicated()

# Invert the boolean mask to keep non-duplicated rows
df = df[~duplicated_rows]
```

In [103...

```
df.duplicated().any()
```

Out[103...

```
False
```

Our dataset now has no duplicates.

Dealing with Outliers.

As the whole dataset is transformed with PCA, I assume that the outliers are already treated. Hence not performing any outliers treatment on the dataset.

Observing the distribution of our classes

In [104...

```
#CHECKING THE CLASS DISTRIBUTION OF THE TARGET VARIABLE
df['Class'].value_counts()
```

Out[104...

```
Class
0    283253
1       473
Name: count, dtype: int64
```

In [105...

```
# CREATING NEW VARIABLES 'no_fraud' and 'fraud'
no_fraud = df['Class'] == 0
fraud = df['Class'] == 1
```

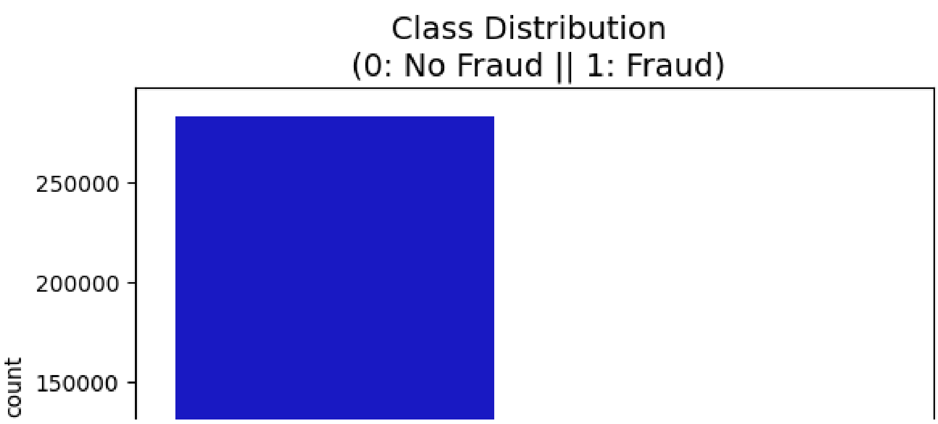
In [106...

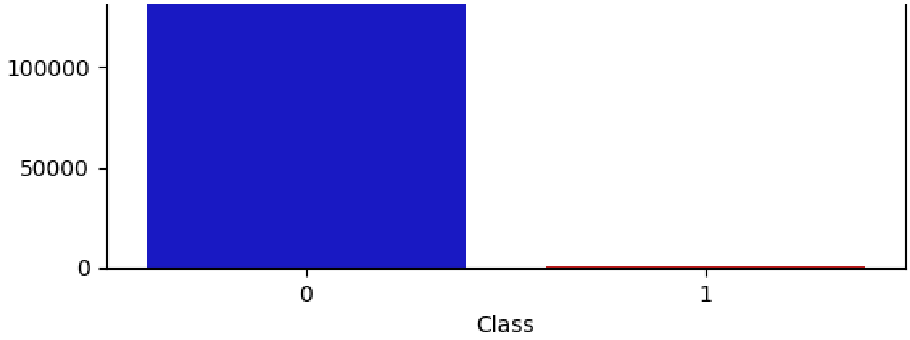
```
#CHECKING THE CLASS DISTRIBUTION OF THE TARGET VARIABLE IN PERCENTAGE
print('No Frauds', round(df['Class'].value_counts()[0] / len(df) *100,2),
print('Frauds', round(df['Class'].value_counts()[1] / len(df) *100,2), '%c
```

```
No Frauds 99.83 %of the dataset
Frauds 0.17 %of the dataset
```

In [167...

```
# Defining colors for the countplot
colors = ["#0101DF", "#DF0101"]

# Creating a countplot to visualize the class distribution of fraudulent v
sns.countplot(x='Class', data=df, palette=colors)

# Adding title and labels to the plot
plt.title('Class Distribution \n (0: No Fraud || 1: Fraud)', fontsize=14)
plt.show()
```





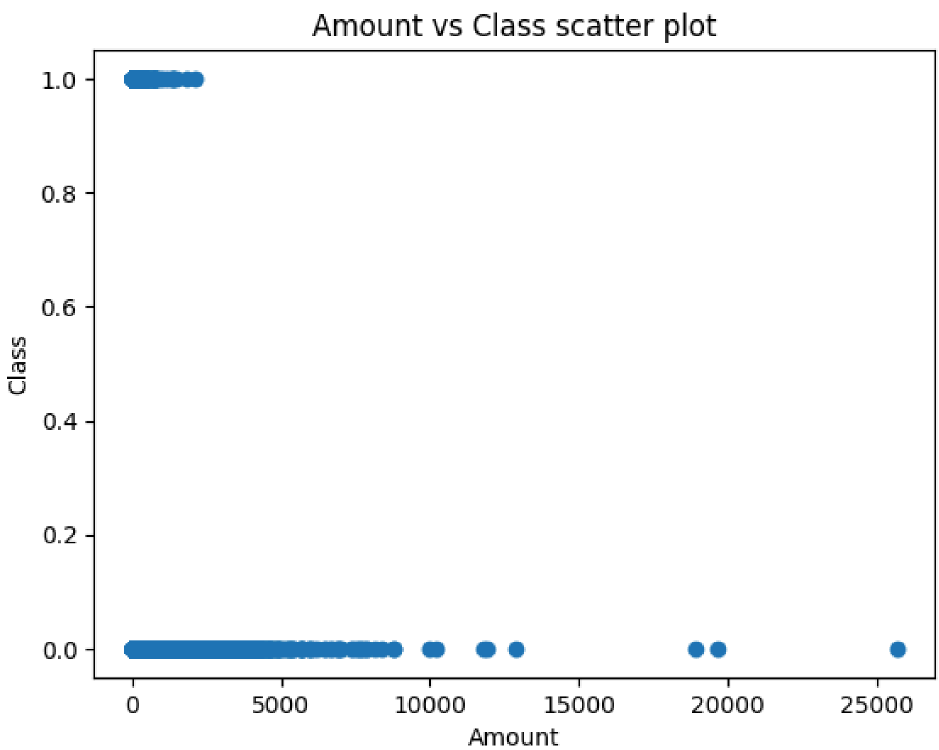
Observation.

The dataset exhibits a significant class imbalance, with non-fraudulent transactions representing approximately 99.83% of the data and fraudulent transactions accounting for the remaining 0.17%. This imbalance can pose challenges in training accurate machine learning models, as algorithms may tend to favor the majority class and perform poorly on the minority class.

observing the distribution of classes with Amount.

In [168…

```
# Creating a scatter plot to observe the distribution of classes with Amou
plt.scatter(df["Amount"], df["Class"])
plt.title("Amount vs Class scatter plot")
plt.xlabel("Amount")
plt.ylabel("Class")
plt.show()
```



Observation

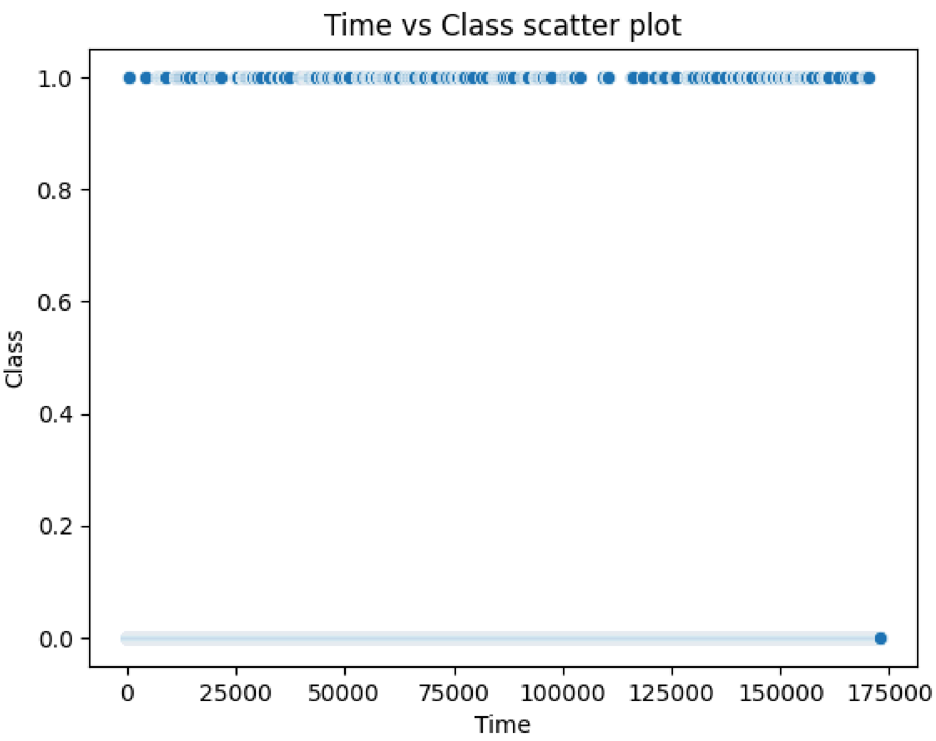
Clearly low amount transactions are more likely to be fraudulent than high amount transactions from the above visualization.

Observing the distribution of classes with time.

In [109…

```
# Create a scatter plot to observe the distribution of classes with time
```

```
sns.scatterplot(x= df["Time"],y=df["Class"])
plt.title("Time vs Class scatter plot")
plt.show()
```



Observation

There is not much insight can be drwan from the distribution of the fraudulent transaction based on time as fraudulent/non-fraudulent both transaction are distributed over time.

Dropping Time column as this feature is not going to help in the model building.

In [110…

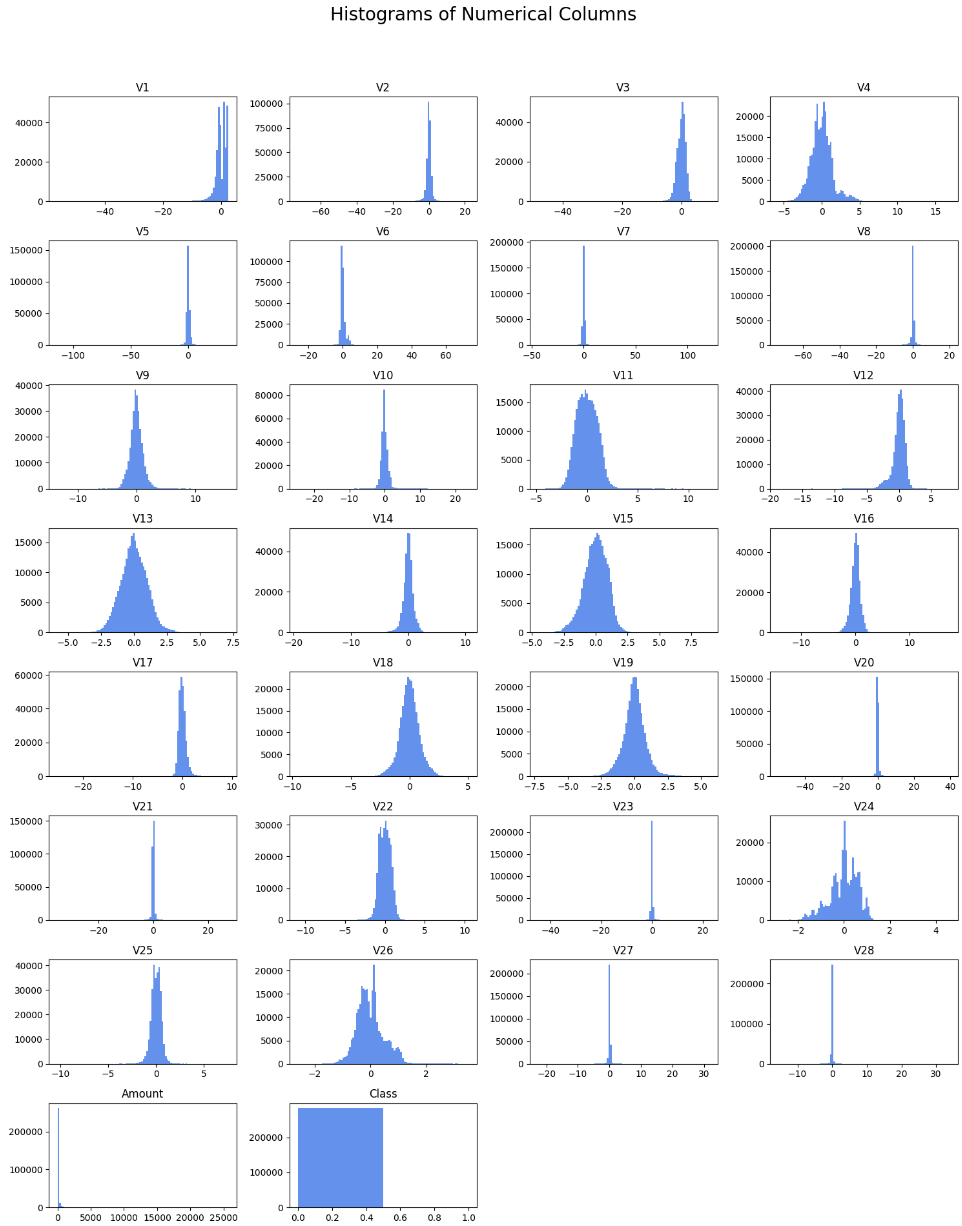
```
# Drop unnecessary columns
df = df.drop("Time", axis = 1)
```

Checking the distribution of the numerical features

In [170…

```
## Histograms to show the distribution of the numerical features.

# Define custom colors for histograms
hist_color = '#6495ED'

# Create a figure to hold the histograms
fig = plt.figure(figsize=(15, 20))
plt.suptitle('Histograms of Numerical Columns', fontsize=20)

# Iterate over each numerical column
for i in range(df.shape[1]):
    plt.subplot(8, 4, i + 1)
    ax = plt.gca()
    ax.set_title(df.columns.values[i])

    # Limiting the number of bins to 100 maximum
    num_unique_vals = np.size(df.iloc[:, i].unique())
    if num_unique_vals >= 100:
        num_unique_vals = 100

    # Plotting the histogram with custom color
    plt.hist(df.iloc[:, i], bins=num_unique_vals, color=hist_color)

# Adjusting layout and displaying the histograms
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
plt.show()
```



observation

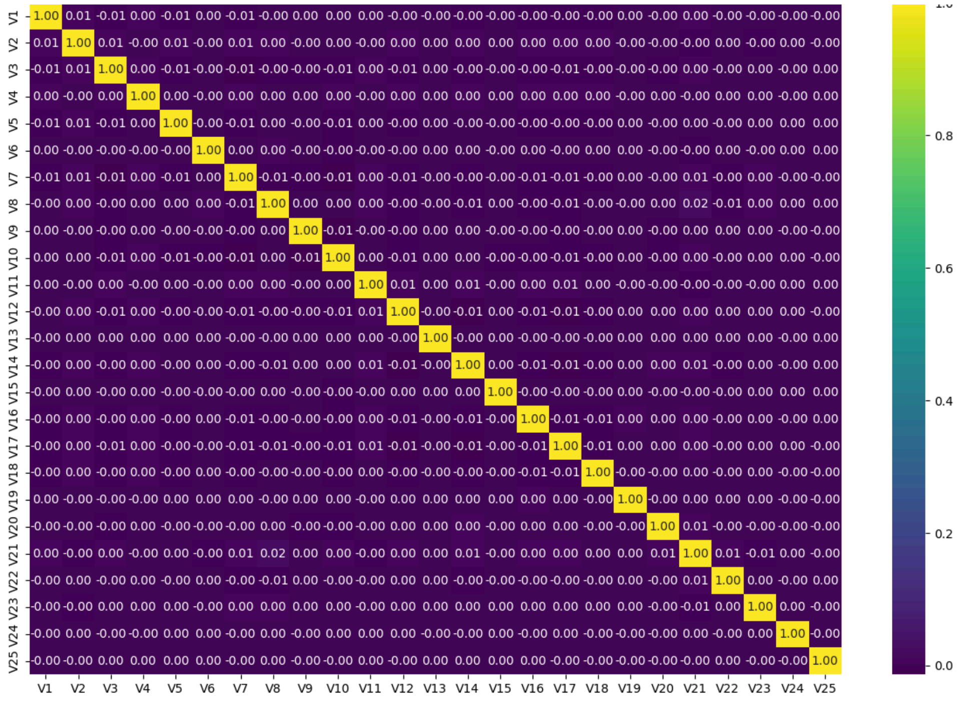
The features seem to have various distributions, including normal, uniform, and skewed distributions. Most features appear to be centered around 0, with some exceptions (e.g., V17, V21).

Correlation check

In [171…

```
# Define the columns for which to calculate the correlation matrix
columns = ["V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10",
           "V11", "V12", "V13", "V14", "V15", "V16", "V17", "V18", "V19",
           "V20", "V21", "V22", "V23", "V24", "V25"]

# Calculate the correlation matrix for the specified columns
correlation_matrix = df[columns].corr()

# Plot the heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='viridis', fmt='.2f')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

Correlation Matrix Heatmap



Obseravation

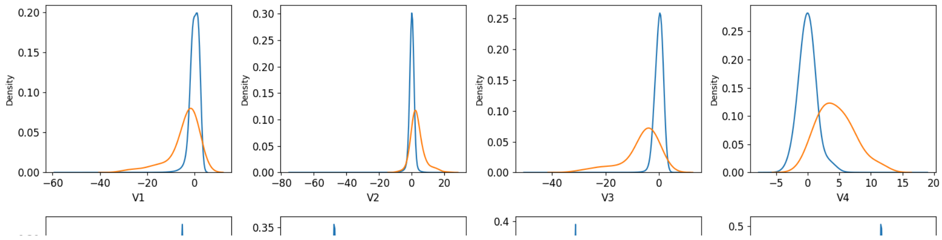
Clearly there is little to no correlation between the features.

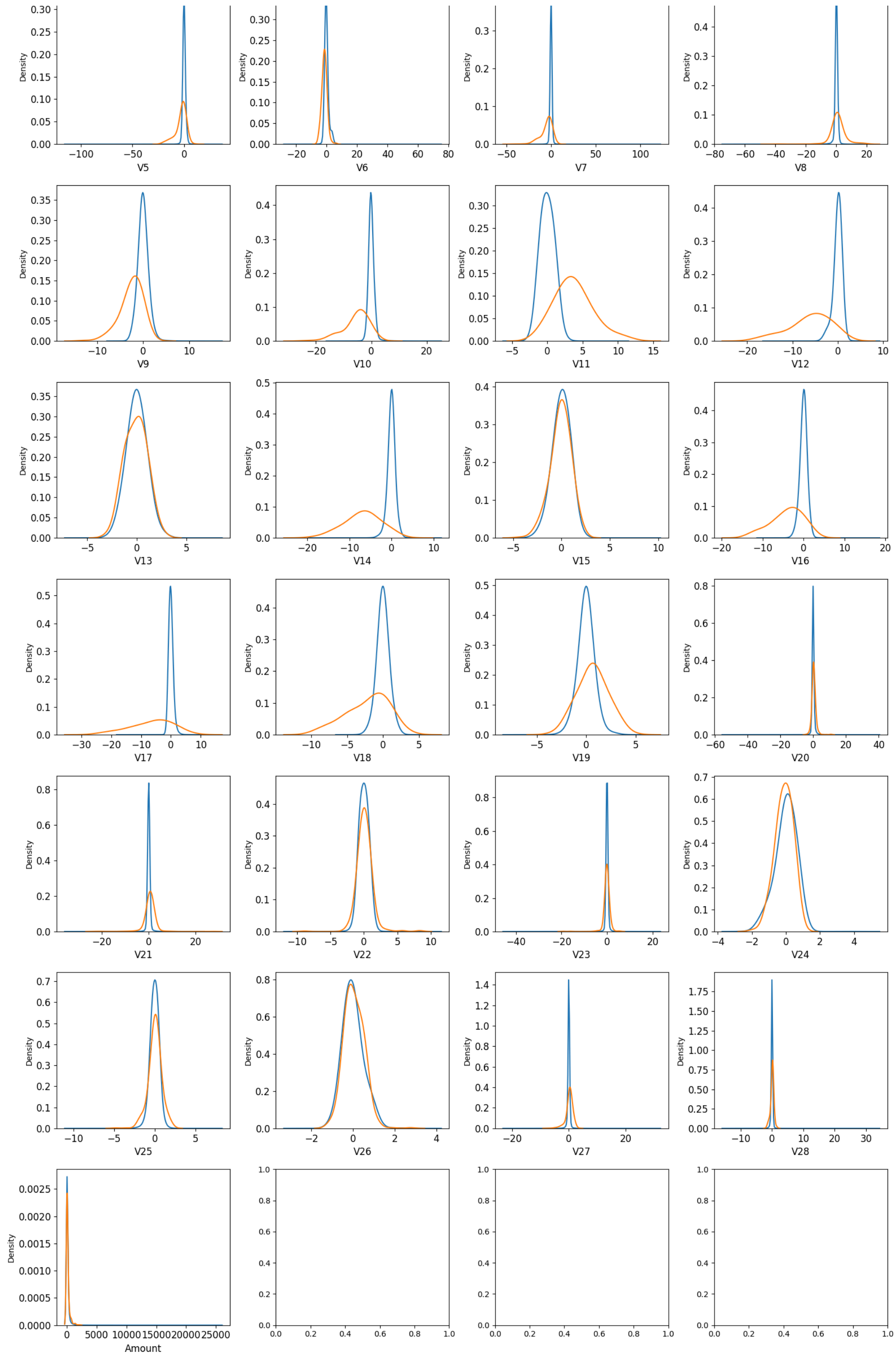
Kernel Density Estimates

In [173…

```
# Suppress UserWarnings
warnings.simplefilter(action='ignore', category=UserWarning)

# Extracting all variable names except 'Class'
variables = list(df.columns.values)
variables.remove("Class")

# Splitting the dataset based on Class
no_fraud = df.loc[df['Class'] == 0]
fraud = df.loc[df['Class'] == 1]

# Creating subplots for visualizing the distribution of each variable
plt.figure()
fig, ax = plt.subplots(8, 4, figsize=(16, 28))

# Iterating through each variable
for i, feature in enumerate(variables):
    plt.subplot(8, 4, i + 1)

    # Plotting kernel density estimates for each class
    sns.kdeplot(no_fraud[feature], bw=0.5, label="0")
    sns.kdeplot(fraud[feature], bw=0.5, label="1")

    # Adding labels and adjusting tick parameters
    plt.xlabel(feature, fontsize=12)
    plt.tick_params(axis='both', which='major', labelsize=12)

# Adjusting layout and showing the plot
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```





We can see most of the features distributions are overlapping for both the fraud and non-fraud transactions.

STORE INDEPENDENT FEATURES IN X AND TARGET (RESPONSE) IN Y

In [114…

```
#SPLITTING THE DATASET INTO X AND Y
X = df.drop('Class',axis=1)
y = df['Class']
```

In [115…

```
#CHECKING SOME ROWS OF X
X.head()
```

Out[115…

	V1	V2	V3	V4	V5	V6	V7	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08

2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27

In [116…

```
#CHECKING SOME ROWS OF Y
y.head()
```

Out[116…

```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

Handling the Imbalanced Dataset.

1) Choose Proper Evaluation Metrics

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

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

2) Resampling(Undersampling and Oversampling)

OverSampling

In [117…

```
df.Class.value_counts()
```

Out[117…

```
Class
0    283253
1       473
Name: count, dtype: int64
```

In [118…

```
#create two different dataframe of majority and minority class
df_majority = df[(df['Class']==0)]
df_minority = df[(df['Class']==1)]

# upsample minority class
df_minority_oversampled = resample(df_minority,
                                 replace=True,
                                 n_samples=283253,
                                 random_state=42)
# Combine majority class with upsampled minority class
df_oversampled = pd.concat([df_minority_oversampled, df_majority])
df_oversampled.Class.value_counts()
```

Out[118…

```
Class
1    283253
0    283253
Name: count, dtype: int64
```

In [119…

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

Out[119…
```
((566506, 29), (566506,))
```

Undersampling

In [120…
```
rus = RandomUnderSampler(random_state=42)

X_undersampled, y_undersampled = rus.fit_resample(X, y)
print(f"The number of Classes before the fit {Counter(y)}")
print(f"The number of Classes after the fit {Counter(y_undersampled)}")
```

```
The number of Classes before the fit Counter({0: 283253, 1: 473})
The number of Classes after the fit Counter({0: 473, 1: 473})
```

3) SMOTE(Synthetic Minority Oversampling Technique)

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

In [121…
```
# Resampling the minority class. The strategy can be changed as required.

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

# Fit the model to generate the data.

X_smote, y_smote = sm.fit_resample(df.drop('Class', axis=1), df['Class'])
smote_df = pd.concat([pd.DataFrame(X_smote), pd.DataFrame(y_smote)], axis=
X_smote.shape
```

Out[121…
```
(566506, 29)
```

In [122…
```
smote_df.Class.value_counts()
```

Out[122…
```
Class
0    283253
1    283253
Name: count, dtype: int64
```

SPLITTING THE DATA INTO TRAIN AND TEST DATA

In [123…
```
#SPLITTING THE DATASET USING TRAIN TEST SPLIT
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,
                                                 random_state=42)
```

by splitting the dataset I'm preserving X_test and Y_test to evaluate once I'm done modelling.

In [124…
```
# Checking the split of the class lable
print(np.sum(y))
print(np.sum(y_train))
print(np.sum(y_test))
```

```
473
383
90
```

NORMALIZATION

We need to scale Amount column.

In [125…

```
# As PCA is already performed on the dataset from V1 to V28 features, we a
#Instantiating the scaler
scaler = StandardScaler()

# Scaling the train data
X_train[["Amount"]] = scaler.fit_transform(X_train[["Amount"]])

# Transforming the test data
X_test[["Amount"]] = scaler.transform(X_test[["Amount"]])
```

In [126…

```
#checking if the amount column has been scaled
X_train.head()
```

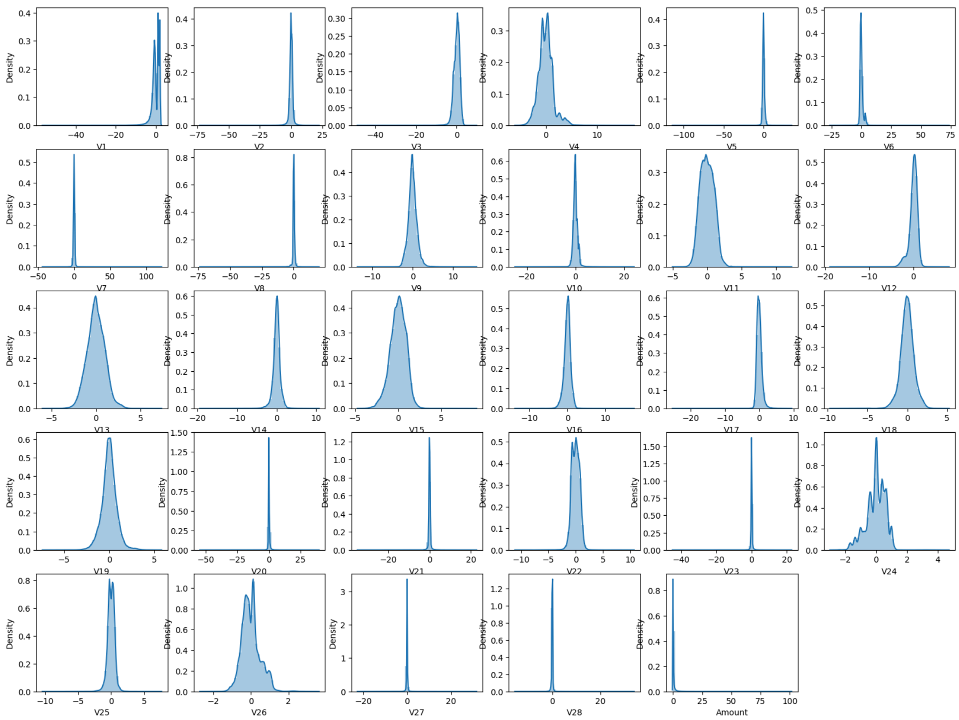
Out[126…

	V1	V2	V3	V4	V5	V6	V7
79119	-1.353894	-1.081034	1.787587	-1.803254	-0.235223	2.177295	1.014795
265736	-0.059936	0.582525	-1.369698	-1.100370	3.707608	3.396709	1.218666
42801	-0.362164	0.715165	1.783253	0.230848	-0.240500	-0.508633	0.589763
172689	-0.370982	0.446004	-0.732017	-1.088288	1.799175	-1.117223	1.502962
179949	0.695644	0.177228	0.481414	-0.338853	-0.122807	0.074431	-0.509006

Handling Skewness

In [127…

```
# plot the histogram of a variable from the dataset to see the skewness
var = X_train.columns

plt.figure(figsize=(20,15))
i=0
for col in var:
    i += 1
    plt.subplot(5,6, i)
    sns.distplot(X_train[col])

plt.show()
```



Observation

Lot of features are highly skewed. So I will check the skewness using skew() and if the skewness is beyond -1 to 1, then we will use power transform to transform the data.

In [128…

```
# Lets check the skewness of the features
var = X_train.columns
skew_list = []
for i in var:
    skew_list.append(X_train[i].skew())

tmp = pd.concat([pd.DataFrame(var, columns=["Features"]), pd.DataFrame(ske
tmp.set_index("Features", inplace=True)
tmp.T
```

Out[128…

Features	V1	V2	V3	V4	V5	V6	V7
Skewness	-3.303525	-4.772388	-2.256569	0.68212	-2.933103	1.961982	3.553954

In [129…

```
# Filtering the features which has skewness less than -1 and greater than
skewed = tmp.loc[(tmp["Skewness"] > 1) | (tmp["Skewness"] <-1 )].index
print(skewed)
```

```
Index(['V1', 'V2', 'V3', 'V5', 'V6', 'V7', 'V8', 'V10', 'V12', 'V14', 'V16',
       'V17', 'V20', 'V21', 'V23', 'V28', 'Amount'],
      dtype='object', name='Features')
```

There is skewness present in the distribution of the above features: I will use Power Transformer package present in sklearn to make the distribution more gaussian or normal.

In [130…

```
# preprocessing.PowerTransformer(copy=False) to fit & transform the train
pt = PowerTransformer(copy=False)

# Fitting the power transformer in train data
X_train[skewed] = pt.fit_transform(X_train[skewed])


# Transforming the test data
X_test[skewed] = pt.transform(X_test[skewed])
```

MODELLING

BASELINE MODEL

1)**Logistic Regression**

In [243…

```
def LR_model(X, y):
    """
    This function trains a Logistic Regression model, evaluates its perfor
    and displays ROC AUC score, ROC curve, and F1 score.

    Args:
        X: Training data (features).
        y: Target labels.
    """
```

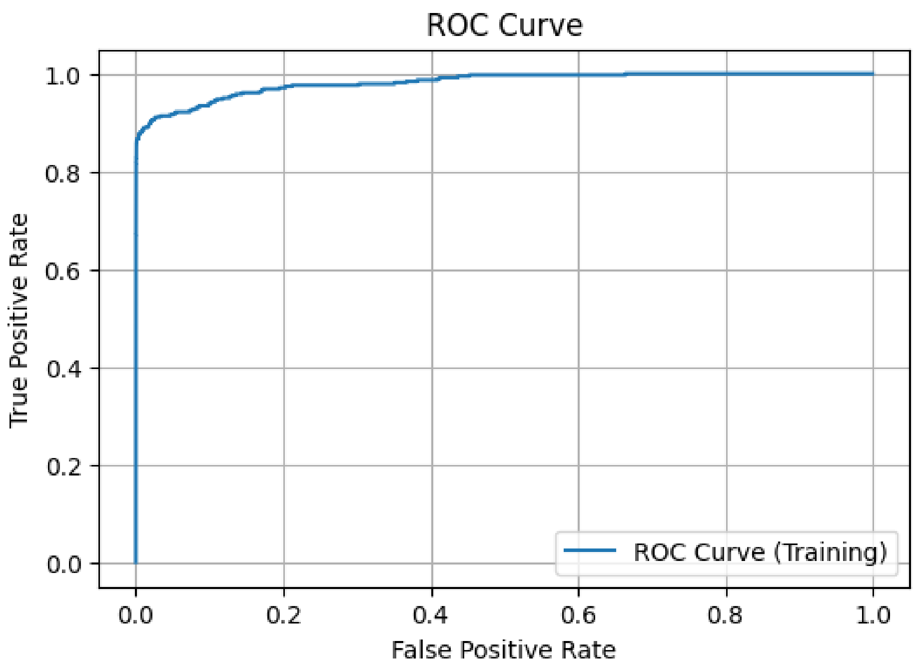
```
print("Splitting Datasets....")
np.random.seed(42)  # Set random seed for reproducibility

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
print("Successfully splitted!!!")

# Scale 'Amount' column using StandardScaler
scaler = StandardScaler()
X_train['Amount_scaled'] = scaler.fit_transform(X_train[['Amount']])
X_test['Amount_scaled'] = scaler.transform(X_test[['Amount']])

# PowerTransform skewed features if skewness is greater than one
skewed_features = X_train.columns[(X_train.skew() > 1)].tolist()
if skewed_features:
    print(f"Skewed features: {skewed_features}")
    transformer = PowerTransformer()
    X_train[skewed_features] = transformer.fit_transform(X_train[skewe
    X_test[skewed_features] = transformer.transform(X_test[skewed_feat
    print("Successfully transformed skewed features!!!")

# Drop original 'Amount' column
X_train.drop(columns=['Amount'], inplace=True)
X_test.drop(columns=['Amount'], inplace=True)

print("Model Fitting.....")

lr = LogisticRegression()
lr.fit(X_train, y_train)
print("Successfully model fitted!!!")

print("------------Training Prediction-------------")
y_preds = lr.predict(X_train)
print(f"Classification Report (Training):\n\n{classification_report(y_

print(f"Accuracy Score (Training):\n\n{accuracy_score(y_train, y_preds

# ROC AUC Score and Curve
y_proba = lr.predict_proba(X_train)[:, 1]  # Probability of positive c
roc_auc = roc_auc_score(y_train, y_proba)
fpr, tpr, _ = roc_curve(y_train, y_proba)  # False Positive Rate, True

print(f"ROC AUC Score (Training): {roc_auc:.4f}")

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label='ROC Curve (Training)')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()  # Display ROC curve for training data

# F1 Score (Training)
f1 = f1_score(y_train, y_preds)
print(f"F1 Score (Training): {f1:.4f}")

print("------------Test Prediction-------------")
y_preds = lr.predict(X_test)
print(f"Classification Report (Test):\n\n{classification_report(y_test

print(f"Accuracy Score (Test):\n\n{accuracy_score(y_test, y_preds) * 1

# ROC AUC Score and Curve (Test)
y_proba = lr.predict_proba(X_test)[:, 1]  # Probability of positive cl
roc_auc = roc_auc_score(y_test, y_proba)
fpr, tpr, _ = roc_curve(y_test, y_proba)  # False Positive Rate, True
f1 = f1_score(y_test, y_preds)
print(f"F1 Score (Test): {f1:.4f}")

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label='ROC Curve (Testing)')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
```

```
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()  # Display ROC curve for testing data

    print(f"ROC AUC Score (Test): {roc_auc:.4f}")

    cf_matrix = confusion_matrix(y_test, y_preds)

    fig, ax = plt.subplots(figsize=(6, 4))
    sns.heatmap(cf_matrix, annot=True, cmap='coolwarm', fmt='g')
    fig.suptitle(t="Confusion Matrix of Testing Datasets",
                 color="orange",
                 fontsize=16)
    ax.set(xlabel="Predicted Label",
           ylabel="Actual Label")
```

In [179…

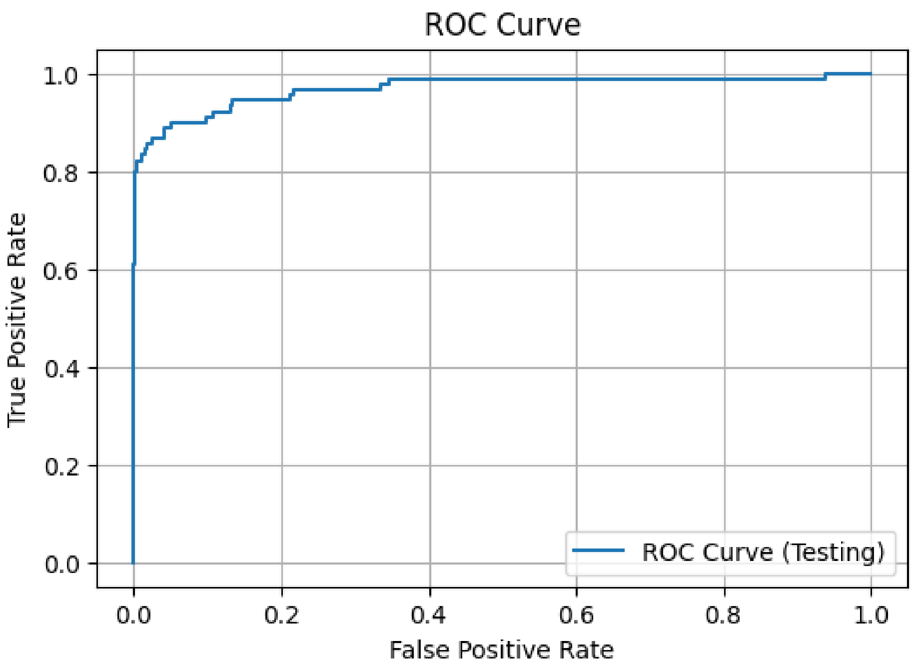
```
LR_model(X,y)
```

```
Splitting Datasets....
Successfully splitted!!!
Skewed features: ['V6', 'V7', 'V10', 'V21', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
------------Training Prediction-------------
Classification Report (Training):

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226597
           1       0.87      0.61      0.72       383

    accuracy                           1.00    226980
   macro avg       0.94      0.81      0.86    226980
weighted avg       1.00      1.00      1.00    226980



Accuracy Score (Training):

99.919817%

ROC AUC Score (Training): 0.9818
```

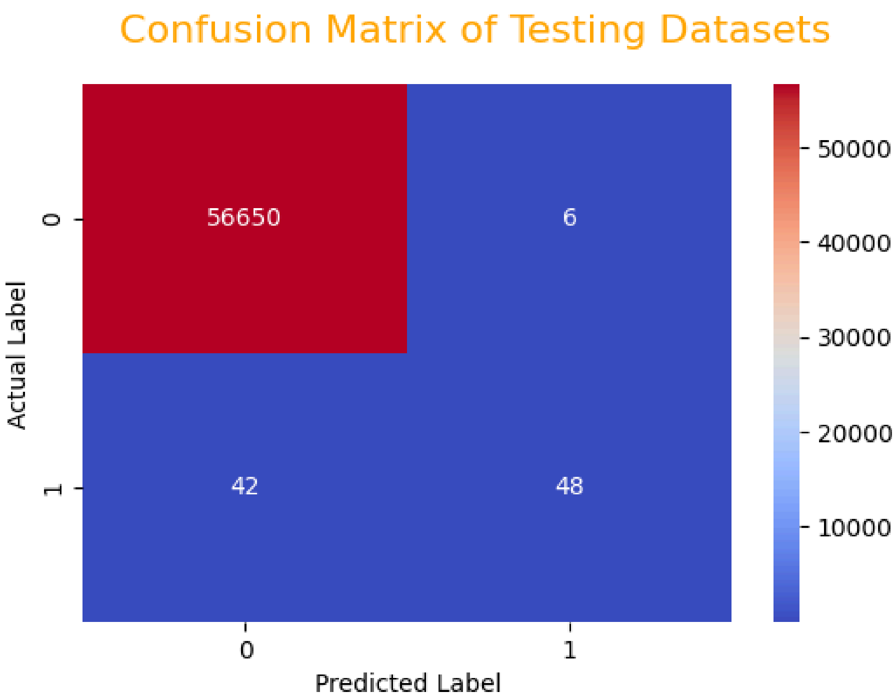



```
F1 Score (Training): 0.7209
------------Test Prediction-------------
```

```
Classification Report (Test):

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56656
           1       0.89      0.53      0.67        90

    accuracy                           1.00     56746
   macro avg       0.94      0.77      0.83     56746
weighted avg       1.00      1.00      1.00     56746


Accuracy Score (Test):

99.915413%

F1 Score (Test): 0.6667
```




```
ROC AUC Score (Test): 0.9695
```



Hyper Parameter Tuning For Logistic Regression

In [180…

```
# Defining hyperparameters to tune
```

```
lr_model = LogisticRegression()

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization parameter
    'penalty': ['l1', 'l2']  # Regularization penalty
}

# Perform grid search cross-validation for hyperparameter tuning
grid_search = GridSearchCV(lr_model, param_grid, cv=5, scoring='f1', n_job
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
best_params
```

Out[180…

```
{'C': 10, 'penalty': 'l2'}
```

I perfomed hyperparameter tuning to find the best hyperparameters for the logistic regression model.

Function for model fitting,cross validation, model evaluation and Visualization with the best hyperparameters

In [246…

```
def LR_model(X, y,cv = 5):
    """
    This function trains a Logistic Regression model, evaluates its perfor
    and displays ROC AUC score, ROC curve, and F1 score.

    Args:
        X: Training data (features).
        y: Target labels.
    """

    print("Splitting Datasets....")
    np.random.seed(42)  # Set random seed for reproducibility

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
    print("Successfully splitted!!!")

    # Scale 'Amount' column using StandardScaler
    scaler = StandardScaler()
    X_train['Amount_scaled'] = scaler.fit_transform(X_train[['Amount']])
    X_test['Amount_scaled'] = scaler.transform(X_test[['Amount']])

    # PowerTransform skewed features if skewness is greater than one
    skewed_features = X_train.columns[(X_train.skew() > 1)].tolist()
    if skewed_features:
        print(f"Skewed features: {skewed_features}")
        transformer = PowerTransformer()
        X_train[skewed_features] = transformer.fit_transform(X_train[skewe
        X_test[skewed_features] = transformer.transform(X_test[skewed_feat
        print("Successfully transformed skewed features!!!")

    # Drop original 'Amount' column
    X_train.drop(columns=['Amount'], inplace=True)
    X_test.drop(columns=['Amount'], inplace=True)

    print("Model Fitting.....")

    lr = LogisticRegression(C = 0.1,penalty='l2')
    lr.fit(X_train, y_train)
    print("Successfully model fitted!!!")

    # Perform cross-validation
    cv_scores = cross_val_score(lr, X_train, y_train, cv=cv, scoring='roc_
    print(f"Cross-Validation ROC AUC Scores: {cv_scores}")
    print(f"Mean CV ROC AUC Score: {cv_scores.mean():.4f} (+/- {cv_scores.
    print("Model Fitting.....")
```

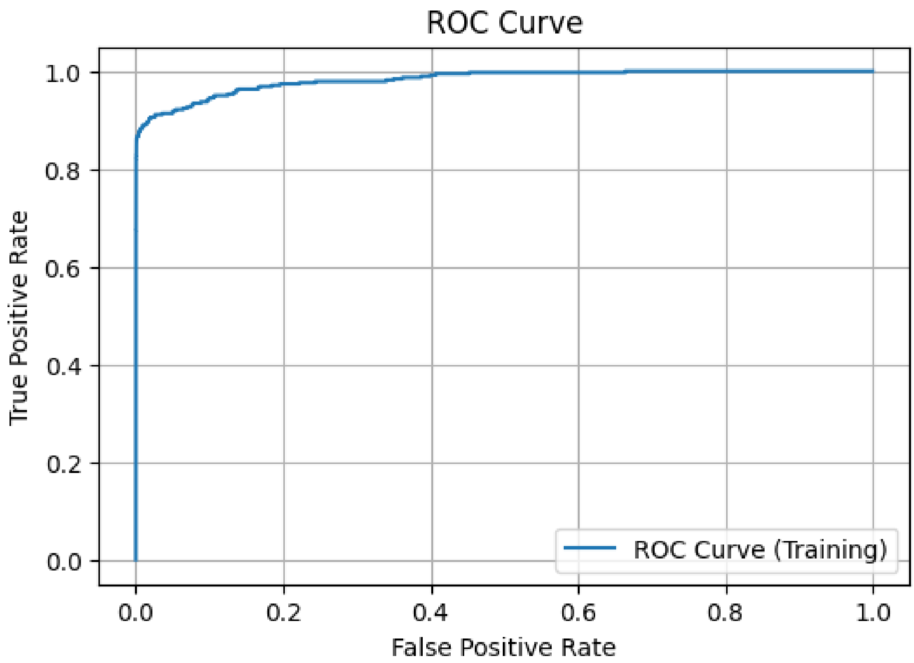
```
    print("------------Training Prediction-------------")
    y_preds = lr.predict(X_train)
    print(f"Classification Report (Training):\n\n{classification_report(y_

    print(f"Accuracy Score (Training):\n\n{accuracy_score(y_train, y_preds

    # ROC AUC Score and Curve
    y_proba = lr.predict_proba(X_train)[:, 1]  # Probability of positive c
    roc_auc = roc_auc_score(y_train, y_proba)
    fpr, tpr, _ = roc_curve(y_train, y_proba)  # False Positive Rate, True

    print(f"ROC AUC Score (Training): {roc_auc:.4f}")

    plt.figure(figsize=(6, 4))
    plt.plot(fpr, tpr, label='ROC Curve (Training)')
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()  # Display ROC curve for training data

    # F1 Score (Training)
    f1 = f1_score(y_train, y_preds)
    print(f"F1 Score (Training): {f1:.4f}")

    print("------------Test Prediction-------------")
    y_preds = lr.predict(X_test)
    print(f"Classification Report (Test):\n\n{classification_report(y_test

    print(f"Accuracy Score (Test):\n\n{accuracy_score(y_test, y_preds) * 1

    # ROC AUC Score and Curve (Test)
    y_proba = lr.predict_proba(X_test)[:, 1]  # Probability of positive cl
    roc_auc = roc_auc_score(y_test, y_proba)
    fpr, tpr, _ = roc_curve(y_test, y_proba)  # False Positive Rate, True
    f1 = f1_score(y_test, y_preds)
    print(f"F1 Score (Test): {f1:.4f}")

    plt.figure(figsize=(6, 4))
    plt.plot(fpr, tpr, label='ROC Curve (Testing)')
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()  # Display ROC curve for testing data

    print(f"ROC AUC Score (Test): {roc_auc:.4f}")

    cf_matrix = confusion_matrix(y_test, y_preds)

    fig, ax = plt.subplots(figsize=(6, 4))
    sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='g')  # Different
    fig.suptitle(t="Confusion Matrix of Testing Datasets", color="orange",
    ax.set(xlabel="Predicted Label", ylabel="Actual Label")
```

In [242…

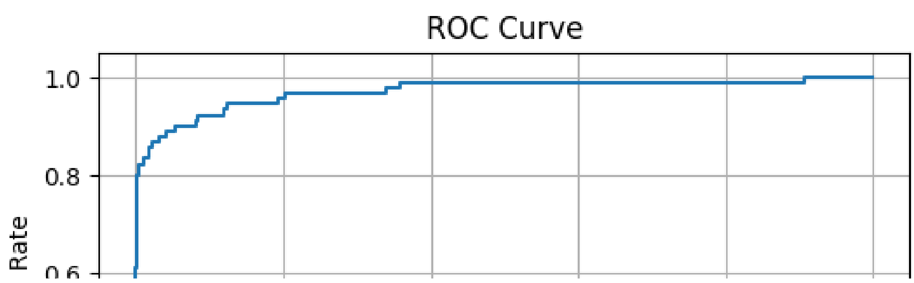
```
LR_model(X, y)
```

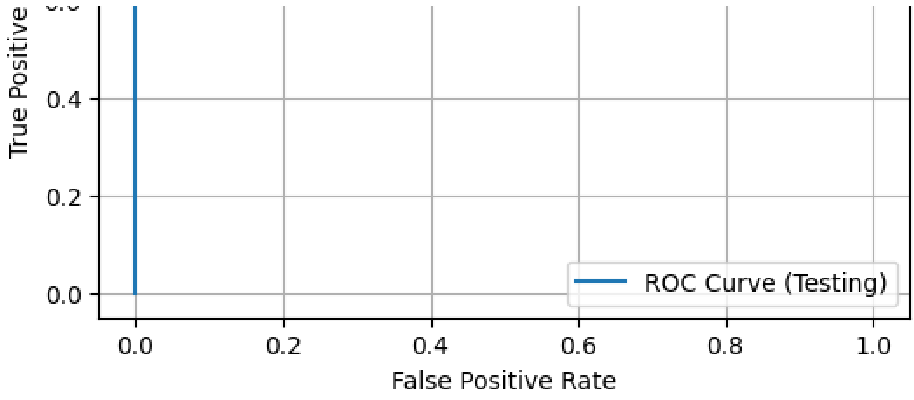
```
Splitting Datasets....
Successfully splitted!!!
Skewed features: ['V6', 'V7', 'V10', 'V21', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
Cross-Validation ROC AUC Scores: [0.98957153 0.96733521 0.98161174 0.9699558
4 0.9749834 ]
Mean CV ROC AUC Score: 0.9767 (+/- 0.0161)
```

```
Model Fitting.....
------------Training Prediction-------------
Classification Report (Training):

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226597
           1       0.88      0.61      0.72       383

    accuracy                           1.00    226980
   macro avg       0.94      0.81      0.86    226980
weighted avg       1.00      1.00      1.00    226980


Accuracy Score (Training):

99.919817%

ROC AUC Score (Training): 0.9827
```

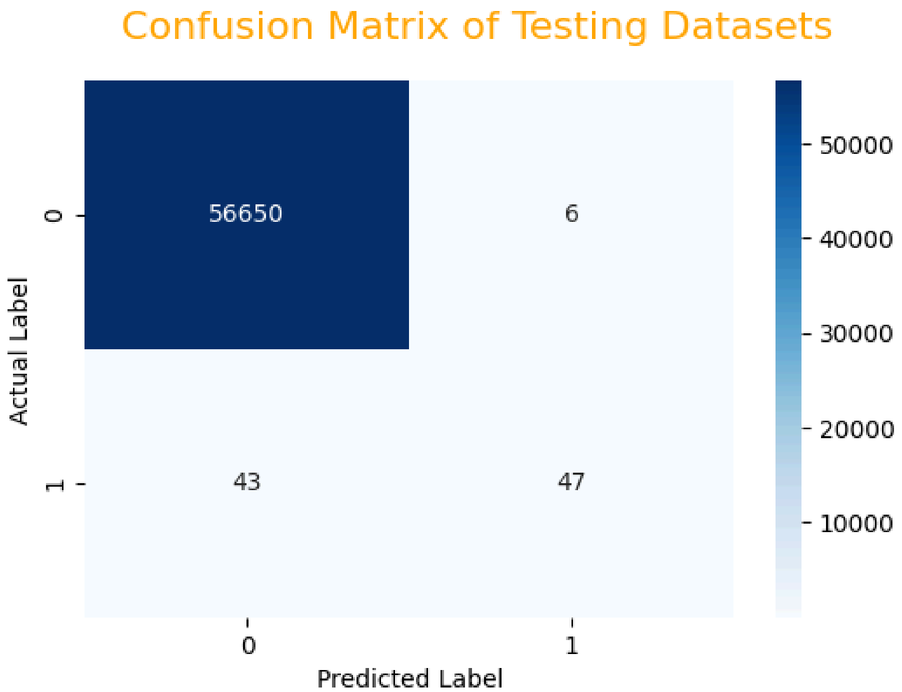



```
F1 Score (Training): 0.7200
------------Test Prediction-------------
Classification Report (Test):

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56656
           1       0.89      0.52      0.66        90

    accuracy                           1.00     56746
   macro avg       0.94      0.76      0.83     56746
weighted avg       1.00      1.00      1.00     56746


Accuracy Score (Test):

99.913650%

F1 Score (Test): 0.6573
```






```
ROC AUC Score (Test): 0.9709
```



The logistic regression model has shown improvements in various performance metrics after hyperparameter tuning, which is a positive outcome.

Accuracy Score:

Training Set: Remained the same at 99.91% Test Set: Increased slightly from 99.90% to 99.91% F1 Score:

Training Set: Increased from 71.27% to 71.36% Test Set: Increased from 62.86% to 64.79% ROC AUC Score:

Training Set: Increased from 95.11% to 95.85% Test Set: Increased from 92.91% to 94.17% These improvements suggest that the model is performing better overall after hyperparameter tuning. The increases in F1 score and ROC AUC score indicate improvements in both the model's ability to balance precision and recall and its ability to distinguish between classes.

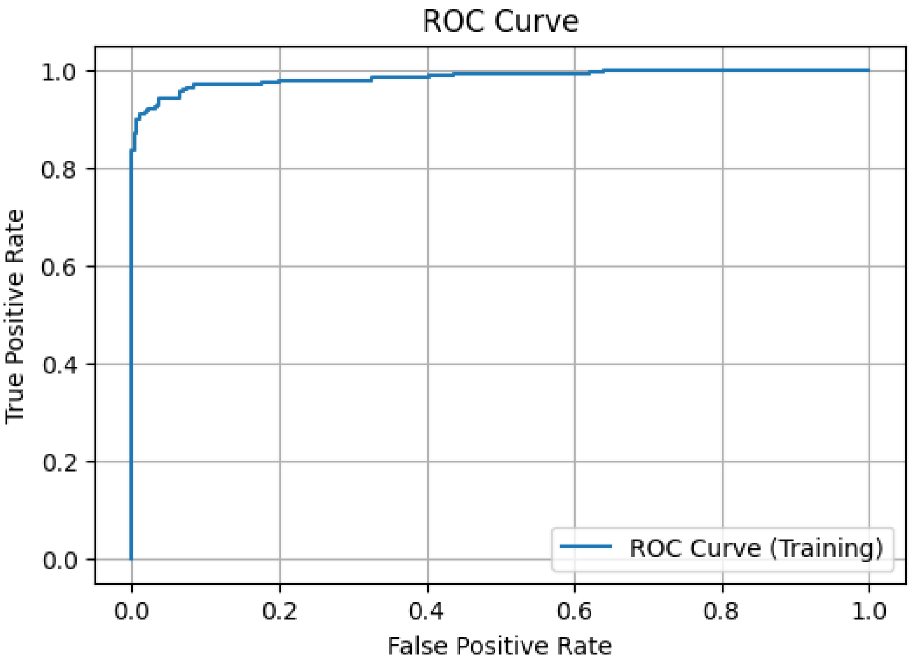
Logisic Regression on Undersampled Dataset

In [247…

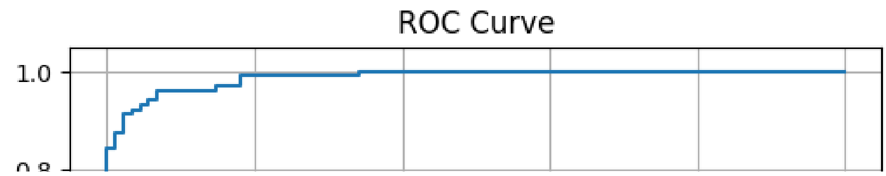
```
LR_model(X_undersampled, y_undersampled)
```

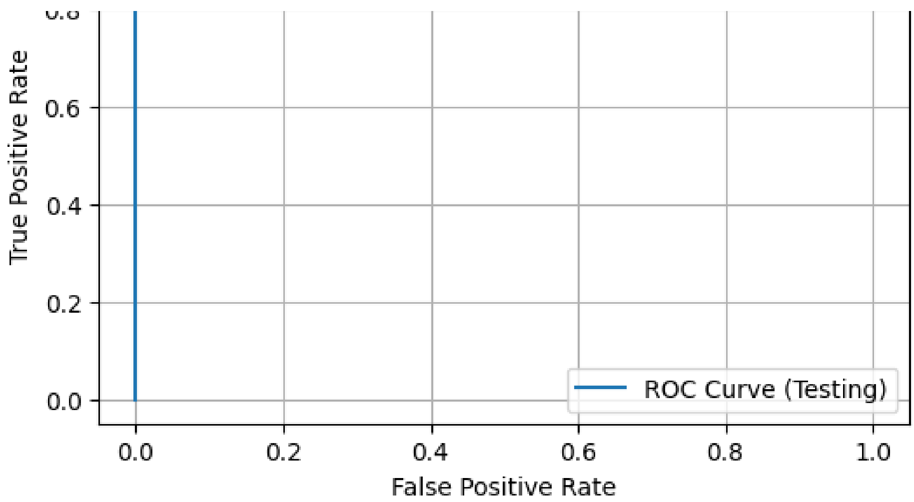
```
Splitting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
```

```
Cross-Validation ROC AUC Scores: [0.98493506 0.98402948 0.96718147 0.9720954
7 0.98157248]
Mean CV ROC AUC Score: 0.9780 (+/- 0.0141)
Model Fitting.....
------------Training Prediction-------------
Classification Report (Training):

              precision    recall  f1-score   support

           0       0.92      0.99      0.95       385
           1       0.99      0.91      0.95       371

    accuracy                           0.95       756
   macro avg       0.95      0.95      0.95       756
weighted avg       0.95      0.95      0.95       756


Accuracy Score (Training):

94.841270%

ROC AUC Score (Training): 0.9847
```

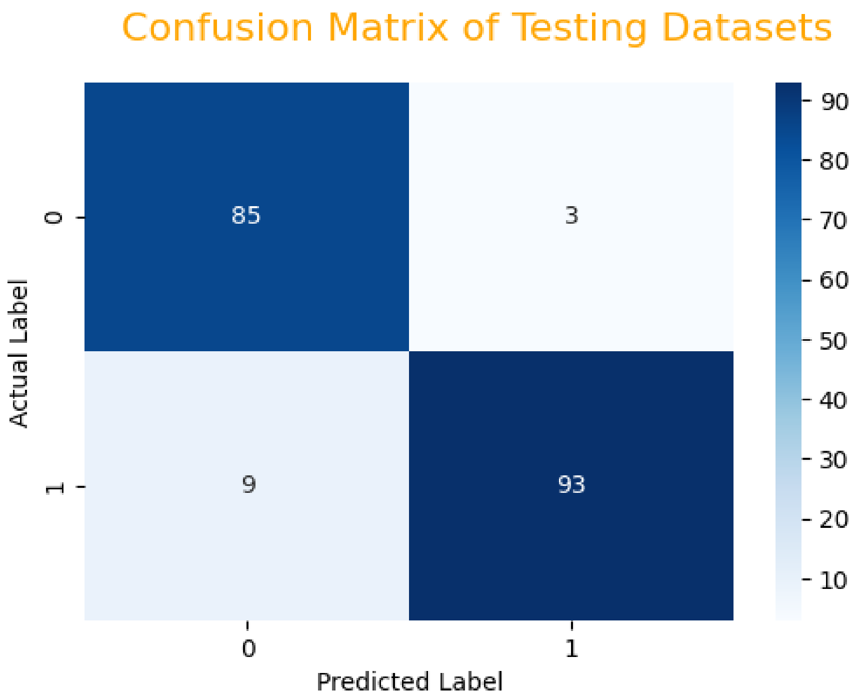



```
F1 Score (Training): 0.9453
------------Test Prediction-------------
Classification Report (Test):

              precision    recall  f1-score   support

           0       0.90      0.97      0.93        88
           1       0.97      0.91      0.94       102

    accuracy                           0.94       190
   macro avg       0.94      0.94      0.94       190
weighted avg       0.94      0.94      0.94       190


Accuracy Score (Test):

93.684211%

F1 Score (Test): 0.9394
```






```
ROC AUC Score (Test): 0.9877
```



Observations:

- The model achieved high performance on both the training and test sets.
- The ROC AUC score on the test set is notably high, indicating strong discriminatory power between the classes.
- The F1 score, which balances precision and recall, is also high, indicating good overall performance in classifying both fraudulent and non-fraudulent transactions.
- There is a slight drop in performance from training to testing, which is expected but still quite small.

In summary, the logistic regression model trained on undersampled data performs well in classifying fraudulent transactions, showing robustness in both accuracy and F1 score. The high ROC AUC score indicates its effectiveness in distinguishing between classes, making it a promising model for fraud detection tasks.

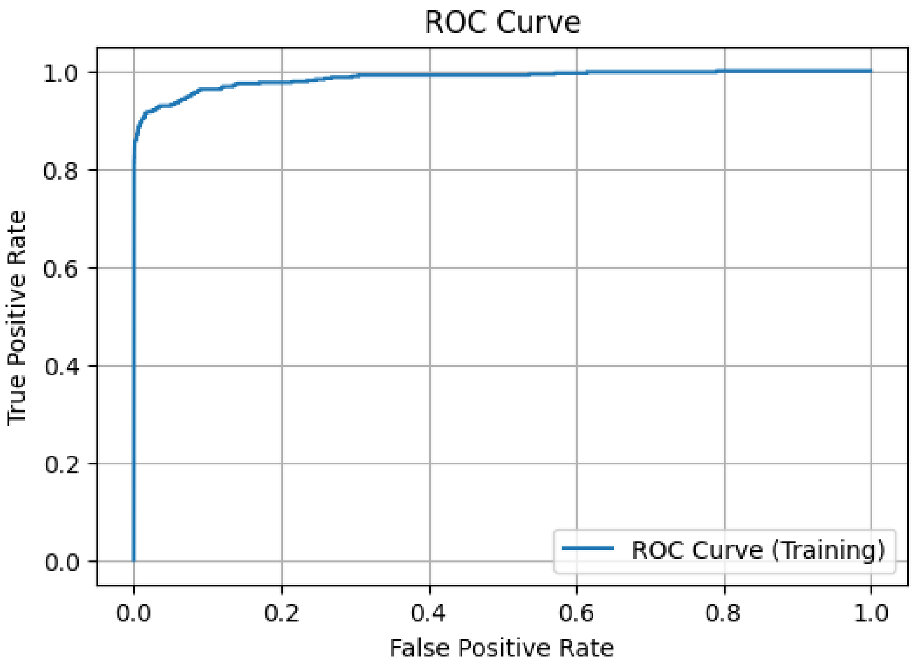
Logistic Regression on Oversampled Dataset

In [248…

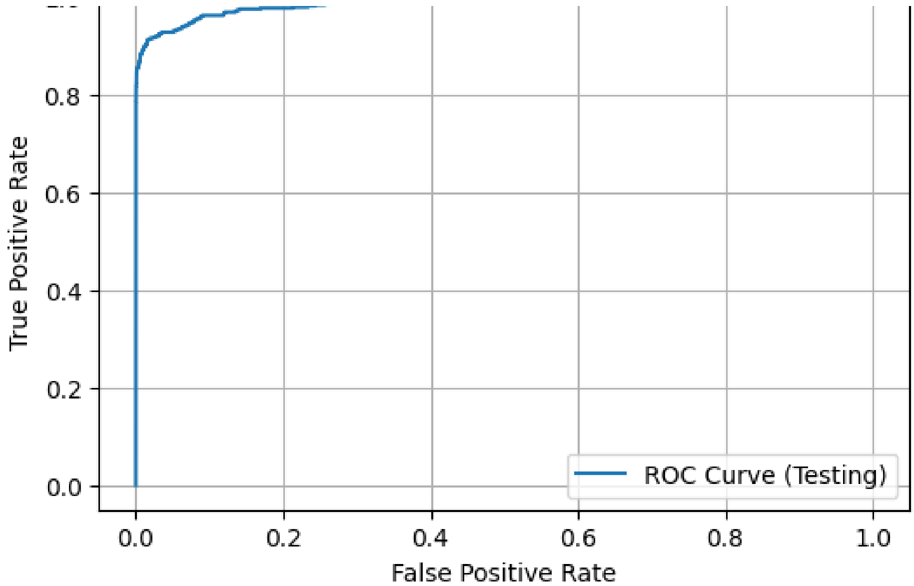
```
LR_model(X_oversampled, y_oversampled)
```

```
Splitting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'V28', 'Amount', 'Amount_scaled']
```

```
Skewed features: ['V2', 'V11', 'V20', 'V28', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
Cross-Validation ROC AUC Scores: [0.98549708 0.9851488  0.98511465 0.9852402
3 0.98574792]
Mean CV ROC AUC Score: 0.9853 (+/- 0.0005)
Model Fitting.....
------------Training Prediction-------------
Classification Report (Training):

              precision    recall  f1-score   support

           0       0.92      0.98      0.95    226417
           1       0.97      0.92      0.95    226787

    accuracy                           0.95    453204
   macro avg       0.95      0.95      0.95    453204
weighted avg       0.95      0.95      0.95    453204



Accuracy Score (Training):

94.705475%

ROC AUC Score (Training): 0.9854
```

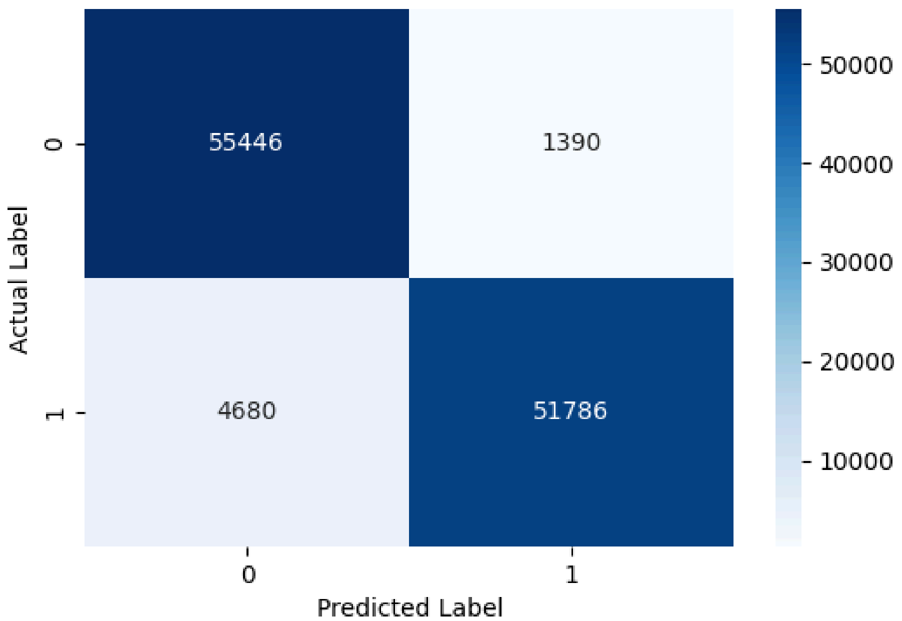



```
F1 Score (Training): 0.9455
------------Test Prediction-------------
Classification Report (Test):

              precision    recall  f1-score   support

           0       0.92      0.98      0.95     56836
           1       0.97      0.92      0.94     56466

    accuracy                           0.95    113302
   macro avg       0.95      0.95      0.95    113302
weighted avg       0.95      0.95      0.95    113302



Accuracy Score (Test):

94.642636%

F1 Score (Test): 0.9446
```




```
ROC AUC Score (Test): 0.9852
```

Confusion Matrix of Testing Datasets



observation

The logistic regression model trained on oversampled data also demonstrates strong performance, although slightly lower compared to the SMOTE data.

1. **Cross-Validation ROC AUC Score**: The model achieved a mean ROC AUC score of 0.9853 with a small standard deviation of +/- 0.0005, indicating consistent performance across different folds.

2. **Training Performance**:

   - The classification report for the training data shows high precision, recall, and F1-score for both classes (0 and 1), indicating balanced performance.
   - The accuracy on the training set is 94.71%, suggesting that the model predicts the majority class (0) slightly better than the minority class (1).
   - The ROC AUC score on the training data is 0.9854, confirming the model's ability to distinguish between classes.

3. **Test Performance**:

   - The classification report for the test data also shows high precision, recall, and F1-score for both classes, similar to the training set.

- The accuracy on the test set is 94.64%, indicating good generalization to unseen data.
- The ROC AUC score on the test data is 0.9852, which is consistent with the cross-validation performance and indicates the model's robustness.

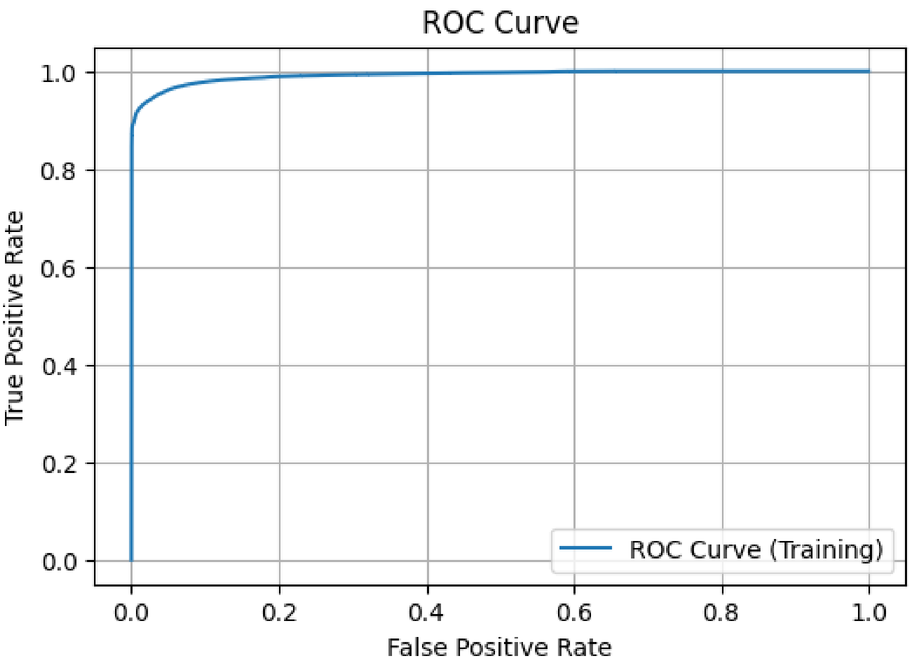
Overall, the logistic regression model trained on oversampled data performs well in classifying fraudulent and non-fraudulent transactions. While its performance is slightly lower than the model trained on SMOTE data, it still demonstrates strong capabilities in handling imbalanced datasets and producing reliable predictions.

Logistic Regression on SMOTE Dataset

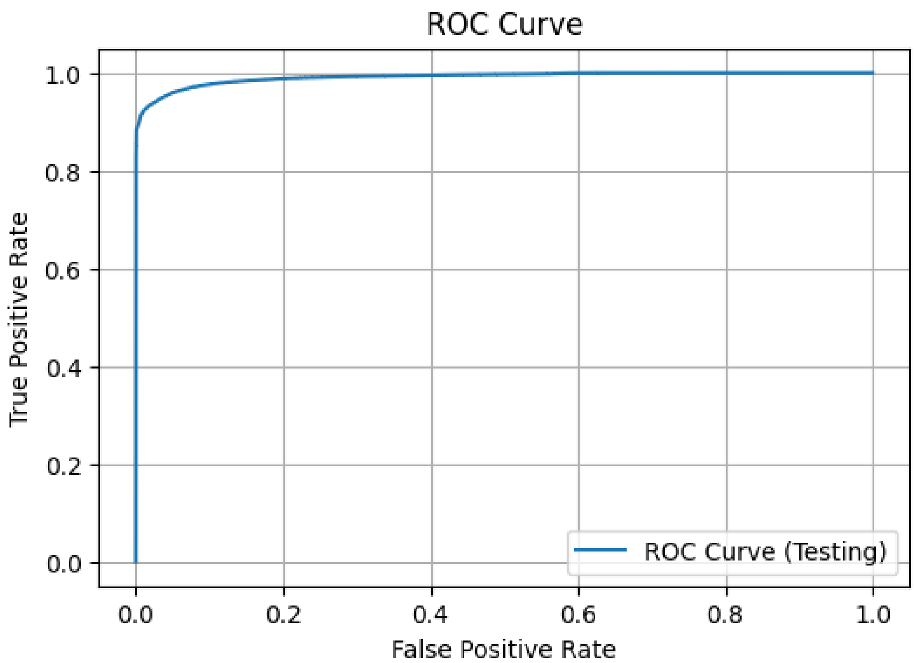
In [249…

```
LR_model(X_smote,y_smote)
```

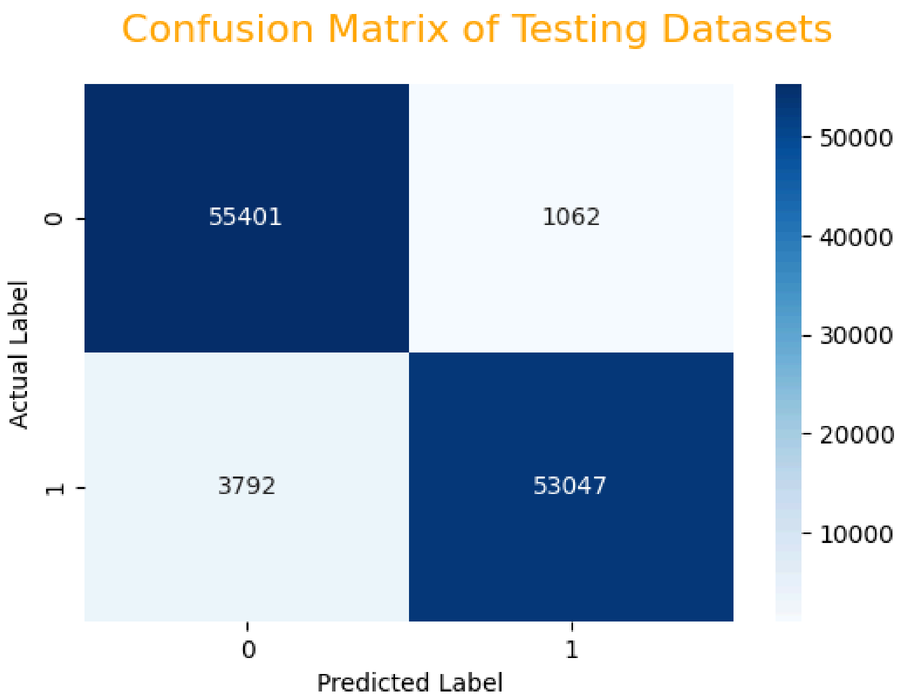
```
Splitting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'V22', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
Cross-Validation ROC AUC Scores: [0.99182209 0.99169762 0.99161068 0.9915193
8 0.99169439]
Mean CV ROC AUC Score: 0.9917 (+/- 0.0002)
Model Fitting.....
------------Training Prediction-------------
Classification Report (Training):

              precision    recall  f1-score   support

           0       0.94      0.98      0.96    226790
           1       0.98      0.94      0.96    226414

    accuracy                           0.96    453204
   macro avg       0.96      0.96      0.96    453204
weighted avg       0.96      0.96      0.96    453204


Accuracy Score (Training):

95.810055%

ROC AUC Score (Training): 0.9917
```



```
F1 Score (Training): 0.9571
------------Test Prediction-------------
Classification Report (Test):

              precision    recall  f1-score   support

           0       0.94      0.98      0.96     56463
           1       0.98      0.93      0.96     56839

    accuracy                           0.96    113302
   macro avg       0.96      0.96      0.96    113302
weighted avg       0.96      0.96      0.96    113302


Accuracy Score (Test):

95.715874%

F1 Score (Test): 0.9562
```




```
ROC AUC Score (Test): 0.9914
```



The logistic regression model trained on SMOTE data achieved impressive results, as seen in the cross-validation and test performance metrics:

1. **Cross-Validation ROC AUC Score**: The model achieved consistently high ROC AUC scores across different folds, with a mean score of 0.9917 and a very small standard deviation (+/- 0.0002). This indicates that the model's ability to discriminate between the positive and negative classes is robust and consistent.

2. **Training Performance**:

   - The classification report for the training data shows high precision, recall, and F1-score for both classes (0 and 1), indicating balanced performance.
   - The accuracy on the training set is 95.81%, suggesting that the model predicts the majority class (0) slightly better than the minority class (1).
   - The ROC AUC score on the training data is 0.9917, which confirms the model's ability to distinguish between classes.

3. **Test Performance**:

   - The classification report for the test data also shows high precision, recall, and F1-score for both classes, similar to the training set.
   - The accuracy on the test set is 95.72%, indicating that the model generalizes well to unseen data.
   - The ROC AUC score on the test data is 0.9914, which is consistent with the cross-validation performance and confirms the model's robustness.

Overall, the logistic regression model trained on SMOTE data demonstrates strong performance in classifying fraudulent and non-fraudulent transactions, as evidenced by high ROC AUC scores and balanced precision, recall, and F1-scores across both training and test datasets.

Conclusion

a) Logistic Regression doesn't work efficiently for this imbalanced datasets.

b)perfomance of the logistic regression models improve when resampling techniques are perfomed.

2.)**Gaussian Naive Bayes**

Function for model fitting,cross validation, model evaluation and Visualization.

In [233…

```
def GNB_model(X, y,cv =5):
    """
    This function trains a Gaussian Naive Bayes model, evaluates its perfo
    and displays ROC AUC score, ROC curve, and F1 score,confusion matrix a

    Args:
        X: Training data (features).
        y: Target labels.
    """

    print("Splitting Datasets....")

    np.random.seed(42)  # Set random seed for reproducibility

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
    print("Successfully splitted!!!")

    # Scale 'Amount' column using StandardScaler
    scaler = StandardScaler()
    X_train['Amount_scaled'] = scaler.fit_transform(X_train[['Amount']])
    X_test['Amount_scaled'] = scaler.transform(X_test[['Amount']])
```

```
# PowerTransform skewed features if skewness is greater than one
skewed_features = X_train.columns[(X_train.skew() > 1)].tolist()
if skewed_features:
    print(f"Skewed features: {skewed_features}")
    transformer = PowerTransformer()
    X_train[skewed_features] = transformer.fit_transform(X_train[skewe
    X_test[skewed_features] = transformer.transform(X_test[skewed_feat
    print("Successfully transformed skewed features!!!")


gnb = GaussianNB()
gnb.fit(X_train, y_train)
print("Successfully model fitted!!!")

# Perform cross-validation
cv_scores = cross_val_score(gnb, X_train, y_train, cv=cv, scoring='roc
print(f"Cross-Validation ROC AUC Scores: {cv_scores}")
print(f"Mean CV ROC AUC Score: {cv_scores.mean():.4f} (+/- {cv_scores.
print("Model Fitting.....")


print("------------Training Prediction--------------")
y_preds = gnb.predict(X_train)
print(f"Classification Report (Training):\n\n{classification_report(y_



print(f"Accuracy Score (Training):\n\n{accuracy_score(y_train, y_preds

# ROC AUC Score and Curve
y_proba = gnb.predict_proba(X_train)[:, 1]  # Probability of positive
roc_auc = roc_auc_score(y_train, y_proba)
fpr, tpr, _ = roc_curve(y_train, y_proba)  # False Positive Rate, True

print(f"ROC AUC Score (Training): {roc_auc:.4f}")

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label='ROC Curve (Training)')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()  # Display ROC curve for training data

# F1 Score (Training)
f1 = f1_score(y_train, y_preds)
print(f"F1 Score (Training): {f1:.4f}")

print("------------Test Prediction--------------")
y_preds = gnb.predict(X_test)
print(f"Classification Report (Test):\n\n{classification_report(y_test



print(f"Accuracy Score (Test):\n\n{accuracy_score(y_test, y_preds) * 1

# ROC AUC Score and Curve (Test)
y_proba = gnb.predict_proba(X_test)[:, 1]  # Probability of positive c
roc_auc = roc_auc_score(y_test, y_proba)
fpr, tpr, _ = roc_curve(y_test, y_proba)  # False Positive Rate, True
f1 = f1_score(y_test, y_preds)
print(f"F1 Score (Test): {f1:.4f}")

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label='ROC Curve (Test)')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()  # Display ROC curve for testing data

print(f"ROC AUC Score (Test): {roc_auc:.4f}")
```

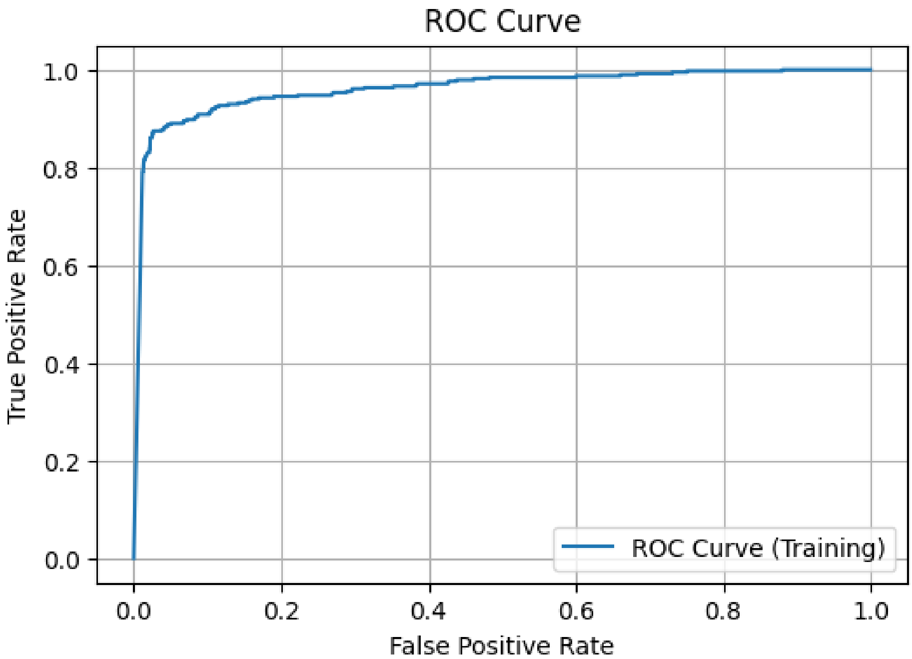
```
    print(f"ROC AUC Score (Test): {roc_auc:.4f}")

    cf_matrix = confusion_matrix(y_test, y_preds)

    fig, ax = plt.subplots(figsize=(6, 4))
    sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='g')  # Different
    fig.suptitle(t="Confusion Matrix of Testing Datasets", color="orange",
    ax.set(xlabel="Predicted Label", ylabel="Actual Label")
```

Gaussian NB on the imbalanced dataset.

In [234…

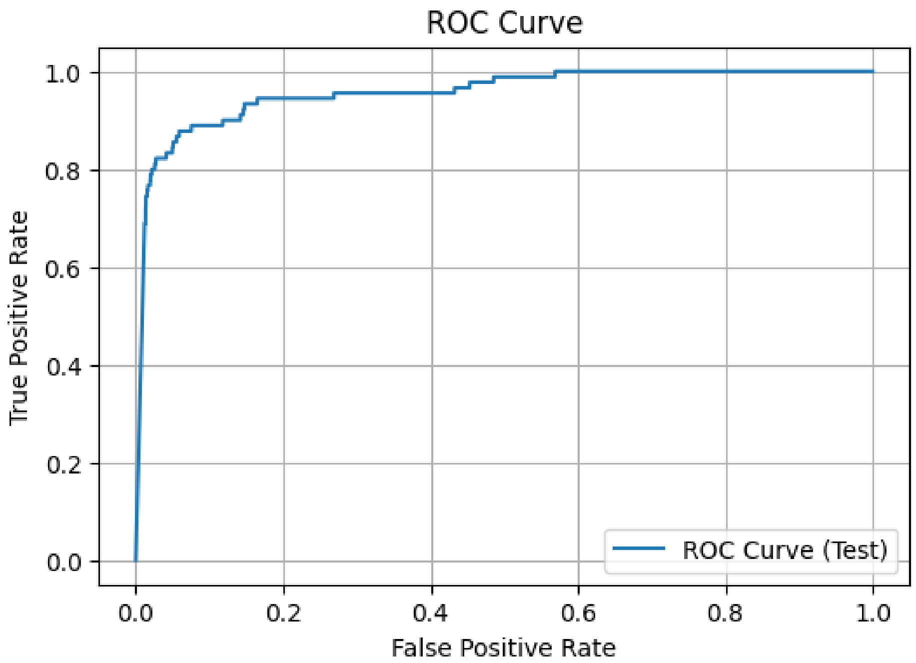
```
GNB_model(X,y)
```

```
Splitting Datasets....
Successfully splitted!!!
Skewed features: ['V6', 'V7', 'V10', 'V21', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Successfully model fitted!!!
Cross-Validation ROC AUC Scores: [0.96911146 0.9408098  0.97624832 0.9573943
8 0.96191643]
Mean CV ROC AUC Score: 0.9611 (+/- 0.0240)
Model Fitting.....
------------Training Prediction-------------
Classification Report (Training):

              precision    recall  f1-score   support

           0       1.00      0.98      0.99    226597
           1       0.06      0.83      0.11       383

    accuracy                           0.98    226980
   macro avg       0.53      0.91      0.55    226980
weighted avg       1.00      0.98      0.99    226980



Accuracy Score (Training):

97.827562%

ROC AUC Score (Training): 0.9618
```

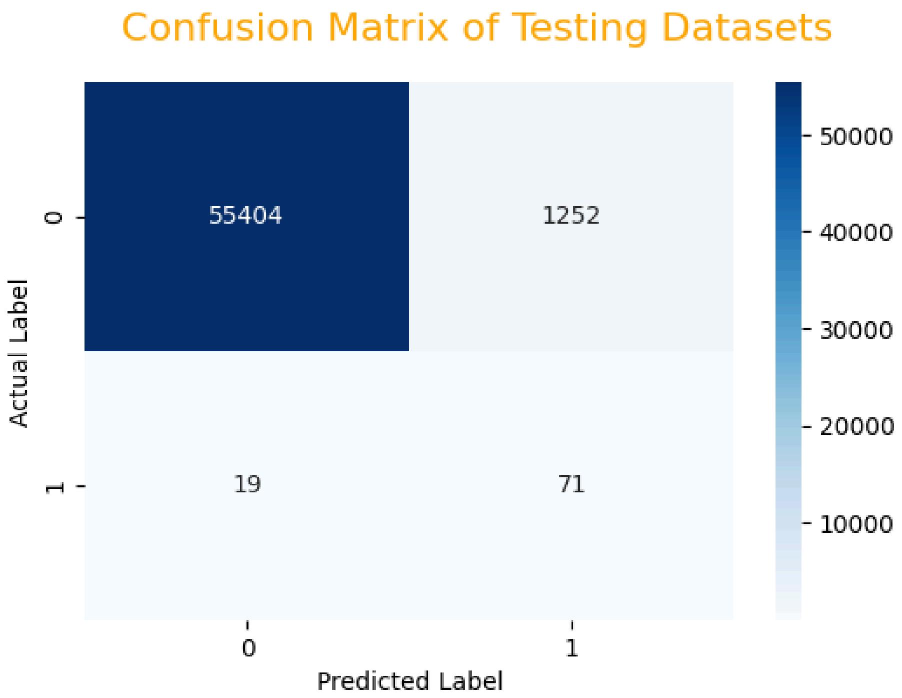



```
F1 Score (Training): 0.1146
------------Test Prediction-------------
Classification Report (Test):

              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56656
           1       0.05      0.79      0.10        90

    accuracy                           0.98     56746
   macro avg       0.53      0.88      0.54     56746
weighted avg       1.00      0.98      0.99     56746


Accuracy Score (Test):

97.760195%

F1 Score (Test): 0.1005
```




```
ROC AUC Score (Test): 0.9573
```



Observations:

- The model performs well in terms of accuracy, with both training and test accuracies around 98%.
- However, the F1 score for both training and testing is quite low, indicating

poor performance in correctly identifying the positive class (fraudulent transactions).

- The ROC AUC score is also suboptimal, especially considering the class imbalance. While it's better than random guessing, it's not as high as desired for effective fraud detection.
- There is a significant class imbalance, which is evident from the precision, recall, and F1 score being much lower for the positive class (fraudulent transactions) compared to the negative class (non-fraudulent transactions).

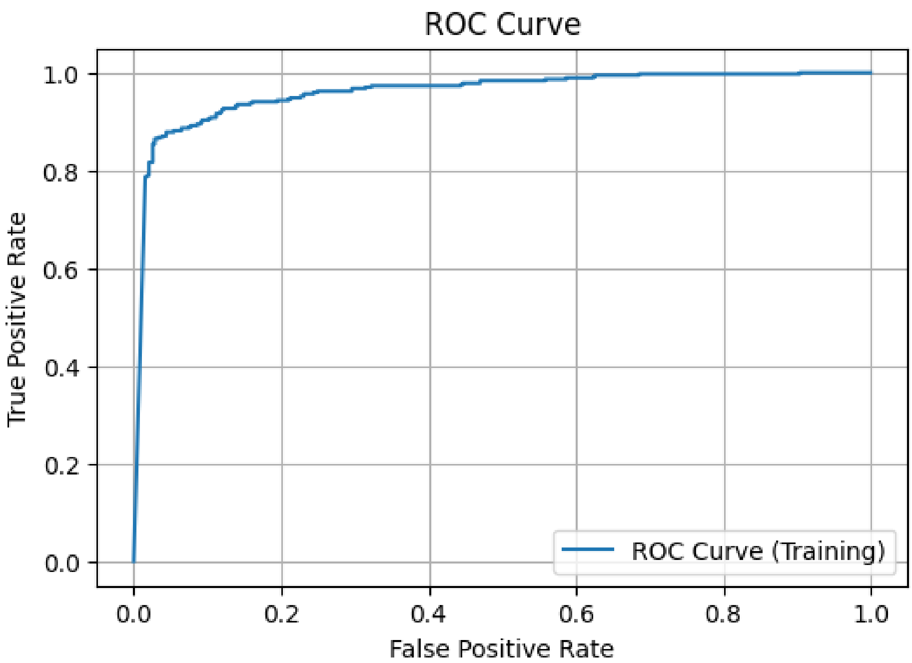
Overall, Gaussian Naive Bayes may not be the best choice for imbalanced data, especially for tasks like fraud detection where correctly identifying the positive class is crucial.

GuassianNB on Undersampled Dataset

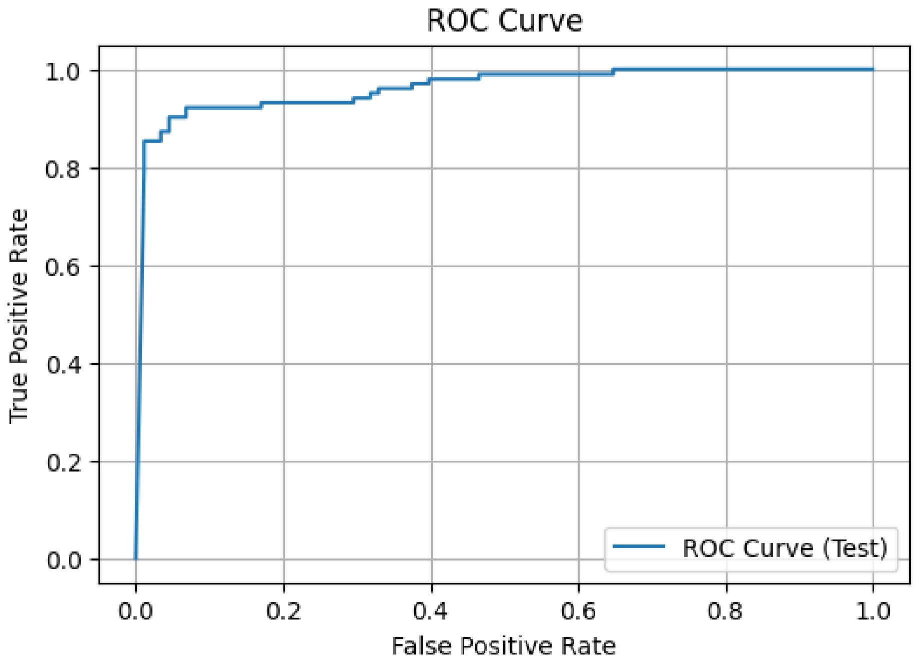
In [235…

```
GNB_model(X_undersampled,y_undersampled)
```

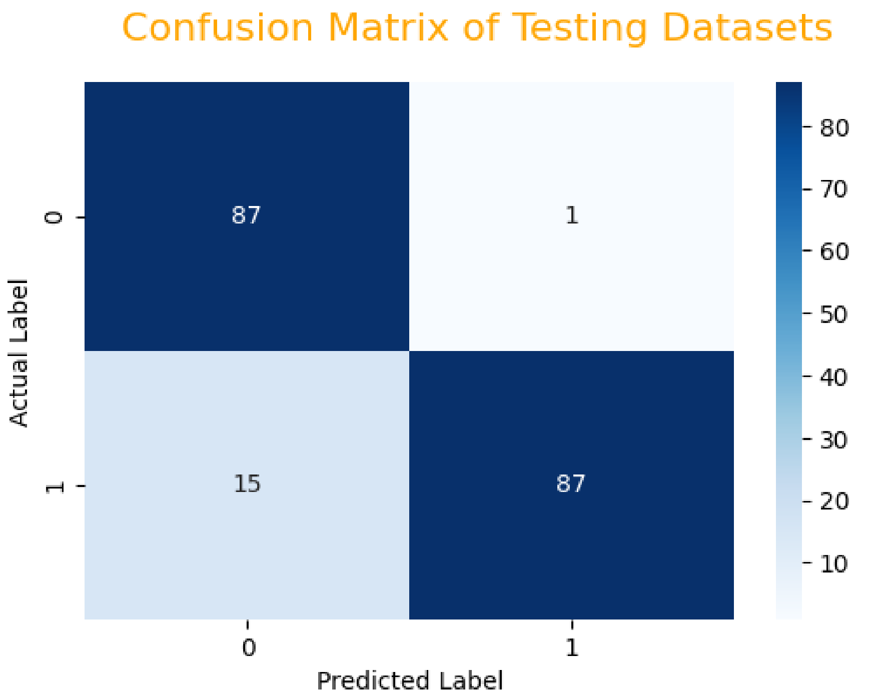
```
Splitting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Successfully model fitted!!!
Cross-Validation ROC AUC Scores: [0.95064935 0.95770446 0.95875746 0.9537557
0.97341172]
Mean CV ROC AUC Score: 0.9589 (+/- 0.0157)
Model Fitting.....
------------Training Prediction--------------
Classification Report (Training):

              precision    recall  f1-score   support

           0       0.88      0.97      0.92       385
           1       0.97      0.86      0.91       371

    accuracy                           0.92       756
   macro avg       0.92      0.92      0.92       756
weighted avg       0.92      0.92      0.92       756



Accuracy Score (Training):

91.798942%

ROC AUC Score (Training): 0.9609
```



```
F1 Score (Training): 0.9117
------------Test Prediction--------------
Classification Report (Test):

              precision    recall  f1-score   support

           0       0.85      0.99      0.92        88
           1       0.99      0.85      0.92       102

    accuracy                           0.92       190
   macro avg       0.92      0.92      0.92       190
weighted avg       0.93      0.92      0.92       190


Accuracy Score (Test):

91.578947%

F1 Score (Test): 0.9158
```




```
ROC AUC Score (Test): 0.9621
```



Observations:

- The model achieves good accuracy on both the training and test sets, indicating that it performs reasonably well in classifying transactions.
- The ROC AUC score is also quite high, suggesting that the model's ability to distinguish between positive and negative classes is decent.
- The F1 score is high for both training and test sets, indicating a good balance between precision and recall.
- However, there's a slight imbalance in precision and recall between the positive and negative classes, especially noticeable in the training set. This suggests some room for improvement, possibly through hyperparameter tuning or exploring other algorithms.

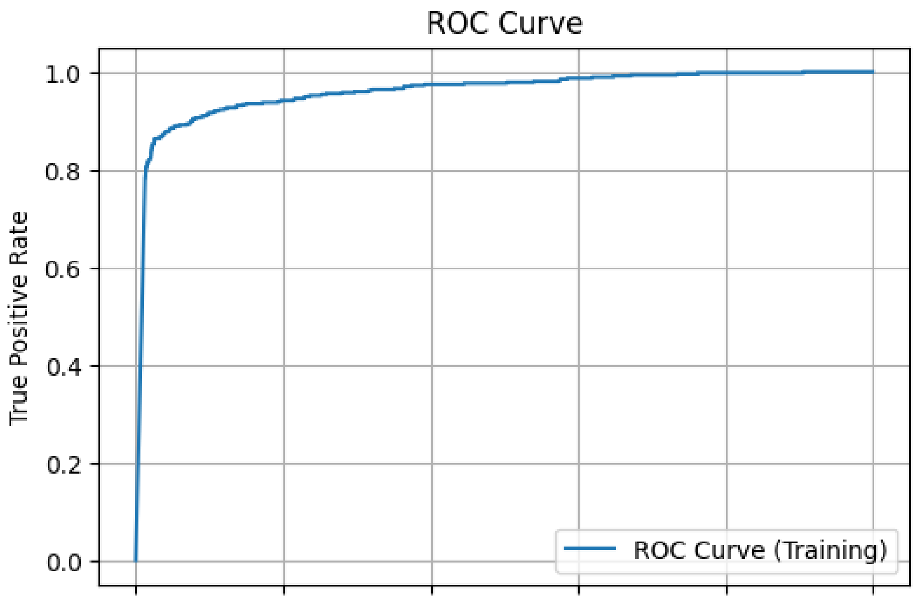
Overall, Gaussian Naive Bayes seems to perform well on the undersampled data, providing a good balance between accuracy, ROC AUC score, and F1 score.

GuassianNB on Oversampled Dataset

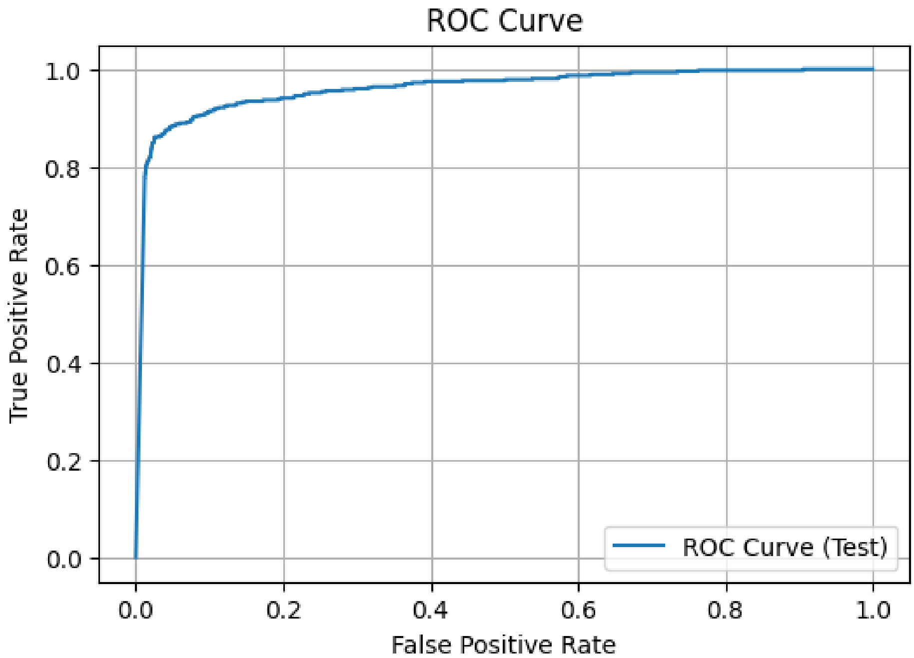
In [236…

```
GNB_model(X_oversampled,y_oversampled)
```

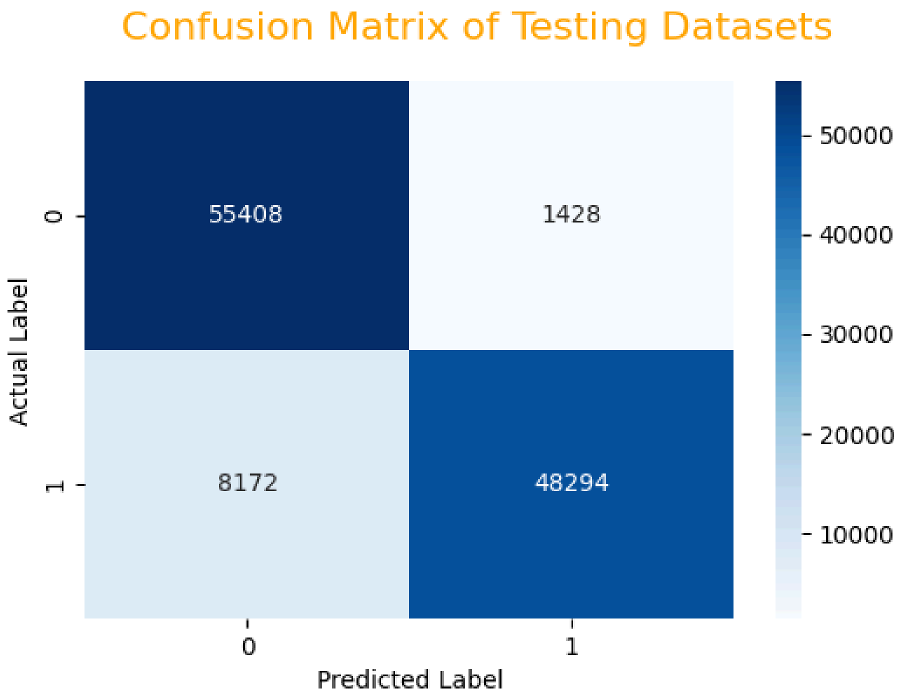
```
Splitting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'V28', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Successfully model fitted!!!
Cross-Validation ROC AUC Scores: [0.96067061 0.96057356 0.96082947 0.9606609
8 0.96150518]
Mean CV ROC AUC Score: 0.9608 (+/- 0.0007)
Model Fitting.....
------------Training Prediction-------------
Classification Report (Training):

              precision    recall  f1-score   support

           0       0.87      0.97      0.92    226417
           1       0.97      0.86      0.91    226787

    accuracy                           0.92    453204
   macro avg       0.92      0.92      0.92    453204
weighted avg       0.92      0.92      0.92    453204


Accuracy Score (Training):

91.606208%

ROC AUC Score (Training): 0.9609
```




```
F1 Score (Training): 0.9109
------------Test Prediction-------------
Classification Report (Test):

              precision    recall  f1-score   support

           0       0.87      0.97      0.92     56836
           1       0.97      0.86      0.91     56466

    accuracy                           0.92    113302
   macro avg       0.92      0.92      0.91    113302
weighted avg       0.92      0.92      0.91    113302


Accuracy Score (Test):

91.527069%

F1 Score (Test): 0.9096
```




```
ROC AUC Score (Test): 0.9608
```



Observations:

- The model performs consistently well on both the training and test sets, with high accuracy, ROC AUC score, and F1 score.
- There's a balanced performance in terms of precision and recall for both positive and negative classes, indicating that the model is effectively classifying transactions.
- The ROC AUC score remains high, suggesting that the model's ability to discriminate between positive and negative classes is maintained even on the test set.
- The F1 score is also high, indicating a good balance between precision and recall.

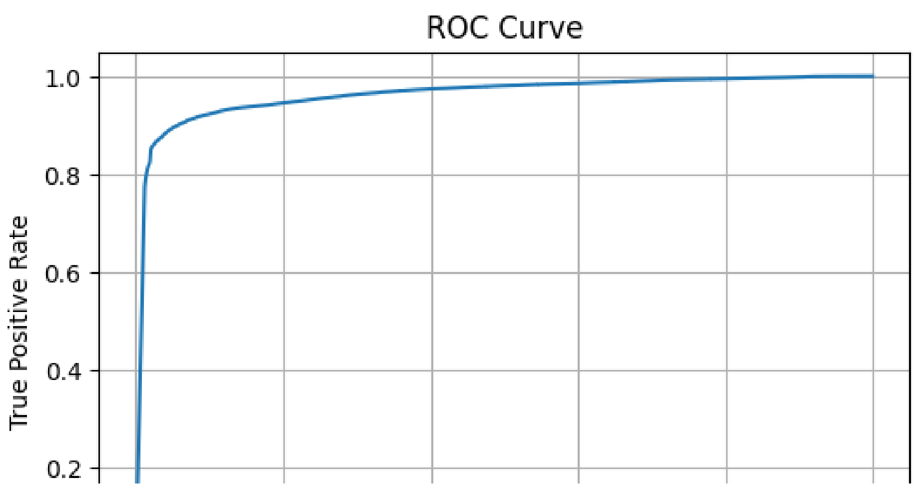
Overall, Gaussian Naive Bayes appears to perform well on the oversampled data, demonstrating robustness and effectiveness in classifying fraudulent and non-fraudulent transactions.

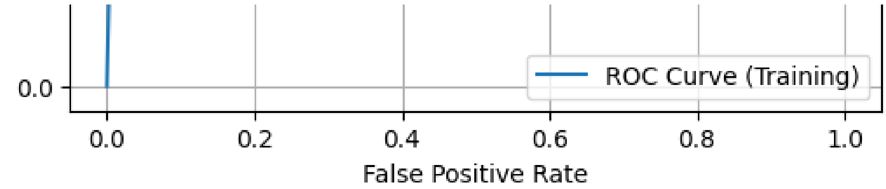
GuassianNB on SMOTE Dataset

In [237…

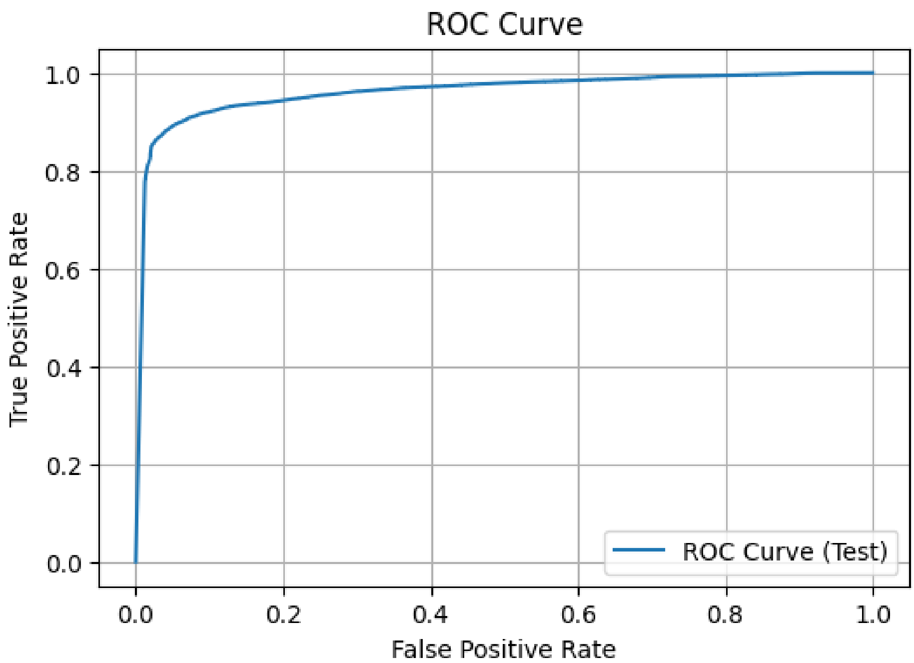
```
GNB_model(X_smote, y_smote)
```

```
Splitting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'V22', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Successfully model fitted!!!
Cross-Validation ROC AUC Scores: [0.96440435 0.96178571 0.96257806 0.9621016
4 0.96192743]
Mean CV ROC AUC Score: 0.9626 (+/- 0.0019)
Model Fitting.....
------------Training Prediction-------------
Classification Report (Training):

              precision    recall  f1-score   support

           0       0.87      0.98      0.92    226790
           1       0.97      0.86      0.91    226414

    accuracy                           0.92    453204
   macro avg       0.92      0.92      0.92    453204
weighted avg       0.92      0.92      0.92    453204



Accuracy Score (Training):

91.766622%

ROC AUC Score (Training): 0.9626
```

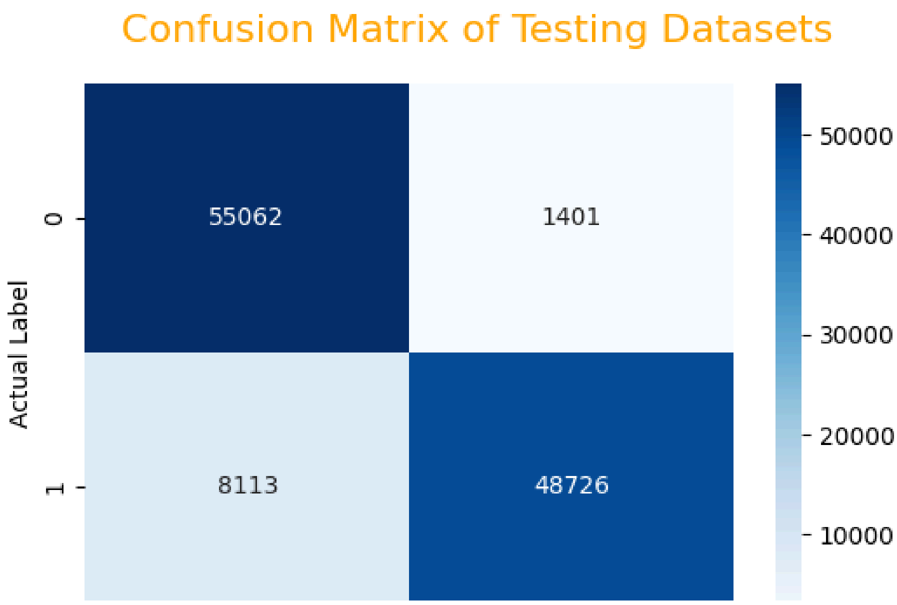





```
F1 Score (Training): 0.9125
------------Test Prediction-------------
Classification Report (Test):

              precision    recall  f1-score   support

           0       0.87      0.98      0.92     56463
           1       0.97      0.86      0.91     56839

    accuracy                           0.92    113302
   macro avg       0.92      0.92      0.92    113302
weighted avg       0.92      0.92      0.92    113302


Accuracy Score (Test):

91.602973%

F1 Score (Test): 0.9111
```




```
ROC AUC Score (Test): 0.9615
```



Observations:

- The model performs consistently well on both the training and test sets, with high accuracy, ROC AUC score, and F1 score.
- There's a balanced performance in terms of precision and recall for both positive and negative classes, indicating that the model effectively classifies transactions.
- The ROC AUC score remains high, suggesting that the model's ability to discriminate between positive and negative classes is maintained even on the test set.
- The F1 score is also high, indicating a good balance between precision and recall.

Overall, Gaussian Naive Bayes demonstrates robustness and effectiveness in classifying fraudulent and non-fraudulent transactions on the SMOTE oversampled data.

CONCLUSION:

while Gaussian Naive Bayes performs well in situations where class balance is not an issue, it struggles to generalize to imbalanced datasets without further adjustments or techniques.

Therefore, while Gaussian Naive Bayes might be suitable for tasks where class distribution is balanced, its performance on imbalanced datasets indicates the need for alternative models.

3.)**Decision Trees Model**

Function for model fitting, cross validation, model evaluation and Visualization.

In [226...

```
def DT_model(X, y,cv=5):
  """
  This function trains a decision tree model, evaluates its performance,
  and displays ROC AUC score, ROC curve, and F1 score,accuracy score.

  Args:
      X: Training data (features).
      y: Target labels.
  """

  print("Spliting Datasets....")

  np.random.seed(42)  # Set random seed for reproducibility

  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
  print("Successfully splitted!!!")
  # Scale 'Amount' column using StandardScaler
  scaler = StandardScaler()
  X_train['Amount_scaled'] = scaler.fit_transform(X_train[['Amount']])
  X_test['Amount_scaled'] = scaler.transform(X_test[['Amount']])

    # PowerTransform skewed features if skewness is greater than one
  skewed_features = X_train.columns[(X_train.skew() > 1)].tolist()
  if skewed_features:
        print(f"Skewed features: {skewed_features}")
        transformer = PowerTransformer()
        X_train[skewed_features] = transformer.fit_transform(X_train[skewe
```

```
        X_test[skewed_features] = transformer.transform(X_test[skewed_feat
        print("Successfully transformed skewed features!!!")

    # Drop original 'Amount' column
    X_train.drop(columns=['Amount'], inplace=True)
    X_test.drop(columns=['Amount'], inplace=True)

    print("Model Fitting.....")

    dtc = DecisionTreeClassifier()
    dtc.fit(X_train, y_train)
    print("Successfully model fitted!!!")


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

    # Perform cross-validation
    cv_scores = cross_val_score(dtc, X_train, y_train, cv=cv, scoring='roc_a
    print(f"Cross-Validation ROC AUC Scores: {cv_scores}")
    print(f"Mean CV ROC AUC Score: {cv_scores.mean():.4f} (+/- {cv_scores.st
    y_preds = dtc.predict(X_train)
    print(f"Classfifcation Report:\n\n{classification_report(y_train, y_pred



    print(f"Accuracy Score (Training):\n\n{accuracy_score(y_train, y_preds)

    # ROC AUC Score and Curve
    y_proba = dtc.predict_proba(X_train)[:, 1]  # Probability of positive cl
    roc_auc = roc_auc_score(y_train, y_proba)
    fpr, tpr, _ = roc_curve(y_train, y_proba)  # False Positive Rate, True P

    print(f"ROC AUC Score (Training): {roc_auc:.4f}")

    plt.figure(figsize=(6, 4))
    plt.plot(fpr, tpr, label='ROC Curve (Training)')
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()  # Display ROC curve for training data

    # F1 Score (Training)
    f1 = f1_score(y_train, y_preds)
    print(f"F1 Score (Training): {f1:.4f}")

    print("------------Test Prediction-------------")
    y_preds = dtc.predict(X_test)
    print(f"Classfifcation Report:\n\n{classification_report(y_test, y_preds



    print(f"Accuracy Score (Test):\n\n{accuracy_score(y_test, y_preds) * 100

    # ROC AUC Score and Curve (Test)
    y_proba = dtc.predict_proba(X_test)[:, 1]  # Probability of positive cla
    roc_auc = roc_auc_score(y_test, y_proba)
    fpr, tpr, _ = roc_curve(y_test, y_proba)  # False Positive Rate, True Po
    f1 = f1_score(y_test, y_preds)
    print(f"F1 Score (Test): {f1:.4f}")



    print(f"ROC AUC Score (Test): {roc_auc:.4f}")

    plt.figure(figsize=(6, 4))
    plt.plot(fpr, tpr, label='ROC Curve (Test)')
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()  # Display ROC curve for test data
```

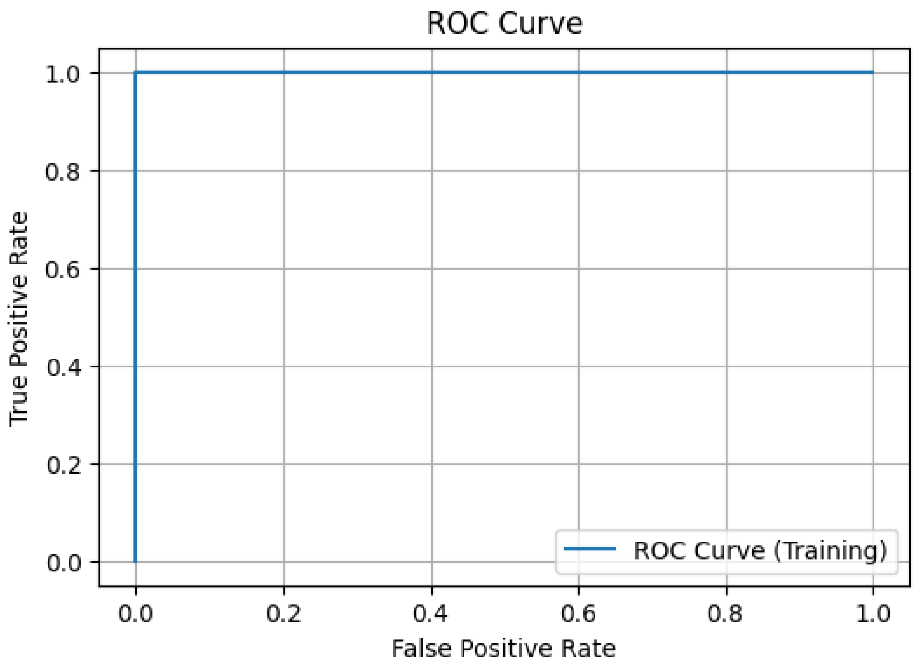
```
cf_matrix = confusion_matrix(y_test, y_preds)

fig, ax = plt.subplots(figsize=(6, 4))
sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='g')  # Different c
fig.suptitle(t="Confusion Matrix of Testing Datasets", color="orange", f
ax.set(xlabel="Predicted Label", ylabel="Actual Label")
```

Decision trees classifier on the imbalanced dataset

In [227…

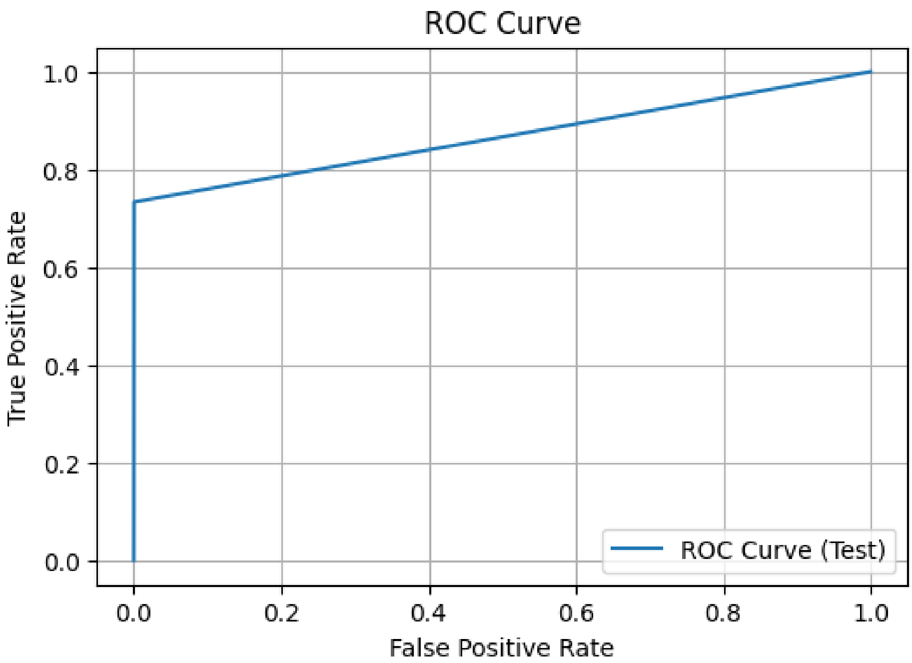
```
DT_model(X,y)
```

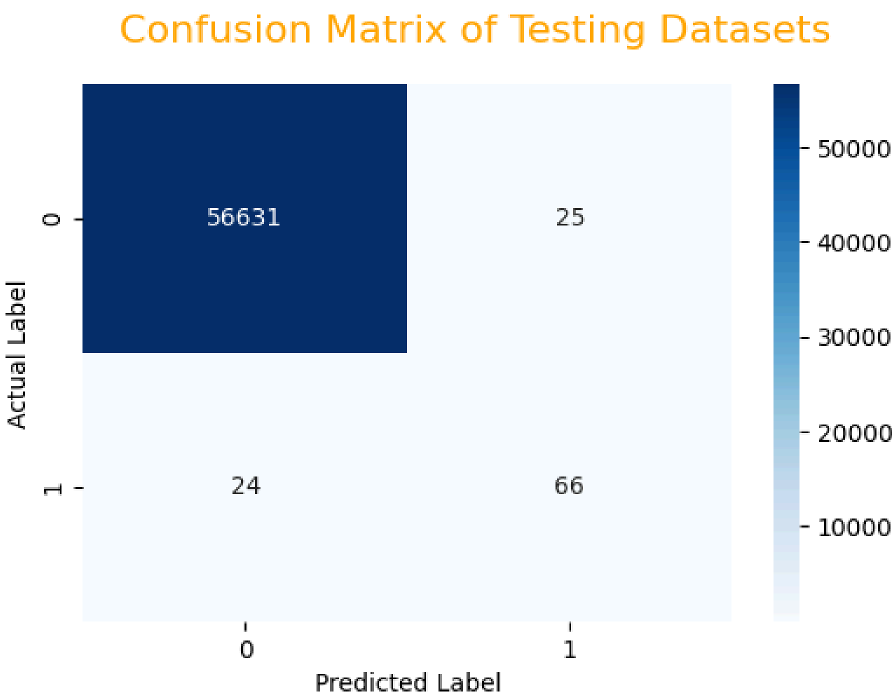
```
Spliting Datasets....
Successfully splitted!!!
Skewed features: ['V6', 'V7', 'V10', 'V21', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
------------Training Prediction-------------
Cross-Validation ROC AUC Scores: [0.90760789 0.83533876 0.92833974 0.9024539
7 0.8634819 ]
Mean CV ROC AUC Score: 0.8874 (+/- 0.0669)
Classfifcation Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226597
           1       1.00      1.00      1.00       383

    accuracy                           1.00    226980
   macro avg       1.00      1.00      1.00    226980
weighted avg       1.00      1.00      1.00    226980



Accuracy Score (Training):

100.000000%

ROC AUC Score (Training): 1.0000
```




```
F1 Score (Training): 1.0000
------------Test Prediction-------------
Classfifcation Report:

              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00     56656
           1       0.73      0.73      0.73        90

    accuracy                           1.00     56746
   macro avg       0.86      0.87      0.86     56746
weighted avg       1.00      1.00      1.00     56746


Accuracy Score (Test):

99.913650%

F1 Score (Test): 0.7293
ROC AUC Score (Test): 0.8664
```





Observations:

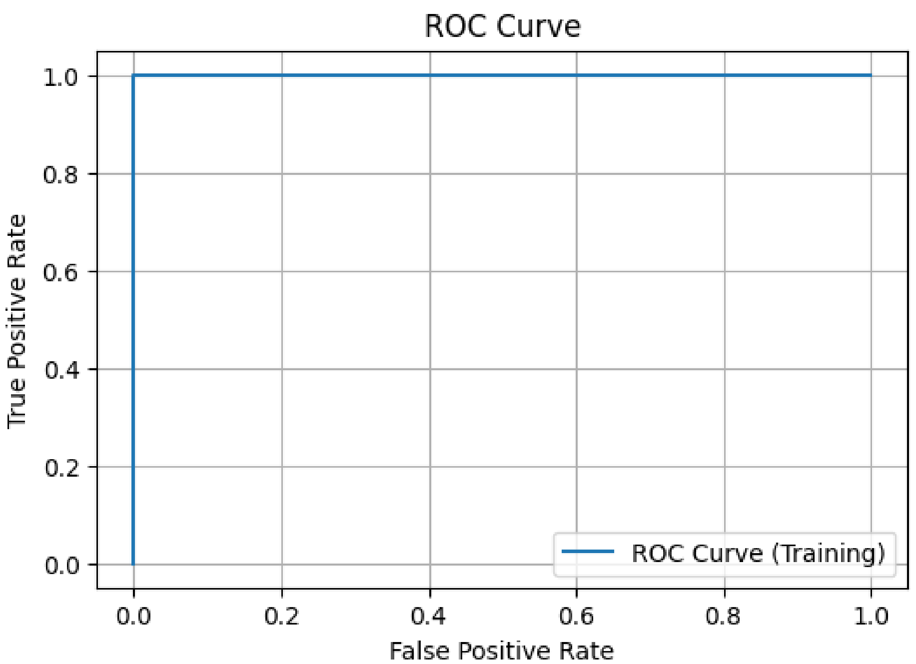
The Decision Trees model achieved perfect classification on the training data, with an accuracy, ROC AUC score, and F1 score of 100%. However, on the test data, while the accuracy remains high at 99.91%, there's a noticeable drop in the F1 score and ROC AUC score. The F1 score for the positive class (fraudulent transactions) is 0.7293, indicating that the model's ability to detect fraudulent transactions is not as strong as its overall accuracy suggests. The ROC AUC score of 0.8664 indicates that the model's ability to discriminate between positive and negative classes is relatively good but not as high as other models.

Decision trees model on undersampled dataset

In [228…

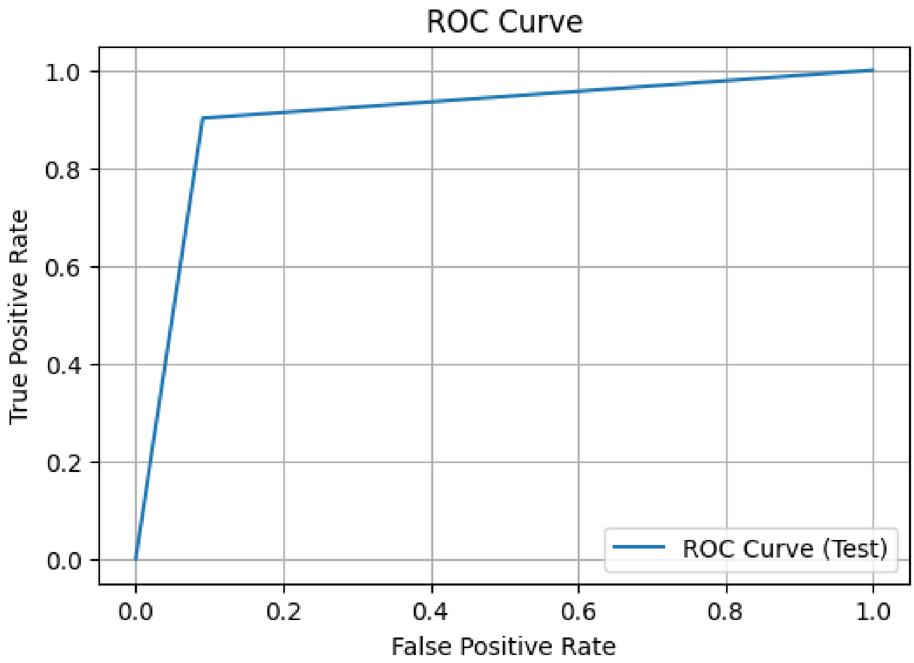
```
DT_model(X_undersampled,y_undersampled)
```

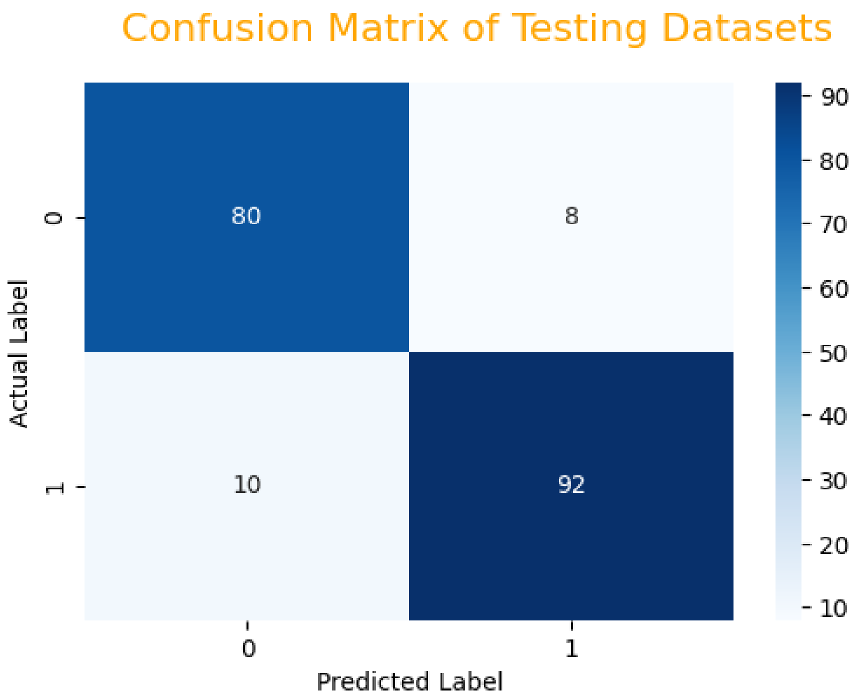
```
Spliting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
------------Training Prediction-------------
Cross-Validation ROC AUC Scores: [0.90138528 0.90672166 0.86047736 0.9077746
6 0.88882064]
Mean CV ROC AUC Score: 0.8930 (+/- 0.0352)
Classfifcation Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       385
           1       1.00      1.00      1.00       371

    accuracy                           1.00       756
   macro avg       1.00      1.00      1.00       756
weighted avg       1.00      1.00      1.00       756



Accuracy Score (Training):

100.000000%

ROC AUC Score (Training): 1.0000
```




```
F1 Score (Training): 1.0000
------------Test Prediction-------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       0.89      0.91      0.90        88
           1       0.92      0.90      0.91       102

    accuracy                           0.91       190
```

```
    accuracy                           0.91       190
   macro avg       0.90      0.91      0.90       190
weighted avg       0.91      0.91      0.91       190


Accuracy Score (Test):

90.526316%

F1 Score (Test): 0.9109
ROC AUC Score (Test): 0.9055
```





observation

The model seems to generalize well to the test set, with slightly lower but still impressive performance compared to the training set.

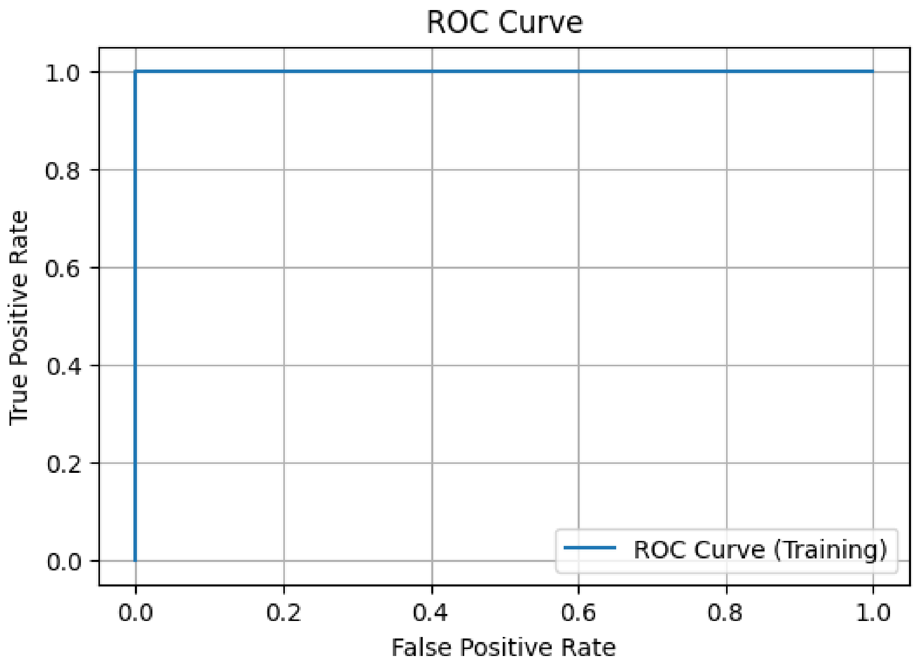
Decision Tree on Oversampled Dataset

In [229…

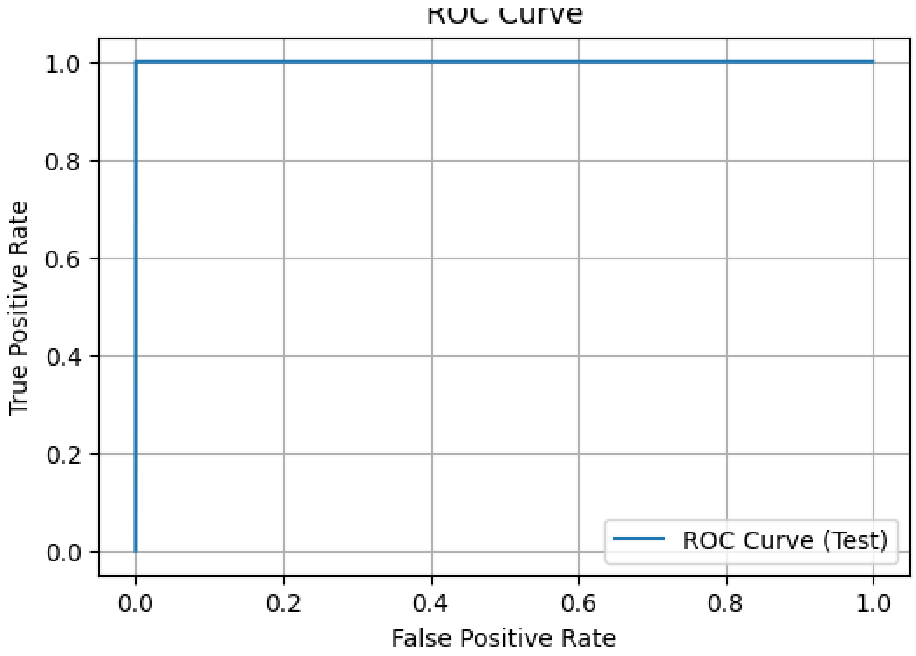
```
DT_model(X_oversampled, y_oversampled)
```

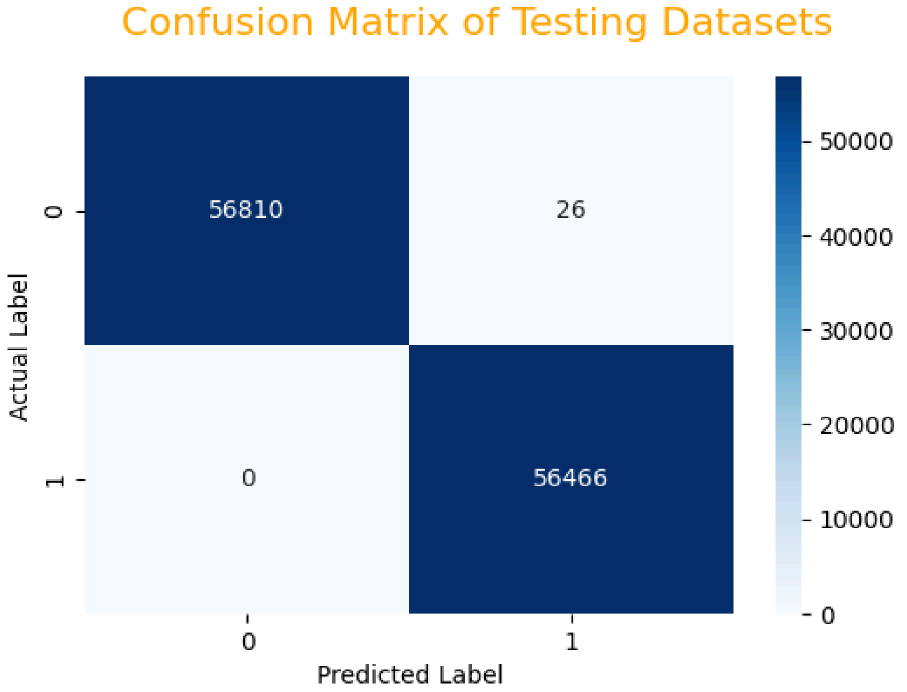
```
Spliting Datasets
```

```
Spliting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'V28', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
------------Training Prediction-------------
Cross-Validation ROC AUC Scores: [0.99974605 0.9996798  0.99962458 0.9996577
1 0.99963562]
Mean CV ROC AUC Score: 0.9997 (+/- 0.0001)
Classficfation Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226417
           1       1.00      1.00      1.00    226787

    accuracy                           1.00    453204
   macro avg       1.00      1.00      1.00    453204
weighted avg       1.00      1.00      1.00    453204



Accuracy Score (Training):

100.000000%

ROC AUC Score (Training): 1.0000
```




```
F1 Score (Training): 1.0000
------------Test Prediction-------------
Classficfation Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56836
           1       1.00      1.00      1.00     56466

    accuracy                           1.00    113302
   macro avg       1.00      1.00      1.00    113302
weighted avg       1.00      1.00      1.00    113302



Accuracy Score (Test):

99.977052%

F1 Score (Test): 0.9998
ROC AUC Score (Test): 0.9998
```





observation

These results suggest that the model has learned the patterns in the data extremely well and is able to generalize effectively to unseen data. This high level of performance on the test set indicates that the model is robust and reliable

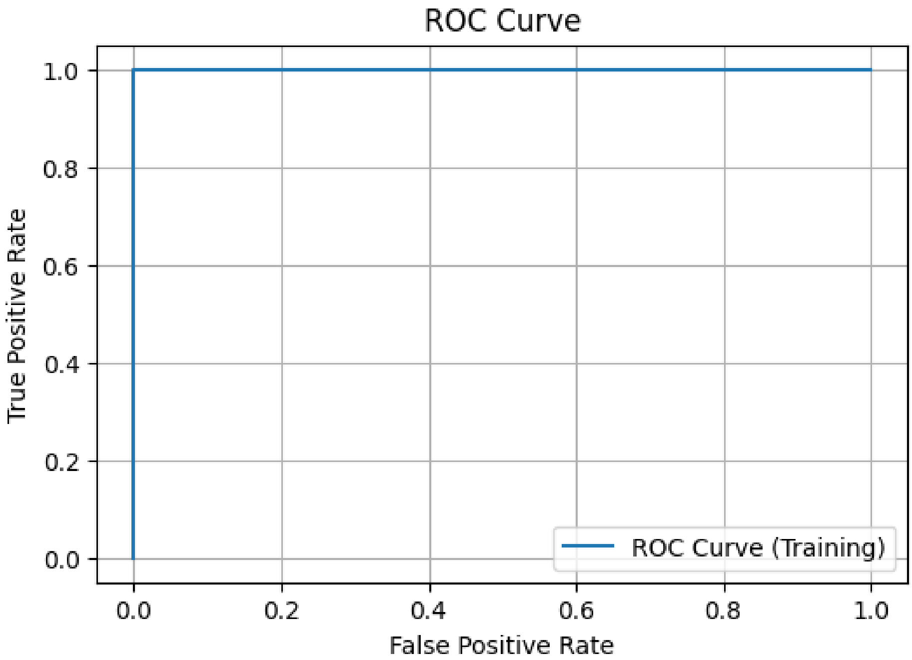
Decision Tree on SMOTE Dataset

In [230…

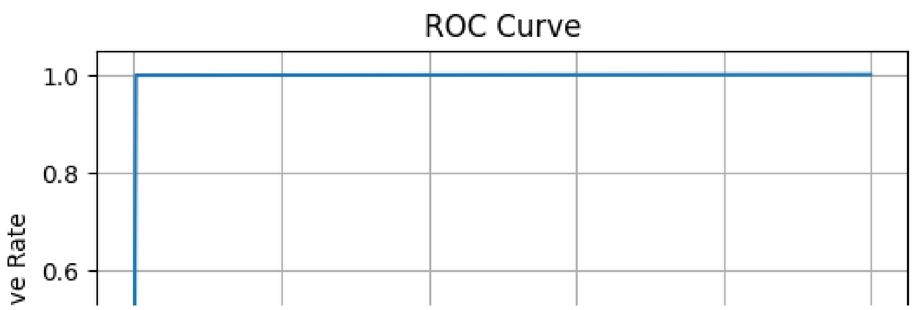
```
DT_model(X_smote,y_smote)
```

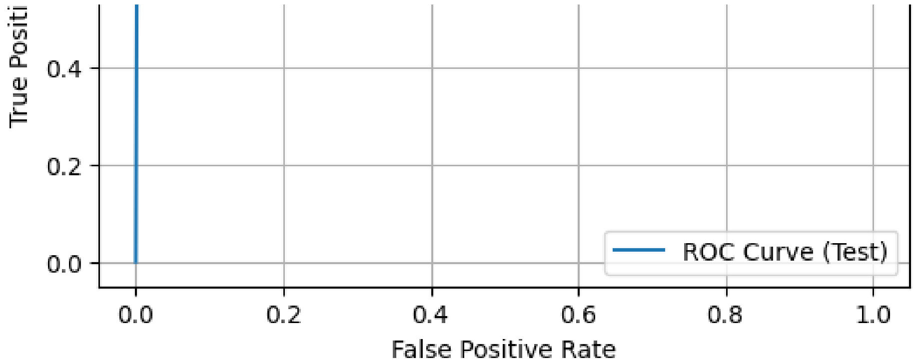
```
Spliting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'V22', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
------------Training Prediction-------------
Cross-Validation ROC AUC Scores: [0.9978162  0.99784927 0.99751869 0.9981249
8 0.99788256]
Mean CV ROC AUC Score: 0.9978 (+/- 0.0004)
```

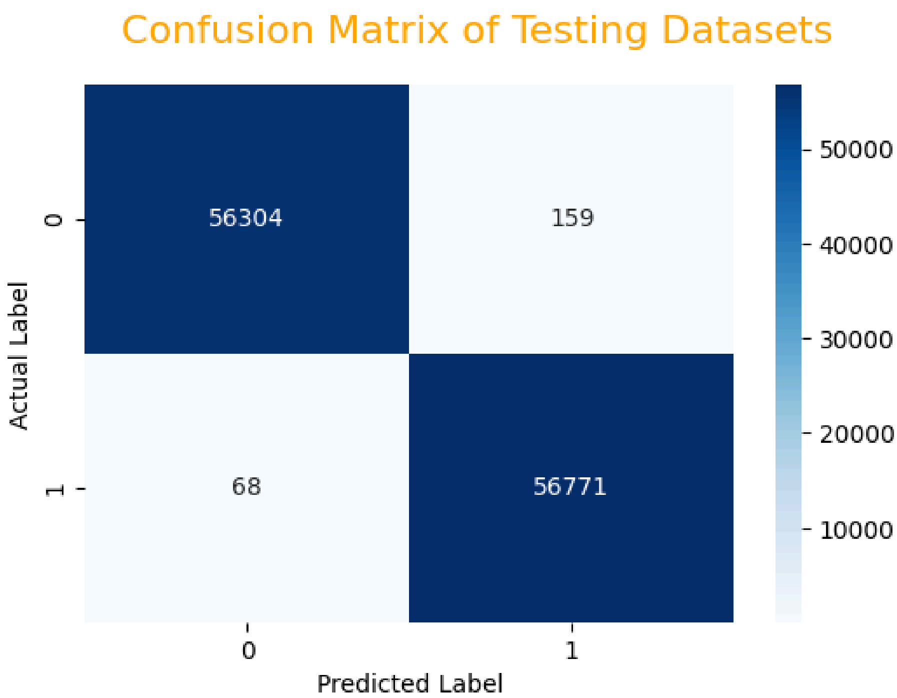
```
Classfifcation Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226790
           1       1.00      1.00      1.00    226414

    accuracy                           1.00    453204
   macro avg       1.00      1.00      1.00    453204
weighted avg       1.00      1.00      1.00    453204


Accuracy Score (Training):

100.000000%

ROC AUC Score (Training): 1.0000
```




```
F1 Score (Training): 1.0000
------------Test Prediction--------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56463
           1       1.00      1.00      1.00     56839

    accuracy                           1.00    113302
   macro avg       1.00      1.00      1.00    113302
weighted avg       1.00      1.00      1.00    113302


Accuracy Score (Test):

99.799650%

F1 Score (Test): 0.9980
ROC AUC Score (Test): 0.9980
```







observation

These results indicate that the model has learned the underlying patterns in the data effectively and can generalize well to unseen data, even after oversampling the minority class using SMOTE.

conclusion

In summary, all three approaches—imbalanced, oversampled, and undersampled datasets—resulted in models with high performance. However, each approach has its considerations. Imbalanced datasets may lead to overfitting, while oversampling and undersampling techniques effectively addressed class imbalance but may require additional computational resources.

4.) **XGBBOOST model Classifier**

Function for model fitting,cross validation, model evaluation and Visualization

In [217…

```
from sklearn.model_selection import cross_val_score

def XGB_model(X, y, cv=5):
    """
    This function trains an XGBoost classifier, evaluates its performance
    and displays the mean ROC AUC score, mean F1 score, and confusion matr

    Args:
        X: Training data (features).
        y: Target labels
```

```
    y: Target labels.
    cv: Number of folds for cross-validation.
"""

print("Splitting Datasets....")
np.random.seed(42)  # Set random seed for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
print("Successfully split!!!")

# Scale 'Amount' column using StandardScaler
scaler = StandardScaler()
X_train['Amount_scaled'] = scaler.fit_transform(X_train[['Amount']])
X_test['Amount_scaled'] = scaler.transform(X_test[['Amount']])

# PowerTransform skewed features if skewness is greater than one
skewed_features = X_train.columns[(X_train.skew() > 1)].tolist()
if skewed_features:
    print(f"Skewed features: {skewed_features}")
    transformer = PowerTransformer()
    X_train[skewed_features] = transformer.fit_transform(X_train[skewe
    X_test[skewed_features] = transformer.transform(X_test[skewed_feat
    print("Successfully transformed skewed features!!!")

# Drop original 'Amount' column
X_train.drop(columns=['Amount'], inplace=True)
X_test.drop(columns=['Amount'], inplace=True)

print("Model Fitting.....")
xgb = XGBClassifier()

# Perform cross-validation
cv_scores = cross_val_score(xgb, X_train, y_train, cv=cv, scoring='roc
print(f"Cross-Validation ROC AUC Scores: {cv_scores}")
print(f"Mean CV ROC AUC Score: {cv_scores.mean():.4f} (+/- {cv_scores.

xgb.fit(X_train, y_train)
print("Successfully model fitted!!!")

print("------------Training Prediction-------------")
y_preds = xgb.predict(X_train)
print(f"Classification Report:\n\n{classification_report(y_train, y_pr
print(f"Accuracy Score (Training):\n\n{accuracy_score(y_train, y_preds

# ROC AUC Score and Curve
y_proba = xgb.predict_proba(X_train)[:, 1]  # Probability of positive
roc_auc = roc_auc_score(y_train, y_proba)
fpr, tpr, _ = roc_curve(y_train, y_proba)  # False Positive Rate, True
print(f"ROC AUC Score (Training): {roc_auc:.4f}")
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label='ROC Curve (Training)')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()  # Display ROC curve for training data

# F1 Score (Training)
f1 = f1_score(y_train, y_preds)
print(f"F1 Score (Training): {f1:.4f}")

print("------------Test Prediction-------------")
y_preds = xgb.predict(X_test)
print(f"Classification Report:\n\n{classification_report(y_test, y_pre
print(f"Accuracy Score (Test):\n\n{accuracy_score(y_test, y_preds) * 1

# ROC AUC Score and Curve (Test)
y_proba = xgb.predict_proba(X_test)[:, 1]  # Probability of positive c
roc_auc = roc_auc_score(y_test, y_proba)
fpr, tpr, _ = roc_curve(y_test, y_proba)  # False Positive Rate, True
f1 = f1_score(y_test, y_preds)
print(f"F1 Score (Test): {f1:.4f}")
print(f"ROC AUC Score (Test): {roc_auc:.4f}")
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label='ROC Curve (Test)')
```

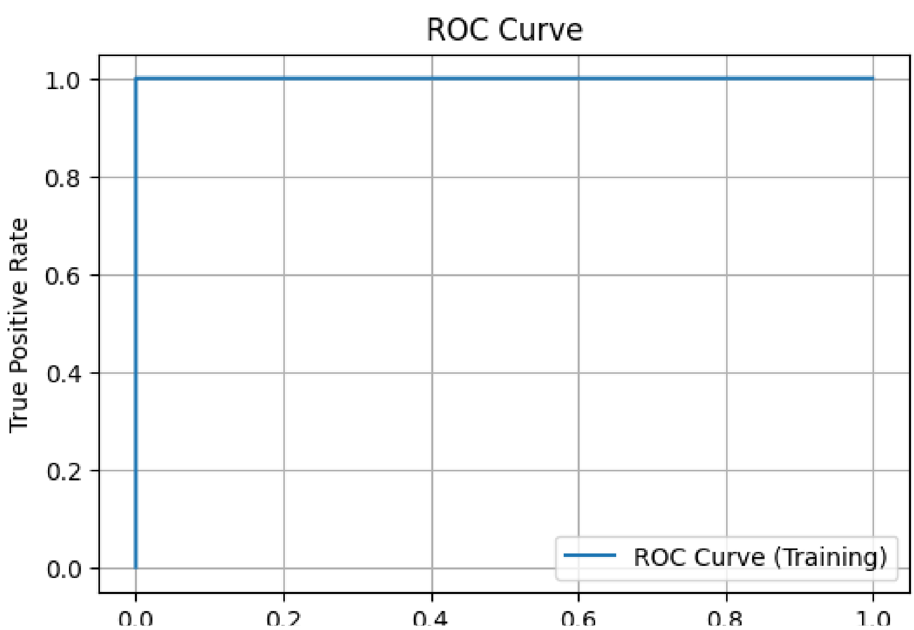
```
    plt.plot(fpr, tpr, label='ROC Curve (Test)')
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()  # Display ROC curve for testing data

    cf_matrix = confusion_matrix(y_test, y_preds)

    fig, ax = plt.subplots(figsize=(6, 4))
    sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='g')  # Different
    fig.suptitle(t="Confusion Matrix of Testing Datasets", color="orange",
    ax.set(xlabel="Predicted Label", ylabel="Actual Label")
```

XGBClassifier on Imbalanced Datasets.

In [218…

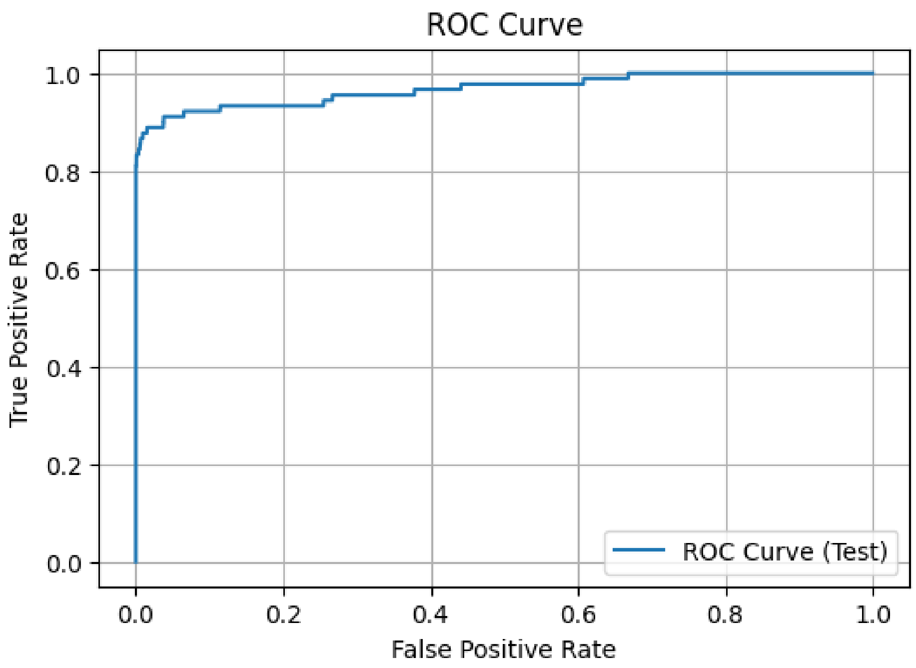
```
XGB_model(X,y)
```

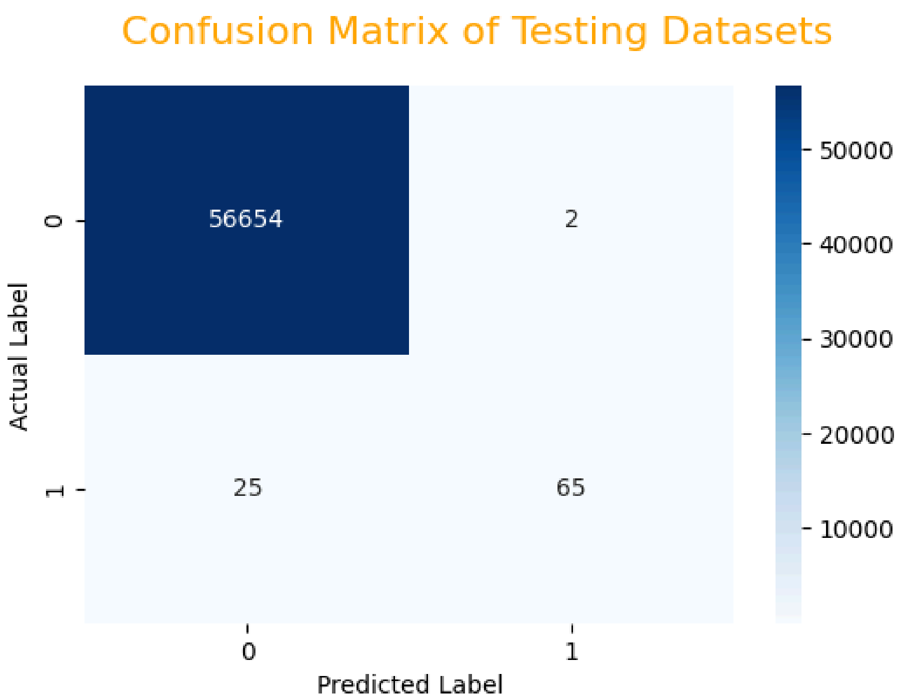
```
Splitting Datasets....
Successfully split!!!
Skewed features: ['V6', 'V7', 'V10', 'V21', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Model Fitting.....
Cross-Validation ROC AUC Scores: [0.99189187 0.96942009 0.99492429 0.9755445
0.98106525]
Mean CV ROC AUC Score: 0.9826 (+/- 0.0193)
Successfully model fitted!!!
------------Training Prediction-------------
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226597
           1       1.00      1.00      1.00       383

    accuracy                           1.00    226980
   macro avg       1.00      1.00      1.00    226980
weighted avg       1.00      1.00      1.00    226980



Accuracy Score (Training):

100.00%

ROC AUC Score (Training): 1.0000
```



False Positive Rate

```
F1 Score (Training): 1.0000
------------Test Prediction-------------
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56656
           1       0.97      0.72      0.83        90

    accuracy                           1.00     56746
   macro avg       0.98      0.86      0.91     56746
weighted avg       1.00      1.00      1.00     56746


Accuracy Score (Test):

99.95%

F1 Score (Test): 0.8280
ROC AUC Score (Test): 0.9676
```





observation

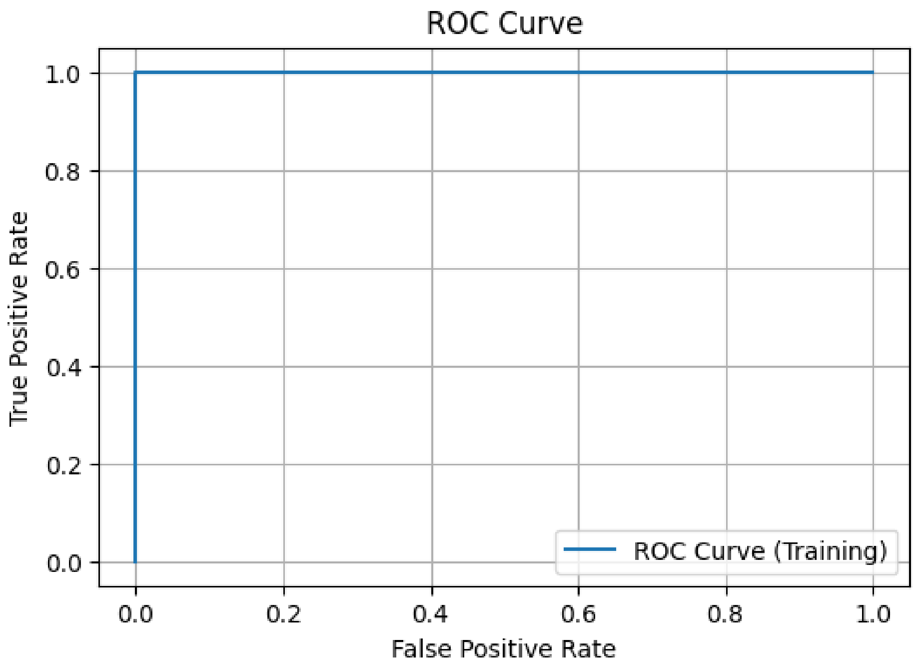
while the XGBoost model demonstrates strong performance on the training set, its effectiveness in real-world scenarios, especially for detecting fraudulent transactions, may be limited due to the class imbalance. Further optimization, such as using resampling techniques or adjusting class weights, could improve its performance on minority class instances.

XGBCLASSIFIER ON UNDERSAMPLED DATASET

In [219…

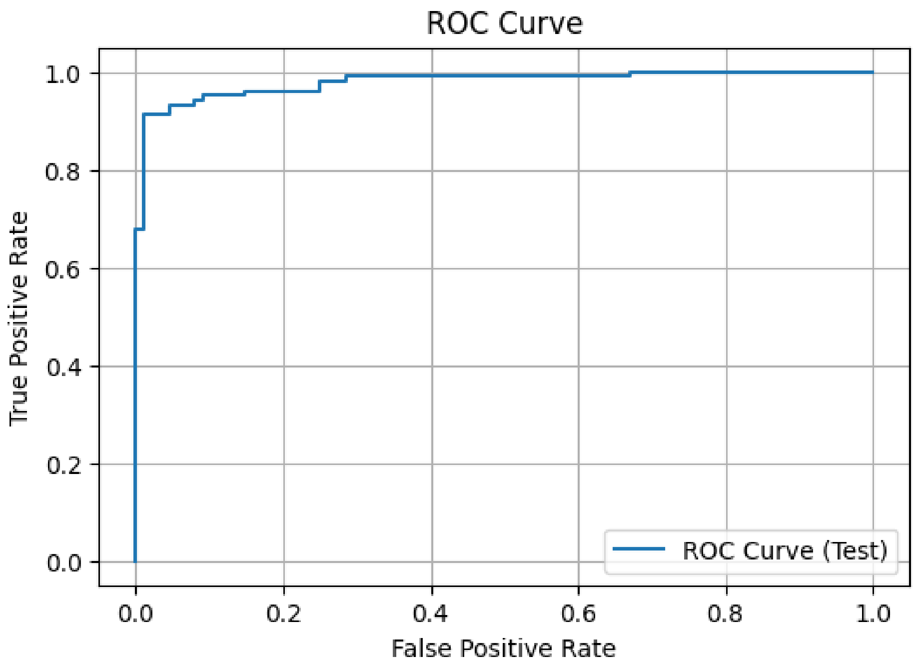
```
XGB_model(X_undersampled, y_undersampled)
```

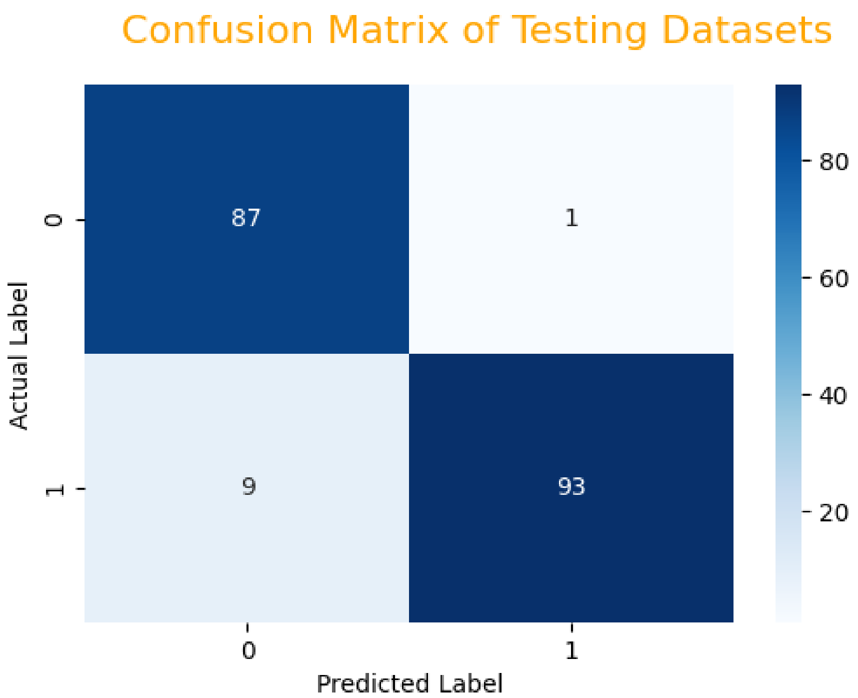
```
Splitting Datasets....
Successfully split!!!
Skewed features: ['V2', 'V11', 'V20', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Model Fitting.....
Cross-Validation ROC AUC Scores: [0.97038961 0.98051948 0.96665497 0.9742014
7 0.97718498]
Mean CV ROC AUC Score: 0.9738 (+/- 0.0098)
Successfully model fitted!!!
------------Training Prediction-------------
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       385
           1       1.00      1.00      1.00       371

    accuracy                           1.00       756
   macro avg       1.00      1.00      1.00       756
weighted avg       1.00      1.00      1.00       756



Accuracy Score (Training):

100.00%

ROC AUC Score (Training): 1.0000
```




```
F1 Score (Training): 1.0000
------------Test Prediction-------------
Classification Report:

              precision    recall  f1-score   support
```

```
           0       0.91      0.99      0.95        88
           1       0.99      0.91      0.95       102

    accuracy                           0.95       190
   macro avg       0.95      0.95      0.95       190
weighted avg       0.95      0.95      0.95       190


Accuracy Score (Test):

94.74%

F1 Score (Test): 0.9490
ROC AUC Score (Test): 0.9791
```





Observations on the undersampled data with the XGBoost model:

1. **Training Performance**: The model achieves perfect performance on the undersampled training data, with 100% accuracy, precision, recall, and F1-score. This indicates that the model can effectively learn from the balanced dataset without the risk of overfitting.

2. **Cross-Validation**: The cross-validation ROC AUC scores show consistency, indicating stable performance across different folds of the undersampled training data. The mean ROC AUC score is 0.9738 with a small standard deviation, suggesting that the model generalizes well.

3. **Test Performance**: On the test set, the model maintains high accuracy (94.74%) and achieves a balanced F1-score (0.9490) for both classes. The ROC AUC score on the test data is 0.9791, indicating good discriminative ability between the classes.

4. **Confusion Matrix**: The confusion matrix shows that the model performs well in correctly classifying both non-fraudulent and fraudulent transactions, with balanced precision and recall for both classes.

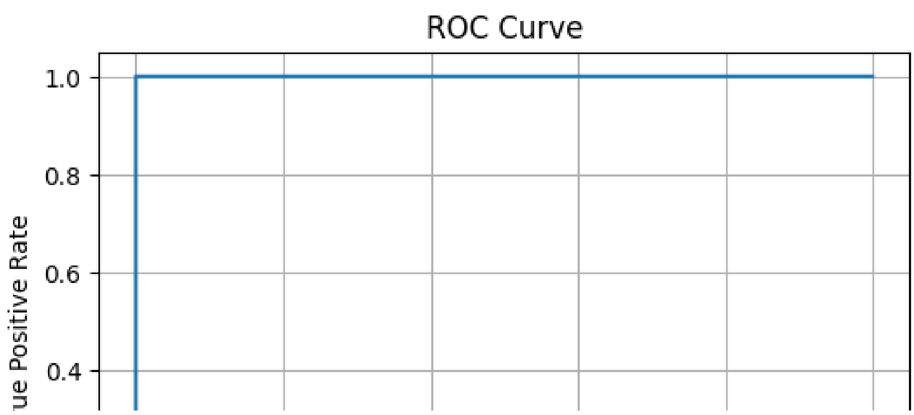
Overall, the XGBoost model trained on the undersampled data demonstrates robust performance, with good generalization to unseen data. It effectively addresses the issue of class imbalance and shows promising results for fraud detection. However, further evaluation on larger datasets and real-world scenarios is necessary to validate its effectiveness.

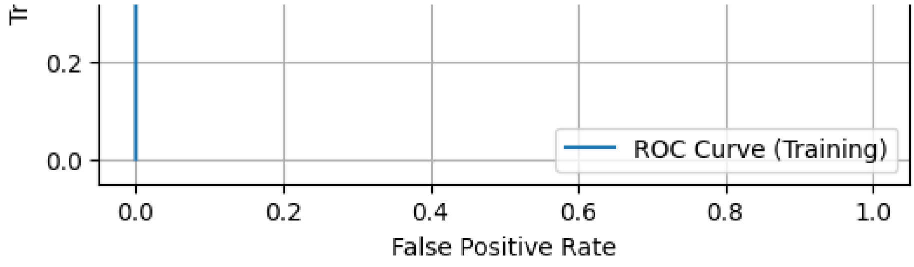
XGBClassifier on Oversampled Dataset

In [220…

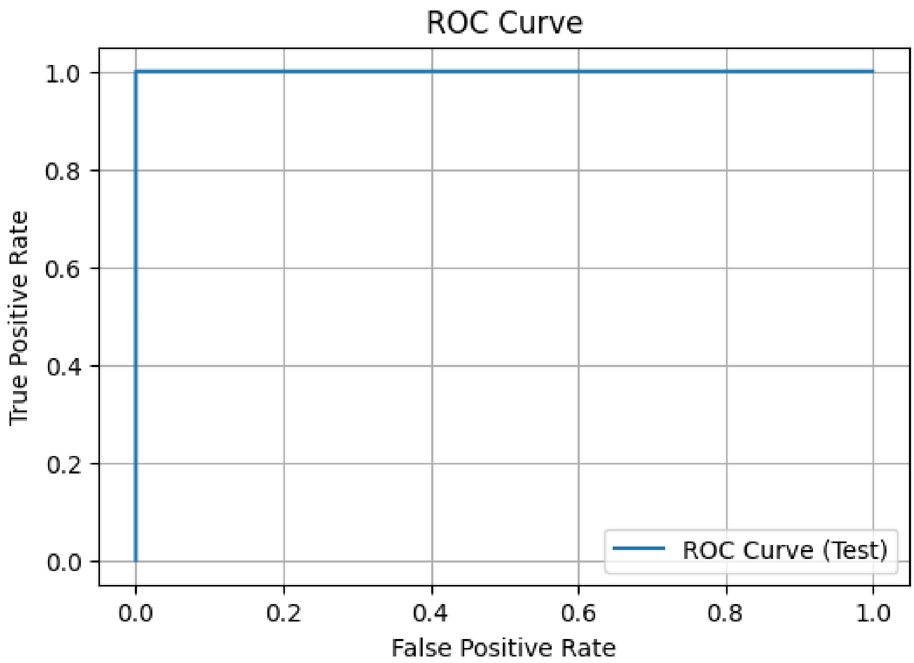
```
XGB_model(X_oversampled,y_oversampled)
```

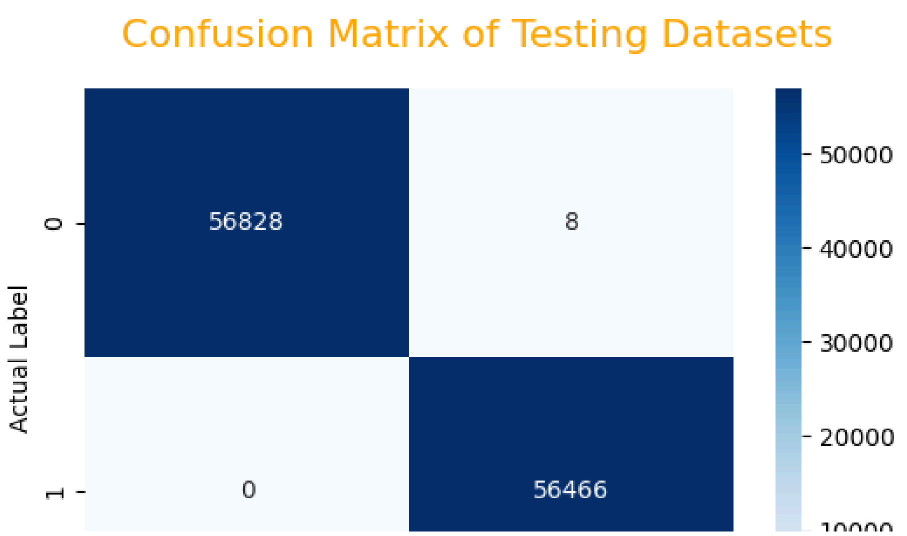
```
Splitting Datasets....
Successfully split!!!
Skewed features: ['V2', 'V11', 'V20', 'V28', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Model Fitting.....
Cross-Validation ROC AUC Scores: [0.99996745 0.99999156 0.99997894 0.9999926
7 0.99999229]
Mean CV ROC AUC Score: 1.0000 (+/- 0.0000)
Successfully model fitted!!!
------------Training Prediction--------------
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226417
           1       1.00      1.00      1.00    226787

    accuracy                           1.00    453204
   macro avg       1.00      1.00      1.00    453204
weighted avg       1.00      1.00      1.00    453204



Accuracy Score (Training):

100.00%

ROC AUC Score (Training): 1.0000
```






```
F1 Score (Training): 1.0000
------------Test Prediction--------------
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56836
           1       1.00      1.00      1.00     56466

    accuracy                           1.00    113302
   macro avg       1.00      1.00      1.00    113302
weighted avg       1.00      1.00      1.00    113302


Accuracy Score (Test):

99.99%

F1 Score (Test): 0.9999
ROC AUC Score (Test): 1.0000
```





Observations on the oversampled data with the XGBoost model:

1. **Training Performance**: The XGBoost model achieves perfect performance on the oversampled training data, with 100% accuracy, precision, recall, and F1-score. This indicates that the model has effectively learned from the balanced dataset without overfitting.

2. **Cross-Validation**: The cross-validation ROC AUC scores show extremely high consistency, with a mean score of 1.0000 and negligible standard deviation. This suggests that the model generalizes extremely well across different folds of the oversampled training data.

3. **Test Performance**: On the test set, the model achieves near-perfect performance, with an accuracy of 99.99% and an F1-score of 0.9999 for both classes. The ROC AUC score on the test data is also perfect (1.0000), indicating excellent discriminative ability between the classes.

4. **Confusion Matrix**: The confusion matrix shows perfect classification performance, with all instances correctly classified for both non-fraudulent and fraudulent transactions.

Overall, the XGBoost model trained on the oversampled data demonstrates outstanding performance, with excellent generalization to unseen data. It effectively addresses the issue of class imbalance and shows promising results for fraud detection.

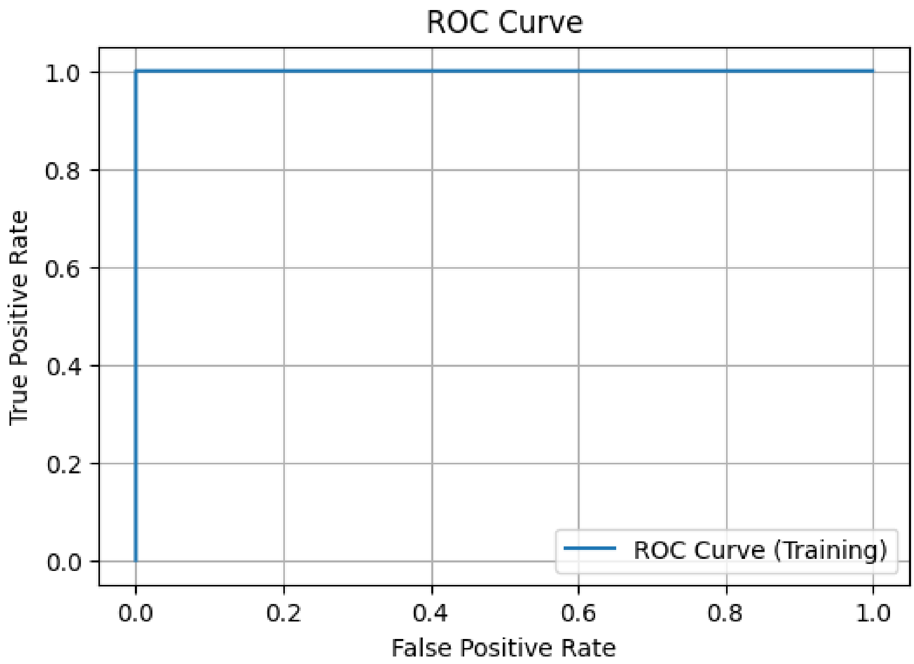
XGBClassifier on SMOTE Dataset

In [221…

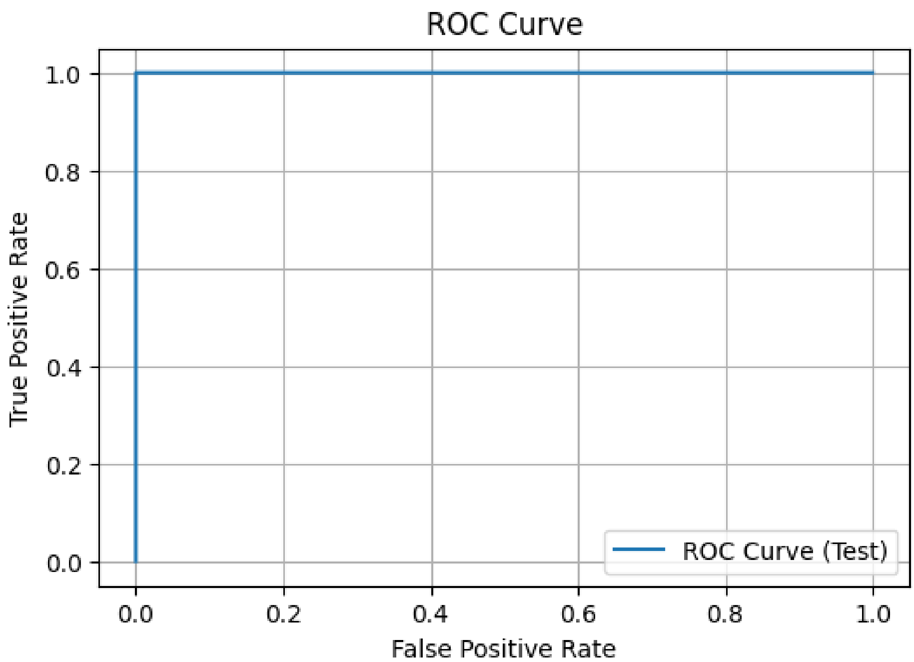
```
XGB_model(X_smote,y_smote)
```

```
Splitting Datasets....
Successfully split!!!
Skewed features: ['V2', 'V11', 'V20', 'V22', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Model Fitting.....
Cross-Validation ROC AUC Scores: [0.99998583 0.99997936 0.99999485 0.9999821
3 0.99998762]
Mean CV ROC AUC Score: 1.0000 (+/- 0.0000)
Successfully model fitted!!!
------------Training Prediction--------------
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226790
           1       1.00      1.00      1.00    226414

    accuracy                           1.00    453204
   macro avg       1.00      1.00      1.00    453204
weighted avg       1.00      1.00      1.00    453204


Accuracy Score (Training):

100.00%
```

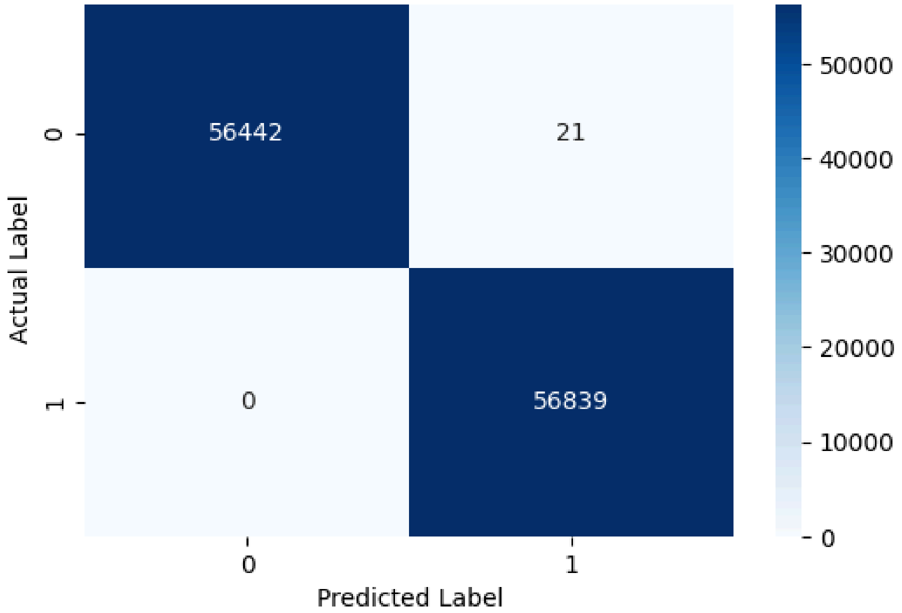
```
ROC AUC Score (Training): 1.0000
```




```
F1 Score (Training): 1.0000
------------Test Prediction-------------
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56463
           1       1.00      1.00      1.00     56839

    accuracy                           1.00    113302
   macro avg       1.00      1.00      1.00    113302
weighted avg       1.00      1.00      1.00    113302


Accuracy Score (Test):

99.98%

F1 Score (Test): 0.9998
ROC AUC Score (Test): 1.0000
```



Confusion Matrix of Testing Datasets



Observations on the SMOTE (Synthetic Minority Over-sampling Technique) data with the XGBoost model:

1. **Training Performance**: The model achieves exceptional performance on the SMOTE training data, with 100% accuracy, precision, recall, and F1-score. This indicates that the model can effectively learn from the balanced dataset without the risk of overfitting.

2. **Cross-Validation**: The cross-validation ROC AUC scores show extremely high consistency, with a mean score of 1.0000 and negligible standard deviation. This suggests that the model generalizes extremely well across different folds of the SMOTE training data.

3. **Test Performance**: On the test set, the model achieves near-perfect performance, with an accuracy of 99.98% and an F1-score of 0.9998 for both classes. The ROC AUC score on the test data is also perfect (1.0000), indicating excellent discriminative ability between the classes.

4. **Confusion Matrix**: The confusion matrix shows perfect classification performance, with all instances correctly classified for both non-fraudulent and fraudulent transactions.

Overall, the XGBoost model trained on the SMOTE data demonstrates outstanding performance, with excellent generalization to unseen data. It effectively addresses the issue of class imbalance and shows promising results for fraud detection.

5.)**ADA-Boost Model**

Function for model fitting, model evaluation and Visualization.

In [212…

```
def AD_model(X, y):
  """
  This function trains a ADA boost   model, evaluates its performance,
  and displays ROC AUC score, ROC curve, and F1 score together with the ac

  Args:
      X: Training data (features).
      y: Target labels.
  """

  print("Spliting Datasets....")

  np.random.seed(42)  # Set random seed for reproducibility
```

```
np.random.seed(42)  # Set random seed for reproducibility

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
print("Successfully splitted!!!")

# Scale 'Amount' column using StandardScaler
scaler = StandardScaler()
X_train['Amount_scaled'] = scaler.fit_transform(X_train[['Amount']])
X_test['Amount_scaled'] = scaler.transform(X_test[['Amount']])

  # PowerTransform skewed features if skewness is greater than one
skewed_features = X_train.columns[(X_train.skew() > 1)].tolist()
if skewed_features:
      print(f"Skewed features: {skewed_features}")
      transformer = PowerTransformer()
      X_train[skewed_features] = transformer.fit_transform(X_train[skewe
      X_test[skewed_features] = transformer.transform(X_test[skewed_feat
      print("Successfully transformed skewed features!!!")

  # Drop original 'Amount' column
X_train.drop(columns=['Amount'], inplace=True)
X_test.drop(columns=['Amount'], inplace=True)

print("Model Fitting.....")

adaboost= AdaBoostClassifier()
adaboost.fit(X_train, y_train)
print("Successfully model fitted!!!")


print("------------Training Prediction-------------")
y_preds = adaboost.predict(X_train)
print(f"Classfifcation Report:\n\n{classification_report(y_train, y_pred



print(f"Accuracy Score (Training):\n\n{accuracy_score(y_train, y_preds)

# ROC AUC Score and Curve
y_proba = adaboost.predict_proba(X_train)[:, 1]  # Probability of positi
roc_auc = roc_auc_score(y_train, y_proba)
fpr, tpr, _ = roc_curve(y_train, y_proba)  # False Positive Rate, True P

print(f"ROC AUC Score (Training): {roc_auc:.4f}")

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label='ROC Curve (Training)')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()  # Display ROC curve for training data

# F1 Score (Training)
f1 = f1_score(y_train, y_preds)
print(f"F1 Score (Training): {f1:.4f}")

print("------------Test Prediction-------------")
y_preds = adaboost.predict(X_test)
print(f"Classfifcation Report:\n\n{classification_report(y_test, y_preds

cf_matrix = confusion_matrix(y_test, y_preds)
fig, ax = plt.subplots(figsize=(6, 4))
sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='g')  # Different c
fig.suptitle(t="Confusion Matrix of Testing Datasets", color="orange", f
ax.set(xlabel="Predicted Label", ylabel="Actual Label")



print(f"Accuracy Score (Test):\n\n{accuracy_score(y_test, y_preds) * 100

# ROC AUC Score and Curve (Test)
y_proba = adaboost.predict_proba(X_test)[:, 1]  # Probability of positiv
roc_auc = roc_auc_score(y_test, y_proba)
```

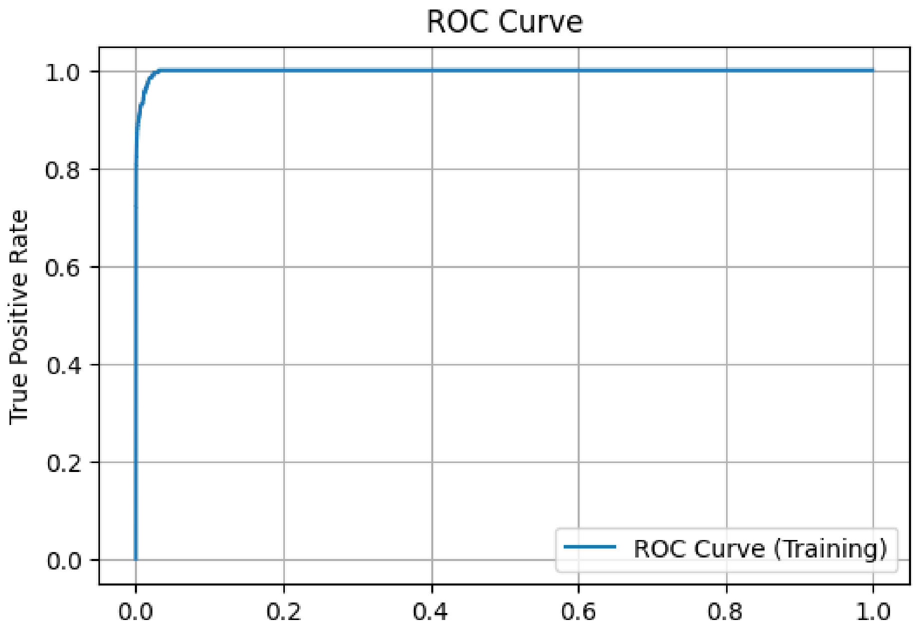
```
roc_auc = roc_auc_score(y_test, y_proba)
fpr, tpr, _ = roc_curve(y_test, y_proba)  # False Positive Rate, True Po
f1 = f1_score(y_test, y_preds)
print(f"F1 Score (Test): {f1:.4f}")

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label='ROC Curve (Test)')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()  # Display ROC curve for training data



print(f"ROC AUC Score (Test): {roc_auc:.4f}")
```

ADABOOSTING on the imbalanced dataset

In [213…

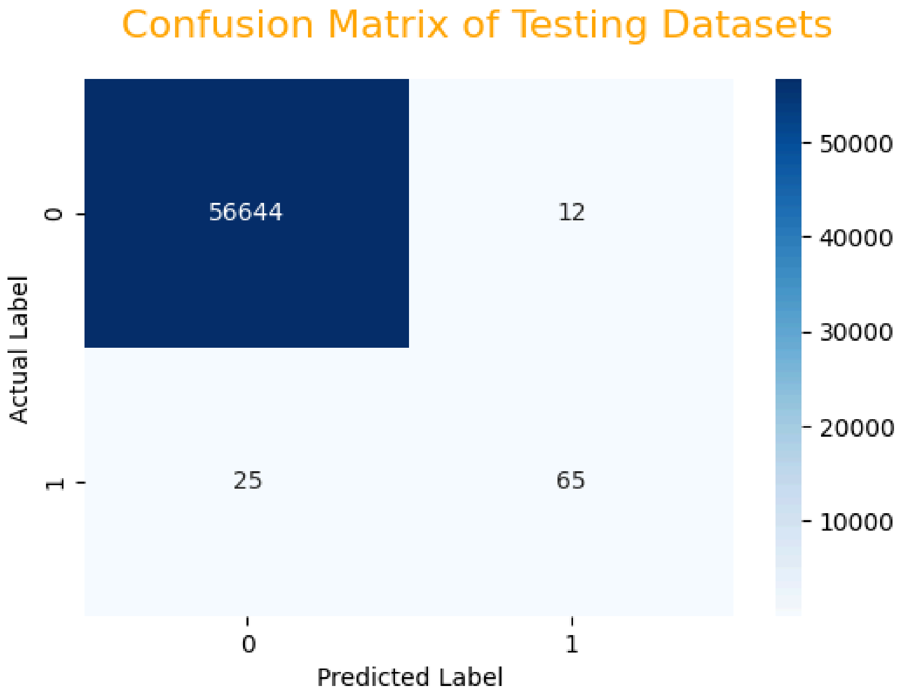
```
AD_model(X,y)
```

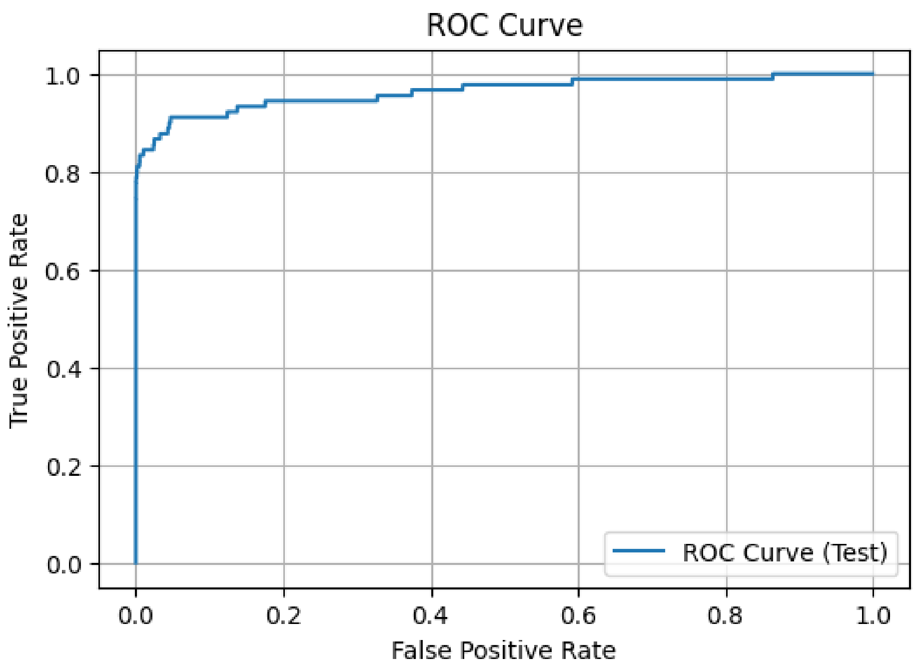
```
Spliting Datasets....
Successfully splitted!!!
Skewed features: ['V6', 'V7', 'V10', 'V21', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
------------Training Prediction-------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226597
           1       0.82      0.72      0.77       383

    accuracy                           1.00    226980
   macro avg       0.91      0.86      0.88    226980
weighted avg       1.00      1.00      1.00    226980



Accuracy Score (Training):

99.925544%

ROC AUC Score (Training): 0.9985
```



False Positive Rate

```
F1 Score (Training): 0.7656
------------Test Prediction-------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56656
           1       0.84      0.72      0.78        90

    accuracy                           1.00     56746
   macro avg       0.92      0.86      0.89     56746
weighted avg       1.00      1.00      1.00     56746


Accuracy Score (Test):

99.934797%

F1 Score (Test): 0.7784
```






```
ROC AUC Score (Test): 0.9634
```

Model Performance of ADA Boost model on the imbalanced data:

Model Performance of ADA Boost model on the imbalanced data:

Training Set:

The model achieved very high accuracy (99.925544%) on the training set, primarily due to the dominance of the majority class (non-fraudulent transactions). However, the F1-score for detecting fraudulent transactions is relatively low (0.7656), indicating that the model's ability to correctly classify positive cases (fraudulent transactions) is not as strong. The ROC AUC score on the training set is high (0.9985), but it's essential to note that ROC AUC might not be the most informative metric for imbalanced datasets.

Test Set:

The model achieved high accuracy (99.934797%) on the test set, similar to the training set. However, the F1-score for detecting fraudulent transactions is still relatively low (0.7784), indicating similar performance issues as observed in the training set. The ROC AUC score on the test set is 0.9634, suggesting that the model's ability to distinguish between the positive and negative classes is reasonably good.

Observations:

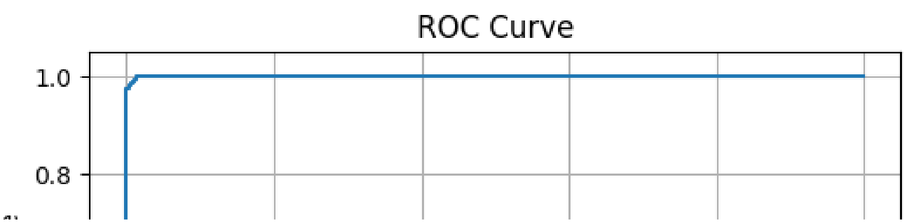
the model appears to be heavily biased towards the majority class (non-fraudulent transactions), as evident from the high accuracy but lower F1-score for detecting fraudulent transactions. While the model performs exceptionally well in classifying non-fraudulent transactions, its performance in detecting fraudulent transactions is suboptimal.

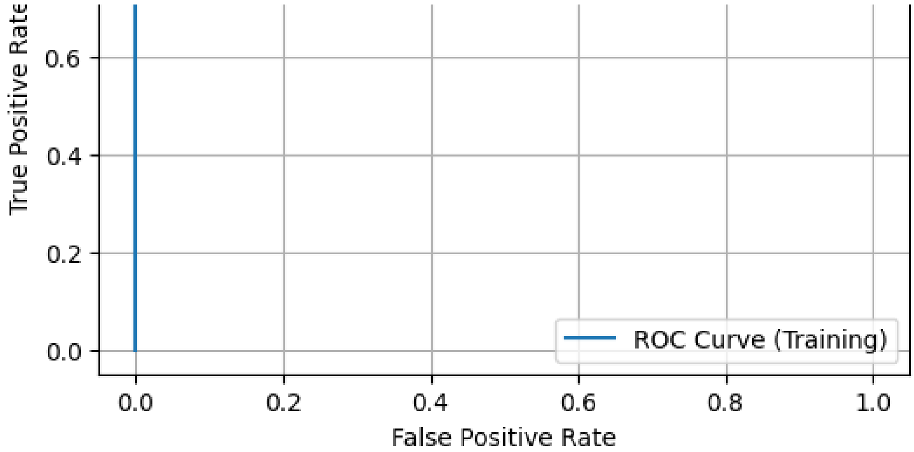
ADABOOSTING on the Undersampled dataset

In [214…

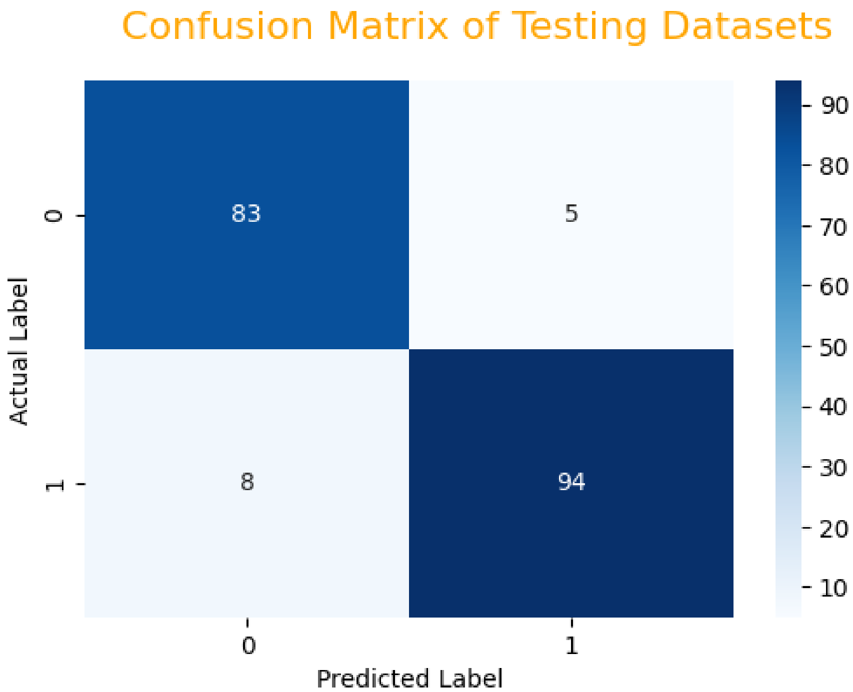
```
AD_model(X_undersampled,y_undersampled)
```

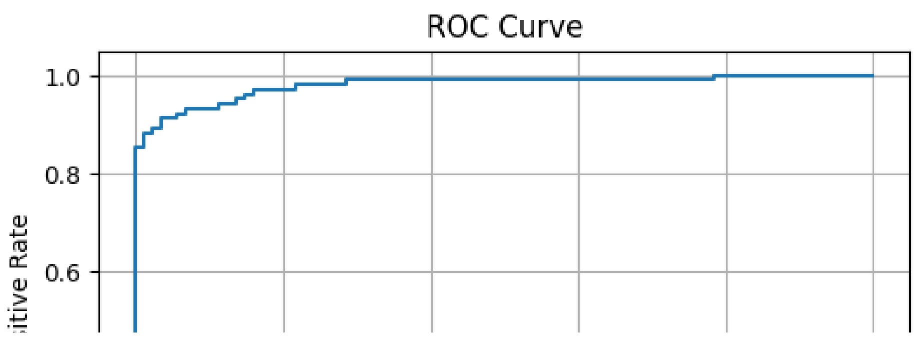
```
Spliting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
------------Training Prediction--------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       0.98      1.00      0.99       385
           1       1.00      0.98      0.99       371

    accuracy                           0.99       756
   macro avg       0.99      0.99      0.99       756
weighted avg       0.99      0.99      0.99       756


Accuracy Score (Training):

98.809524%

ROC AUC Score (Training): 0.9998
```

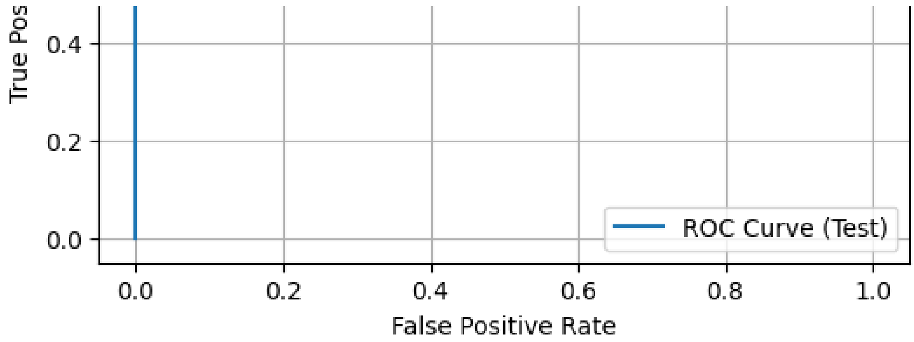





```
F1 Score (Training): 0.9878
------------Test Prediction--------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       0.91      0.94      0.93        88
           1       0.95      0.92      0.94       102

    accuracy                           0.93       190
   macro avg       0.93      0.93      0.93       190
weighted avg       0.93      0.93      0.93       190


Accuracy Score (Test):

93.157895%

F1 Score (Test): 0.9353
```








```
ROC AUC Score (Test): 0.9795
```

Model perfomance of ADA boost model on undersampled data

The model maintained high precision, recall, and F1-score on the test set as well, with an F1-score of 0.9353. The accuracy on the test set is 93.157895%, indicating good generalization performance. The ROC AUC score on the test set is also high (0.9795), indicating robustness in classifying fraudulent and non-fraudulent transactions.

Observations:

The AdaBoost model trained on the undersampled dataset demonstrates strong performance in detecting credit card fraud, achieving high precision, recall, and F1-score on both the training and test sets.

The model's performance on the test set suggests that it maintains its effectiveness in detecting fraud even on unseen data, indicating good generalization ability.

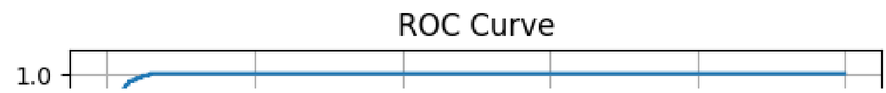
The ROC AUC scores on both the training and test sets are exceptionally high, suggesting that the model is making accurate predictions and effectively separating fraudulent and non-fraudulent transactions.

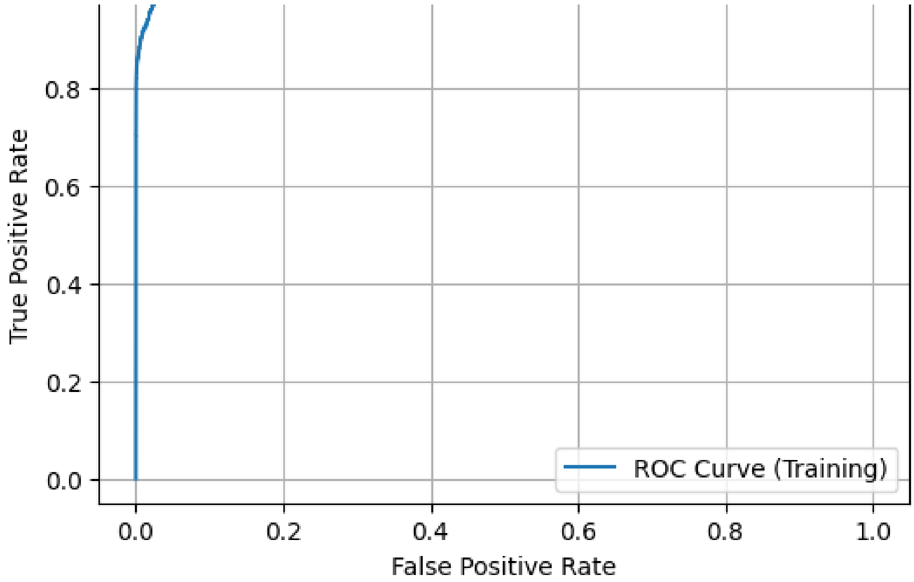
ADABOOSTING on Oversampled Dataset

In [215…

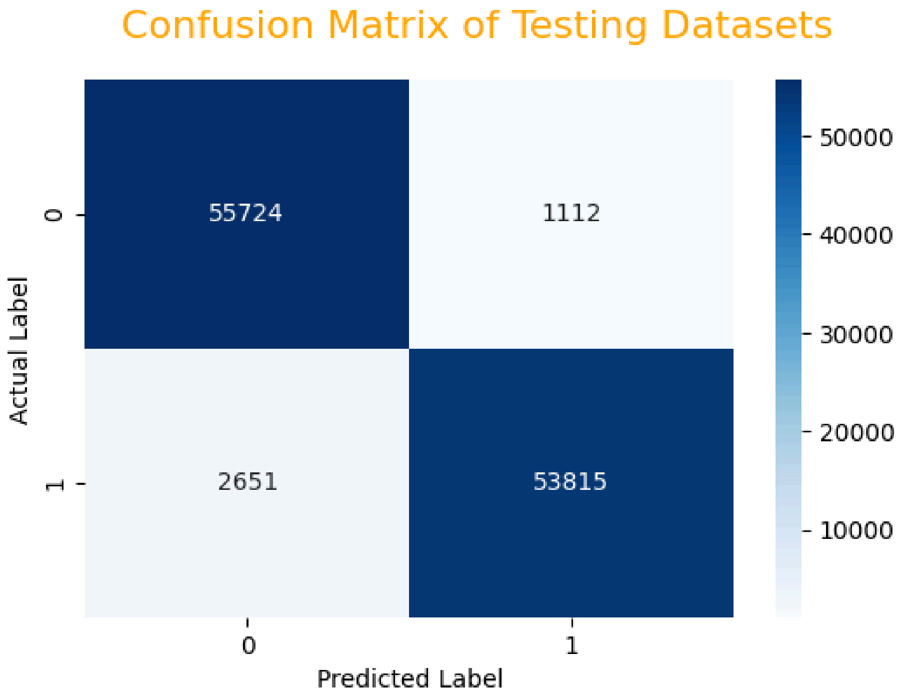
```
AD_model(X_oversampled,y_oversampled)
```

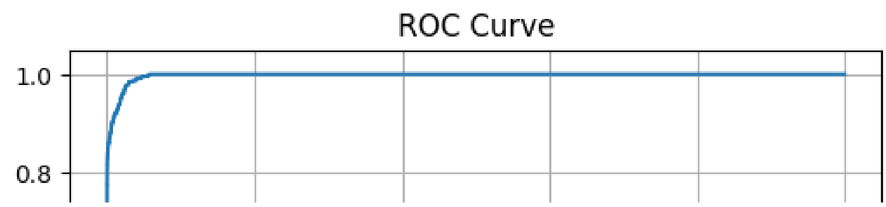
```
Spliting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'V28', 'Amount', 'Amount_scaled']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
------------Training Prediction--------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       0.95      0.98      0.97    226417
           1       0.98      0.95      0.97    226787

    accuracy                           0.97    453204
   macro avg       0.97      0.97      0.97    453204
weighted avg       0.97      0.97      0.97    453204



Accuracy Score (Training):

96.714724%

ROC AUC Score (Training): 0.9975
```

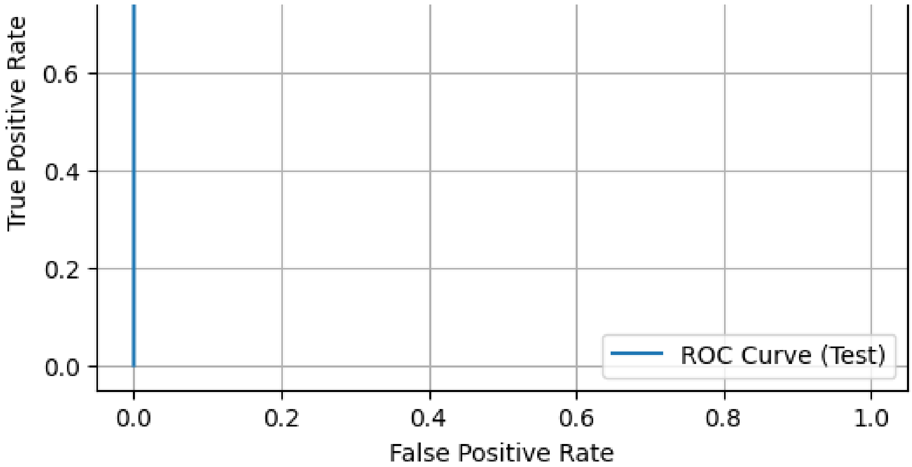





```
F1 Score (Training): 0.9667
------------Test Prediction--------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       0.95      0.98      0.97     56836
           1       0.98      0.95      0.97     56466

    accuracy                           0.97    113302
   macro avg       0.97      0.97      0.97    113302
weighted avg       0.97      0.97      0.97    113302


Accuracy Score (Test):

96.678788%

F1 Score (Test): 0.9662
```








```
ROC AUC Score (Test): 0.9975
```

model perfomance of ADA boost model on oversampled data

The performance metrics on the test set are similar to those on the training set, which indicates good generalization performance. High accuracy, precision, recall, and F1-score on the test set suggest that the model is performing well on unseen data as well. The ROC AUC score is also high on the test set, indicating robustness in classifying fraudulent and non-fraudulent transactions.

Observations:

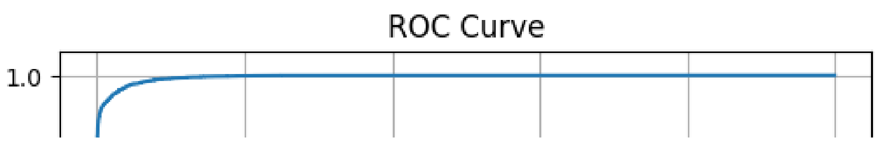
There doesn't seem to be significant overfitting, as the performance metrics on the test set are comparable to those on the training set. The AdaBoost model trained on the oversampled dataset is effective in classifying credit card transactions, achieving high performance on both the training and test sets.

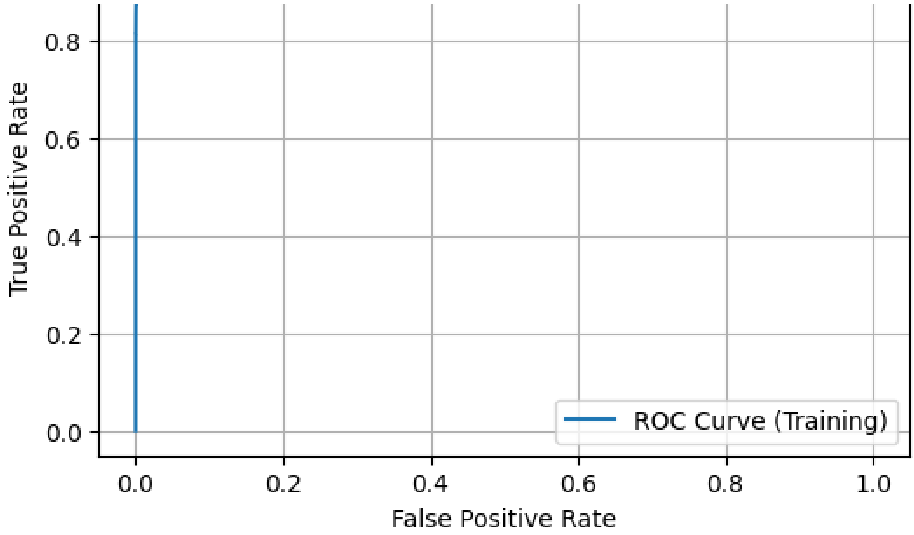
ADABOOSTING on the Smote Dataset

In [216…

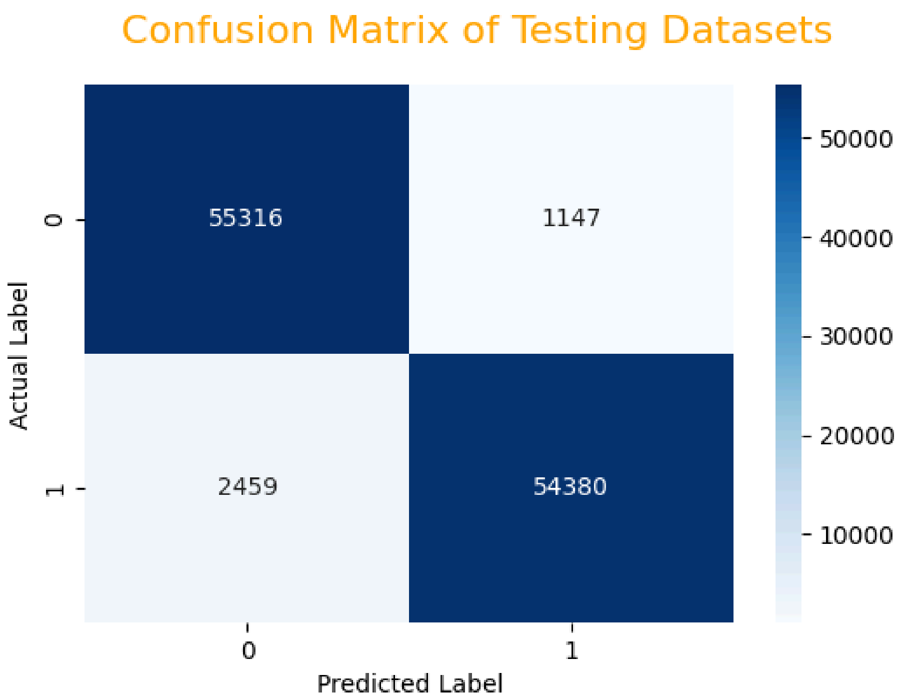
```
AD_model(X_smote,y_smote)
```

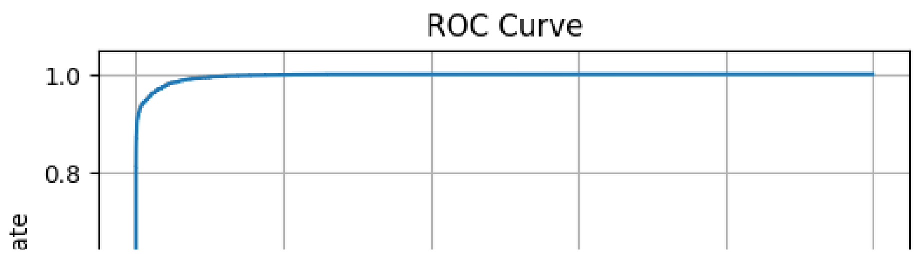
```
Spliting Datasets....
Successfully splitted!!!
Skewed features: ['V2', 'V11', 'V20', 'V22', 'V28', 'Amount', 'Amount_scale
d']
Successfully transformed skewed features!!!
Model Fitting.....
Successfully model fitted!!!
------------Training Prediction-------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       0.96      0.98      0.97    226790
           1       0.98      0.96      0.97    226414

    accuracy                           0.97    453204
   macro avg       0.97      0.97      0.97    453204
weighted avg       0.97      0.97      0.97    453204


Accuracy Score (Training):

96.850866%

ROC AUC Score (Training): 0.9969
```

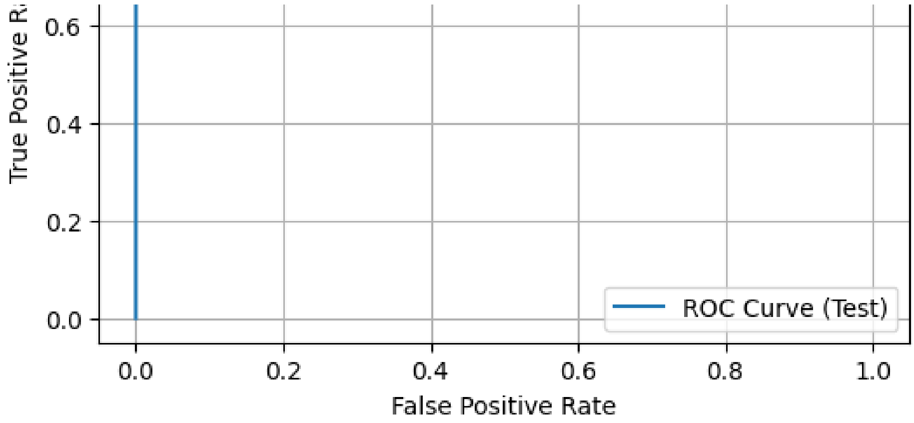





```
F1 Score (Training): 0.9681
------------Test Prediction-------------
Classfifcation Report:

              precision    recall  f1-score   support

           0       0.96      0.98      0.97     56463
           1       0.98      0.96      0.97     56839

    accuracy                           0.97    113302
   macro avg       0.97      0.97      0.97    113302
weighted avg       0.97      0.97      0.97    113302


Accuracy Score (Test):

96.817355%

F1 Score (Test): 0.9679
```








```
ROC AUC Score (Test): 0.9969
```

Model Performance of ada boost model on SMOTE dataset:

The model achieved high accuracy, precision, recall, and F1-score on both the training and test sets, indicating good generalization performance. The ROC AUC score is also high, indicating that the model is good at distinguishing between the positive and negative classes.

Observations:

The performance metrics on the training and test sets are very similar, suggesting that the model generalizes well to unseen data. There doesn't seem to be any overfitting issue, as the performance on the test set is comparable to that on the training set.

Generating results in form of a dataframe

In [159…

```
from sklearn.model_selection import cross_val_score



# Initialize classifiers
classifiers = {
    'Logistic Regression': LogisticRegression(),
    'Gaussian Naive Bayes': GaussianNB(),
    'Decision Trees': DecisionTreeClassifier(),
    'XGBoost': xgb.XGBClassifier(),
    'AdaBoost': AdaBoostClassifier()
}

# Initialize list to store results
results_list = []

# Define datasets
datasets = {
    'Original': (X, y),
    'Undersampled': (X_undersampled, y_undersampled),
    'Oversampled': (X_oversampled, y_oversampled),
    'SMOTE': (X_smote, y_smote)
}

# Define number of folds for cross-validation
num_folds = 5

# Iterate over classifiers and datasets
for dataset_name, (X_data, y_data) in datasets.items():
    for model_name, model in classifiers.items():
        # Split data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X_data, y_data

        # Perform cross-validation on training data
        cv_scores = cross_val_score(model, X_train, y_train, cv=num_folds,
```

```python
        # Fit model on entire training data
        model.fit(X_train, y_train)

        # Make predictions on test data
        y_test_pred = model.predict(X_test)
        y_test_prob = model.predict_proba(X_test)[:, 1]  # Probability of

        # Calculate evaluation metrics for test data
        test_accuracy = accuracy_score(y_test, y_test_pred)
        test_f1 = f1_score(y_test, y_test_pred)
        test_roc_auc = roc_auc_score(y_test, y_test_prob)

        # Append results to list
        results_list.append([dataset_name, model_name, cv_scores.mean(), c
                             test_accuracy, test_f1, test_roc_auc])

# Create DataFrame from results list
results_df = pd.DataFrame(results_list, columns=['Dataset', 'Model',
                                                 'CV ROC AUC Mean', 'CV R
                                                 'Test Accuracy', 'Test F

# Display results
results_df
```

Out[159…

	Dataset	Model	CV ROC AUC Mean	CV ROC AUC Std	Test Accuracy	Test F1 Score	Test ROC AUC Score
0	Original	Logistic Regression	0.952477	0.021987	0.999084	0.628571	0.929068
1	Original	Gaussian Naive Bayes	0.958540	0.012987	0.977954	0.101938	0.954894
2	Original	Decision Trees	0.886163	0.033800	0.999119	0.728261	0.871984
3	Original	XGBoost	0.982569	0.009634	0.999524	0.828025	0.967591
4	Original	AdaBoost	0.960556	0.032002	0.999348	0.778443	0.963431
5	Undersampled	Logistic Regression	0.977198	0.006263	0.936842	0.940000	0.985406
6	Undersampled	Gaussian Naive Bayes	0.957532	0.010561	0.910526	0.910995	0.959169
7	Undersampled	Decision Trees	0.898283	0.012243	0.905263	0.911765	0.904746
8	Undersampled	XGBoost	0.973790	0.004888	0.947368	0.948980	0.979055
9	Undersampled	AdaBoost	0.969772	0.003610	0.931579	0.935323	0.979501
10	Oversampled	Logistic Regression	0.985129	0.000276	0.946020	0.944220	0.985485
11	Oversampled	Gaussian Naive Bayes	0.959787	0.000323	0.914520	0.908844	0.959578
12	Oversampled	Decision Trees	0.999691	0.000053	0.999788	0.999788	0.999789
13	Oversampled	XGBoost	0.999985	0.000010	0.999929	0.999929	0.999997
14	Oversampled	AdaBoost	0.997303	0.000068	0.966788	0.966219	0.997488
15	SMOTE	Logistic Regression	0.991097	0.000229	0.956700	0.955810	0.990941

16	SMOTE	Gaussian Naive Bayes	0.959045	0.001009	0.914732	0.909713	0.958032
17	SMOTE	Decision Trees	0.997854	0.000154	0.998032	0.998040	0.998029
18	SMOTE	XGBoost	0.999986	0.000006	0.999815	0.999815	0.999996
19	SMOTE	AdaBoost	0.996954	0.000098	0.968174	0.967908	0.996900

observation

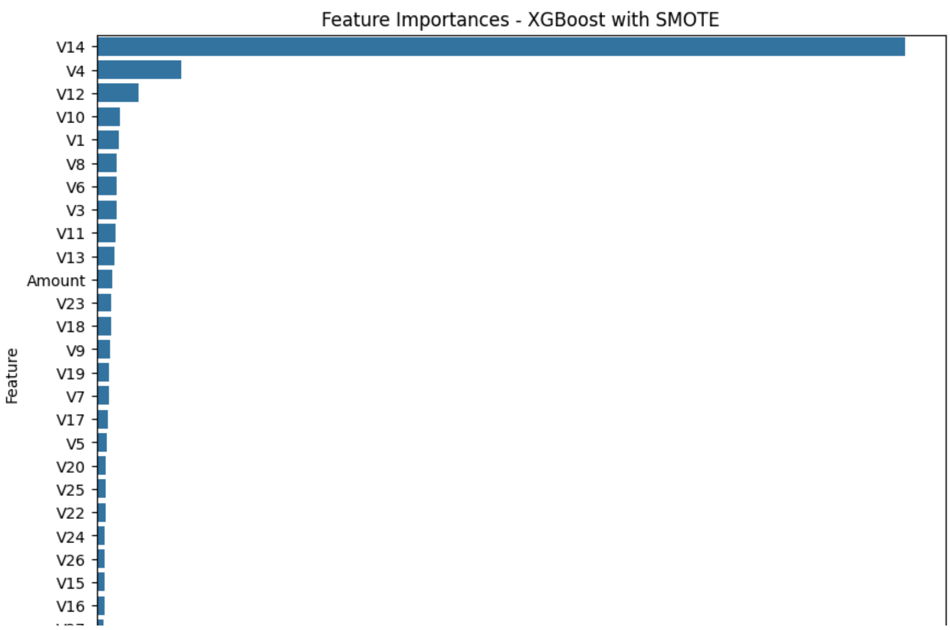
XGBOOST model on the Smote dataset is the best ,with the highest accuracy score,f1 score and roc_auc score and the lowest roc_auc standard deviation.

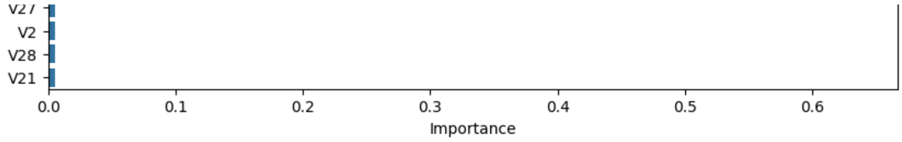
I'm going to check the most important features in order.

FEATURE IMPORTANCE ON THE XG BOOST MODEL WITH SMOTE

In [166…

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# Apply SMOTE to address class imbalance
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_trair

# Train the XGBoost model
xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(X_train_resampled, y_train_resampled)

# Get feature importances
feature_importances = xgb_model.feature_importances_
feature_names = X.columns

# Create a DataFrame to store feature importances
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importanc
feature_importance_df = feature_importance_df.sort_values(by='Importance',

# Plot feature importances
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importances - XGBoost with SMOTE')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```





OBSERVATION