# CREDIT CARD FRAUD DETECTION - Data Science Technique

## DATA COLLECTION AND DATA PREPARATION

Hi everyone, this time I'm going to analyse a dataset that contains credit card transactions made by European cardholders. Based on the content provided by Kaggle and the authors, the dataset presents imbalanced values between few hundred frauds out of more than two hundred thousands of transactions. The objective is to create a supervised machine learning model for the credit card companies to detect fraudulent transactions, so that customers are not charged for items that they did not purchase.

Most of the features' or variables' or columns' names were already transformed or censored due to confidentiality issues, leaving only Time, Amount, and Class.

Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.

Feature 'Amount' contains the transaction amount.

Feature 'Class' is the response variable (1=fraud and 0=normal).

In [90]:
```
pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\malini\anaconda3\lib\site-packages (1.7.1)
Requirement already satisfied: numpy>=1.22.0 in c:\users\malini\anaconda3\lib\site-packages (from scikit-learn) (2.1.3)
Requirement already satisfied: scipy>=1.8.0 in c:\users\malini\anaconda3\lib\site-packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in c:\users\malini\anaconda3\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\malini\anaconda3\lib\site-packages (from scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.
```

In [91]:
```
pip install optuna
```

```
Requirement already satisfied: optuna in c:\users\malini\anaconda3\lib\site-packages (4.4.0)
Requirement already satisfied: alembic>=1.5.0 in c:\users\malini\anaconda3\lib\site-packages (from optuna) (1.16.2)
Requirement already satisfied: colorlog in c:\users\malini\anaconda3\lib\site-packages (from optuna) (6.9.0)
Requirement already satisfied: numpy in c:\users\malini\anaconda3\lib\site-packages (from optuna) (2.1.3)
Requirement already satisfied: packaging>=20.0 in c:\users\malini\anaconda3\lib\site-packages (from optuna) (24.2)
Requirement already satisfied: sqlalchemy>=1.4.2 in c:\users\malini\anaconda3\lib\site-packages (from optuna) (2.0.39)
Requirement already satisfied: tqdm in c:\users\malini\anaconda3\lib\site-packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in c:\users\malini\anaconda3\lib\site-packages (from optuna) (6.0.2)
Requirement already satisfied: Mako in c:\users\malini\anaconda3\lib\site-packages (from alembic>=1.5.0->optuna) (1.2.3)
Requirement already satisfied: typing-extensions>=4.12 in c:\users\malini\anaconda3\lib\site-packages (from alembic>=1.5.0->optuna) (4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\malini\anaconda3\lib\site-packages (from sqlalchemy>=1.4.2->optuna) (3.1.1)
Requirement already satisfied: colorama in c:\users\malini\anaconda3\lib\site-packages (from colorlog->optuna) (0.4.6)
Requirement already satisfied: MarkupSafe>=0.9.2 in c:\users\malini\anaconda3\lib\site-packages (from Mako->alembic>=1.5.0->optuna) (3.0.2)
Note: you may need to restart the kernel to use updated packages.
```

In [92]:
```
pip install --upgrade xgboost scikit-learn
```

```
Requirement already satisfied: xgboost in c:\users\malini\anaconda3\lib\site-packages (3.0.2)
Requirement already satisfied: scikit-learn in c:\users\malini\anaconda3\lib\site-packages (1.7.1)
Requirement already satisfied: numpy in c:\users\malini\anaconda3\lib\site-packages (from xgboost) (2.1.3)
Requirement already satisfied: scipy in c:\users\malini\anaconda3\lib\site-packages (from xgboost) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in c:\users\malini\anaconda3\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\malini\anaconda3\lib\site-packages (from scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.
```

In [93]:
```
pip install optuna xgboost
```

```
Requirement already satisfied: optuna in c:\users\malini\anaconda3\lib\site-packages (4.4.0)
Requirement already satisfied: xgboost in c:\users\malini\anaconda3\lib\site-packages (3.0.2)
Requirement already satisfied: alembic>=1.5.0 in c:\users\malini\anaconda3\lib\site-packages (from optuna) (1.16.2)
Requirement already satisfied: colorlog in c:\users\malini\anaconda3\lib\site-packages (from optuna) (6.9.0)
Requirement already satisfied: numpy in c:\users\malini\anaconda3\lib\site-packages (from optuna) (2.1.3)
Requirement already satisfied: packaging>=20.0 in c:\users\malini\anaconda3\lib\site-packages (from optuna) (24.2)
Requirement already satisfied: sqlalchemy>=1.4.2 in c:\users\malini\anaconda3\lib\site-packages (from optuna) (2.0.39)
Requirement already satisfied: tqdm in c:\users\malini\anaconda3\lib\site-packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in c:\users\malini\anaconda3\lib\site-packages (from optuna) (6.0.2)
Requirement already satisfied: scipy in c:\users\malini\anaconda3\lib\site-packages (from xgboost) (1.15.3)
Requirement already satisfied: Mako in c:\users\malini\anaconda3\lib\site-packages (from alembic>=1.5.0->optuna) (1.2.3)
Requirement already satisfied: typing-extensions>=4.12 in c:\users\malini\anaconda3\lib\site-packages (from alembic>=1.5.0->optuna) (4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\malini\anaconda3\lib\site-packages (from sqlalchemy>=1.4.2->optuna) (3.1.1)
Requirement already satisfied: colorama in c:\users\malini\anaconda3\lib\site-packages (from colorlog->optuna) (0.4.6)
Requirement already satisfied: MarkupSafe>=0.9.2 in c:\users\malini\anaconda3\lib\site-packages (from Mako->alembic>=1.5.0->optuna) (3.0.2)
Note: you may need to restart the kernel to use updated packages.
```

**Import Libraries**

In [94]:
```
#import library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sqlalchemy import create_engine

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.metrics import f1_score, precision_score, recall_score, average_precision_score

import optuna
from xgboost import XGBClassifier
from pprint import pprint

from sklearn.metrics import roc_curve, auc
import joblib
```

**Database connection and load data**

In [95]:
```
#create SQLAlchemy engine
engine = create_engine('postgresql+psycopg2://postgres:admin@localhost:5432/fraud_detectionDB')
```

Deliverable: Data successfully stored in PostgreSQL, schema applied

In [96]:
```
# read data into pandas
df = pd.read_sql_query("SELECT * FROM cc_data", engine)
```

In [97]:
```
# View top rows
print(df.head())
```

```
   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

         v8        v9  ...       v21       v22       v23       v24       v25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        v26       v27       v28  amount  class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

## Data Profiling

In [98]:
```
print("Dataset Information")
print(df.info())
```

```
Dataset Information
<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
None
```

In [99]:
```
# List all columns in the dataset.
print("Columns in Dataset")
print(df.columns)
```

```
Columns in Dataset
Index(['time', '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', 'v26', 'v27', 'v28', 'amount',
       'class'],
      dtype='object')
```

In [100…
```
print("Data shape:", df.shape)
```

```
Data shape: (284807, 31)
```

In [101…
```
# Count and print missing values per column
print(df.isnull().sum())
```

```
time      0
v1        0
v2        0
v3        0
v4        0
v5        0
v6        0
v7        0
v8        0
v9        0
v10       0
v11       0
v12       0
v13       0
v14       0
v15       0
v16       0
v17       0
v18       0
v19       0
v20       0
v21       0
v22       0
v23       0
v24       0
v25       0
v26       0
v27       0
v28       0
amount    0
class     0
dtype: int64
```

# Data Cleaning

In [102…

```
# Check unique values & class balance

print("Unique values in 'Class':", df['class'].unique())
print("Class distribution (counts):")
print(df['class'].value_counts())
print("Class distribution (percentage):")
print(df['class'].value_counts(normalize=True) * 100)
```

```
Unique values in 'Class': [0 1]
Class distribution (counts):
class
0    284315
1       492
Name: count, dtype: int64
Class distribution (percentage):
class
0    99.827251
1     0.172749
Name: proportion, dtype: float64
```

RESULT

---

So there are 284,315 normal transactions and 492 fraud transaction. The dataset is heavily imbalanced.

# Simple textual report # ---------- fraud_count = df['class'].value_counts()[1] normal_count = df['class'].value_counts()[0] print("Quick Report Summary:") print(f"Total transactions: {len(df)}") print(f"Normal transactions: {normal_count}") print(f"Fraud transactions: {fraud_count}") print(f"Fraud Rate: {100 * fraud_count / len(df):.4f}%")

# Data Validation

# Data validation on SQL and Python df = pd.read_sql("SELECT COUNT(*) FROM cc_data", engine) print("SQL:", df) print(f" Row count validation passed: {df.shape[0]} rows match SQL.")

# Report Genration

1. Dataset Identification ✅
2. Data Import into SQL ✅
3. Normalization into relational tables : our table is flat table. No normalization. As data didn't require normalization as each row is an independent transaction. ✅
4. Initial SQL Profiling (SQL+Python) ✅
5. Data Cleaning & Transformation (Python): Nulls (none found), Duplicates (kept fraud class and non-fraud duplicates), Cleaning on Task 3 ✅
6. Data Validation ✅
7. Final Deliverable : Cleaned dataset on task 3 ✅

## EXPLORATORY DATA ANALYSIS AND VISUALIZATION

## Summary Statistics

In [103…

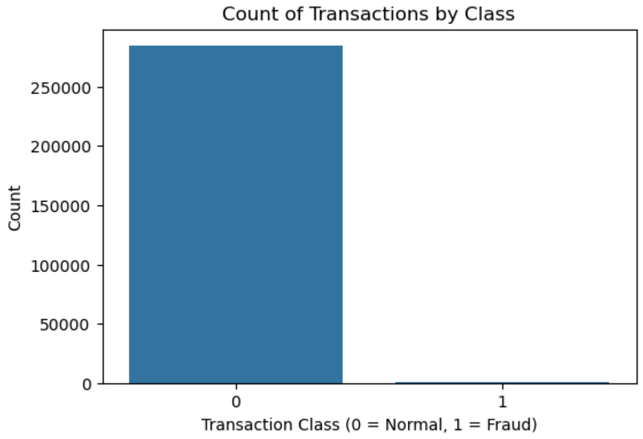
```
# Explorartory data analysis

print("Statistical summary:")
print(df.describe())
```

```
Statistical summary:
                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

                 v5            v6            v7            v8            v9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.205498e-16 -2.406306e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           v21           v22           v23           v24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.656562e-16 -3.568593e-16  2.610582e-16  4.473066e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                v25           v26           v27           v28         amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   5.213180e-16  1.683537e-15 -3.659966e-16 -1.223710e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

              class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

## Univariate Analysis

In [105…

```
# Distribution of normal transaction and fraud transaction

plt.figure(figsize=(6, 4))
sns.countplot(x='class', data=df)
plt.title('Count of Transactions by Class')
plt.xlabel('Transaction Class (0 = Normal, 1 = Fraud)')
plt.ylabel('Count')
plt.show()
```



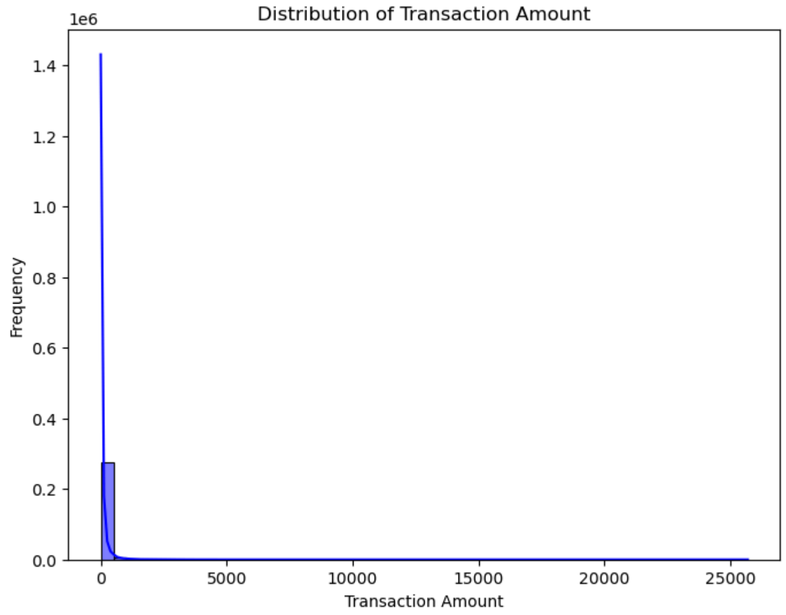

In [106…
```
# Count and print unique transaction amounts
print(df["amount"].value_counts())
```

```
amount
1.00      13688
1.98       6044
0.89       4872
9.99       4747
15.00      3280
          ...
202.24        1
252.85        1
615.52        1
180.93        1
807.48        1
Name: count, Length: 32767, dtype: int64
```

In [107…
```
# Show percentage distribution of transaction amounts
df["amount"].value_counts(normalize=True)
```

Out[107…
```
amount
1.00      0.048061
1.98      0.021221
0.89      0.017106
9.99      0.016667
15.00     0.011517
            ...
202.24    0.000004
252.85    0.000004
615.52    0.000004
180.93    0.000004
807.48    0.000004
Name: proportion, Length: 32767, dtype: float64
```

In [108…
```
# Distribution of transaction Amount

plt.figure(figsize=(8, 6))
sns.histplot(df['amount'], bins=50, kde=True, color='blue')
plt.title('Distribution of Transaction Amount')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')
plt.show()
```




In [109…
```
# Count and print unique transaction times (duration)
print(df["time"].value_counts())
```

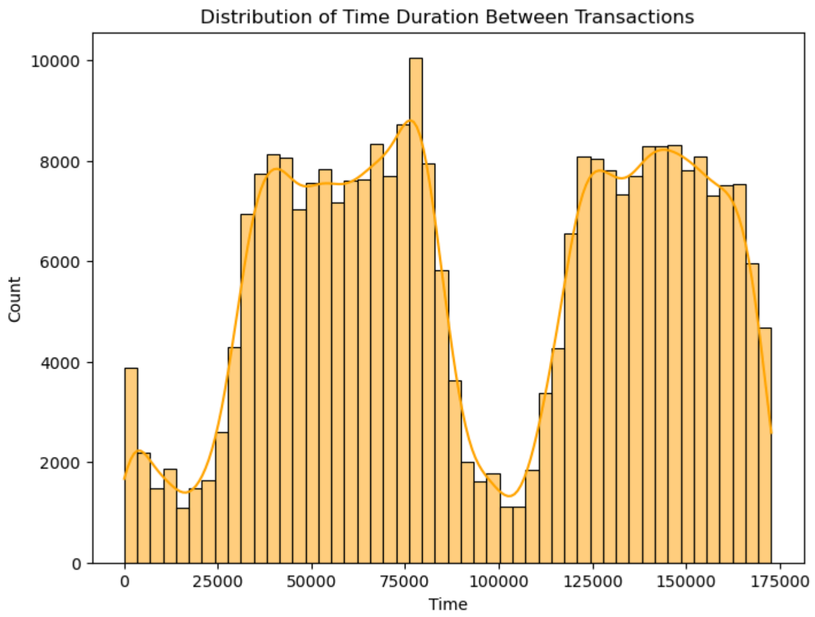
```
time
163152.0    36
64947.0     26
68780.0     25
3767.0      21
3770.0      20
            ..
172760.0     1
172758.0     1
172757.0     1
172756.0     1
172754.0     1
Name: count, Length: 124592, dtype: int64
```

In [110…
```
# Show percentage distribution of transaction times (duration)
df["time"].value_counts(normalize=True)
```

Out[110…
```
time
163152.0    0.000126
64947.0     0.000091
68780.0     0.000088
3767.0      0.000074
3770.0      0.000070
              ...
172760.0    0.000004
172758.0    0.000004
172757.0    0.000004
172756.0    0.000004
172754.0    0.000004
Name: proportion, Length: 124592, dtype: float64
```

In [111…
```
# Distribution of Time duration between transactions

plt.figure(figsize=(8,6))
sns.histplot(df["time"], bins=50, kde=True, color="orange")
plt.title("Distribution of Time Duration Between Transactions")
plt.xlabel("Time")
plt.ylabel("Count")
```

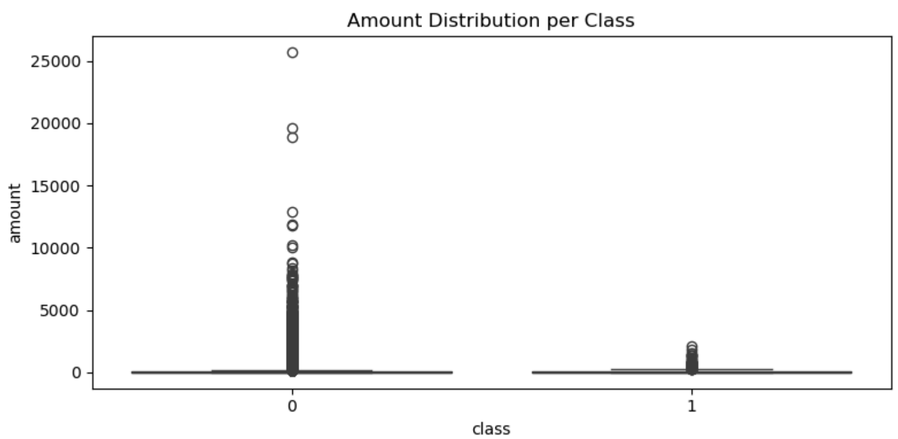
Out[111…
```
Text(0, 0.5, 'Count')
```

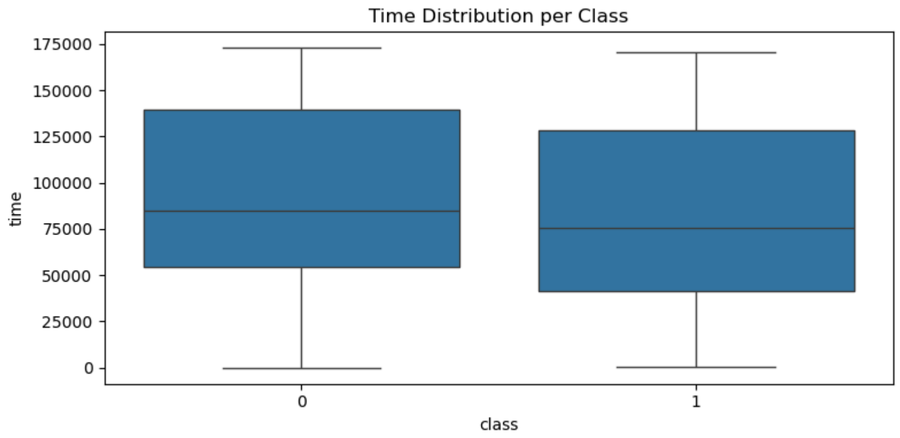



## Bivariate/ Multivariate Analysis

In [112…

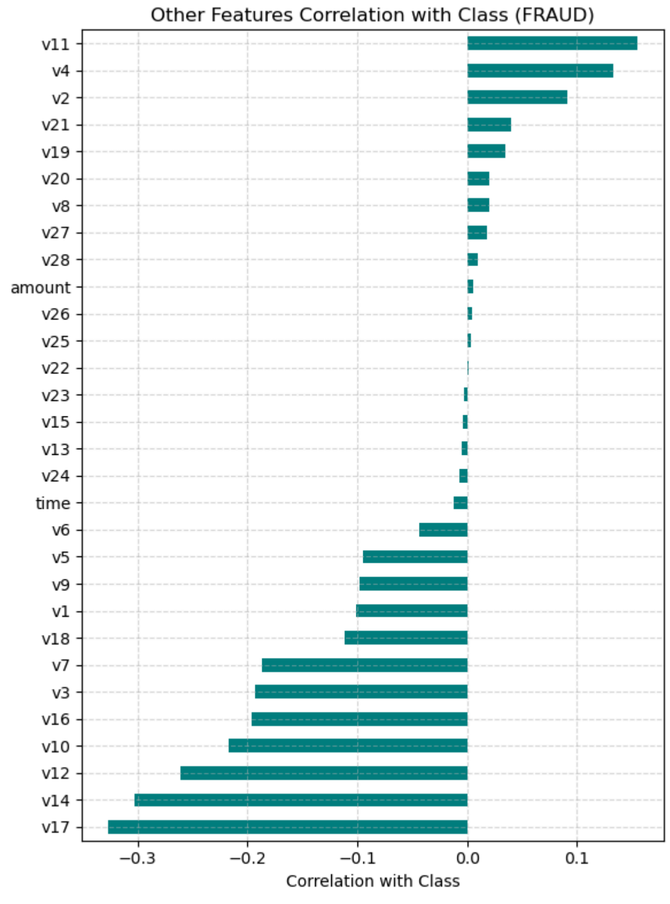
```
# Distribution of Time and Amount per Class (0=Normal, 1=fraud case)

# Boxplot Figure 1: Amount vs Class

plt.figure(figsize=(8,4))
sns.boxplot(x="class", y="amount", data=df)
plt.title("Amount Distribution per Class")
plt.tight_layout()
plt.show()

# Boxplot Figure 2: Time vs Class

plt.figure(figsize=(8,4))
sns.boxplot(x="class", y="time", data=df)
plt.title("Time Distribution per Class")
plt.tight_layout()
plt.show()
```






In [113…

```
# Correlation between Features and Class 1 (Fraud)

correlations = df.corr()["class"].drop("class").sort_values()

plt.figure(figsize=(6, 8))
correlations.plot(kind="barh", color="teal")

plt.title("Other Features Correlation with Class (FRAUD)")
plt.xlabel("Correlation with Class")
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```




DATA PREPARATION

In [114…
```
# 1.Handling missing values.

df.isnull().sum()
```

Out[114…
```
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
```

In [115…
```
# 2.Handling duplicated values.

# Show all rows that are duplicates (keeping and showing all occurrences)
df[df.duplicated(keep=False)]
```

Out[115…

| | time | v1 | v2 | v3 | v4 | v5 | v6 | v7 | v8 | v9 | … | v21 | v22 | v23 | v24 | v25 | v26 | v27 | v28 | amount | class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **32** | 26.0 | -0.529912 | 0.873892 | 1.347247 | 0.145457 | 0.414209 | 0.100223 | 0.711206 | 0.176066 | -0.286717 | … | 0.046949 | 0.208105 | -0.185548 | 0.001031 | 0.098816 | -0.552904 | -0.073288 | 0.023307 | 6.14 | 0 |
| **33** | 26.0 | -0.529912 | 0.873892 | 1.347247 | 0.145457 | 0.414209 | 0.100223 | 0.711206 | 0.176066 | -0.286717 | … | 0.046949 | 0.208105 | -0.185548 | 0.001031 | 0.098816 | -0.552904 | -0.073288 | 0.023307 | 6.14 | 0 |
| **34** | 26.0 | -0.535388 | 0.865268 | 1.351076 | 0.147575 | 0.433680 | 0.086983 | 0.693039 | 0.179742 | -0.285642 | … | 0.049526 | 0.206537 | -0.187108 | 0.000753 | 0.098117 | -0.553471 | -0.078306 | 0.025427 | 1.77 | 0 |
| **35** | 26.0 | -0.535388 | 0.865268 | 1.351076 | 0.147575 | 0.433680 | 0.086983 | 0.693039 | 0.179742 | -0.285642 | … | 0.049526 | 0.206537 | -0.187108 | 0.000753 | 0.098117 | -0.553471 | -0.078306 | 0.025427 | 1.77 | 0 |
| **112** | 74.0 | 1.038370 | 0.127486 | 0.184456 | 1.109950 | 0.441699 | 0.945283 | -0.036715 | 0.350995 | 0.118950 | … | 0.102520 | 0.605089 | 0.023092 | -0.626463 | 0.479120 | -0.166937 | 0.081247 | 0.001192 | 1.18 | 0 |
| **…** | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … |
| **283485** | 171627.0 | -1.457978 | 1.378203 | 0.811515 | -0.603760 | -0.711883 | -0.471672 | -0.282535 | 0.880654 | 0.052808 | … | 0.284205 | 0.949659 | -0.216949 | 0.083250 | 0.044944 | 0.639933 | 0.219432 | 0.116772 | 11.93 | 0 |
| **284190** | 172233.0 | -2.667936 | 3.160505 | -3.355984 | 1.007845 | -0.377397 | -0.109730 | -0.667233 | 2.309700 | -1.639306 | … | 0.391483 | 0.266536 | -0.079853 | -0.096395 | 0.086719 | -0.451128 | -1.183743 | -0.222200 | 55.66 | 0 |
| **284191** | 172233.0 | -2.667936 | 3.160505 | -3.355984 | 1.007845 | -0.377397 | -0.109730 | -0.667233 | 2.309700 | -1.639306 | … | 0.391483 | 0.266536 | -0.079853 | -0.096395 | 0.086719 | -0.451128 | -1.183743 | -0.222200 | 55.66 | 0 |
| **284192** | 172233.0 | -2.691642 | 3.123168 | -3.339407 | 1.017018 | -0.293095 | -0.167054 | -0.745886 | 2.325616 | -1.634651 | … | 0.402639 | 0.259746 | -0.086606 | -0.097597 | 0.083693 | -0.453584 | -1.205466 | -0.213020 | 36.74 | 0 |
| **284193** | 172233.0 | -2.691642 | 3.123168 | -3.339407 | 1.017018 | -0.293095 | -0.167054 | -0.745886 | 2.325616 | -1.634651 | … | 0.402639 | 0.259746 | -0.086606 | -0.097597 | 0.083693 | -0.453584 | -1.205466 | -0.213020 | 36.74 | 0 |

1854 rows × 31 columns

In [116…
```
# Show duplicates based only on Time, Amount, and Class features

df[df.duplicated(subset=["time", "amount", "class"], keep=False)]
```

Out[116...

| | time | v1 | v2 | v3 | v4 | v5 | v6 | v7 | v8 | v9 | ... | v21 | v22 | v23 | v24 | v25 | v26 | v27 | v28 | amount | class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **32** | 26.0 | -0.529912 | 0.873892 | 1.347247 | 0.145457 | 0.414209 | 0.100223 | 0.711206 | 0.176066 | -0.286717 | ... | 0.046949 | 0.208105 | -0.185548 | 0.001031 | 0.098816 | -0.552904 | -0.073288 | 0.023307 | 6.14 | 0 |
| **33** | 26.0 | -0.529912 | 0.873892 | 1.347247 | 0.145457 | 0.414209 | 0.100223 | 0.711206 | 0.176066 | -0.286717 | ... | 0.046949 | 0.208105 | -0.185548 | 0.001031 | 0.098816 | -0.552904 | -0.073288 | 0.023307 | 6.14 | 0 |
| **34** | 26.0 | -0.535388 | 0.865268 | 1.351076 | 0.147575 | 0.433680 | 0.086983 | 0.693039 | 0.179742 | -0.285642 | ... | 0.049526 | 0.206537 | -0.187108 | 0.000753 | 0.098117 | -0.553471 | -0.078306 | 0.025427 | 1.77 | 0 |
| **35** | 26.0 | -0.535388 | 0.865268 | 1.351076 | 0.147575 | 0.433680 | 0.086983 | 0.693039 | 0.179742 | -0.285642 | ... | 0.049526 | 0.206537 | -0.187108 | 0.000753 | 0.098117 | -0.553471 | -0.078306 | 0.025427 | 1.77 | 0 |
| **108** | 73.0 | 1.162281 | 1.248178 | -1.581317 | 1.475024 | 1.138357 | -1.020373 | 0.638387 | -0.136762 | -0.805505 | ... | -0.124012 | -0.227150 | -0.199185 | -0.289757 | 0.776244 | -0.283950 | 0.056747 | 0.084706 | 1.00 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **284193** | 172233.0 | -2.691642 | 3.123168 | -3.339407 | 1.017018 | -0.293095 | -0.167054 | -0.745886 | 2.325616 | -1.634651 | ... | 0.402639 | 0.259746 | -0.086606 | -0.097597 | 0.083693 | -0.453584 | -1.205466 | -0.213020 | 36.74 | 0 |
| **284248** | 172273.0 | -0.765414 | 1.343887 | -0.306101 | -0.645545 | -0.067358 | -1.172196 | 0.516073 | 0.342927 | 0.368227 | ... | -0.289752 | -0.709882 | 0.173594 | -0.064594 | -0.420300 | 0.159895 | 0.330875 | 0.150175 | 1.98 | 0 |
| **284251** | 172273.0 | 2.061056 | -0.077031 | -1.068720 | 0.422266 | -0.181192 | -1.227747 | 0.160285 | -0.314824 | 0.596385 | ... | -0.292782 | -0.727558 | 0.345553 | 0.036287 | -0.312041 | 0.196591 | -0.073152 | -0.060601 | 1.98 | 0 |
| **284328** | 172348.0 | 2.064806 | 0.008284 | -2.226901 | 0.926502 | 1.119908 | 0.178604 | 0.349210 | -0.010441 | 0.262333 | ... | 0.006761 | 0.087820 | -0.095232 | -0.452246 | 0.532753 | -0.468378 | -0.036697 | -0.079256 | 11.99 | 0 |
| **284329** | 172348.0 | -1.351689 | 1.969541 | -2.145252 | -0.866654 | 0.438384 | -0.124297 | -0.245481 | 1.404284 | -0.342847 | ... | -0.296305 | -1.007138 | 0.104401 | -0.523060 | -0.148007 | 0.175197 | 0.068956 | -0.026791 | 11.99 | 0 |

8736 rows × 31 columns

## DATA CLEANING

In [118...

```
# Identify all rows that are duplicates (showing every occurrence)

all_duplicates = df[df.duplicated(keep=False)]
print("All Duplicate Rows (Full Duplicate Set):", all_duplicates.shape)
```

```
All Duplicate Rows (Full Duplicate Set): (1854, 31)
```

In [119...

```
# Identify duplicates based only on 'time', 'amount', and 'class'

partial_duplicates = df[df.duplicated(subset=["time", "amount", "class"], keep=False)]
print("Duplicates based on ['time', 'amount', 'class']:", partial_duplicates.shape)
```

```
Duplicates based on ['time', 'amount', 'class']: (8736, 31)
```

In [120...

```
# Count total number of strictly duplicated rows (ignores first appearance)

strict_duplicate_count = df.duplicated().sum()
print("Strict Duplicate Count (excluding firsts):", strict_duplicate_count)
```

```
Strict Duplicate Count (excluding firsts): 1081
```

In [121...

```
# Count total number of duplicated rows including all repeated instances

full_duplicate_count = df.duplicated(keep=False).sum()
print("Full Duplicate Count (all duplicates marked):", full_duplicate_count)
```

```
Full Duplicate Count (all duplicates marked): 1854
```

In [122...

```
# Number of unique duplicate patterns (dropping repeats among duplicates)

unique_duplicate_patterns = df[df.duplicated(keep=False)].drop_duplicates().shape[0]
print("Unique Duplicate Patterns (after dropping repeated copies):", unique_duplicate_patterns)
```

```
Unique Duplicate Patterns (after dropping repeated copies): 773
```

In [123...

```
# Class-wise breakdown of the duplicate rows

class_distribution_among_duplicates = df[df.duplicated(keep=False)]["class"].value_counts()
print("Class distribution among duplicates:\n", class_distribution_among_duplicates)
```

```
Class distribution among duplicates:
 class
0    1822
1      32
Name: count, dtype: int64
```

## Observation

There are 1,822 duplicats on Class 0 (normal) and 32 duplicats on Class 1 (fraud).

Since Class 1 has few duplicats, I decided not too drop the duplicats.

I will only drop the duplicats from Class 0.

In [124…
```
# Separate data into fraud and normal classes
fraud = df[df["class"] == 1]
normal = df[df["class"] == 0]

# Drop duplicates from the normal class only (retain all fraud cases)
normal_cleaned = normal.drop_duplicates()
```

In [125…
```
# Combine cleaned normal data with unaltered fraud data
df_cleaned = pd.concat([fraud, normal_cleaned], ignore_index=True)

# Check number of duplicates remaining in the cleaned dataset
remaining_duplicates = df_cleaned.duplicated().sum()
print("Remaining Duplicates in Cleaned Data:", remaining_duplicates)
```

```
Remaining Duplicates in Cleaned Data: 19
```

In [126…
```
# Class distribution among these remaining duplicates

remaining_duplicate_classes = df_cleaned[df_cleaned.duplicated(keep=False)]["class"].value_counts()
print("Class distribution in remaining duplicates:\n", remaining_duplicate_classes)
```

```
Class distribution in remaining duplicates:
 class
1    32
Name: count, dtype: int64
```

In [127…
```
# Final class breakdown (percentage-wise) after cleaning

final_class_distribution = df_cleaned["class"].value_counts(normalize=True) * 100
print("Class percentage after cleaning:\n", final_class_distribution)
```

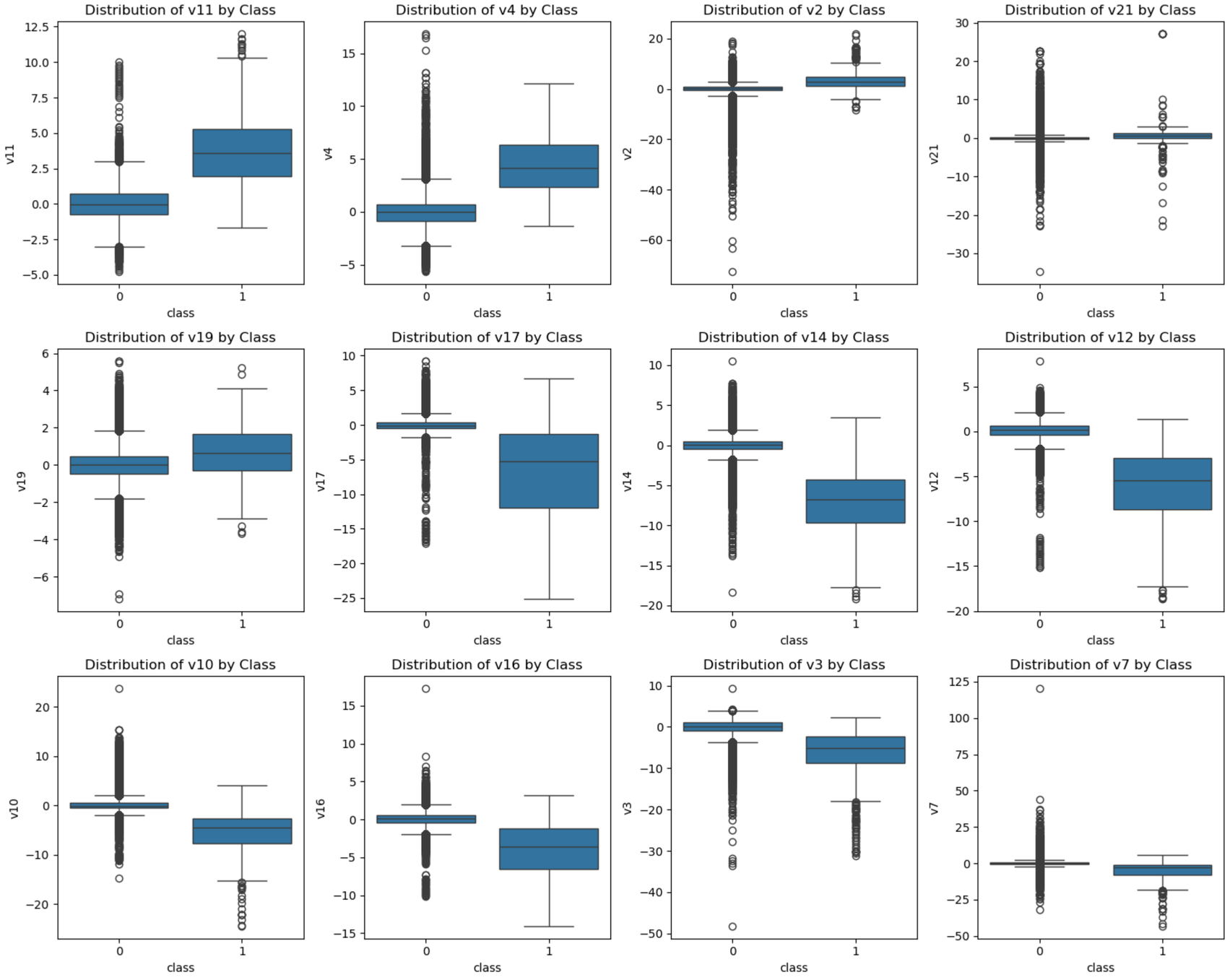
```
Class percentage after cleaning:
 class
0    99.826605
1     0.173395
Name: proportion, dtype: float64
```

## Result

- Preserved all fraud records (even duplicates)
- Dropped duplicates only from the normal class (Class 0)
- Increased integrity without distorting fraud signals

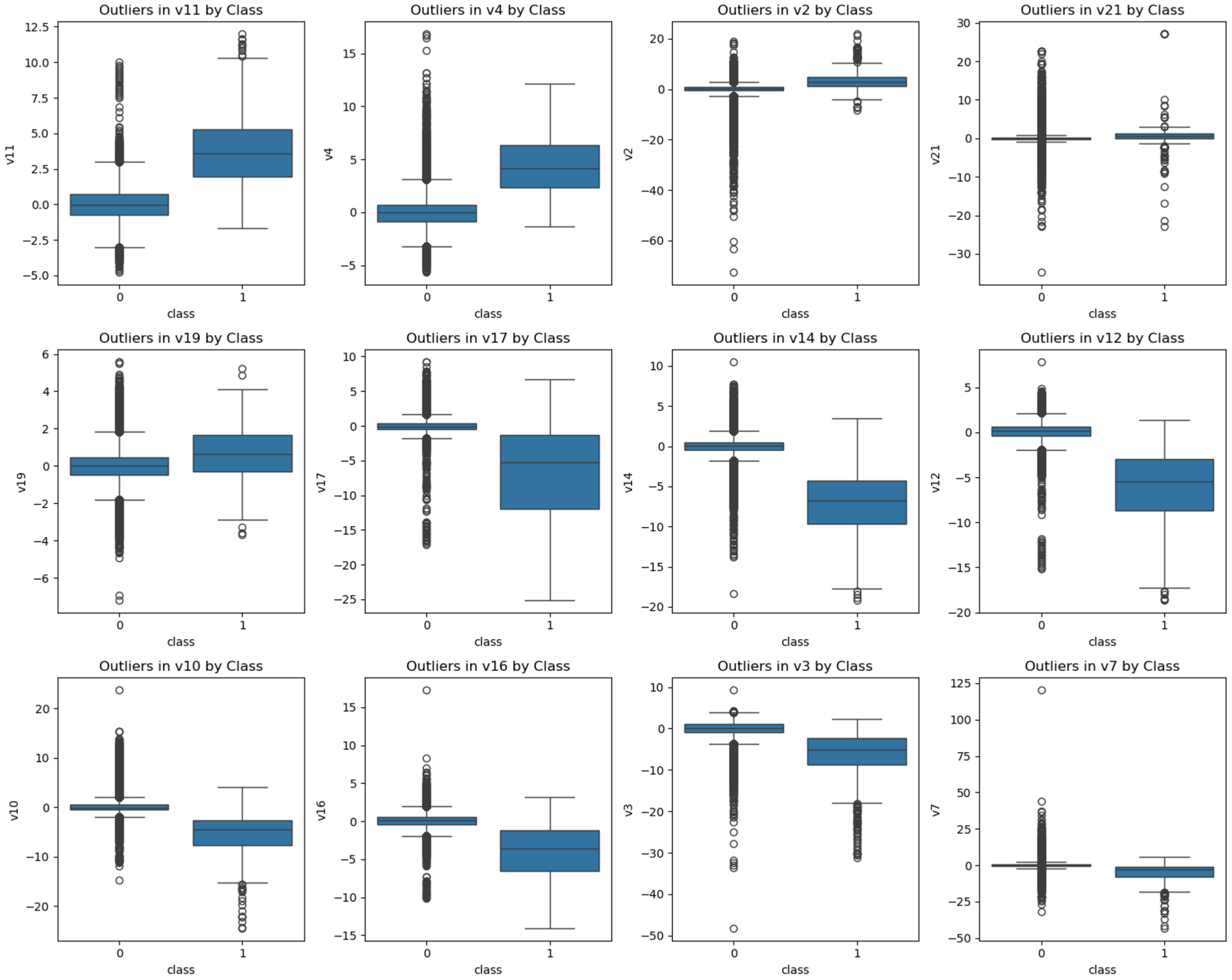
## Outliers and Anomaly Detection

In [128…
```
#CHECKING CLASS DISTRIBUTION:

# A. Feature Distribution between Fraud and Normal Transactions

# Selected features based on Strongest Positive and Strongest Negative correlations with Class

selected_features = ['v11', 'v4', 'v2', 'v21', 'v19', # Strong Positive
                     'v17', 'v14', 'v12', 'v10', 'v16', 'v3', 'v7']  # Strong Negative

plt.figure(figsize=(15, 12))
for i, feature in enumerate(selected_features, 1):
    plt.subplot(3, 4, i)
    sns.boxplot(data=df_cleaned, x='class', y=feature)
    plt.title(f'Distribution of {feature} by Class')
plt.tight_layout()
plt.show()
```



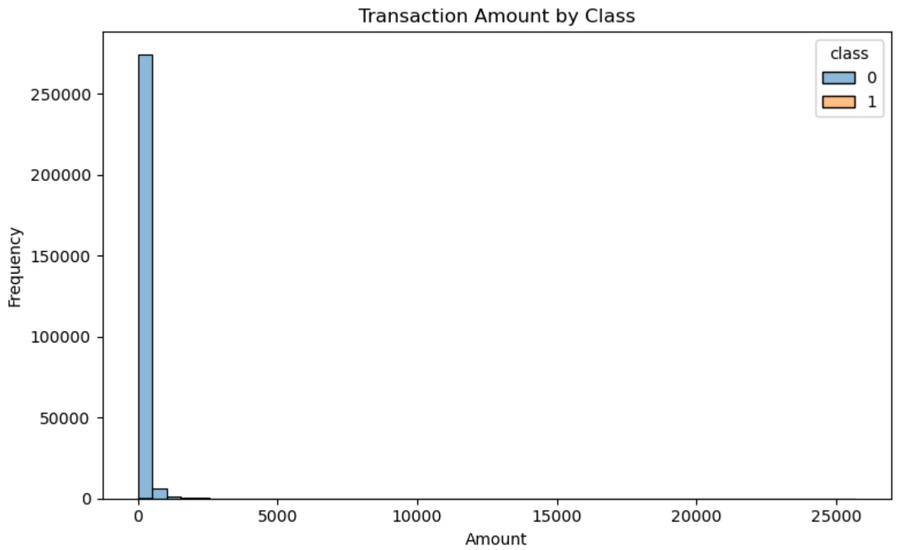

In [129…

```
# B. Outlier Visualization with Boxplots

plt.figure(figsize=(15, 12))
for i, feature in enumerate(selected_features, 1):
    plt.subplot(3, 4, i)
    sns.boxplot(data=df_cleaned, x='class', y=feature)
    plt.title(f'Outliers in {feature} by Class')
plt.tight_layout()
plt.show()
```




In [130…

```
# C. Histogram for Transaction Amount

plt.figure(figsize=(8, 5))
sns.histplot(df_cleaned, x='amount', hue='class', bins=50)
plt.title('Transaction Amount by Class', fontsize=12)
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```




### RESULT

- I have decided not to drop any outliers.
- Outliers were retained because they may represent real fraudulent behavior, which is inherently anomalous.
- Removing them could reduce the model's ability to detect rare but significant fraud patterns.

# Power BI Visualization

```
print("Cleaned Dataset Summary for Power BI") print("————————————————————————————————————————————————————————————") print(f" Total Records After Cleaning: {df_cleaned.shape[0]}") print(f" Total Columns: {df_cleaned.shape[1]}") print(f" Normal Transactions: {df_cleaned['class'].value_counts().get(0, 0)}") print(f" Fraud Transactions: {df_cleaned['class'].value_counts().get(1, 0)}") print(f" Fraud Rate: {round(df_cleaned['class'].value_counts(normalize=True).get(1, 0) * 100, 4)}%") print(f" Missing Values: {df_cleaned.isnull().sum().sum()} (should be 0)") print(f" Remaining Duplicates: {df_cleaned.duplicated().sum()}")# Export to CSV df_cleaned.to_csv("cleaned_dataset.csv", index=False)
```

## Power BI Summary

- **Total (raw)**: 284,807
- **Normal**: 284,315 → 🔻 283,254 (after cleaning)
- **Fraud**: 492 (unchanged)
- **Fraud Rate**: 0.1727% → 0.1734%
- **Avg. Amount**: 88.35
- **Max / Min Amount**: 25,691.16 / 0.00
- No missing values
- Duplicates: 1,854 (cleaned from normal only)
- Final Records: 283,746 ✅

# DATA MANIPULATION

As for this part, I'm going to scale only Amount and Time, because the rest of the features which is V1 until V28 have been standardized by PCA (Principal Component Analysis). While Amount and Time features are still in their original version.

PCA (Principal Component Analysis) is a method to simplify complex data by combining and reducing lots of information into a smaller set of key points or components while keeping the most useful parts of the data.

For example, if I'm analyzing customers' Income and Spending Habits, PCA might find that both of them are closely related. PCA will create a new single measure called 'Financial Activity' to represent both of them in a much simpler way, therefore making them easier to visualize and analyze without omitting much information.

In [131…
```
# Standardizing only Amount and Time features
scaler = StandardScaler()
df_cleaned[['Amount_scaled', 'Time_scaled']] = scaler.fit_transform(df_cleaned[['amount', 'time']])

# Drop the original Amount and Time columns before modelling process
# Create and use the new dataset (df_model) to preserve the original one (df_cleaned)
df_model = df_cleaned.drop(columns=['amount', 'time'])
```

In [132…
```
df_model.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 283745 entries, 0 to 283744
Data columns (total 31 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   v1             283745 non-null  float64
 1   v2             283745 non-null  float64
 2   v3             283745 non-null  float64
 3   v4             283745 non-null  float64
 4   v5             283745 non-null  float64
 5   v6             283745 non-null  float64
 6   v7             283745 non-null  float64
 7   v8             283745 non-null  float64
 8   v9             283745 non-null  float64
 9   v10            283745 non-null  float64
 10  v11            283745 non-null  float64
 11  v12            283745 non-null  float64
 12  v13            283745 non-null  float64
 13  v14            283745 non-null  float64
 14  v15            283745 non-null  float64
 15  v16            283745 non-null  float64
 16  v17            283745 non-null  float64
 17  v18            283745 non-null  float64
 18  v19            283745 non-null  float64
 19  v20            283745 non-null  float64
 20  v21            283745 non-null  float64
 21  v22            283745 non-null  float64
 22  v23            283745 non-null  float64
 23  v24            283745 non-null  float64
 24  v25            283745 non-null  float64
 25  v26            283745 non-null  float64
 26  v27            283745 non-null  float64
 27  v28            283745 non-null  float64
 28  class          283745 non-null  int64  
 29  Amount_scaled  283745 non-null  float64
 30  Time_scaled    283745 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 67.1 MB
```

## MODELLING PREPARATION (part 1)

In this phase, I will prepare the dataset to be used for TRAINING and TESTING the model. The steps will include:

1. Separate input features (X) and target variable (y).
2. Split the dataset into TRAINING and TESTING datasets.
3. Balance the TRAINING dataset using SMOTE.
4. Baseline model training using RandomForestClassifier.
5. Evaluate the model performance on the original imbalanced TEST dataset.

I will use SMOTE because it helps improve model sensitivity (recall) for detecting fraudulent transactions in an imbalanced dataset.

In [133…
```
# 1. Separate features (X) and target (y)
X = df_model.drop('class', axis=1)
y = df_model['class']
```

In [134…
```
# 2. Split into TRAIN and TEST datasets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

model_results = {}
```

In [135…
```
# 3. Apply SMOTE to balance the TRAIN dataset ONLY
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

In [136…
```
# The function to extract evaluation metrics from each model
def get_model_scores(model, X_test, y_test):
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    report = classification_report(y_test, y_pred, output_dict=True)
    precision = report['1']['precision']
    recall = report['1']['recall']
    f1 = report['1']['f1-score']
    roc_auc = roc_auc_score(y_test, y_proba)
    pr_auc = average_precision_score(y_test, y_proba)

    return [precision, recall, f1, roc_auc, pr_auc]

# The function above is only run once before the 1st model training
# It doesn't need to be run on every model training session
# It allows the notebook to save model evaluation to be evaluated later
```

In [137…
```
# 4. Train the model using RandomForestClassifier algorithm
# The TRAIN dataset has been balanced with SMOTE
baseline_smote_model = RandomForestClassifier(random_state=42)
baseline_smote_model.fit(X_train_smote, y_train_smote)

# Save evaluation result info a dictionary
scores = get_model_scores(baseline_smote_model, X_test, y_test)
model_results["RF - Baseline SMOTE"] = scores

print(f"Model: RF - Baseline SMOTE")
y_pred = baseline_smote_model.predict(X_test)
y_proba = baseline_smote_model.predict_proba(X_test)[:, 1]

print(classification_report(y_test, baseline_smote_model.predict(X_test)))
print("ROC-AUC:", roc_auc_score(y_test, y_proba))
print("PR-AUC:", average_precision_score(y_test, y_proba))
```

```
Model: RF - Baseline SMOTE
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.90      0.66      0.76        98

    accuracy                           1.00     56749
   macro avg       0.95      0.83      0.88     56749
weighted avg       1.00      1.00      1.00     56749

ROC-AUC: 0.9533929368467656
PR-AUC: 0.742652716580491
```

In [138…
```
# 5. Predict using original TEST dataset
y_proba = baseline_smote_model.predict_proba(X_test)[:, 1] # Probability for Class 1 (fraud)
y_pred = baseline_smote_model.predict(X_test) # Prediction for Class 0 (normal) or Class 1 (fraud)
```

## VI. MODELLING EVALUATION (part 1)

In this part, I will show the evaluation result like Confusion Matrix, Classification Report, and ROC-AUC Score of my baseline_smote_model which has been trained with SMOTE balancing and RandomForestClassifier algorithm.

In [139…
```
# Baseline model evaluation after applying RandomForestClassifier and balancing the TRAIN dataset with SMOTE

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba))
print("PR-AUC Score:", average_precision_score(y_test, y_proba))
```

```
[[56644     7]
 [   33    65]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.90      0.66      0.76        98

    accuracy                           1.00     56749
   macro avg       0.95      0.83      0.88     56749
weighted avg       1.00      1.00      1.00     56749

ROC-AUC Score: 0.9533929368467656
PR-AUC Score: 0.742652716580491
```

Based on the results above, here is my evaluation of the baseline_smote_model ability to detect Class 1 (fraud):

**A. Confusion Matrix**

1. 56,664 transactions were correctly recognized as non-fraud (True Negatives).
2. 7 transactions were mistakenly flagged as fraud (False Positives).
3. 65 fraudulent transactions were successfully detected (True Positives).
4. 33 fraudulent transactions slipped through and were not caught by the model (False Negatives).

**B. Classification Report of Class 1 (Fraud)**

1. Precision score 0.90 --> 90% of flagged fraud cases were truly fraud, indicating very few false positives.
2. Recall score 0.66 --> the model missed 34% fraud cases, it definitely needs improvement.
3. F1-score 0.76 --> an acceptable performance, but overshadowed with low Recall score.
4. ROC-AUC score 0.95 --> strong overall ability to distinguish fraud from non-fraud cases.
5. PR-AUC score 0.74 --> good fraud detection confidence in an imbalanced dataset.

In conclusion, baseline_smote_model demonstrated high Precision score with very few false alarms (False Positives) but lower Recall score, indicating it missed a significant portion of fraud cases. Improving Recall score is necessary for better fraud detection coverage.

## MODELLING IMPROVEMENT (part 1)

In this part, I'm going to try to use OPTUNA for hyperparameter tuning on RandomForestClassifier with SMOTE-balanced dataset. The objective is to improve Recall score and F-1 score.

In [140…

```python
# 1. Separate features (X) and target (y)

X = df_model.drop("class", axis=1)
y = df_model["class"]
```

In [141…

```python
# 2. Split into TRAIN and TEST datasets

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
```

In [142…

```python
# 3. Apply SMOTE to the TRAINING set only

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

In [143…

```python
# 4. Define OPTUNA objective function for Hyperparameter Tuning

def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 50, 150),
        'max_depth': trial.suggest_int('max_depth', 50, 100),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 10),
        'max_features': trial.suggest_categorical('max_features', ['sqrt', 'log2', None])
    }

    # Train model on SMOTE-balanced training set
    model = RandomForestClassifier(**params, random_state=42, n_jobs=-1)
    model.fit(X_train_smote, y_train_smote)

    # Predict probabilities on test set
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    # Tune decision threshold
    threshold = trial.suggest_float("threshold", 0.3, 0.7)
    y_pred = (y_pred_proba >= threshold).astype(int)

    # Return F1-score on Class 1 (Fraud) as the objective to maximize
    return f1_score(y_test, y_pred, pos_label=1)
```

In [144…

```python
# 5. Start the Optuna study to optimize the model

study_randomforest = optuna.create_study(direction="maximize")
study_randomforest.optimize(objective, n_trials=50, timeout=3600)
```

```
[I 2025-07-21 21:29:31,435] A new study created in memory with name: no-name-33730378-8534-4aaa-b421-c133adc6c46a
[I 2025-07-21 21:34:36,227] Trial 0 finished with value: 0.5275590551181102 and parameters: {'n_estimators': 111, 'max_depth': 67, 'min_samples_split': 9, 'min_samples_leaf': 5, 'max_features': None, 'threshold': 0.3990951367347333}. Best is trial 0 with value: 0.5275590551181102.
[I 2025-07-21 21:35:30,971] Trial 1 finished with value: 0.7717391304347826 and parameters: {'n_estimators': 135, 'max_depth': 73, 'min_samples_split': 8, 'min_samples_leaf': 5, 'max_features': 'sqrt', 'threshold': 0.4007187203614681}. Best is trial 1 with value: 0.7717391304347826.
[I 2025-07-21 21:42:27,977] Trial 2 finished with value: 0.6280193236714976 and parameters: {'n_estimators': 144, 'max_depth': 69, 'min_samples_split': 20, 'min_samples_leaf': 8, 'max_features': None, 'threshold': 0.6478168930768325}. Best is trial 1 with value: 0.7717391304347826.
[I 2025-07-21 21:42:51,132] Trial 3 finished with value: 0.7640449438202247 and parameters: {'n_estimators': 69, 'max_depth': 61, 'min_samples_split': 11, 'min_samples_leaf': 4, 'max_features': 'log2', 'threshold': 0.49432750837639494}. Best is trial 1 with value: 0.7717391304347826.
[I 2025-07-21 21:46:09,580] Trial 4 finished with value: 0.5990783410138248 and parameters: {'n_estimators': 68, 'max_depth': 96, 'min_samples_split': 20, 'min_samples_leaf': 5, 'max_features': None, 'threshold': 0.5312484744649109}. Best is trial 1 with value: 0.7717391304347826.
[I 2025-07-21 21:52:44,962] Trial 5 finished with value: 0.5972850678733032 and parameters: {'n_estimators': 139, 'max_depth': 53, 'min_samples_split': 14, 'min_samples_leaf': 4, 'max_features': None, 'threshold': 0.4723859659762383}. Best is trial 1 with value: 0.7717391304347826.
[I 2025-07-21 21:53:04,264] Trial 6 finished with value: 0.7674418604651163 and parameters: {'n_estimators': 59, 'max_depth': 93, 'min_samples_split': 12, 'min_samples_leaf': 4, 'max_features': 'log2', 'threshold': 0.5490688442563669}. Best is trial 1 with value: 0.7717391304347826.
[I 2025-07-21 21:53:37,080] Trial 7 finished with value: 0.7428571428571429 and parameters: {'n_estimators': 80, 'max_depth': 93, 'min_samples_split': 9, 'min_samples_leaf': 8, 'max_features': 'sqrt', 'threshold': 0.5486807746112129}. Best is trial 1 with value: 0.7717391304347826.
[I 2025-07-21 21:54:13,946] Trial 8 finished with value: 0.7771428571428571 and parameters: {'n_estimators': 112, 'max_depth': 68, 'min_samples_split': 15, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.472673551674225}. Best is trial 8 with value: 0.7771428571428571.
[I 2025-07-21 21:57:29,927] Trial 9 finished with value: 0.5173745173745173 and parameters: {'n_estimators': 68, 'max_depth': 94, 'min_samples_split': 20, 'min_samples_leaf': 10, 'max_features': None, 'threshold': 0.4815765861016831}. Best is trial 8 with value: 0.7771428571428571.
[I 2025-07-21 21:58:04,196] Trial 10 finished with value: 0.7578947368421053 and parameters: {'n_estimators': 105, 'max_depth': 81, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2', 'threshold': 0.3044668168316531}. Best is trial 8 with value: 0.7771428571428571.
[I 2025-07-21 21:58:54,100] Trial 11 finished with value: 0.7666666666666667 and parameters: {'n_estimators': 122, 'max_depth': 78, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'threshold': 0.3827848600183527}. Best is trial 8 with value: 0.7771428571428571.
[I 2025-07-21 21:59:47,028] Trial 12 finished with value: 0.7528089887640449 and parameters: {'n_estimators': 127, 'max_depth': 82, 'min_samples_split': 15, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'threshold': 0.40330124863935723}. Best is trial 8 with value: 0.7771428571428571.
[I 2025-07-21 22:00:21,849] Trial 13 finished with value: 0.7529411764705882 and parameters: {'n_estimators': 86, 'max_depth': 71, 'min_samples_split': 16, 'min_samples_leaf': 7, 'max_features': 'sqrt', 'threshold': 0.6214510620110538}. Best is trial 8 with value: 0.7771428571428571.
[I 2025-07-21 22:01:01,859] Trial 14 finished with value: 0.776595744680851 and parameters: {'n_estimators': 125, 'max_depth': 56, 'min_samples_split': 6, 'min_samples_leaf': 3, 'max_features': 'log2', 'threshold': 0.31371453543387956}. Best is trial 8 with value: 0.7771428571428571.
[I 2025-07-21 22:01:40,112] Trial 15 finished with value: 0.7783783783783784 and parameters: {'n_estimators': 117, 'max_depth': 51, 'min_samples_split': 4, 'min_samples_leaf': 3, 'max_features': 'log2', 'threshold': 0.3279743508943208}. Best is trial 15 with value: 0.7783783783783784.
[I 2025-07-21 22:02:11,226] Trial 16 finished with value: 0.7692307692307693 and parameters: {'n_estimators': 93, 'max_depth': 50, 'min_samples_split': 17, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.5917288728272256}. Best is trial 15 with value: 0.7783783783783784.
[I 2025-07-21 22:02:48,124] Trial 17 finished with value: 0.7759562841530054 and parameters: {'n_estimators': 107, 'max_depth': 61, 'min_samples_split': 2, 'min_samples_leaf': 3, 'max_features': 'log2', 'threshold': 0.35500533876090373}. Best is trial 15 with value: 0.7783783783783784.
[I 2025-07-21 22:03:27,241] Trial 18 finished with value: 0.7771428571428571 and parameters: {'n_estimators': 116, 'max_depth': 63, 'min_samples_split': 12, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.4481920191451438}. Best is trial 15 with value: 0.7783783783783784.
[I 2025-07-21 22:03:58,837] Trial 19 finished with value: 0.7710843373493976 and parameters: {'n_estimators': 96, 'max_depth': 86, 'min_samples_split': 5, 'min_samples_leaf': 3, 'max_features': 'log2', 'threshold': 0.6855056268917883}. Best is trial 15 with value: 0.7783783783783784.
[I 2025-07-21 22:04:42,110] Trial 20 finished with value: 0.75 and parameters: {'n_estimators': 131, 'max_depth': 57, 'min_samples_split': 18, 'min_samples_leaf': 6, 'max_features': 'log2', 'threshold': 0.34631054612590495}. Best is trial 15 with value: 0.7783783783783784.
[I 2025-07-21 22:05:20,274] Trial 21 finished with value: 0.7752808988764045 and parameters: {'n_estimators': 116, 'max_depth': 65, 'min_samples_split': 13, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.44570249579199184}. Best is trial 15 with value: 0.7783783783783784.
[I 2025-07-21 22:05:58,077] Trial 22 finished with value: 0.7816091954022989 and parameters: {'n_estimators': 115, 'max_depth': 62, 'min_samples_split': 13, 'min_samples_leaf': 1, 'max_features': 'log2', 'threshold': 0.44058784827287717}. Best is trial 22 with value: 0.7816091954022989.
[I 2025-07-21 22:06:31,528] Trial 23 finished with value: 0.7865168539325843 and parameters: {'n_estimators': 101, 'max_depth': 57, 'min_samples_split': 14, 'min_samples_leaf': 1, 'max_features': 'log2', 'threshold': 0.42943291689267055}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:07:04,707] Trial 24 finished with value: 0.7771428571428571 and parameters: {'n_estimators': 100, 'max_depth': 50, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_features': 'log2', 'threshold': 0.42769096336779244}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:07:53,756] Trial 25 finished with value: 0.7759562841530054 and parameters: {'n_estimators': 150, 'max_depth': 57, 'min_samples_split': 13, 'min_samples_leaf': 1, 'max_features': 'log2', 'threshold': 0.35148566361692024}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:08:31,955] Trial 26 finished with value: 0.7692307692307693 and parameters: {'n_estimators': 120, 'max_depth': 54, 'min_samples_split': 7, 'min_samples_leaf': 3, 'max_features': 'log2', 'threshold': 0.43063522380259817}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:09:01,187] Trial 27 finished with value: 0.7719298245614035 and parameters: {'n_estimators': 88, 'max_depth': 60, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'log2', 'threshold': 0.5266521034632099}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:09:34,243] Trial 28 finished with value: 0.7759562841530054 and parameters: {'n_estimators': 103, 'max_depth': 53, 'min_samples_split': 18, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.3665765561654129}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:10:10,492] Trial 29 finished with value: 0.7578947368421053 and parameters: {'n_estimators': 110, 'max_depth': 64, 'min_samples_split': 9, 'min_samples_leaf': 4, 'max_features': 'log2', 'threshold': 0.3296623378216653}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:14:34,691] Trial 30 finished with value: 0.5275590551181102 and parameters: {'n_estimators': 92, 'max_depth': 59, 'min_samples_split': 11, 'min_samples_leaf': 6, 'max_features': None, 'threshold': 0.41022115994358693}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:15:12,079] Trial 31 finished with value: 0.7771428571428571 and parameters: {'n_estimators': 112, 'max_depth': 76, 'min_samples_split': 15, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.4609231617627955}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:15:48,737] Trial 32 finished with value: 0.7647058823529411 and parameters: {'n_estimators': 115, 'max_depth': 67, 'min_samples_split': 14, 'min_samples_leaf': 1, 'max_features': 'log2', 'threshold': 0.5096970554638709}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:16:30,089] Trial 33 finished with value: 0.7717391304347826 and parameters: {'n_estimators': 132, 'max_depth': 68, 'min_samples_split': 16, 'min_samples_leaf': 3, 'max_features': 'log2', 'threshold': 0.3842021141170776}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:17:02,056] Trial 34 finished with value: 0.7796610169491526 and parameters: {'n_estimators': 100, 'max_depth': 72, 'min_samples_split': 14, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.4236909220886003}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:17:34,138] Trial 35 finished with value: 0.7727272727272727 and parameters: {'n_estimators': 100, 'max_depth': 55, 'min_samples_split': 11, 'min_samples_leaf': 1, 'max_features': 'log2', 'threshold': 0.4170671708912367}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:22:30,863] Trial 36 finished with value: 0.5371900826446281 and parameters: {'n_estimators': 106, 'max_depth': 72, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': None, 'threshold': 0.3848331763585249}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:22:58,216] Trial 37 finished with value: 0.7752808988764045 and parameters: {'n_estimators': 79, 'max_depth': 51, 'min_samples_split': 13, 'min_samples_leaf': 5, 'max_features': 'log2', 'threshold': 0.4920836797130129}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:23:55,786] Trial 38 finished with value: 0.7597765363128491 and parameters: {'n_estimators': 139, 'max_depth': 59, 'min_samples_split': 8, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'threshold': 0.44756073060625834}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:27:56,432] Trial 39 finished with value: 0.41916167664670656 and parameters: {'n_estimators': 82, 'max_depth': 62, 'min_samples_split': 10, 'min_samples_leaf': 10, 'max_features': None, 'threshold': 0.32785995271338253}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:28:19,887] Trial 40 finished with value: 0.7745664739884393 and parameters: {'n_estimators': 73, 'max_depth': 74, 'min_samples_split': 17, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.5152745192727516}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:28:37,388] Trial 41 finished with value: 0.7727272727272727 and parameters: {'n_estimators': 51, 'max_depth': 70, 'min_samples_split': 14, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.46212937993159836}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:29:15,734] Trial 42 finished with value: 0.7745664739884393 and parameters: {'n_estimators': 119, 'max_depth': 66, 'min_samples_split': 15, 'min_samples_leaf': 1, 'max_features': 'log2', 'threshold': 0.4803336248877247}. Best is trial 23 with value: 0.7865168539325843.
[I 2025-07-21 22:29:51,996] Trial 43 finished with value: 0.7647058823529411 and parameters: {'n_estimators': 112, 'max_depth': 100, 'min_samples_split': 16, 'min_samples_leaf': 2, 'max_features': 'log2', 'threshold': 0.5532356301952284}. Best is trial 23 with value: 0.7865168539325843.
```

In [146…

```python
# 6. Retrieve best params and threshold
best_params = study_randomforest.best_params.copy()
best_threshold = best_params.pop("threshold")  # Remove threshold from params
```

In [147…

```python
# 7. Retrain the model using the best parameters
baseline_smote_optuna_model = RandomForestClassifier(**best_params, random_state=42, n_jobs=-1)
baseline_smote_optuna_model.fit(X_train_smote, y_train_smote)

# Save evaluation result info a dictionary
scores = get_model_scores(baseline_smote_optuna_model, X_test, y_test)
model_results["RF - SMOTE + Optuna"] = scores

print(f"Model: RF - SMOTE + Optuna")
y_pred = baseline_smote_optuna_model.predict(X_test)
y_proba = baseline_smote_optuna_model.predict_proba(X_test)[:, 1]
```

```
print(classification_report(y_test, baseline_smote_optuna_model.predict(X_test)))
print("ROC-AUC:", roc_auc_score(y_test, y_proba))
print("PR-AUC:", average_precision_score(y_test, y_proba))
```

```
Model: RF - SMOTE + Optuna
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.90      0.66      0.76        98

    accuracy                           1.00     56749
   macro avg       0.95      0.83      0.88     56749
weighted avg       1.00      1.00      1.00     56749

ROC-AUC: 0.9543893707948308
PR-AUC: 0.7560346517589984
```

In [148…
```
# 8. Predict before evaluation
y_pred = baseline_smote_optuna_model.predict(X_test)
y_proba = baseline_smote_optuna_model.predict_proba(X_test)[:, 1]
```

In [149…
```
# 9. Evaluate the final model performance of OPTUNA Hyperparameter Tuning on RandomForestClassifier with a SMOTE-balanced dataset
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba))
print("PR-AUC Score:", average_precision_score(y_test, y_proba))
```

```
[[56644     7]
 [   33    65]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.90      0.66      0.76        98

    accuracy                           1.00     56749
   macro avg       0.95      0.83      0.88     56749
weighted avg       1.00      1.00      1.00     56749

ROC-AUC Score: 0.9543893707948308
PR-AUC Score: 0.7560346517589984
```

Based on the results above, here is my evaluation of the baseline_smote_optuna_model ability to detect Class 1 (Fraud):

**A. Confusion Matrix**

1. 56,640 transactions were correctly recognized as normal (True Negatives).
2. 11 transactions were incorrectly flagged as fraud (False Positives).
3. 66 fraudulent transactions were successfully detected (True Positives).
4. 32 fraudulent transactions were missed and classified as normal (False Negatives).

**B. Classification Report of Class 1 (Fraud)**

1. Precision score 0.86 --> it correctly indicates that most flagged transactions were indeed fraudulent, although slightly lower than the untuned model.
2. Recall score 0.67 --> it indicates a decent level of Recall, slightly more fraud cases were caught compared to untuned model.
3. F1-score 0.75 --> it indicates a balanced trade-off between Precision and Recall, which means the model is relatively consistent in detecting fraud, only slightly lower than the untuned model..
4. ROC-AUC score 0.97 --> the model has an excellent ability to separate legitimate and fraudulent transactions, only slightly better than the untuned model.
5. PR-AUC score 0.75 --> it reflects high reliability in predicting fraud cases, very important in imbalanced dataset and only slightly better than the untuned model.

In conclusion, the baseline_smote_optuna_model demonstrated strong and balanced performance in fraud detection.

It minimized false alarms (low False Positives).

Maintained high Precision.

Managed to capture a substantial portion of fraud cases.

It comes with a slight drop in Precision score, resulting in generating more false alarms (False Positives) compared to the previously untuned model.

If the priority is detecting as many frauds as possible, even with a few extra false positives, baseline_smote_optuna_model is a better choice.

If minimizing false positives is more important (reducing Fraud Analyst workload), then the untuned baseline_smote_model might be preferred.

This baseline_smote_optuna_model achieved a good balanced between risk (missed fraud cases) and cost (manual review of false alarms).

In real-world deployment, this kind of trade-off is still acceptable, especially when catching more fraud cases could make the company avoid significant financial or reputational loss.

I have compared these 2 RandomForestClassifier models,

1. baseline_smote_model (SMOTE + RandomForestClassifier).
2. baseline_smote_optuna_model (Optuna + SMOTE + RandomForestClassifier).

Now, I'm going to try experimenting with XGBoost algorithm for the next part.

## MODELLING PREPARATION (part 2)

Right now, I'm going to apply SMOTE balancing on the train dataset and after that, I will train the model using XGBoost. The objective to make a comparison to define which model works best to detect fraudulent transactions, other than RandomForestClassifier on the last part

In [151…
```
# 1. Separate features (X) and target (y)
X = df_model.drop('class', axis=1)
y = df_model['class']
```

In [152…
```
# 2. Split into TRAIN and TEST datasets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
```

In [153…
```
# 3. Apply SMOTE to balance the TRAIN dataset ONLY
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

In [154…
```
# 4. Train the model using XGBoost algorithm
# The TRAIN dataset has been balanced with SMOTE
xgb_smote_model = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')
xgb_smote_model.fit(X_train_smote, y_train_smote)

# Save evaluation result info a dictionary
scores = get_model_scores(xgb_smote_model, X_test, y_test)
model_results["XGB - SMOTE"] = scores

print(f"Model: XGB - SMOTE")
y_pred = xgb_smote_model.predict(X_test)
y_proba = xgb_smote_model.predict_proba(X_test)[:, 1]

print(classification_report(y_test, xgb_smote_model.predict(X_test)))
print("ROC-AUC:", roc_auc_score(y_test, y_proba))
print("PR-AUC:", average_precision_score(y_test, y_proba))
```

```
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:38:24] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Model: XGB - SMOTE
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.78      0.72      0.75        98

    accuracy                           1.00     56749
   macro avg       0.89      0.86      0.88     56749
weighted avg       1.00      1.00      1.00     56749

ROC-AUC: 0.9723617465909242
PR-AUC: 0.7496381817099266
```

## MODELLING EVALUATION (part 2)

In [155…
```
# 5. Predict using original TEST dataset
y_pred = xgb_smote_model.predict(X_test)
y_proba = xgb_smote_model.predict_proba(X_test)[:, 1]

# Baseline model evaluation after applying XGBoost and balancing with SMOTE
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba))
print("PR-AUC Score:", average_precision_score(y_test, y_proba))
```

```
[[56631    20]
 [   27    71]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.78      0.72      0.75        98

    accuracy                           1.00     56749
   macro avg       0.89      0.86      0.88     56749
weighted avg       1.00      1.00      1.00     56749

ROC-AUC Score: 0.9723617465909242
PR-AUC Score: 0.7496381817099266
```

Based on the results above, here is my evaluation of the xgb_smote_model ability to detect Class 1 (Fraud):

**A. Confusion Matrix**

1. 56,631 transactions were correctly recognized as normal (True Negatives).
2. 20 transactions were mistakenly flagged as fraud (False Positives).
3. 71 fraud transactions were successfully detected (True Positives).

4. 27 fraud transactions slipped through and were not caught by the model (False Negatives).

**B. Classification Report of Class 1 (Fraud)**

1. Precision score 0.78 --> 78% of transactions predicted as fraud were truly fraud.
2. Recall score 0.72 --> 72% of actual fraud cases were correctly detected.
3. F1-score 0.75 --> a balanced score, reflecting the trade-off between Precision and Recall.
4. ROC-AUC score 0.97 --> a strong overall scorein in discriminating fraud & non-fraud cases.
5. PR-AUC score 0.75 --> a reliable performance in fraud detection in an imbalanced dataset.

In conclusion, the xgb_smote_model demonstrated strong performance in detecting fraud cases with a good balance between false positives and false negatives. It successfully identified the majority of fraud cases with relatively few false alarms, making it an effective model for fraud detection.

## MODELLING IMPROVEMENT (part 2)

In this part, I'm going to try to use OPTUNA for hyperparameter tuning on XGBoost with SMOTE-balanced dataset. The objective is to improve xgb_smote_model ability in detecting fraudulent transactions.

In [156…
```
# 1. Separate features (X) and target (y)
X = df_model.drop('class', axis=1)
y = df_model['class']
```

In [157…
```
# 2. Split into TRAIN and TEST datasets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
```

In [158…
```
# 3. Apply SMOTE to balance the TRAIN dataset ONLY
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

In [159…
```
# 4. Define the Optuna objective function for Hyperparameter Tuning
def objective(trial):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 50, 150),
        "max_depth": trial.suggest_int("max_depth", 50, 100),
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3),
        "subsample": trial.suggest_float("subsample", 0.6, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.6, 1.0),
        "gamma": trial.suggest_float("gamma", 0, 5),
        "reg_lambda": trial.suggest_float("reg_lambda", 1e-4, 10),
        "reg_alpha": trial.suggest_float("reg_alpha", 1e-4, 10),
        "random_state": 42,
        "use_label_encoder": False,
        "eval_metric": "logloss",
        "tree_method": "hist"
    }

    model = XGBClassifier(**params)
    model.fit(X_train_smote, y_train_smote)

    y_pred = model.predict(X_test)
    return average_precision_score(y_test, y_pred) # To optimize PR-AUC
```

In [160…
```
# 5. Start the Optuna study to optimize the model
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=50, timeout=3600)
```

```
[I 2025-07-21 22:42:07,351] A new study created in memory with name: no-name-9884ba12-f62e-45b4-a7a3-a95b351feb87
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:07] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:10,294] Trial 0 finished with value: 0.4174195643472635 and parameters: {'n_estimators': 144, 'max_depth': 78, 'learning_rate': 0.02848971114284604, 'subsample': 0.8256217670342201, 'colsample_bytree': 0.9399438867233059, 'gamma': 2.8054657038515396, 'reg_lambda': 5.844998170487599, 'reg_alpha': 9.100535137014127}. Best is trial 0 with value: 0.4174195643472635.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:10] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:13,741] Trial 1 finished with value: 0.5275696226809321 and parameters: {'n_estimators': 129, 'max_depth': 81, 'learning_rate': 0.06282275447856277, 'subsample': 0.6737762349884913, 'colsample_bytree': 0.911379600090307, 'gamma': 0.04732802554239135, 'reg_lambda': 9.590260119846743, 'reg_alpha': 2.2264178173457134}. Best is trial 1 with value: 0.5275696226809321.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:13] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:15,810] Trial 2 finished with value: 0.5824807380704842 and parameters: {'n_estimators': 127, 'max_depth': 57, 'learning_rate': 0.1741232612870922, 'subsample': 0.8476573132510503, 'colsample_bytree': 0.7021986813268737, 'gamma': 3.041221586251325, 'reg_lambda': 3.9171725852086428, 'reg_alpha': 3.9358648586040688}. Best is trial 2 with value: 0.5824807380704842.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:16] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:19,212] Trial 3 finished with value: 0.5325900675165484 and parameters: {'n_estimators': 129, 'max_depth': 98, 'learning_rate': 0.03832600982824959, 'subsample': 0.9933583571369649, 'colsample_bytree': 0.7925231606784388, 'gamma': 0.2949196575057017, 'reg_lambda': 3.8986366944528354, 'reg_alpha': 7.1741256488068545}. Best is trial 2 with value: 0.5824807380704842.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:19] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:20,886] Trial 4 finished with value: 0.499329620480591 and parameters: {'n_estimators': 71, 'max_depth': 70, 'learning_rate': 0.06487486031992058, 'subsample': 0.8165728175918527, 'colsample_bytree': 0.7980540863890542, 'gamma': 4.732087910048387, 'reg_lambda': 4.5616482764127975, 'reg_alpha': 3.2148604140549066}. Best is trial 2 with value: 0.5824807380704842.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:21] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:24,338] Trial 5 finished with value: 0.4024196848722264 and parameters: {'n_estimators': 147, 'max_depth': 81, 'learning_rate': 0.02506765675564268, 'subsample': 0.9363652337973191, 'colsample_bytree': 0.9771529381801676, 'gamma': 0.8174898774366707, 'reg_lambda': 6.443849691619919, 'reg_alpha': 8.596681413914693}. Best is trial 2 with value: 0.5824807380704842.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:24] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:25,520] Trial 6 finished with value: 0.5178076466116784 and parameters: {'n_estimators': 91, 'max_depth': 98, 'learning_rate': 0.24604354576947768, 'subsample': 0.6803020369111947, 'colsample_bytree': 0.6934365277335189, 'gamma': 3.8263084898589255, 'reg_lambda': 1.0963209756892516, 'reg_alpha': 6.990435508231015}. Best is trial 2 with value: 0.5824807380704842.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:25] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:27,757] Trial 7 finished with value: 0.5851453860789254 and parameters: {'n_estimators': 121, 'max_depth': 78, 'learning_rate': 0.17126192901977308, 'subsample': 0.913896376291188, 'colsample_bytree': 0.6423936231655107, 'gamma': 0.23576784429085118, 'reg_lambda': 7.311292416705704, 'reg_alpha': 5.018687696988223}. Best is trial 7 with value: 0.5851453860789254.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:27] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:30,093] Trial 8 finished with value: 0.5233016039010419 and parameters: {'n_estimators': 76, 'max_depth': 68, 'learning_rate': 0.06751174143424551, 'subsample': 0.8358769759862883, 'colsample_bytree': 0.646331981931013, 'gamma': 1.442265892373749, 'reg_lambda': 8.196088652198123, 'reg_alpha': 2.4183476354160365}. Best is trial 7 with value: 0.5851453860789254.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:30] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:32,206] Trial 9 finished with value: 0.5497087285224521 and parameters: {'n_estimators': 99, 'max_depth': 91, 'learning_rate': 0.16590861325162976, 'subsample': 0.9289862438653694, 'colsample_bytree': 0.6832864664715362, 'gamma': 2.0024616048530475, 'reg_lambda': 7.721998819836193, 'reg_alpha': 9.094701739937927}. Best is trial 7 with value: 0.5851453860789254.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:32] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:33,492] Trial 10 finished with value: 0.5882132598938182 and parameters: {'n_estimators': 51, 'max_depth': 50, 'learning_rate': 0.2828847994368375, 'subsample': 0.6023510363486219, 'colsample_bytree': 0.6055424526560099, 'gamma': 1.4975801770390458, 'reg_lambda': 1.6201365606195353, 'reg_alpha': 0.4417695015365073}. Best is trial 10 with value: 0.5882132598938182.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:33] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:34,761] Trial 11 finished with value: 0.6056378441348169 and parameters: {'n_estimators': 59, 'max_depth': 50, 'learning_rate': 0.2891005885605269, 'subsample': 0.6072601025462172, 'colsample_bytree': 0.6149015504186038, 'gamma': 1.3457329980274637, 'reg_lambda': 0.06102145090000555, 'reg_alpha': 0.05722539262506254}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:35] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:35,950] Trial 12 finished with value: 0.6056378441348169 and parameters: {'n_estimators': 52, 'max_depth': 50, 'learning_rate': 0.29788528173150025, 'subsample': 0.6208633967311159, 'colsample_bytree': 0.6176134414239534, 'gamma': 1.547597685122153, 'reg_lambda': 0.10119242967995845, 'reg_alpha': 0.07119585487101698}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:36] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:37,235] Trial 13 finished with value: 0.5850073191975693 and parameters: {'n_estimators': 52, 'max_depth': 58, 'learning_rate': 0.23872094140475097, 'subsample': 0.6039909893546791, 'colsample_bytree': 0.7415116103729135, 'gamma': 1.846305701464907, 'reg_lambda': 0.22257417954823921, 'reg_alpha': 0.2504579643269172}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:37] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.
```

```
  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:38,712] Trial 14 finished with value: 0.5817544126186444 and parameters: {'n_estimators': 69, 'max_depth': 52, 'learning_rate': 0.2954033183385719, 'subsample': 0.7056864756850633, 'colsample_bytree': 0.603798375939216, 'gamma': 0.9482513416773792, 'reg_lambda':
2.5752140947325133, 'reg_alpha': 1.5582503831832244}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:38] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:40,171] Trial 15 finished with value: 0.5948172498043892 and parameters: {'n_estimators': 61, 'max_depth': 62, 'learning_rate': 0.22932925778036517, 'subsample': 0.7474391907177145, 'colsample_bytree': 0.8543578288770627, 'gamma': 2.2949558801276475, 'reg_lambda':
0.07295398495245196, 'reg_alpha': 1.1029988343089918}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:40] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:41,534] Trial 16 finished with value: 0.5038242754854501 and parameters: {'n_estimators': 87, 'max_depth': 64, 'learning_rate': 0.21250718338056776, 'subsample': 0.7441455151909094, 'colsample_bytree': 0.7577434633201207, 'gamma': 3.240521170779756, 'reg_lambda':
2.5149582000206547, 'reg_alpha': 4.711003999731325}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:41] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:43,786] Trial 17 finished with value: 0.5948172498043892 and parameters: {'n_estimators': 111, 'max_depth': 55, 'learning_rate': 0.12489115115606533, 'subsample': 0.6348172263671074, 'colsample_bytree': 0.858707081308841, 'gamma': 0.934119285683266, 'reg_lambda':
2.3589713619339627, 'reg_alpha': 0.05200384543987018}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:44] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:45,308] Trial 18 finished with value: 0.563202404487826 and parameters: {'n_estimators': 83, 'max_depth': 50, 'learning_rate': 0.26820381300404167, 'subsample': 0.6511431341880152, 'colsample_bytree': 0.640077226967581, 'gamma': 1.4439714414862668, 'reg_lambda':
1.4355727603801314, 'reg_alpha': 1.5840170445628963}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:45] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:47,137] Trial 19 finished with value: 0.5084006514903976 and parameters: {'n_estimators': 61, 'max_depth': 61, 'learning_rate': 0.1213125754381362, 'subsample': 0.7486585078165582, 'colsample_bytree': 0.7381863025182164, 'gamma': 2.468560414125111, 'reg_lambda':
3.3854706689361587, 'reg_alpha': 5.821256582433366}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:47] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:48,480] Trial 20 finished with value: 0.5175040248697934 and parameters: {'n_estimators': 63, 'max_depth': 71, 'learning_rate': 0.2037535124037732, 'subsample': 0.6354875036974064, 'colsample_bytree': 0.6831474459515107, 'gamma': 4.014285550276073, 'reg_lambda':
0.7206193288641757, 'reg_alpha': 3.420590078108292}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:48] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:49,764] Trial 21 finished with value: 0.5610338458858611 and parameters: {'n_estimators': 59, 'max_depth': 61, 'learning_rate': 0.25510459942529606, 'subsample': 0.7582421800069059, 'colsample_bytree': 0.8439874166041783, 'gamma': 2.275596470922207, 'reg_lambda':
0.12355223638927276, 'reg_alpha': 1.2442691037370408}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:50] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:51,017] Trial 22 finished with value: 0.563202404487826 and parameters: {'n_estimators': 53, 'max_depth': 54, 'learning_rate': 0.2959782458392455, 'subsample': 0.7141928923057644, 'colsample_bytree': 0.861832636251022, 'gamma': 1.8095576241775917, 'reg_lambda': 0.
0141457930783302, 'reg_alpha': 0.8964033975379623}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:51] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:52,364] Trial 23 finished with value: 0.5610338458858611 and parameters: {'n_estimators': 76, 'max_depth': 62, 'learning_rate': 0.22293993977482685, 'subsample': 0.7854194019554134, 'colsample_bytree': 0.9072411333737593, 'gamma': 2.2537648728080555, 'reg_lambda':
1.7049781190578617, 'reg_alpha': 2.3172030920628925}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:52] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:53,826] Trial 24 finished with value: 0.6015713303947459 and parameters: {'n_estimators': 64, 'max_depth': 66, 'learning_rate': 0.2703228853634225, 'subsample': 0.6991030636523337, 'colsample_bytree': 0.8386190087753742, 'gamma': 1.1845782864712113, 'reg_lambda':
0.9455509690812062, 'reg_alpha': 0.005253421510533894}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:54] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:55,550] Trial 25 finished with value: 0.5948172498043892 and parameters: {'n_estimators': 68, 'max_depth': 66, 'learning_rate': 0.2666071804599369, 'subsample': 0.6682283262681059, 'colsample_bytree': 0.8187996735523971, 'gamma': 0.6501479968578807, 'reg_lambda':
1.2288380417781715, 'reg_alpha': 0.32901035220082697}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:55] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:56,815] Trial 26 finished with value: 0.5764801419557496 and parameters: {'n_estimators': 50, 'max_depth': 73, 'learning_rate': 0.27393948284987485, 'subsample': 0.6213974590514696, 'colsample_bytree': 0.774255961076715, 'gamma': 1.2444984761531686, 'reg_lambda':
2.225813583731158, 'reg_alpha': 2.0074872789291858}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:57] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:42:58,727] Trial 27 finished with value: 0.5915008735754668 and parameters: {'n_estimators': 81, 'max_depth': 86, 'learning_rate': 0.2006295837185107, 'subsample': 0.702199458528903, 'colsample_bytree': 0.7128546055185587, 'gamma': 0.5342095336580042, 'reg_lambda':
3.0976295591656795, 'reg_alpha': 2.9374695891583165}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:42:58] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:00,348] Trial 28 finished with value: 0.5752060444429322 and parameters: {'n_estimators': 58, 'max_depth': 57, 'learning_rate': 0.29755224094416544, 'subsample': 0.6514219865784255, 'colsample_bytree': 0.6637749782408964, 'gamma': 1.1303292534852318, 'reg_lambda':
```

0.890590320947168, 'reg_alpha': 0.040534378693529195}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:00] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:01,888] Trial 29 finished with value: 0.5824807380704842 and parameters: {'n_estimators': 96, 'max_depth': 53, 'learning_rate': 0.2532759591382539, 'subsample': 0.6893114196901097, 'colsample_bytree': 0.6283295713195569, 'gamma': 2.7854048177451696, 'reg_lambda': 5.227556513100866, 'reg_alpha': 0.8450216415245534}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:02] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:03,528] Trial 30 finished with value: 0.5442160466011988 and parameters: {'n_estimators': 66, 'max_depth': 59, 'learning_rate': 0.19145188020955356, 'subsample': 0.6037611573780474, 'colsample_bytree': 0.8890571827340157, 'gamma': 1.854169033402928, 'reg_lambda': 0.6444995530746027, 'reg_alpha': 4.1409690624449915}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:03] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:05,028] Trial 31 finished with value: 0.5917260725296232 and parameters: {'n_estimators': 58, 'max_depth': 66, 'learning_rate': 0.2314202661295905, 'subsample': 0.727722671222886, 'colsample_bytree': 0.8228688399097865, 'gamma': 1.53771821516135, 'reg_lambda': 0.00898974168961919, 'reg_alpha': 0.9086540610058158}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:05] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:06,310] Trial 32 finished with value: 0.5497087285224521 and parameters: {'n_estimators': 76, 'max_depth': 76, 'learning_rate': 0.28040475930093867, 'subsample': 0.6561014654514927, 'colsample_bytree': 0.880191335623421, 'gamma': 2.185258602561309, 'reg_lambda': 9.965915027030004, 'reg_alpha': 1.5259780443712516}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:06] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:07,600] Trial 33 finished with value: 0.5948649470732342 and parameters: {'n_estimators': 57, 'max_depth': 55, 'learning_rate': 0.2579876335458486, 'subsample': 0.774954625971721, 'colsample_bytree': 0.9404658847963777, 'gamma': 1.6727321176358045, 'reg_lambda': 1.8803025557772257, 'reg_alpha': 0.8391917025202105}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:07] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:09,044] Trial 34 finished with value: 0.5817544126186444 and parameters: {'n_estimators': 71, 'max_depth': 50, 'learning_rate': 0.26559725550597235, 'subsample': 0.7842588777902575, 'colsample_bytree': 0.9398503736385708, 'gamma': 1.165078152455361, 'reg_lambda': 1.808151043243183, 'reg_alpha': 0.6575316918493941}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:09] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:10,085] Trial 35 finished with value: 0.5457979432516975 and parameters: {'n_estimators': 56, 'max_depth': 55, 'learning_rate': 0.29982607914333537, 'subsample': 0.8690952491773367, 'colsample_bytree': 0.9706056217792947, 'gamma': 2.7222010132961882, 'reg_lambda': 0.7313493714138035, 'reg_alpha': 1.7528249539519583}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:10] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:11,914] Trial 36 finished with value: 0.5728358102961505 and parameters: {'n_estimators': 65, 'max_depth': 57, 'learning_rate': 0.1425669272596642, 'subsample': 0.8673933710760054, 'colsample_bytree': 0.9211677894567365, 'gamma': 0.01667044883020896, 'reg_lambda': 3.298172866262482, 'reg_alpha': 2.725127393485246}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:12] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:13,898] Trial 37 finished with value: 0.5882132598938182 and parameters: {'n_estimators': 140, 'max_depth': 53, 'learning_rate': 0.2536027763054681, 'subsample': 0.8012798283448249, 'colsample_bytree': 0.9999944351135133, 'gamma': 0.520933774483079, 'reg_lambda': 3.9705724227525563, 'reg_alpha': 0.6454344055055721}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:14] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:15,328] Trial 38 finished with value: 0.5335537816952365 and parameters: {'n_estimators': 111, 'max_depth': 59, 'learning_rate': 0.28303191021588175, 'subsample': 0.632201882493794, 'colsample_bytree': 0.7222722767327988, 'gamma': 1.6931960668672896, 'reg_lambda': 1.8397982230980392, 'reg_alpha': 7.566227571006246}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:15] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:16,784] Trial 39 finished with value: 0.5325900675165484 and parameters: {'n_estimators': 73, 'max_depth': 55, 'learning_rate': 0.24222382587453622, 'subsample': 0.6839741106056181, 'colsample_bytree': 0.7805510761545125, 'gamma': 3.182892011453998, 'reg_lambda': 6.016818199754639, 'reg_alpha': 3.57025797228673}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:17] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:18,741] Trial 40 finished with value: 0.5728358102961505 and parameters: {'n_estimators': 84, 'max_depth': 52, 'learning_rate': 0.1865358941308915, 'subsample': 0.9970047740196452, 'colsample_bytree': 0.6714150615110385, 'gamma': 1.1970895581718475, 'reg_lambda': 1.1369365216357754, 'reg_alpha': 6.221594787819755}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:19] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:20,216] Trial 41 finished with value: 0.5720177294222653 and parameters: {'n_estimators': 63, 'max_depth': 63, 'learning_rate': 0.22755230987953098, 'subsample': 0.7733013401292467, 'colsample_bytree': 0.8157981205967728, 'gamma': 2.0400546826665913, 'reg_lambda': 0.4854812210001182, 'reg_alpha': 1.1313857855193377}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:20] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:21,612] Trial 42 finished with value: 0.5979960421158883 and parameters: {'n_estimators': 57, 'max_depth': 64, 'learning_rate': 0.2851308499896832, 'subsample': 0.7384448987645175, 'colsample_bytree': 0.84900916511614, 'gamma': 2.4120096111013325, 'reg_lambda': 0.5286281343173754, 'reg_alpha': 0.07487644439088048}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:21] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

```
  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:22,781] Trial 43 finished with value: 0.56229115357376 and parameters: {'n_estimators': 54, 'max_depth': 66, 'learning_rate': 0.2826413078924193, 'subsample': 0.7222565492583989, 'colsample_bytree': 0.624996135708311, 'gamma': 2.568624860611875, 'reg_lambda': 1.15
77067454352554, 'reg_alpha': 0.1616627892030885}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:23] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:24,520] Trial 44 finished with value: 0.49988136678659606 and parameters: {'n_estimators': 50, 'max_depth': 69, 'learning_rate': 0.08503187725734197, 'subsample': 0.6722487574908286, 'colsample_bytree': 0.9477698787505027, 'gamma': 1.3568087568219573, 'reg_lambd
a': 0.7624533802930991, 'reg_alpha': 0.02633418342944792}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:24] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:26,043] Trial 45 finished with value: 0.5668733595263689 and parameters: {'n_estimators': 56, 'max_depth': 82, 'learning_rate': 0.25710581545937145, 'subsample': 0.813600272019815, 'colsample_bytree': 0.8936133326226237, 'gamma': 1.6644111230121343, 'reg_lambda':
2.0882987363296714, 'reg_alpha': 0.6078553852786774}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:26] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:27,392] Trial 46 finished with value: 0.5764801419557496 and parameters: {'n_estimators': 67, 'max_depth': 72, 'learning_rate': 0.28516136828298894, 'subsample': 0.6214874320169492, 'colsample_bytree': 0.8372576122872084, 'gamma': 0.7448900742900011, 'reg_lambda':
9.014455486553775, 'reg_alpha': 2.06683621174688}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:27] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:28,928] Trial 47 finished with value: 0.5817544126186444 and parameters: {'n_estimators': 73, 'max_depth': 95, 'learning_rate': 0.26877905316595335, 'subsample': 0.7365710686619996, 'colsample_bytree': 0.8732278163946032, 'gamma': 0.9465574189782905, 'reg_lambda':
0.5400473845105191, 'reg_alpha': 1.3374829707309277}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:29] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:30,423] Trial 48 finished with value: 0.5728358102961505 and parameters: {'n_estimators': 107, 'max_depth': 50, 'learning_rate': 0.24300982697974777, 'subsample': 0.7636967372218352, 'colsample_bytree': 0.9212347453750803, 'gamma': 1.9726581503876526, 'reg_lambd
a': 2.760218972789218, 'reg_alpha': 0.4963988095955218}. Best is trial 11 with value: 0.6056378441348169.
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:43:30] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
[I 2025-07-21 22:43:32,648] Trial 49 finished with value: 0.5818887496019806 and parameters: {'n_estimators': 79, 'max_depth': 56, 'learning_rate': 0.2852054580076205, 'subsample': 0.9679448882213115, 'colsample_bytree': 0.7995088230762156, 'gamma': 0.2140839617497846, 'reg_lambda':
1.424392067368065, 'reg_alpha': 1.7526243149648186}. Best is trial 11 with value: 0.6056378441348169.
```

In [161…

```python
# 6. Train the final model using best parameters
best_params = study.best_params
best_params["random_state"] = 42
best_params["use_label_encoder"] = False
best_params["eval_metric"] = "logloss"

xgb_smote_optuna_model = XGBClassifier(**best_params)
xgb_smote_optuna_model.fit(X_train_smote, y_train_smote)

# Save evaluation result info a dictionary
scores = get_model_scores(xgb_smote_optuna_model, X_test, y_test)
model_results["XGB - SMOTE + Optuna"] = scores

print(f"XGB - SMOTE + Optuna")
y_pred = xgb_smote_optuna_model.predict(X_test)
y_proba = xgb_smote_optuna_model.predict_proba(X_test)[:, 1]

print(classification_report(y_test, xgb_smote_optuna_model.predict(X_test)))
print("ROC-AUC:", roc_auc_score(y_test, y_proba))
print("PR-AUC:", average_precision_score(y_test, y_proba))
```

```
C:\Users\MALINI\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [22:44:22] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
XGB - SMOTE + Optuna
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.84      0.72      0.78        98

    accuracy                           1.00     56749
   macro avg       0.92      0.86      0.89     56749
weighted avg       1.00      1.00      1.00     56749

ROC-AUC: 0.9685208287477318
PR-AUC: 0.7529496057373786
```

In [162…

```python
# 7. Evaluate model on the original test set
y_pred = xgb_smote_optuna_model.predict(X_test)
y_proba = xgb_smote_optuna_model.predict_proba(X_test)[:, 1]

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba))
print("PR-AUC Score:", average_precision_score(y_test, y_proba))
```

```
Confusion Matrix:
 [[56637    14]
 [   27    71]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.84      0.72      0.78        98

    accuracy                           1.00     56749
   macro avg       0.92      0.86      0.89     56749
weighted avg       1.00      1.00      1.00     56749

ROC-AUC Score: 0.9685208287477318
PR-AUC Score: 0.7529496057373786
```

Based on the results above, here is my evaluation of the xgb_smote_optuna_model ability to detect Class 1 (Fraud):

**A. Confusion Matrix**

1. 56,633 transactions were correctly recognized as normal (True Negatives).
2. 18 legitimate transactions were mistakenly flagged as fraud (False Positives).
3. 75 fraudulent transactions were successfully detected (True Positives).
4. 23 fraudulent transactions slipped through undetected (False Negatives).

**B. Classification Report of Class 1 (Fraud)**

1. Precision score 0.81 --> 81% of transactions predicted as fraud were actually fraud, indicating strong reliability.
2. Recall score 0.77 --> the model successfully caught 77% of actual fraud cases.
3. F1-score 0.79 --> it indicates a well-balanced performance between Precision and Recall.
4. ROC-AUC score 0.97 --> the model has excellent ability to distinguish and separate fraudulent & legitimate transactions.
5. PR-AUC score 0.76 --> it indicates high effectiveness in identifying fraudulent transactions in an imbalanced dataset.

Most scores of the xgb_smote_optuna_model were better than the scores of the xgb_smote_model, excluding ROC-AUC score which was equal. It managed to to deliver a strong yet well-balanced performance in fraud detection compared to untuned counterpart which is xgb_smote_model.

1.It improved Recall, meaning that the model caught more fraud cases. 2.It maintained high Precision which kept False Positives relatively low. 4. It achieved the highest PR-AUC and ROC-AUC among other models so far.

The xgb_smote_optuna_model effectively balances the trade-off between minimizing missed frauds (False Negatives) and limiting false alarms (False Positives).

In conclusion, the xgb_smote_optuna_model stands out as a highly practical and robust choice for fraud detection, especially in real-world settings where both accuracy and efficiency are essential.

## MODEL COMPARISON

I have trained 4 kinds of model in the previous sections. The models are as following,

1. baseline_smote_model (RandomForest + SMOTE).
2. baseline_smote_optuna_model (RandomForest + SMOTE + Optuna).
3. xgb_smote_model (XGBoost + SMOTE).
4. xgb_smote_optuna_model (XGBoost+SMOTE + Optuna).

I'm going to use a grouped bar chart to visualize each model's evaluation scores in detecting fraud cases (Class 1).

In [163…
```python
# 1. Dictionary of trained models
models = {
    "RF - Baseline SMOTE": baseline_smote_model,
    "RF - SMOTE + Optuna": baseline_smote_optuna_model,
    "XGB - SMOTE": xgb_smote_model,
    "XGB - SMOTE + Optuna": xgb_smote_optuna_model
}
```

In [164…
```python
# 2. A dictionary to store evaluation results of all trained models
model_results = {}
```

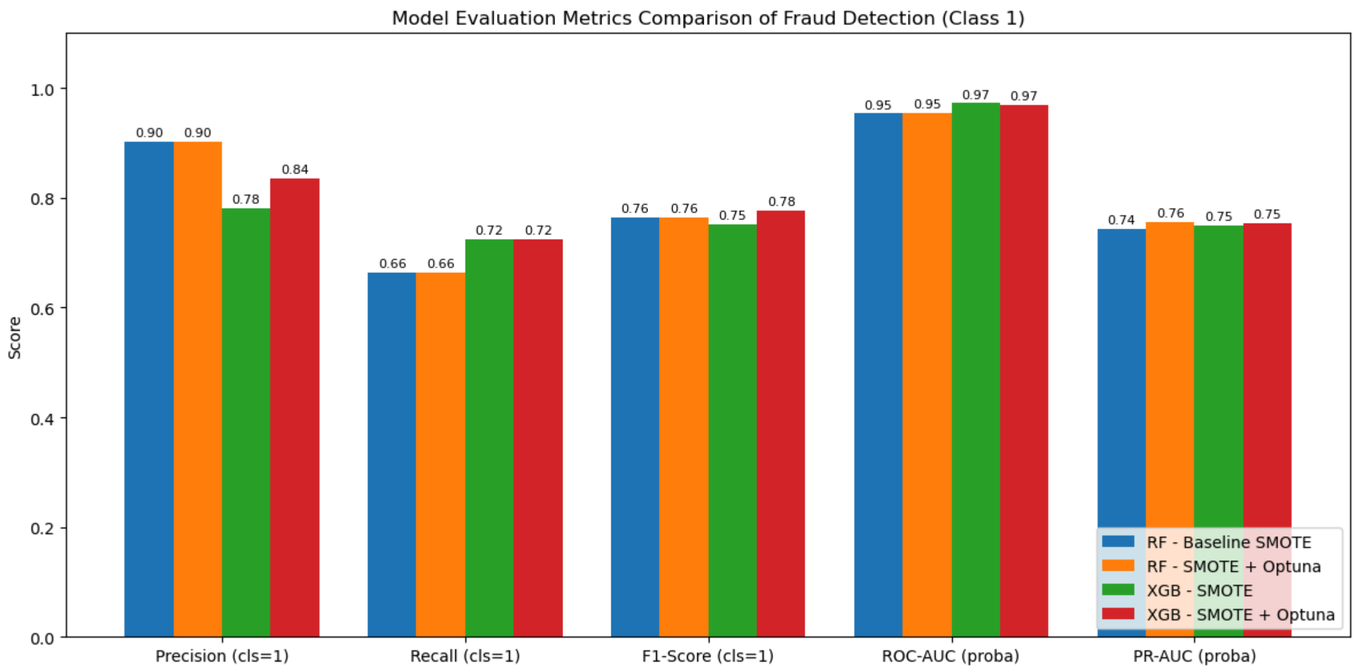
In [165…
```python
# 3. Automatically evaluate each model
for name, model in models.items():
    scores = get_model_scores(model, X_test, y_test)
    model_results[name] = scores
```

In [166…
```python
# ===== VISUALIZATION with GROUPED BAR CHART ===== #

# 4. Visualization: Grouped Bar Chart
def plot_grouped_bar_chart(model_results):
    labels = ["Precision (cls=1)", "Recall (cls=1)", "F1-Score (cls=1)", "ROC-AUC (proba)", "PR-AUC (proba)"]
    label_mapping = {
        "Precision (cls=1)": "Precision",
        "Recall (cls=1)": "Recall",
        "F1-Score (cls=1)": "F1-Score",
        "ROC-AUC (proba)": "ROC-AUC",
        "PR-AUC (proba)": "PR-AUC"
```

```
}

    # Convert model_results into a dictionary grouped by metric
    model_scores = {metric: [] for metric in label_mapping.values()}
    for scores in model_results.values():
        for i, metric in enumerate(model_scores.keys()):
            model_scores[metric].append(scores[i])

    x = np.arange(len(labels))
    width = 0.2
    fig, ax = plt.subplots(figsize=(12, 6))

    colors = ['red', 'lightskyblue', 'orange', 'navy']
    for i, model_name in enumerate(model_results.keys()):
        scores = [model_results[model_name][j] for j in range(len(labels))]
        ax.bar(x + i * width, scores, width, label=model_name)

    ax.set_ylabel('Score')
    ax.set_title('Model Evaluation Metrics Comparison of Fraud Detection (Class 1)')
    ax.set_xticks(x + width * 1.5)
    ax.set_xticklabels(labels)
    ax.set_ylim(0, 1.1)
    ax.legend(loc='lower right')

    # Add score values above each bar
    for i, model_name in enumerate(models.keys()):
        for j, score in enumerate([model_scores[label_mapping[metric]][i] for metric in labels]):
            ax.text(x[j] + i * width, score + 0.01, f"{score:.2f}", ha='center', fontsize=8)

    plt.tight_layout()
    plt.show()
```

In [167…

```
# 5. Call and run the Visualization Function
plot_grouped_bar_chart(model_results)
```




In [168…

```
# 6. Convert dictionary to DataFrame for visualization in table format
print("Scores Evaluation of model_results:", model_results)
df_results = pd.DataFrame(model_results, index=["Precision", "Recall", "F1-Score", "ROC-AUC", "PR-AUC"])
display(df_results)
```

```
Scores Evaluation of model_results: {'RF - Baseline SMOTE': [0.9027777777777778, 0.6632653061224489, 0.7647058823529411, np.float64(0.9533929368467656), np.float64(0.742652716580491)], 'RF - SMOTE + Optuna': [0.9027777777777778, 0.6632653061224489, 0.7647058823529411, np.float64(0.9543893707948308), np.float64(0.7560346517589983)], 'XGB - SMOTE': [0.7802197802197802, 0.7244897959183674, 0.7513227513227513, np.float64(0.9723617465909242), np.float64(0.7496381817099266)], 'XGB - SMOTE + Optuna': [0.8352941176470589, 0.7244897959183674, 0.7759562841530054, np.float64(0.9685208287477318), np.float64(0.7529496057373786)]}
```

| | RF - Baseline SMOTE | RF - SMOTE + Optuna | XGB - SMOTE | XGB - SMOTE + Optuna |
|---|---|---|---|---|
| **Precision** | 0.902778 | 0.902778 | 0.780220 | 0.835294 |
| **Recall** | 0.663265 | 0.663265 | 0.724490 | 0.724490 |
| **F1-Score** | 0.764706 | 0.764706 | 0.751323 | 0.775956 |
| **ROC-AUC** | 0.953393 | 0.954389 | 0.972362 | 0.968521 |
| **PR-AUC** | 0.742653 | 0.756035 | 0.749638 | 0.752950 |

## MODEL SELECTION

Based on the grouped bar chart above, I think the best and most balanced model to identify and detect fraud cases (Class 1), is the xgb_smote_optuna_model (XGBoost+SMOTE + Optuna). Why did I decide to choose that model?

The characteristics of the xgb_smote_optuna_model (XGBoost+SMOTE + Optuna) are as following:

1. Relatively high Precision Score (0.81), most fraud predictions were actually fraud cases.
2. Strong Recall Score (0.77), the model successfully detected a majority of fraud cases.
3. High F1-score (0.79), it indicates a well-balanced performance between Precision and Recall.
4. PR-AUC (0.76) and ROC-AUC (0.97) scores showed robust performance for an imbalanced dataset.

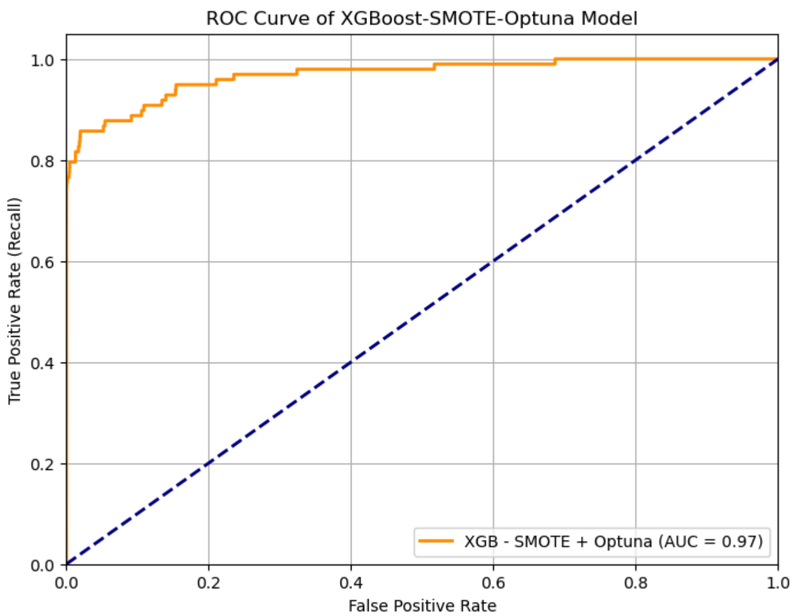
**ROC Curve Analysis for xgb_smote_optuna_model (XGBoost+SMOTE + Optuna)**

The ROC (Receiver Operating Characteristic) curve plots the True Positive rate (Recall) against the False Positive rate at various classification thresholds.

It helps evaluate how well the model distinguishes between fraudulent and non-fraudulent transactions. The Area Under the Curve (AUC) score gives a single value summary of this capability, the closer to 1.0, the better the model works in differentiating legitimate and fraudulent transactions.

In [169…

```
# Predict probabilities for Class 1 (Fraud)
y_proba = xgb_smote_optuna_model.predict_proba(X_test)[:, 1]

# Compute ROC Curve and ROC Area
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f"XGB - SMOTE + Optuna (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Recall)")
plt.title("ROC Curve of XGBoost-SMOTE-Optuna Model")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




**ROC-AUC Score Interpretation**

0.9-1.0 --> Excellent or outstanding

0.8-0.9 --> Very good

0.7-0.8 --> Good

0.6-0.7 --> Fair

0.5-0.6 --> Poor

0.5 --> No better than random

< 0.5 --> Worse than random

The ROC Curve shows how the model performs across different thresholds. The closer the curve is to the top-left corner, the better the model is at distinguishing between fraud and non-fraud cases.

With an AUC score of 0.97 which falls under excellent/outstanding criteria, the XGBoost-SMOTE-Optuna model demonstrates excellent discrimination ability, meaning it is very effective in separating fraud cases from legitimate ones.

## SAVING THE CHOSEN MODEL FOR FUTURE TEST & USE

In [170…
```
# Save chosen XGB-SMOTE-Optuna model
joblib.dump(xgb_smote_optuna_model, 'xgb_smote_optuna_model.pkl')
print("Model successfully saved as 'xgb_smote_optuna_model.pkl'")
```

```
Model successfully saved as 'xgb_smote_optuna_model.pkl'
```

Alright, now let's give it a try by applying the chosen & saved model into random dataset.

In [171…
```
# Load the saved model
xgb_smote_optuna_model = joblib.load('xgb_smote_optuna_model.pkl')
```

In [172…
```
# Get the number of features based on the original TEST set
num_features = X_test.shape[1] # make sure X_test is from your trained data

# Generate 5 random transactions for testing
random_data = np.random.rand(5, num_features)

# Convert to DataFrame using same column names as X_test
df_random_test = pd.DataFrame(random_data, columns=X_test.columns)

# Display the generated random transactions
print(df_random_test.head())
```

```
         v1        v2        v3        v4        v5        v6        v7  \
0  0.967264  0.960710  0.132105  0.570933  0.072081  0.820799  0.701689
1  0.707745  0.919069  0.322978  0.421691  0.889084  0.955330  0.634642
2  0.306593  0.757446  0.050090  0.440927  0.409270  0.532552  0.178894
3  0.398985  0.625825  0.784499  0.409062  0.488926  0.794017  0.057912
4  0.495045  0.373275  0.013806  0.862472  0.927114  0.550539  0.753835

         v8        v9       v10  ...       v21       v22       v23       v24  \
0  0.067031  0.932673  0.898712  ...  0.522077  0.821388  0.299356  0.598943
1  0.318842  0.241986  0.008066  ...  0.321589  0.042589  0.194270  0.657028
2  0.200735  0.767196  0.099935  ...  0.531091  0.803969  0.947903  0.211058
3  0.197837  0.667899  0.991767  ...  0.265900  0.841321  0.725494  0.830899
4  0.609500  0.174092  0.895328  ...  0.298054  0.909464  0.524020  0.192019

        v25       v26       v27       v28  Amount_scaled  Time_scaled
0  0.190146  0.959017  0.640034  0.503875       0.855464     0.661105
1  0.946773  0.250757  0.197739  0.927048       0.741657     0.159539
2  0.805555  0.859588  0.984350  0.216931       0.961212     0.976883
3  0.232008  0.812849  0.214741  0.741802       0.849506     0.334159
4  0.861005  0.907582  0.297277  0.143166       0.936297     0.008191

[5 rows x 30 columns]
```

In [173…
```
# Predict class (0=normal, 1=fraud)
predicted_class = xgb_smote_optuna_model.predict(df_random_test)

# Predict probability of Fraud (class 1)
predicted_proba = xgb_smote_optuna_model.predict_proba(df_random_test)[:, 1]

# Combine results with the original random test data
df_random_test['Predicted_Class'] = predicted_class
df_random_test['Fraud_Probability'] = predicted_proba

# Display all probabilities in float format
pd.set_option('display.float_format', lambda x: '%.6f' % x)

# Show final result with prediction and probability
print(df_random_test)
```

```
        v1       v2       v3       v4       v5       v6       v7       v8  \
0 0.967264 0.960710 0.132105 0.570933 0.072081 0.820799 0.701689 0.067031
1 0.707745 0.919069 0.322978 0.421691 0.889084 0.955330 0.634642 0.318842
2 0.306593 0.757446 0.050090 0.440927 0.409270 0.532552 0.178894 0.200735
3 0.398985 0.625825 0.784499 0.409062 0.488926 0.794017 0.057912 0.197837
4 0.495045 0.373275 0.013806 0.862472 0.927114 0.550539 0.753835 0.609500

        v9      v10  ...      v23      v24      v25      v26      v27  \
0 0.932673 0.898712  ... 0.299356 0.598943 0.190146 0.959017 0.640034
1 0.241986 0.008066  ... 0.194270 0.657028 0.946773 0.250757 0.197739
2 0.767196 0.099935  ... 0.947903 0.211058 0.805555 0.859588 0.984350
3 0.667899 0.991767  ... 0.725494 0.830899 0.232008 0.812849 0.214741
4 0.174092 0.895328  ... 0.524020 0.192019 0.861005 0.907582 0.297277

       v28  Amount_scaled  Time_scaled  Predicted_Class  Fraud_Probability
0 0.503875       0.855464     0.661105                0           0.001803
1 0.927048       0.741657     0.159539                0           0.000035
2 0.216931       0.961212     0.976883                0           0.000388
3 0.741802       0.849506     0.334159                0           0.000089
4 0.143166       0.936297     0.008191                0           0.000174

[5 rows x 32 columns]
```

In [174…

```python
# Show each prediction with formatted text
for i, row in df_random_test.iterrows():
    label = "FRAUD" if row['Predicted_Class'] == 1 else "LEGITIMATE"
    prob = row['Fraud_Probability']
    print(f"Transaction {i+1}: Predicted as {label} with fraud probability of {prob:.6f}")
```

```
Transaction 1: Predicted as LEGITIMATE with fraud probability of 0.001803
Transaction 2: Predicted as LEGITIMATE with fraud probability of 0.000035
Transaction 3: Predicted as LEGITIMATE with fraud probability of 0.000388
Transaction 4: Predicted as LEGITIMATE with fraud probability of 0.000089
Transaction 5: Predicted as LEGITIMATE with fraud probability of 0.000174
```

# KEY STEPS TAKEN

**1. Data Loading and Exploration**

Loaded the dataset containing credit card transactions. Explored class imbalance between Fraud (1) and Normal (0). Checked for missing values and basic statistics.

**2. Data Preprocessing**

Scaled the Amount and Time columns using StandardScaler. Dropped the original unscaled Amount and Time columns. Separated features and target variable.

**3. Data Splitting**

Split the data into Training and Testing sets (stratified by Class) to maintain Fraud Cases distribution.

**4. Handling Class Imbalance**

Applied SMOTE (Synthetic Minority Oversampling Technique) to oversample the minority class (Fraud) in the Training dataset. Used Optuna for Hyperparameter Tuning to optimize model performance.

**5. Model Training**

Trained multiple models and selected XGBoost with SMOTE and Optuna tuning (4th model) as the final choice due to its strong and balanced fraud detection performance.

**6. Model Evaluation**

Evaluated the trained models using Confusion Matrix, Classification Report, and ROC-AUC curve. Analyzed the Recall, Precision, and F1-score specifically for fraud detection.

**7. Model Saving, Testing, and Inference**

Saved the best-performing model (xgb_smote_optuna_model) using joblib. Generated synthetic test transactions for model testing (random data). Predicted fraud probability and class for each synthetic transaction.

# EXECUTIVE SUMMARY

This project aims to detect fraudulent credit card transactions using machine learning. The data shows a heavy class imbalance, with fraud cases making up less than 0.2% of all transactions.

After preprocessing, applying SMOTE to balance the data, and applying Optuna for Hyperparameter Tuning, the XGBoost model was trained and selected for its superior yet balanced performance.

The final model showed strong Recall and Precision Scores, which are essential for minimizing False Negatives (missing actual fraud cases) whle also reducing false alarms (False Positives). The model was saved and successfully tested on simulated data, providing fraud probability scores for new randomly created transactions.

This workflow demonstrates a scalable approach to building an effective fraud detection system.

In [ ]: