Credit Card Fraud Detection

This project aims to detect fraudulent transactions using credit card data. The dataset, containing 284,807 transactions, includes 492 fraud cases, making it highly imbalanced, with fraud accounting for only 0.172% of transactions.

Key Points:

1. Features: The dataset contains only numerical input variables, which are the result of Principal Component Analysis (PCA). Features V1 to V28 represent these components. The 'Time' feature indicates the seconds elapsed since the first transaction, and 'Amount' represents the transaction amount.
2. Target Variable: The 'Class' feature indicates whether a transaction is fraudulent (1) or not (0).
3. Challenge: The dataset is imbalanced, and using accuracy as a performance measure is not reliable. Therefore, we recommend evaluating the model performance using the Area Under the Precision-Recall Curve (AUPRC), which is better suited for imbalanced datasets.

Import Necessory Libraries

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

# Set seaborn theme for better visual aesthetics
sns.set_theme(style="whitegrid")

# Import necessary libraries
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (confusion_matrix, accuracy_score,
                             classification_report, precision_recall_curve, auc)
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import RandomizedSearchCV, ShuffleSplit

import warnings
# Suppress warnings for cleaner output
warnings.filterwarnings("ignore")
```

Load DataSet

- **Load Data directly through using kaggle API because of data set is little huge which take a long time on google colab**

```
from google.colab import files
files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 66 bytes, last modified: 2/7/2025 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"samikhan25","key":"e79a0dc30de81afae830d9d1e172ca2c"}'}
```

```
# !pip install kaggle  # Install Kaggle API
!mkdir -p ~/.kaggle  # Create Kaggle directory
!cp kaggle.json ~/.kaggle/  # Copy your Kaggle API key (Upload kaggle.json first)
!chmod 600 ~/.kaggle/kaggle.json  # Set file permissions
```

```
!kaggle datasets download -d mlg-ulb/creditcardfraud
```

```
Dataset URL: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
License(s): DbCL-1.0
Downloading creditcardfraud.zip to /content
 73% 48.0M/66.0M [00:00<00:00, 169MB/s]
100% 66.0M/66.0M [00:00<00:00, 161MB/s]
```

Unzip dataset

```
from zipfile import ZipFile

dataset="/content/creditcardfraud.zip"

with ZipFile(dataset, 'r') as zip:
  zip.extractall()
  print("The DataSet us extracted")
```

```
The DataSet us extracted
```

```
df = pd.read_csv("/content/creditcard.csv")
df.shape
```

```
(284807, 31)
```

Show Top 5 Rows

```
df.head().T
```

	0	1	2	3	4
Time	0.000000	0.000000	1.000000	1.000000	2.000000
V1	-1.359807	1.191857	-1.358354	-0.966272	-1.158233
V2	-0.072781	0.266151	-1.340163	-0.185226	0.877737
V3	2.536347	0.166480	1.773209	1.792993	1.548718
V4	1.378155	0.448154	0.379780	-0.863291	0.403034
V5	-0.338321	0.060018	-0.503198	-0.010309	-0.407193
V6	0.462388	-0.082361	1.800499	1.247203	0.095921
V7	0.239599	-0.078803	0.791461	0.237609	0.592941
V8	0.098698	0.085102	0.247676	0.377436	-0.270533
V9	0.363787	-0.255425	-1.514654	-1.387024	0.817739
V10	0.090794	-0.166974	0.207643	-0.054952	0.753074
V11	-0.551600	1.612727	0.624501	-0.226487	-0.822843
V12	-0.617801	1.065235	0.066084	0.178228	0.538196
V13	-0.991390	0.489095	0.717293	0.507757	1.345852
V14	-0.311169	-0.143772	-0.165946	-0.287924	-1.119670
V15	1.468177	0.635558	2.345865	-0.631418	0.175121
V16	-0.470401	0.463917	-2.890083	-1.059647	-0.451449
V17	0.207971	-0.114805	1.109969	-0.684093	-0.237033
V18	0.025791	-0.183361	-0.121359	1.965775	-0.038195
V19	0.403993	-0.145783	-2.261857	-1.232622	0.803487
V20	0.251412	-0.069083	0.524980	-0.208038	0.408542
V21	-0.018307	-0.225775	0.247998	-0.108300	-0.009431
V22	0.277838	-0.638672	0.771679	0.005274	0.798278
V23	-0.110474	0.101288	0.909412	-0.190321	-0.137458
V24	0.066928	-0.339846	-0.689281	-1.175575	0.141267
V25	0.128539	0.167170	-0.327642	0.647376	-0.206010
V26	-0.189115	0.125895	-0.139097	-0.221929	0.502292
V27	0.133558	-0.008983	-0.055353	0.062723	0.219422
V28	-0.021053	0.014724	-0.059752	0.061458	0.215153
Amount	149.620000	2.690000	378.660000	123.500000	69.990000
Class	0.000000	0.000000	0.000000	0.000000	0.000000

Dataset information and Statistical Analysis

```
print(df.info())
df.describe()
```

```
<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
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	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	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01

8 rows × 31 columns

PREPROCESSING

```
print("before dataset size -- >> ",df.shape)
print("Missing Values -- >> ",df.isnull().sum().sum())
print("Duplicated values -- >> ",df.duplicated().sum())

df.drop_duplicates(inplace=True)
print("\nDrop Sucessfully\n")

print("After drop Missing Values -- >> ",df.isnull().sum().sum())
print("After drop Duplicated values -- >> ",df.duplicated().sum())
print("After Drop dataset size -- >> ",df.shape)
```

```
before dataset size -- >>  (284807, 31)
Missing Values -- >>  0
Duplicated values -- >>  1081

Drop Sucessfully
```

```
After drop Missing Values -- >>  0
After drop Duplicated values -- >>  0
After Drop dataset size -- >>  (283726, 31)
```

Data Visualization

Class Distribution:

- Highly Imbalanced Dataset (0 vs 1)
- Since Class 0 has 283,253 instances and Class 1 has only 473, it indicates a severe class imbalance.

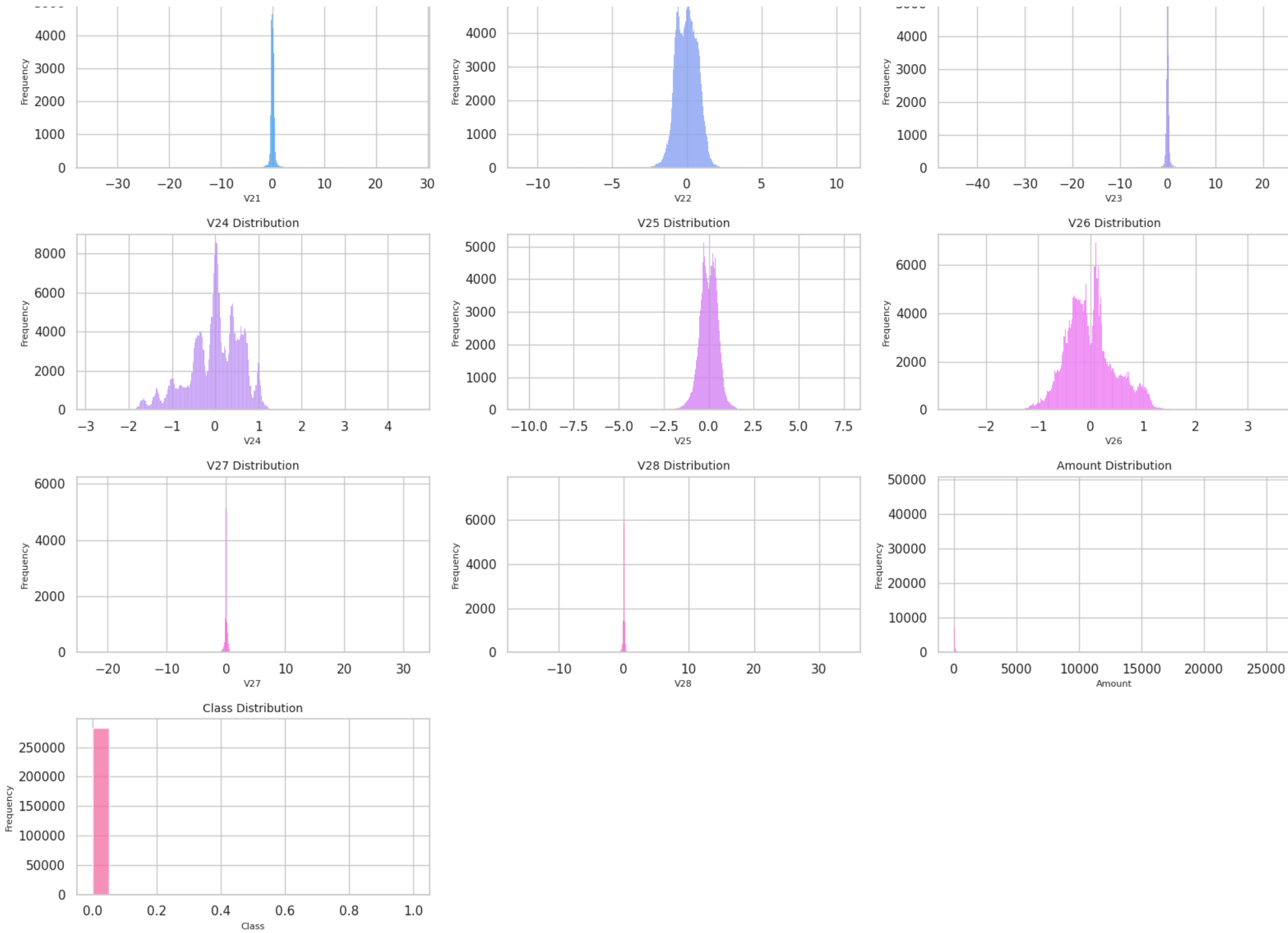
```
# Set up the figure and subplots
fig, ax = plt.subplots(1, 2, figsize=(12, 4))

# Define colors
colors = ["#ff9999", "#66b3ff"]

# Create a count plot on the first subplot
sns.countplot(x=df["Class"], palette=colors, ax=ax[0])
ax[0].set_title("Class Distribution", fontsize=14, fontweight="bold")
ax[0].set_xlabel("Class", fontsize=12)
ax[0].set_ylabel("Count", fontsize=12)

# Show value counts on top of bars
for p in ax[0].patches:
    ax[0].annotate(f'{p.get_height()}',
                   (p.get_x() + p.get_width() / 2., p.get_height()),
                   ha='center', va='baseline', fontsize=10, fontweight="bold", color='black')

# Create a pie chart on the second subplot
class_counts = df["Class"].value_counts()
ax[1].pie(class_counts, labels=class_counts.index, autopct='%1.1f%%', colors=colors, startangle=90, wedgeprops={'edgecolor': 'black'})

# Set the title for the pie chart
ax[1].set_title("Class Proportion", fontsize=14, fontweight="bold")

# Adjust layout
plt.tight_layout()
plt.show()
```

Numerical Features Distributions

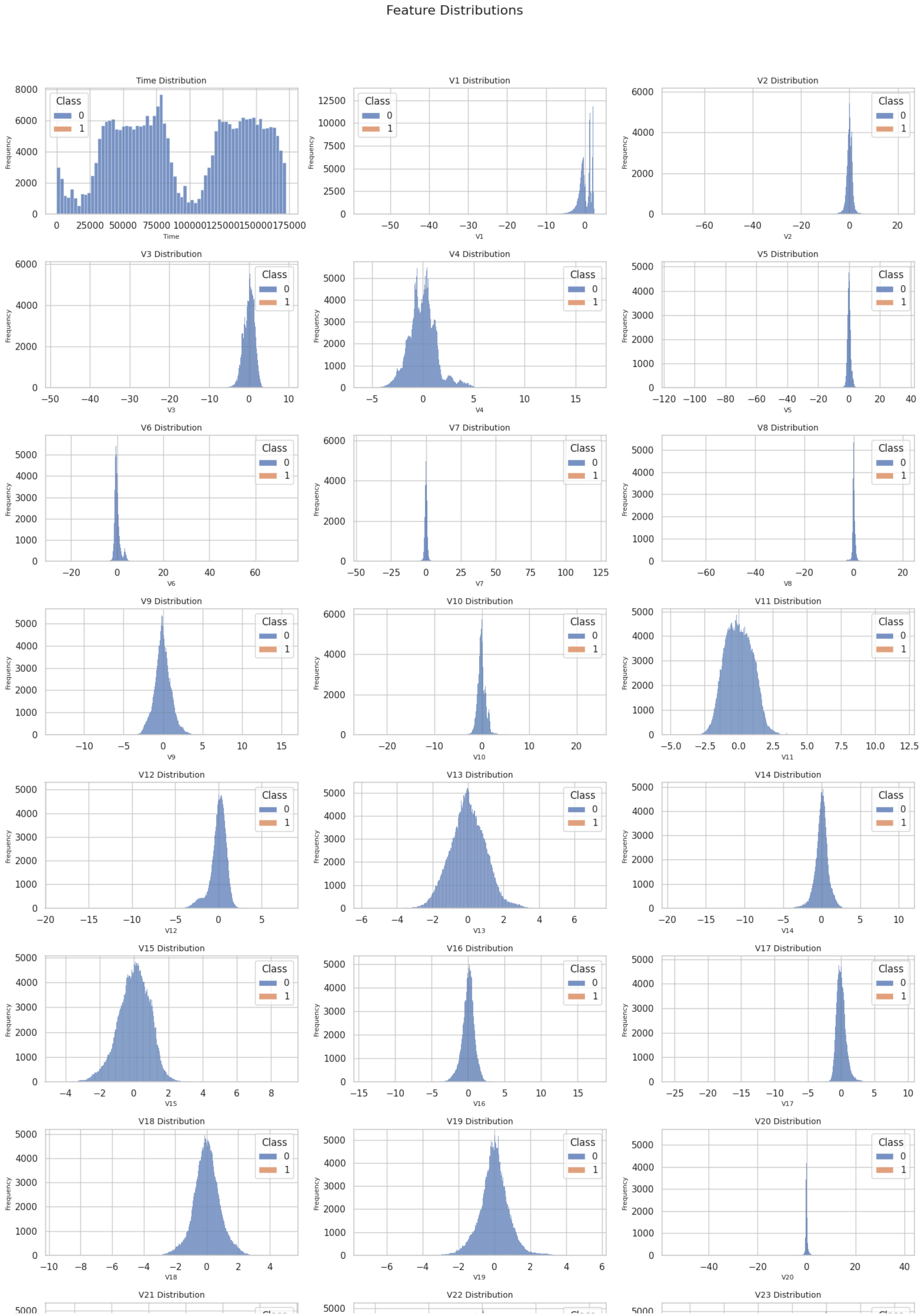
```
# Define the number of features
num_features = len(df.columns)

num_cols = 3  # Fixed number of columns
num_rows = int(np.ceil(num_features / num_cols))  # Calculate required rows
```

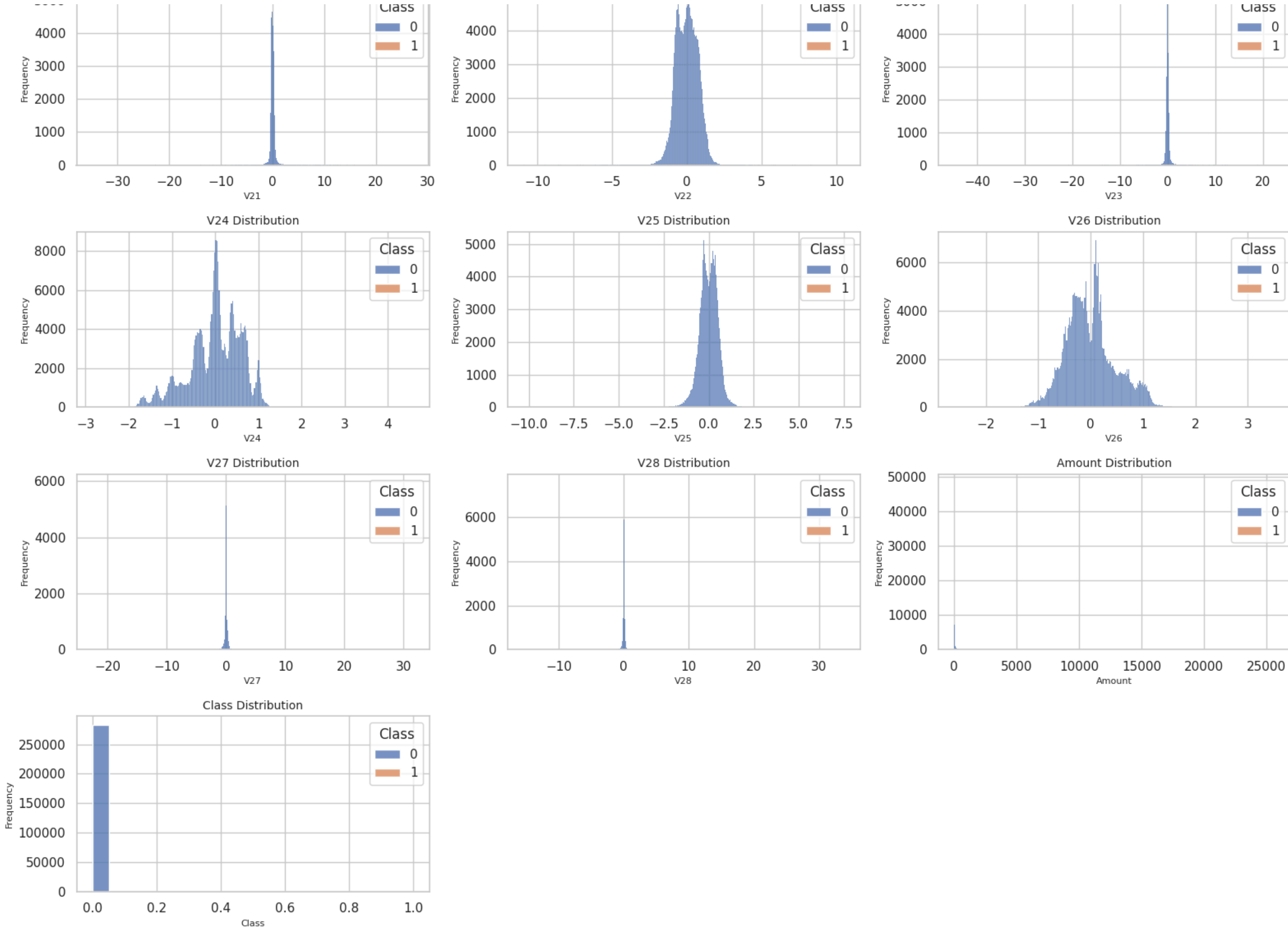
```
# Create subplots
fig, ax = plt.subplots(num_rows, num_cols, figsize=(16, num_rows * 3))
ax = ax.flatten()

# Define a color palette
colors = sns.color_palette("husl", num_features)

for i, feature in enumerate(df.columns):
    sns.histplot(df[feature], ax=ax[i], color=colors[i])

    # Set titles and labels
    ax[i].set_title(f"{feature} Distribution", fontsize=10)
    ax[i].set_xlabel(feature, fontsize=8)
    ax[i].set_ylabel("Frequency", fontsize=8)

# Hide any unused subplots
for i in range(num_features, len(ax)):
    fig.delaxes(ax[i])

# Set a main title for the entire figure
fig.suptitle("Feature Distributions", fontsize=16, y=1.02)

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


Feature Distributions
Time Distribution
V1 Distribution
V2 Distribution
V3 Distribution
V4 Distribution
V5 Distribution
V6 Distribution
V7 Distribution
V8 Distribution
V9 Distribution
V10 Distribution
V11 Distribution
V12 Distribution
V13 Distribution
V14 Distribution
V15 Distribution
V16 Distribution
V17 Distribution
V18 Distribution
V19 Distribution
V20 Distribution
V21 Distribution
V22 Distribution
V23 Distribution
Frequency
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 Distribution
V24
V25 Distribution
V25
V26 Distribution
V26
V27 Distribution
V27
V28 Distribution
V28
Amount Distribution
Amount
Class Distribution
Class
Frequency

```
num_features = len(df.columns)

num_cols = 3  # Fixed number of columns
num_rows = int(np.ceil(num_features / num_cols))  # Calculate required rows

# Create subplots
fig, ax = plt.subplots(num_rows, num_cols, figsize=(16, num_rows * 3))
ax = ax.flatten()

# Define a color palette
colors = sns.color_palette("husl", num_features)

for i, feature in enumerate(df):
    sns.histplot(df, x=feature, hue=df["Class"].astype(str), ax=ax[i], color=colors[i],multiple='stack')

    # Set titles and labels
    ax[i].set_title(f"{feature} Distribution", fontsize=10)
    ax[i].set_xlabel(feature, fontsize=8)
    ax[i].set_ylabel("Frequency", fontsize=8)

# Hide any unused subplots
for i in range(num_features, len(ax)):
    fig.delaxes(ax[i])

# Set a main title for the entire figure
fig.suptitle("Feature Distributions", fontsize=16, y=1.02)

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


Feature Distributions
Time Distribution
V1 Distribution
V2 Distribution
V3 Distribution
V4 Distribution
V5 Distribution
V6 Distribution
V7 Distribution
V8 Distribution
V9 Distribution
V10 Distribution
V11 Distribution
V12 Distribution
V13 Distribution
V14 Distribution
V15 Distribution
V16 Distribution
V17 Distribution
V18 Distribution
V19 Distribution
V20 Distribution
V21 Distribution
V22 Distribution
V23 Distribution
Frequency
Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
Class
0
1


Class
0
1
V21
V22
V23
V24 Distribution
V24
V25 Distribution
V25
V26 Distribution
V26
V27 Distribution
V27
V28 Distribution
V28
Amount Distribution
Amount
Class Distribution
Class
Frequency

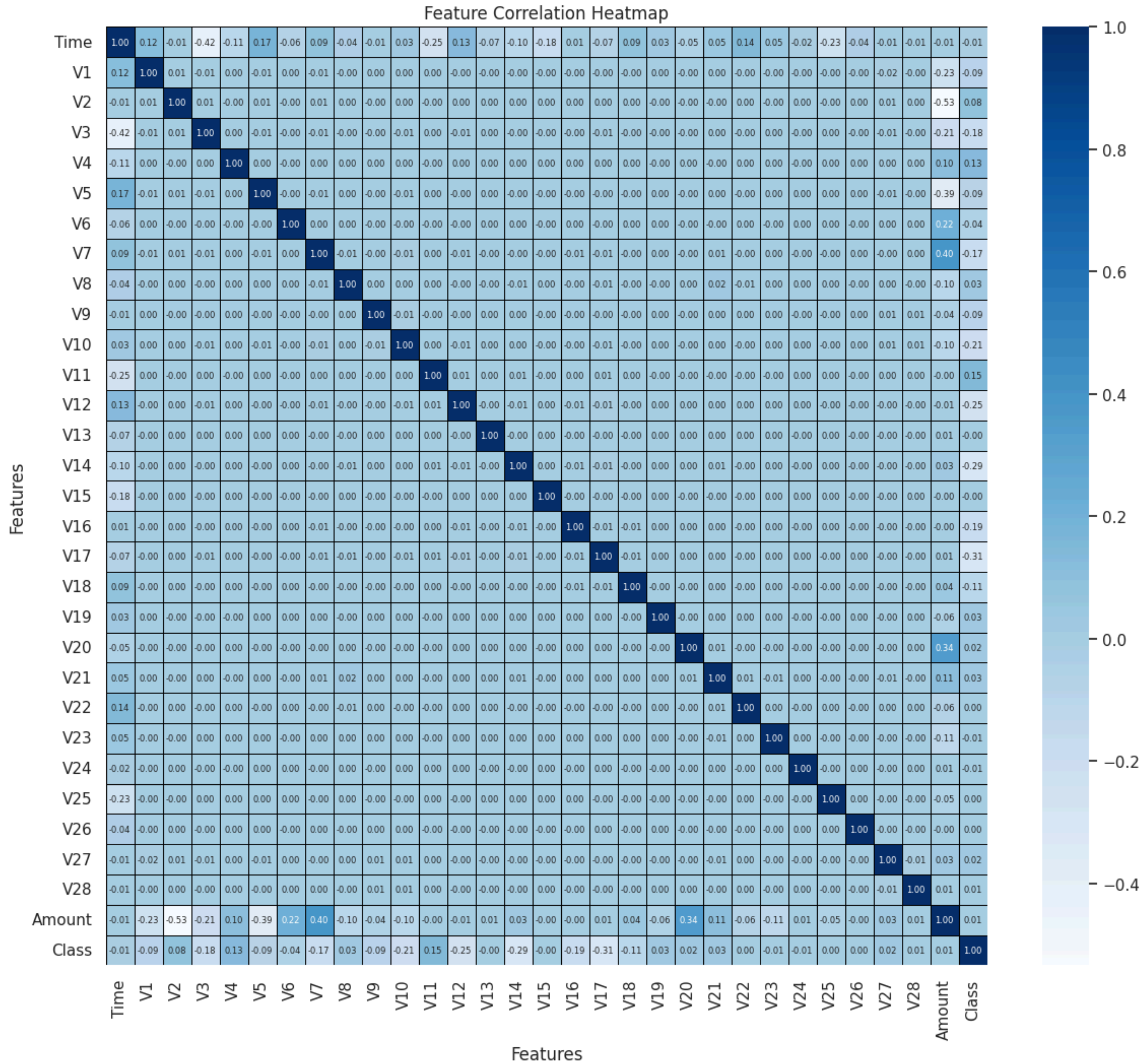
Find Corrleation between Features

```
metrix = df.corr()
metrix
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
Time	1.000000	0.117927	-0.010556	-0.422054	-0.105845	0.173223	-0.063279	0.085335	-0.038203	-0.007861	...	0.045913	0.143727	(
V1	0.117927	1.000000	0.006875	-0.008112	0.002257	-0.007036	0.000413	-0.009173	-0.001168	0.001828	...	0.002818	-0.001436	-(
V2	-0.010556	0.006875	1.000000	0.005278	-0.001495	0.005210	-0.000594	0.007425	0.002899	-0.000274	...	-0.004897	0.001237	-(
V3	-0.422054	-0.008112	0.005278	1.000000	0.002829	-0.006879	-0.001511	-0.011721	-0.001815	-0.003579	...	0.003500	-0.000275	(
V4	-0.105845	0.002257	-0.001495	0.002829	1.000000	0.001744	-0.000880	0.004657	0.000890	0.002154	...	-0.001034	0.000115	(
V5	0.173223	-0.007036	0.005210	-0.006879	0.001744	1.000000	-0.000938	-0.008709	0.001430	-0.001213	...	0.001622	-0.000559	(
V6	-0.063279	0.000413	-0.000594	-0.001511	-0.000880	-0.000938	1.000000	0.000436	0.003036	-0.000734	...	-0.002134	0.001104	-(
V7	0.085335	-0.009173	0.007425	-0.011721	0.004657	-0.008709	0.000436	1.000000	-0.006419	-0.004921	...	0.009010	-0.002280	(
V8	-0.038203	-0.001168	0.002899	-0.001815	0.000890	0.001430	0.003036	-0.006419	1.000000	0.001038	...	0.018892	-0.006156	(
V9	-0.007861	0.001828	-0.000274	-0.003579	0.002154	-0.001213	-0.000734	-0.004921	0.001038	1.000000	...	0.000679	0.000785	(
V10	0.031068	0.000815	0.000620	-0.009632	0.002753	-0.006050	-0.002180	-0.013617	0.000481	-0.012613	...	0.003777	-0.000481	(
V11	-0.248536	0.001028	-0.000633	0.002339	-0.001223	0.000411	-0.000211	0.002454	0.004688	-0.000217	...	-0.002760	-0.000150	-(
V12	0.125500	-0.001524	0.002266	-0.005900	0.003366	-0.002342	-0.001185	-0.006153	-0.004414	-0.002385	...	0.003285	0.000151	(
V13	-0.065958	-0.000568	0.000680	0.000113	0.000177	0.000019	0.000397	-0.000170	-0.001381	0.000745	...	0.000522	0.000016	(
V14	-0.100316	-0.002663	0.002711	-0.003027	0.002801	-0.001000	0.000184	-0.003816	-0.008387	0.001981	...	0.005633	-0.001906	(
V15	-0.184392	-0.000602	0.001538	-0.001230	0.000572	-0.001171	-0.000470	-0.001394	0.001044	-0.000283	...	-0.000271	-0.001197	(
V16	0.011286	-0.003345	0.004013	-0.004430	0.003346	-0.002373	0.000122	-0.005944	-0.004376	-0.000086	...	0.004326	-0.000820	(
V17	-0.073819	-0.003491	0.003244	-0.008159	0.003655	-0.004466	-0.001716	-0.008794	-0.005576	-0.002318	...	0.003560	-0.000162	(
V18	0.090305	-0.003535	0.002477	-0.003495	0.002325	-0.002685	0.000541	-0.004279	-0.001323	-0.000373	...	0.001629	-0.000533	(
V19	0.029537	0.000919	-0.000358	-0.000016	-0.000560	0.000436	0.000106	0.000846	-0.000626	0.000247	...	0.000244	0.001342	(
V20	-0.051022	-0.001393	-0.001287	-0.002269	0.000318	-0.001185	-0.000181	-0.001192	0.000271	-0.001838	...	0.005372	-0.001617	-(
V21	0.045913	0.002818	-0.004897	0.003500	-0.001034	0.001622	-0.002134	0.009010	0.018892	0.000679	...	1.000000	0.009645	-(
V22	0.143727	-0.001436	0.001237	-0.000275	0.000115	-0.000559	0.001104	-0.002280	-0.006156	0.000785	...	0.009645	1.000000	(
V23	0.051474	-0.001330	-0.003855	0.000449	0.000732	0.001183	-0.000755	0.003303	0.004994	0.000677	...	-0.006391	0.001929	'
V24	-0.015954	-0.000723	0.000701	-0.000072	-0.000120	0.000198	0.001202	-0.000384	0.000113	-0.000103	...	0.001210	-0.000031	(
V25	-0.233262	-0.000222	-0.001569	0.000425	0.000162	0.000069	0.000697	-0.000072	0.000011	-0.000275	...	-0.000872	0.000197	-(
V26	-0.041818	-0.000684	0.000253	-0.000094	0.000777	0.000390	-0.000028	0.000624	-0.001407	0.001253	...	-0.000874	-0.001495	-(
V27	-0.005171	-0.015706	0.007555	-0.007051	0.001322	-0.005798	0.000289	-0.004537	0.000613	0.008221	...	-0.005216	0.003037	-(
V28	-0.009305	-0.004861	0.001611	-0.000134	0.000231	-0.000820	0.000925	0.001657	-0.000099	0.005591	...	-0.004436	0.001392	-(
Amount	-0.010559	-0.230105	-0.533428	-0.212410	0.099514	-0.387685	0.216389	0.400408	-0.104662	-0.044123	...	0.108058	-0.064965	-(
Class	-0.012359	-0.094486	0.084624	-0.182322	0.129326	-0.087812	-0.043915	-0.172347	0.033068	-0.094021	...	0.026357	0.004887	-(

31 rows × 31 columns

```
# Plot heatmap
plt.figure(figsize=(14, 12))  # Set figure size
sns.heatmap(metrix, annot=True, cmap="Blues", fmt=".2f", linewidths=0.5,
            linecolor="black", annot_kws={"size": 6})

# title and labels for correlation matrix
plt.title("Feature Correlation Heatmap")
plt.xlabel("Features")
plt.ylabel("Features")

plt.show()
```



Feature Engineering

1. Define X and y: Separate the features and target variable.
2. Split the Data: Split the dataset into training and testing sets.
3. Scale the Dataset: Normalize the feature values for improved model performance.

```
# Define features (X) and target variable (y)
X = df.drop(columns=["Class"])  # Dropping the target column
y = df["Class"]  # Target variable
```

```
# Splitting dataset (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Print dataset sizes with better formatting
print(f"X_train Size: {X_train.shape}")
print(f"y_train Size: {y_train.shape}")
print(f"X_test Size: {X_test.shape}")
```

```
print(f"y_test Size: {y_test.shape}")
```

```
X_train Size: (226980, 30)
y_train Size: (226980,)
X_test Size: (56746, 30)
y_test Size: (56746,)
```

```
# Initialize the scaler
scaler = StandardScaler()

# Fit on training data and transform both sets
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)  # Only transform test data

# Print confirmation
print("Feature scaling applied both X_train,X_test")
```

```
Feature scaling applied both X_train,X_test
```

Model Trainings

1. LogisticRegression
2. Decission Tree
3. RandomForest
4. Gradien Boosting
5. XGBoost
6. ANN(Artificial Neural Network)

Function include:

- Model Training
- Evaluation
- Performance Metrics Visualization

Resampling

```
# Initialize SMOTE
smote = SMOTE(random_state=42)

# Resample the training data
X_train_resample, y_train_resample = smote.fit_resample(X_train, y_train)

# Check the shape of the resampled data
print(f"Original X_train shape: {X_train.shape}")
print(f"Original y_train shape: {y_train.shape}")
print(f"Resampled X_train shape: {X_train_resample.shape}")
print(f"Resampled y_train shape: {y_train_resample.shape}")
```

```
Original X_train shape: (226980, 30)
Original y_train shape: (226980,)
Resampled X_train shape: (453204, 30)
Resampled y_train shape: (453204,)
```

Model Training on Resample Dataset

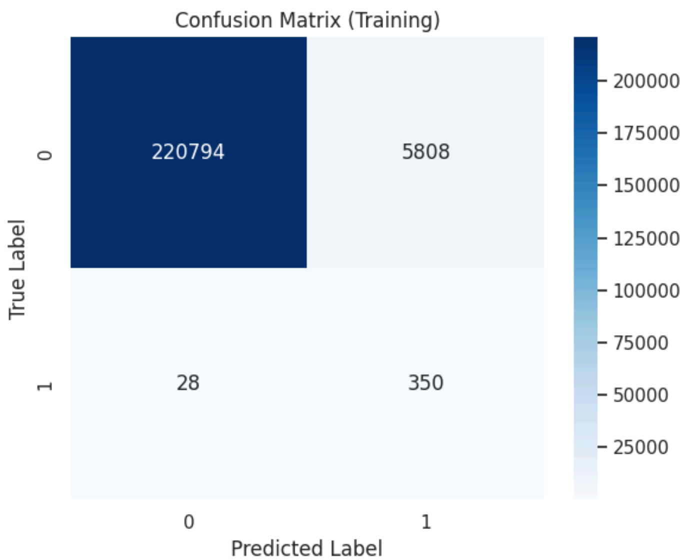
```
def model_train_evaluation_resampling(model):

    model.fit(X_train_resample, y_train_resample)

    # Make predictions on both training and test data
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
```

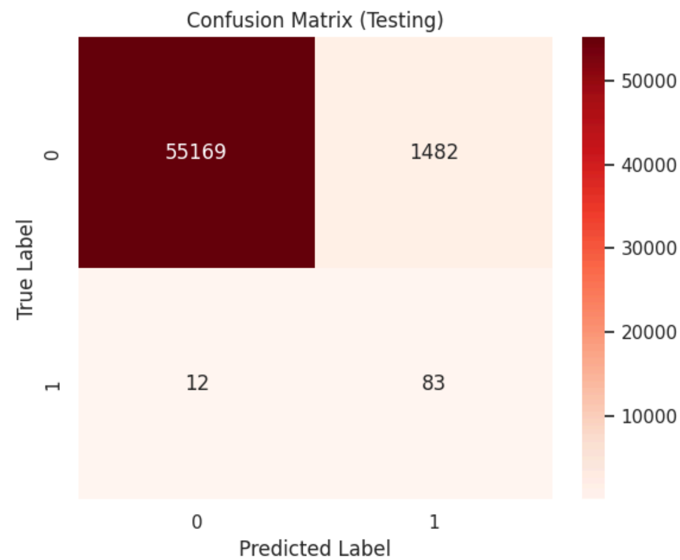
```
    # Evaluate performance
    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)

    # Print accuracy scores
    print(f"Training Accuracy: {train_acc:.4f}")
    print(f"Testing Accuracy: {test_acc:.4f}")

    # Generate classification reports
    print("\nClassification Report (Training Data):\n", classification_report(y_train, y_train_pred))
    print("\nClassification Report (Testing Data):\n", classification_report(y_test, y_test_pred))

    # Generate confusion matrices
    train_cm = confusion_matrix(y_train, y_train_pred)
    test_cm = confusion_matrix(y_test, y_test_pred)

    # Plot confusion matrix heatmaps
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))

    # Training Confusion Matrix
    sns.heatmap(train_cm, annot=True, fmt="d", cmap="Blues", ax=ax[0])
    ax[0].set_title("Confusion Matrix (Training)")
    ax[0].set_xlabel("Predicted Label")
    ax[0].set_ylabel("True Label")

    # Testing Confusion Matrix
    sns.heatmap(test_cm, annot=True, fmt="d", cmap="Reds", ax=ax[1])
    ax[1].set_title("Confusion Matrix (Testing)")
    ax[1].set_xlabel("Predicted Label")
    ax[1].set_ylabel("True Label")

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

Model --> 1 -- >> LogisticRegression

```
log_model_2 = LogisticRegression(class_weight='balanced' , random_state=42)
model_train_evaluation_resampling(log_model_2)
```

```
Training Accuracy: 0.9743
Testing Accuracy: 0.9737

Classification Report (Training Data):
              precision    recall  f1-score   support

           0       1.00      0.97      0.99    226602
           1       0.06      0.93      0.11       378

    accuracy                           0.97    226980
   macro avg       0.53      0.95      0.55    226980
weighted avg       1.00      0.97      0.99    226980


Classification Report (Testing Data):
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56651
           1       0.05      0.87      0.10        95

    accuracy                           0.97     56746
   macro avg       0.53      0.92      0.54     56746
weighted avg       1.00      0.97      0.99     56746
```

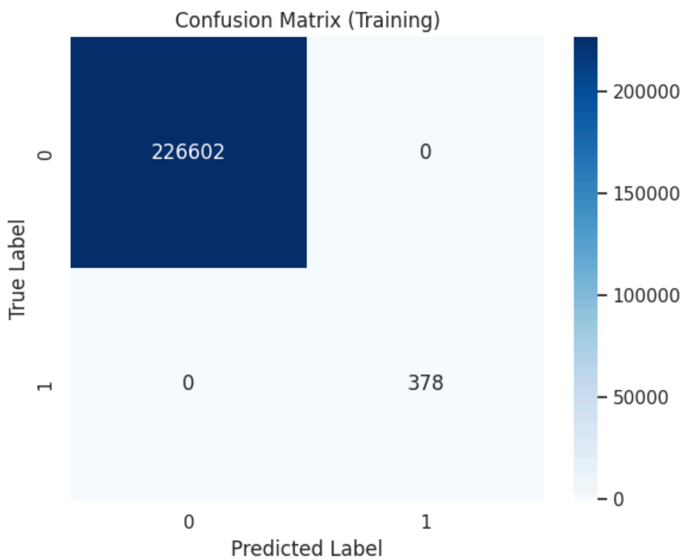


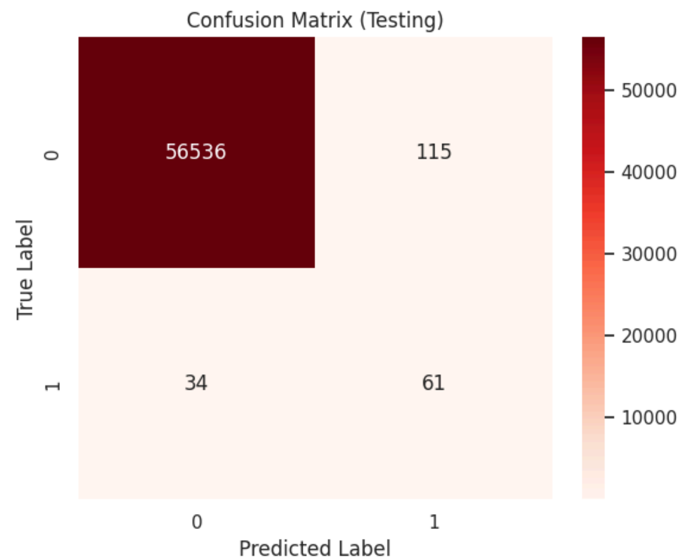


Model --> 2 -- >>Decission Tree

```
dt_model_2 = DecisionTreeClassifier()
model_train_evaluation_resampling(dt_model_2)
```

```
Training Accuracy: 1.0000
Testing Accuracy: 0.9974

Classification Report (Training Data):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226602
           1       1.00      1.00      1.00       378

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


Classification Report (Testing Data):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.35      0.64      0.45        95

    accuracy                           1.00     56746
   macro avg       0.67      0.82      0.72     56746
weighted avg       1.00      1.00      1.00     56746
```





Model --> 3 -- >> Random Forest

```
rf_model_2 = RandomForestClassifier(n_estimators=500,class_weight='balanced',random_state=42,criterion='gini',n_jobs=-1)
model_train_evaluation_resampling(rf_model_2)
```