project2

May 15, 2025

Project - Predictive Analysis for Credit Card Fraud Detection Using Classification Models

---

Problem Statement:

In the financial industry, fraudulent credit card transactions can cause significant losses and damage customer trust. The challenge lies in identifying fraudulent transactions among millions of legitimate ones, especially when fraud cases represent less than 0.2% of all transactions. This project aims to develop and evaluate classification models that can accurately detect credit card fraud, even in the presence of severe class imbalance.

Dataset Information:

Source: Kaggle - Credit Card Fraud Detection Dataset

Link: Kaggle Dataset

Records: 284,807 transactions

Features: 30 columns (28 PCA features V1-V28, plus Time, Amount)

Target Column: Class (0 = legitimate, 1 = fraud)

Initial Exploration & Preprocessing:

1 **Data Loading from ZIP:**

Loaded creditcard.csv directly from ZIP using zipfile.ZipFile

2 **Exploratory Data Analysis (EDA):**

Used df.describe() to understand feature distributions

Observed heavy class imbalance (0 » 1)

Univariate Analysis:

Plotted histograms for individual feature columns using sns.histplot

Identified skewness in many features

Bivariate Analysis:

Plotted distributions of each feature vs. Class using overlaid histograms

Helped identify which features separate frauds from normal transactions

Target Analysis:

Fraud class (1) is only 0.17% of data

3 **Data Preprocessing:**

Feature Scaling:

Scaled Amount and Time using StandardScaler

Splitting Data:

Used train_test_split(X, y, test_size=0.2, stratify=y) to ensure balanced splits

Handling Class Imbalance:

Used SMOTE to oversample minority class in training data

Post-SMOTE: Balanced dataset with equal 0s and 1s

4 **Model Building:**

Algorithm used to create a model:

Logistic Regression (Baseline)

Random Forest Classifier

K-Nearest Neighbors (KNN)

Loading libraries and Data

[2]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
# Suppress future warnings
import warnings
warnings.filterwarnings('ignore')
```

[4]:
```
# Define the path to the zip file dataset
zip_path = r'C:\Users\hp\Downloads\creditcard.csv.zip'

# Open the zip file
with zipfile.ZipFile(zip_path) as z:

     # Get the CSV filename (first file in the archive)
    csv_file = z.namelist()[0]

    # Read the CSV file
    with z.open(csv_file) as f:
```

```python
        # Read the CSV file
        data = pd.read_csv(f)

# Display the first 5 rows
print(data.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]
```

[5]:
```python
# Returns a tuple representing (rows, columns)
data.shape
```

[5]:
```
(284807, 31)
```

concept of pca , what are principle componenets , aurthonalaty , multyquonarity , eigen values , eigen vector

Column Dictionary

Time - Time in seconds elapsed between this transaction and the first transaction in the dataset.

V1-V28 - Principal components obtained from a PCA (Principal Component Analysis) transformation to anonymize the features (original features like name, merchant, location are not available).

Amount - The transaction amount. Useful for scaling and normalization.

Class - Target variable: 0 for non-fraud, 1 for fraud.

[7]:
```python
# see column data type and some info
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

[9]:
```
# see missing value in each column
data.isna().sum()
```

[9]:
```
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
```

[19]:
```
data.describe()
```

[19]:
```
                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  1.213481e-16 -2.406331e-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
```

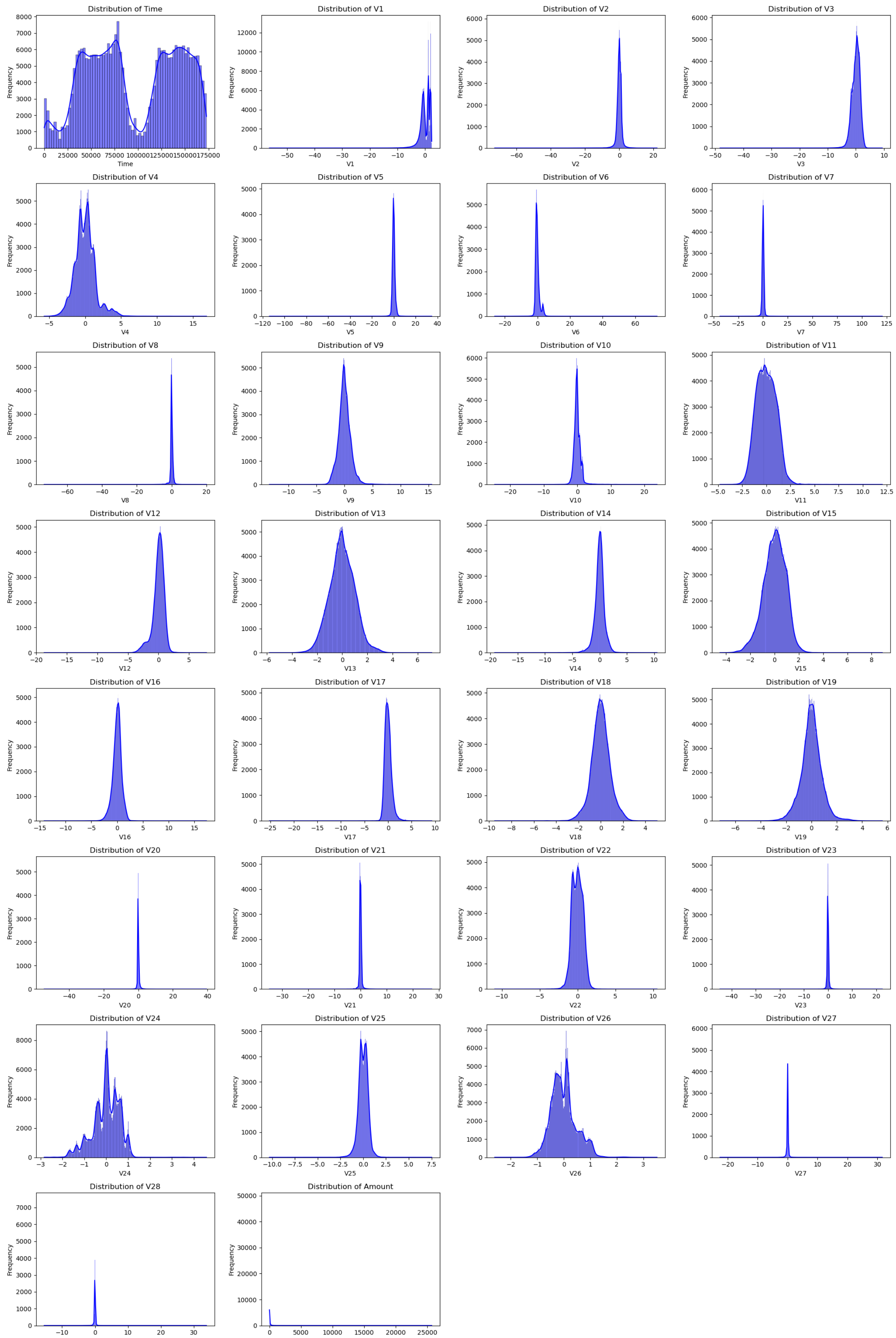
```
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       …           V21           V22           V23           V24  \
count  …  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   …  1.654067e-16 -3.568593e-16  2.578648e-16  4.473266e-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   5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-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 & Visualizations

[10]:
```
#separate the features and target value
A = data.drop("Class", axis=1)  # Features
B = data["Class"]               # Target
```

[35]:
```
#Visualizations of Features value
# Create a histplot for the feature variable

plt.figure(figsize=(20, 30))

for i , col in enumerate(A):
    plt.subplot(8, 4, i + 1)
```

```
    sns.histplot(data=data, x=col, kde=True, color='blue')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

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


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

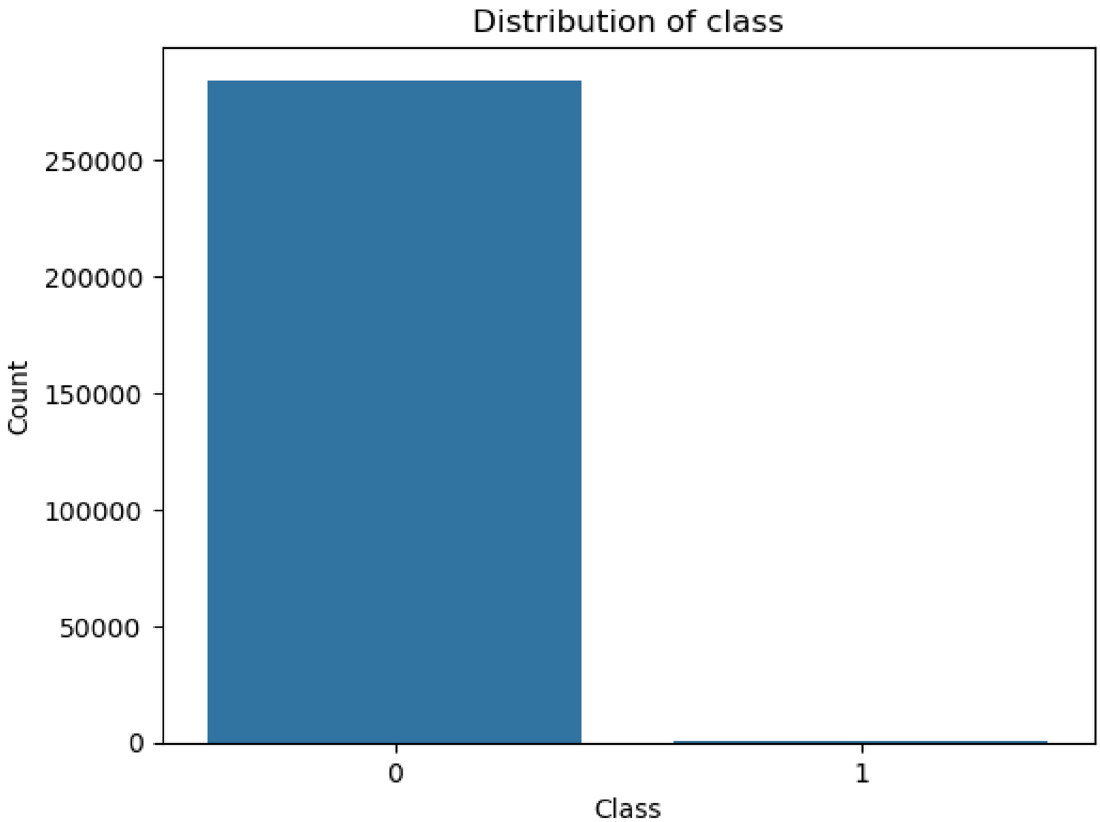
Columns Distribution

Time - Insight : The distribution shows a right skew, indicating that most transactions occur within a certain time frame, with fewer transactions occurring at the extremes. This suggests that the majority of transactions are clustered around specific times, while a few occur at much later times.

V1-V28 - Insight : These features exhibit various distributions. Some columns show a more concentrated range of values, while others have a wider spread. The presence of both positive and negative values indicates that these features capture different aspects of the transactions, possibly related to the underlying patterns of the data.

Amount - Insight : The histogram indicates that most transactions are relatively small, with a few outliers representing significantly larger amounts. This is typical in financial datasets where most transactions are low-value, but there are occasional high-value transactions that could be significant.

[11]:
```
#Visualizations of Target value
# Create a count plot for the target variable 'Class'
sns.countplot(x=B, data=data)
plt.title(f'Distribution of {"class"}')
plt.xlabel("Class")
plt.ylabel('Count')
plt.show()
```



Class - Insight : The count plot shows a significant imbalance between the classes, with a vast majority of transactions being non- fraudulent (0) compared to fraudulent (1). This imbalance is crucial for modeling, as it may require techniques to handle class imbalance effectively.

Bivariate Analysis & Visualizations

[15]:
```
# Calculate correlations between all features and the 'Class' column
correlations = data.corr()["Class"].sort_values(ascending=False)
print(correlations)
```

```
Class     1.000000
V11       0.154876
V4        0.133447
V2        0.091289
V21       0.040413
V19       0.034783
V20       0.020090
V8        0.019875
V27       0.017580
V28       0.009536
Amount    0.005632
V26       0.004455
```

```
V25       0.003308
V22       0.000805
V23      -0.002685
V15      -0.004223
V13      -0.004570
V24      -0.007221
Time     -0.012323
V6       -0.043643
V5       -0.094974
V9       -0.097733
V1       -0.101347
V18      -0.111485
V7       -0.187257
V3       -0.192961
V16      -0.196539
V10      -0.216883
V12      -0.260593
V14      -0.302544
V17      -0.326481
Name: Class, dtype: float64
```

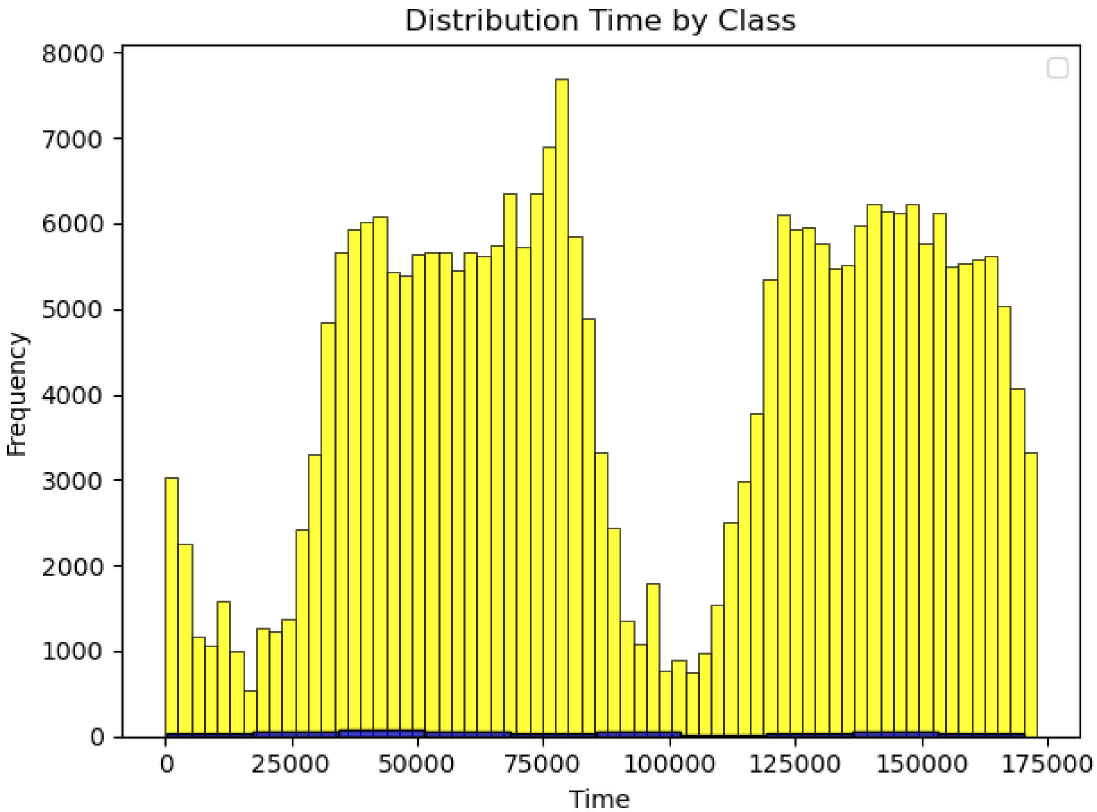
Observations

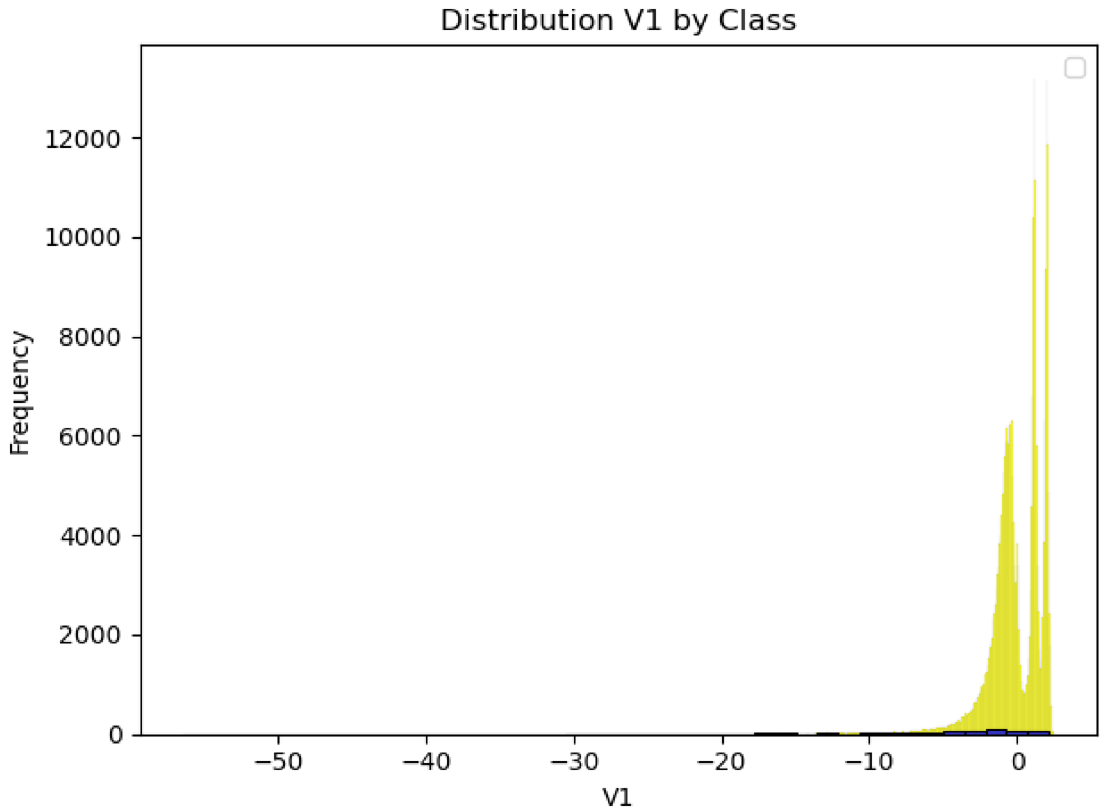
Strong Correlations: The features V11 and V4 show a strong positive correlation, indicating that as one increases, the other tends to increase as well. This could suggest that they are capturing similar underlying patterns in the data.

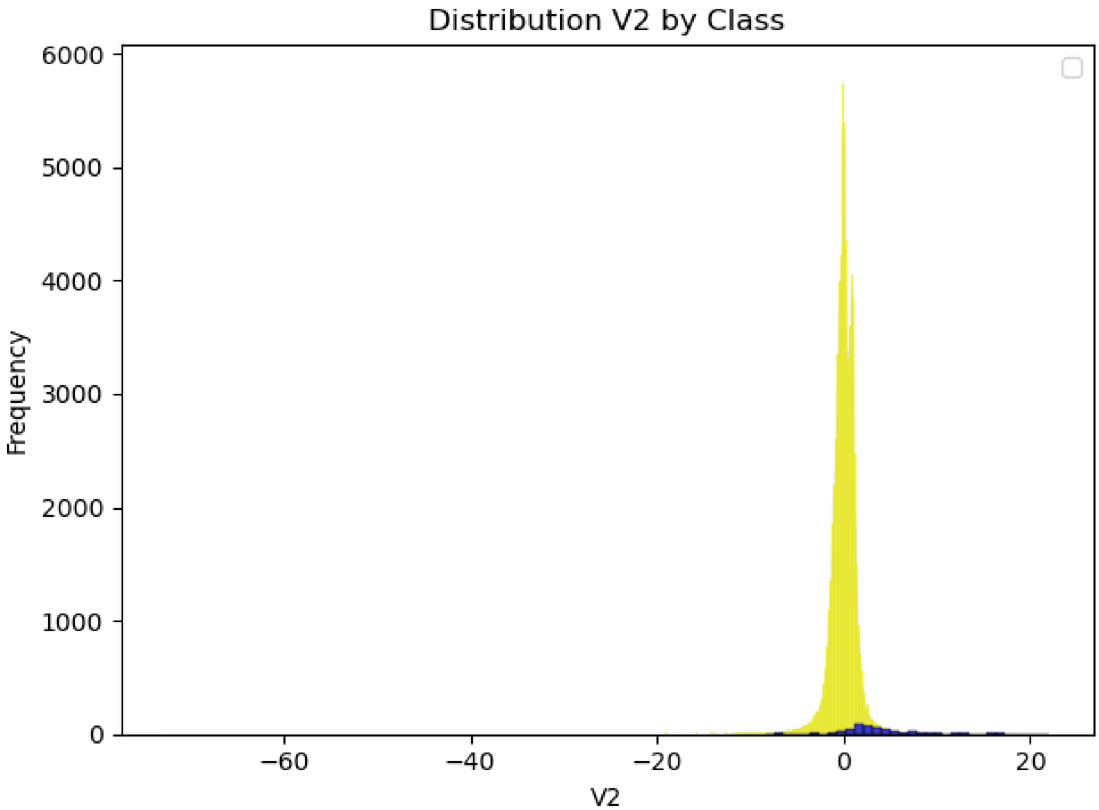
Weak Correlations: Most of the other features exhibit weak correlations with each other, which is common in high-dimensional datasets. This suggests that the features are relatively independent of one another.

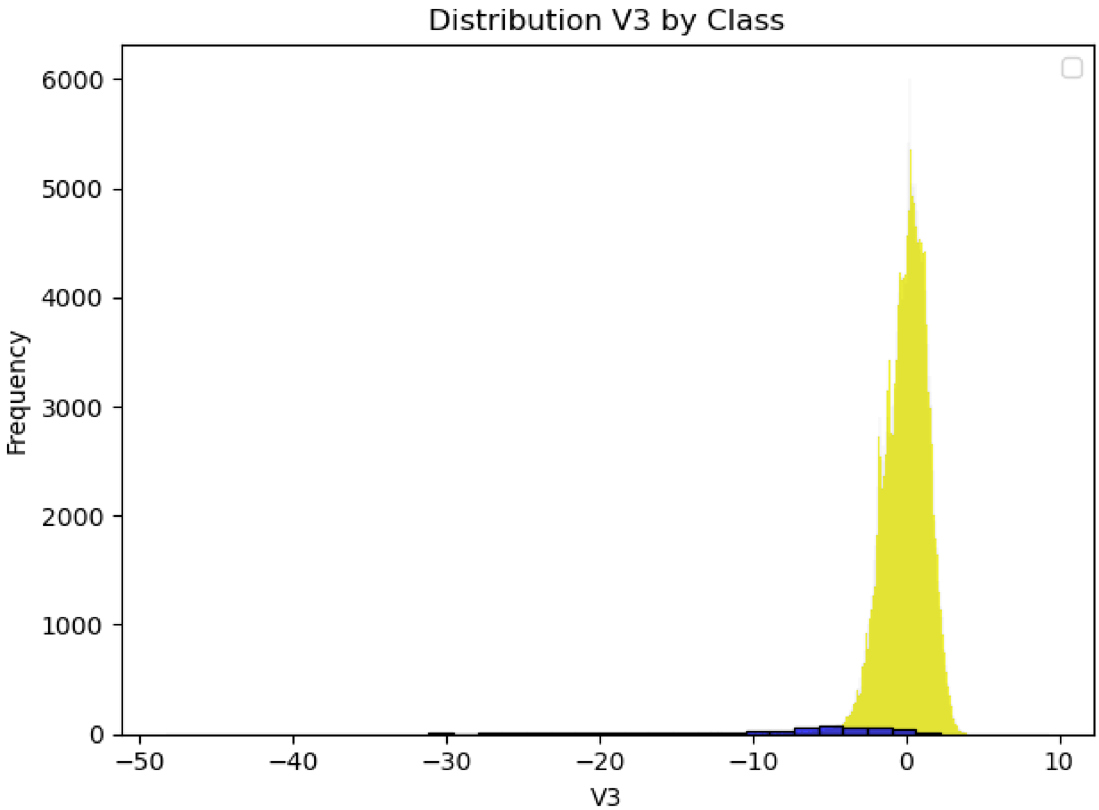
Target Variable (Class): The target variable (Class) shows a weak negative correlation with several features, indicating that there is no strong linear relationship between the features and the likelihood of fraud. This is typical in fraud detection datasets, where the relationship may be more complex.

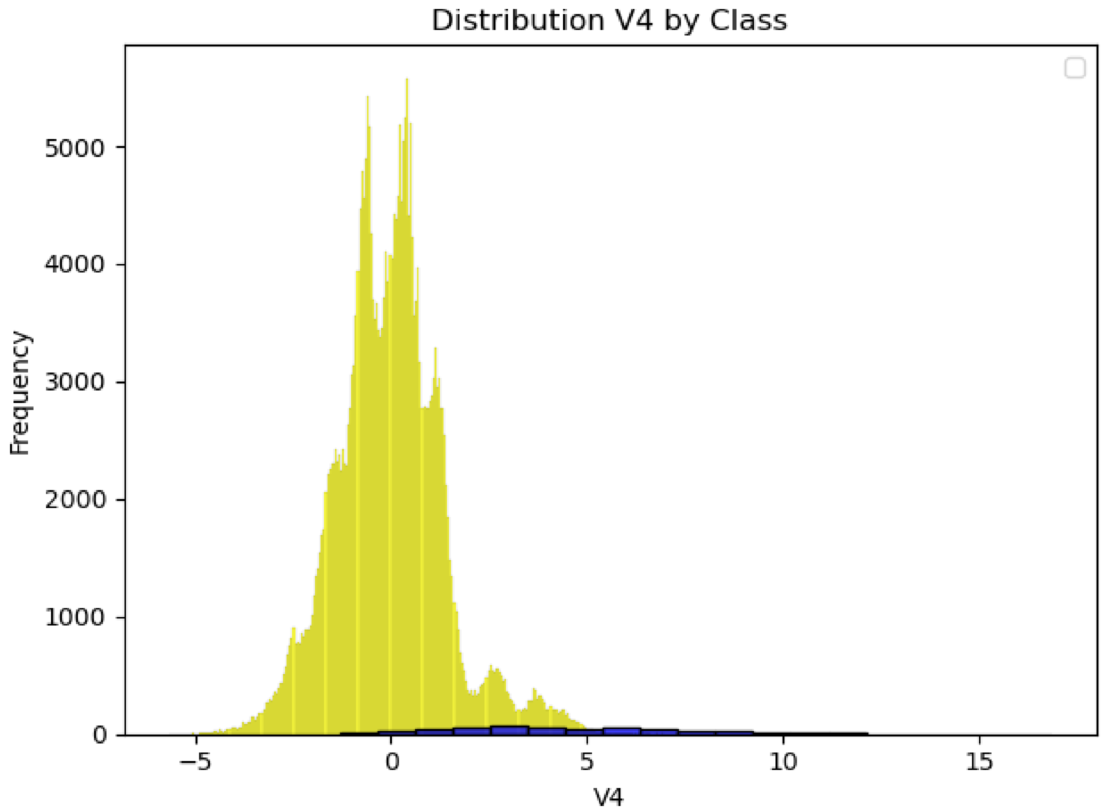
[9]:
```
# Compare feature  vs. target columns
for i in A:
    sns.histplot(x=A[i][data['Class'] == 0], color='yellow')
    sns.histplot(x=A[i][data['Class'] == 1], color='blue')

    plt.title(f'Distribution {i} by Class')
    plt.xlabel(i)
    plt.ylabel('Frequency')
    plt.legend()
    plt.tight_layout()
    plt.show()
```


Distribution Time by Class
8000
7000
6000
5000
4000
3000
2000
1000
0
Frequency
0
25000
50000
75000
100000
125000
150000
175000
Time

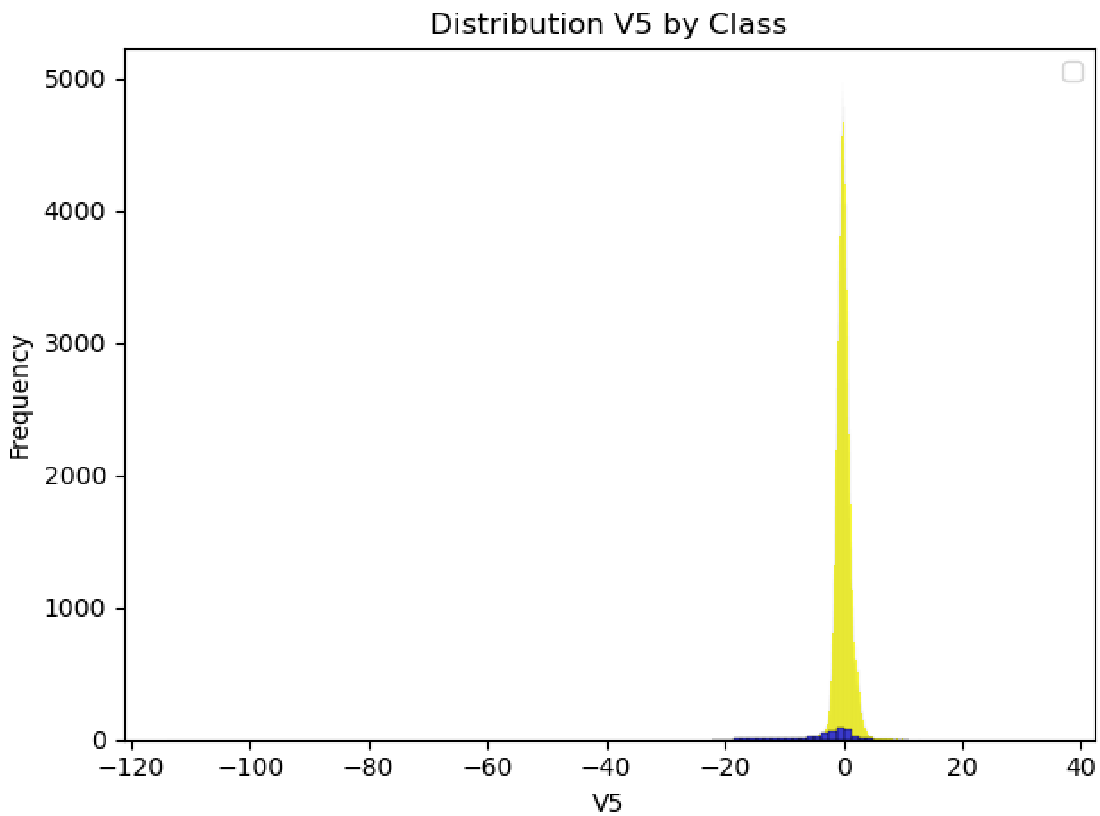

Distribution V1 by Class
Frequency
12000
10000
8000
6000
4000
2000
0
−50
−40
−30
−20
−10
0
V1

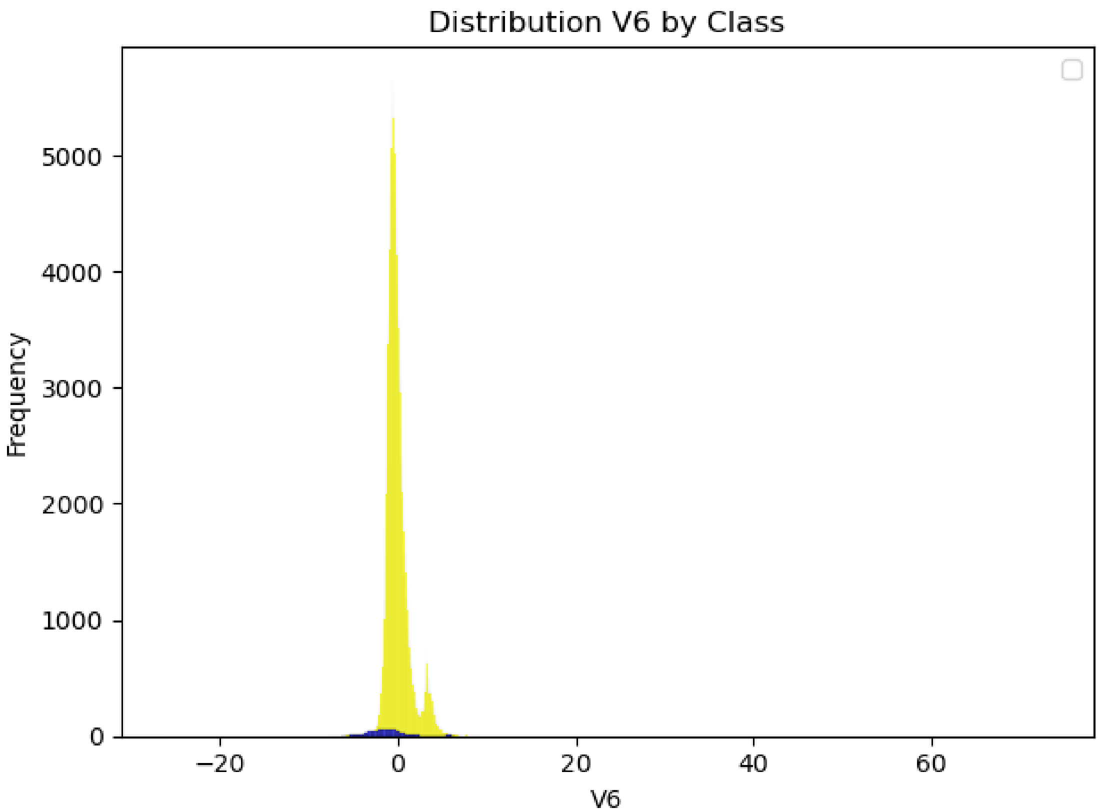

Distribution V2 by Class
6000
5000
4000
3000
2000
1000
0
Frequency
−60
−40
−20
0
20
V2

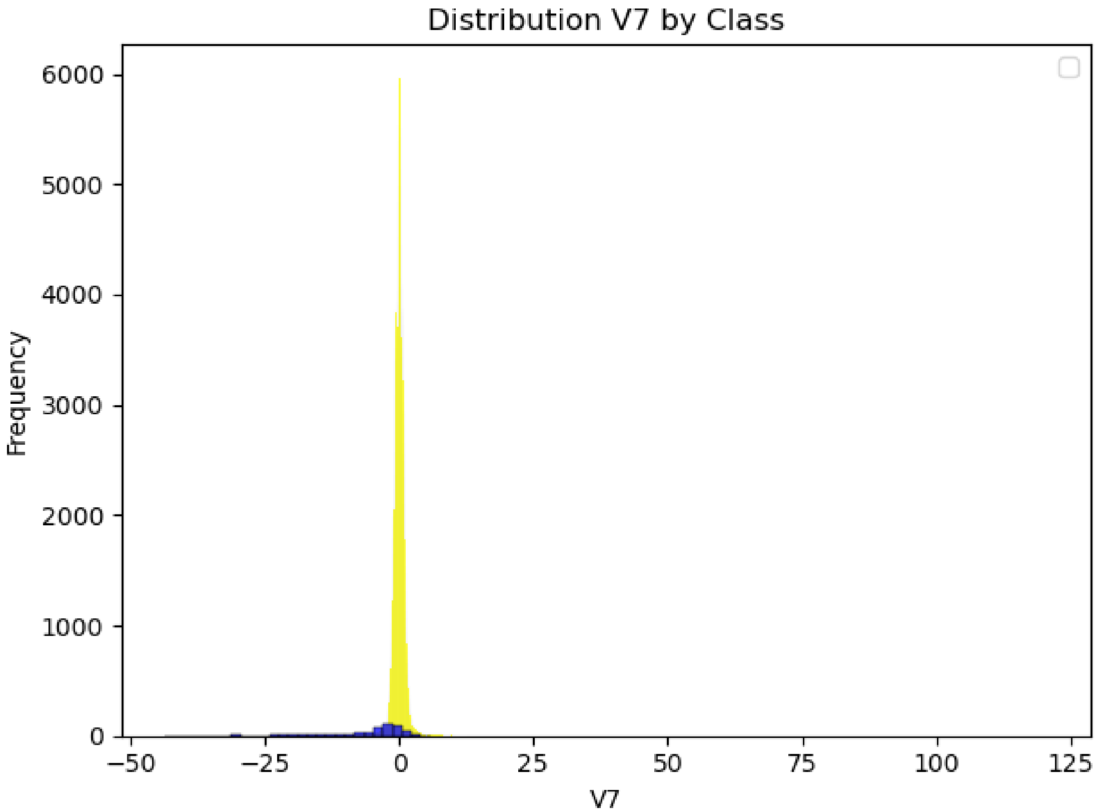

Distribution V3 by Class
Frequency
V3
6000
5000
4000
3000
2000
1000
0
−50
−40
−30
−20
−10
0
10



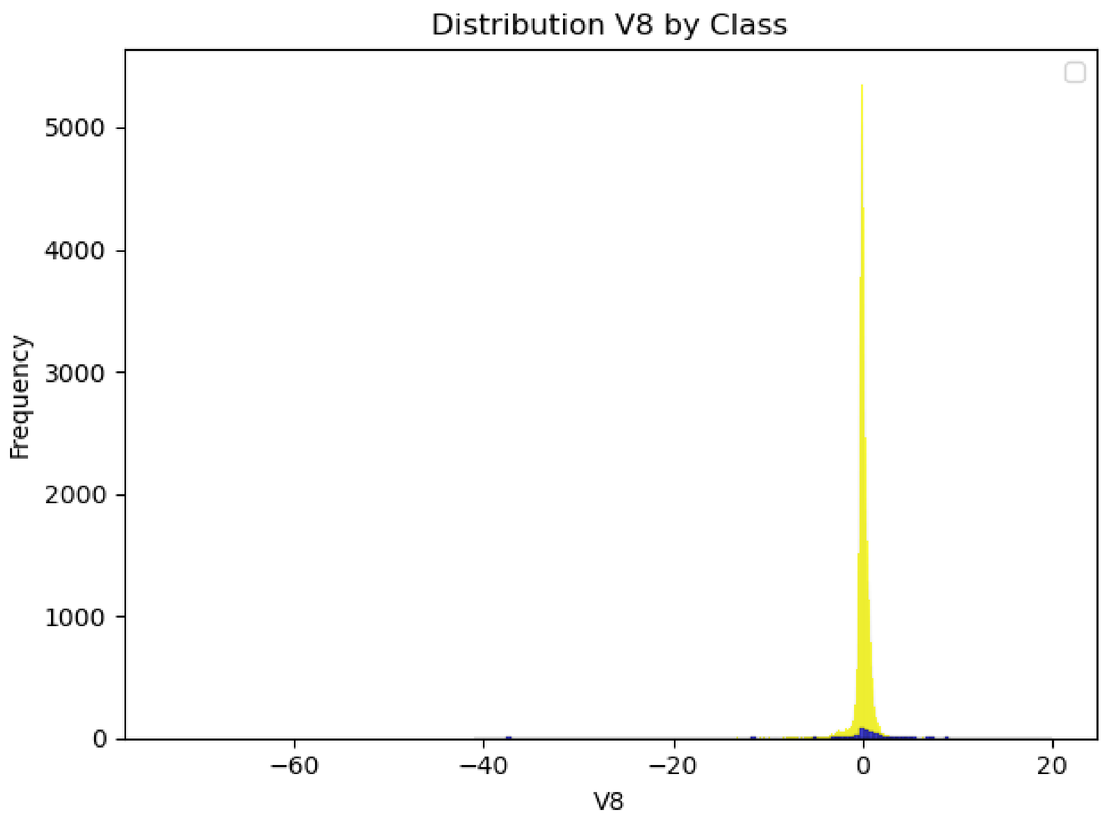
Distribution V4 by Class
Frequency
V4
5000
4000
3000
2000
1000
0
−5
0
5
10
15


Distribution V5 by Class
5000
4000
3000
2000
1000
0
Frequency
−120
−100
−80
−60
−40
−20
0
20
40
V5

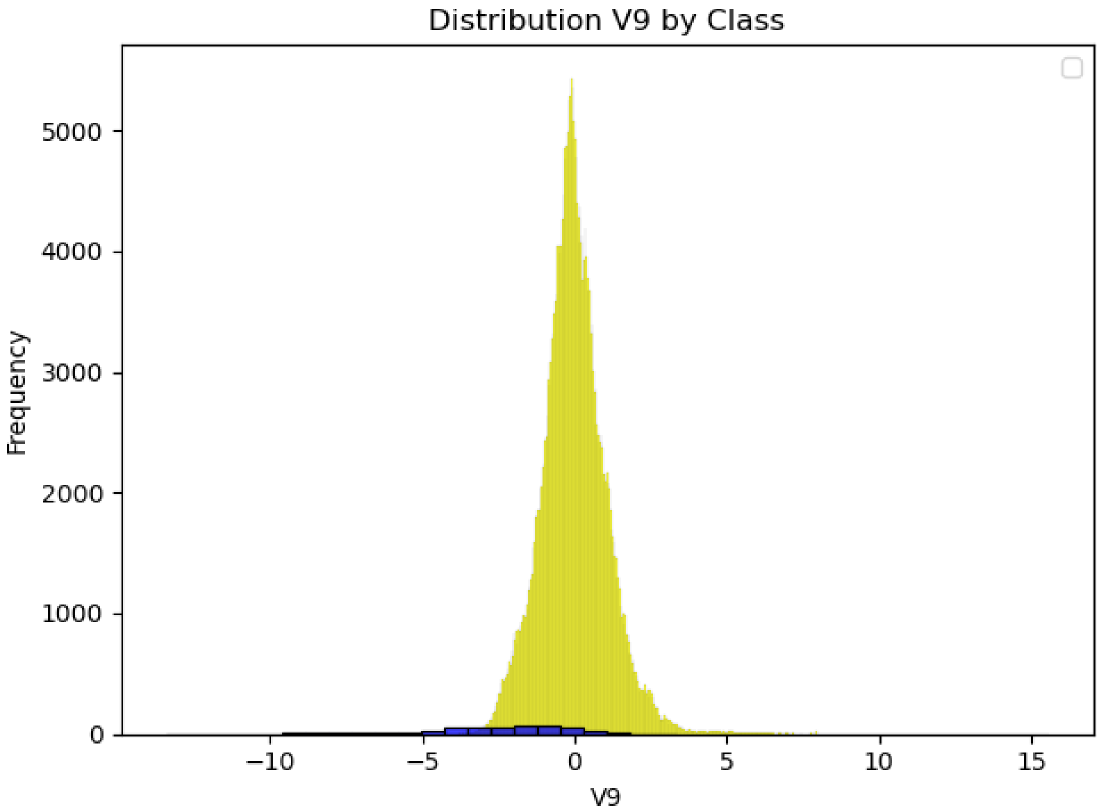

Distribution V6 by Class
Frequency
V6
5000
4000
3000
2000
1000
0
−20
0
20
40
60

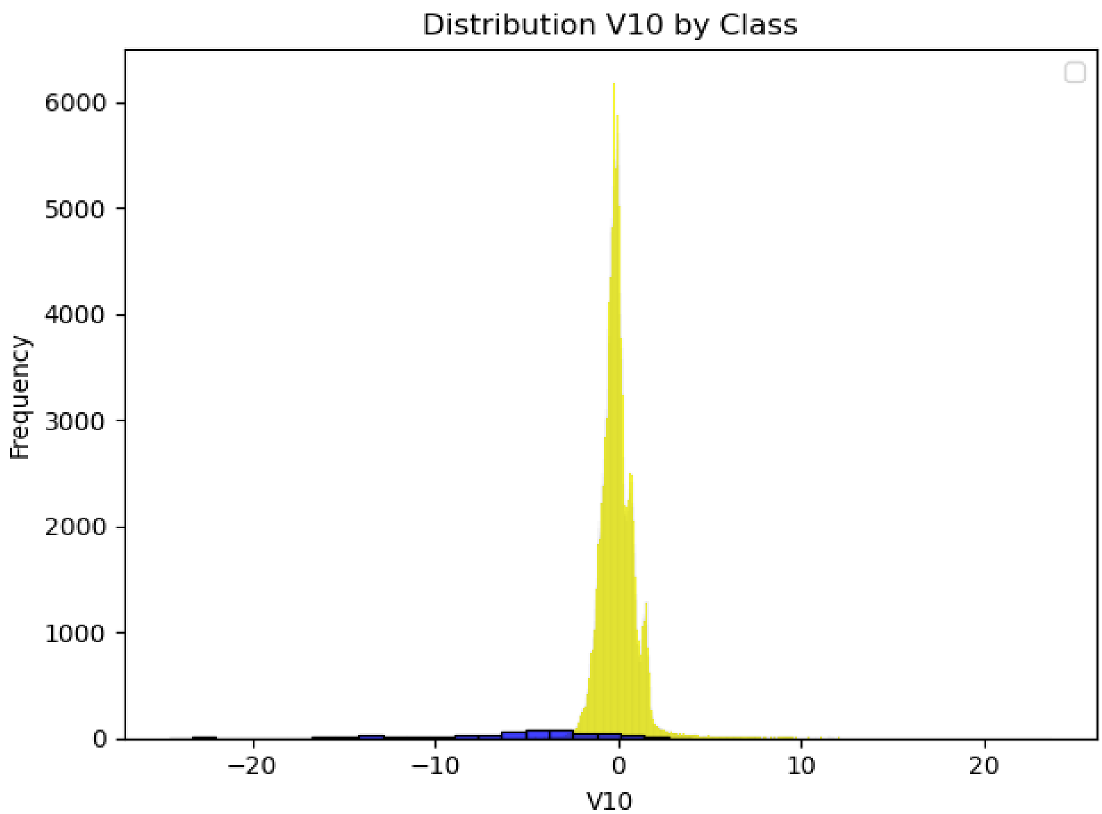


Distribution V7 by Class
6000
5000
4000
3000
2000
1000
0
Frequency
−50
−25
0
25
50
75
100
125
V7

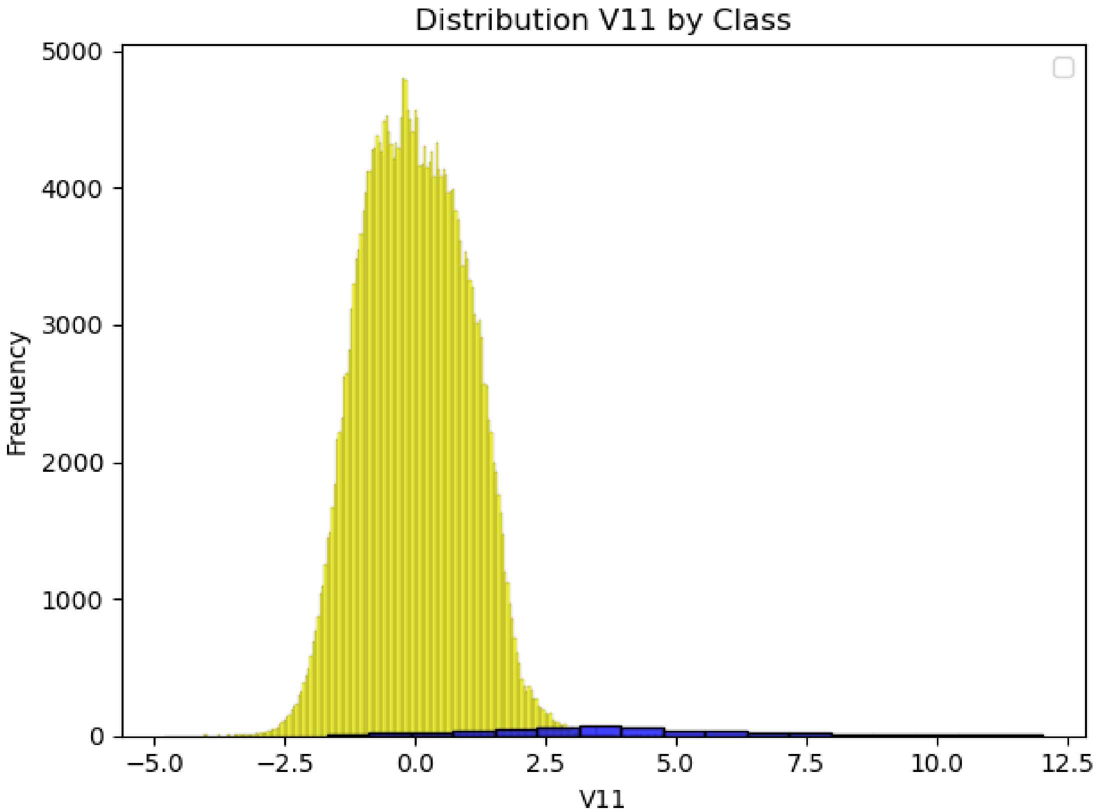


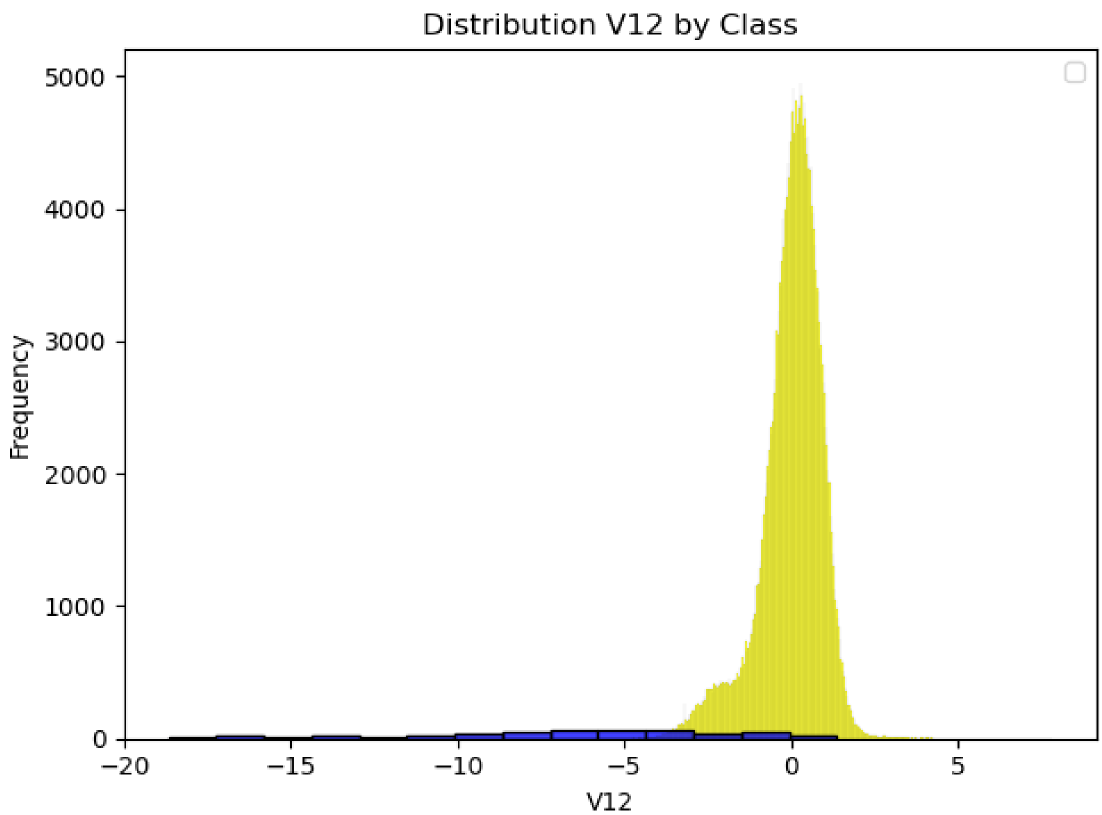
Distribution V8 by Class
Frequency
5000
4000
3000
2000
1000
0
−60
−40
−20
0
20
V8

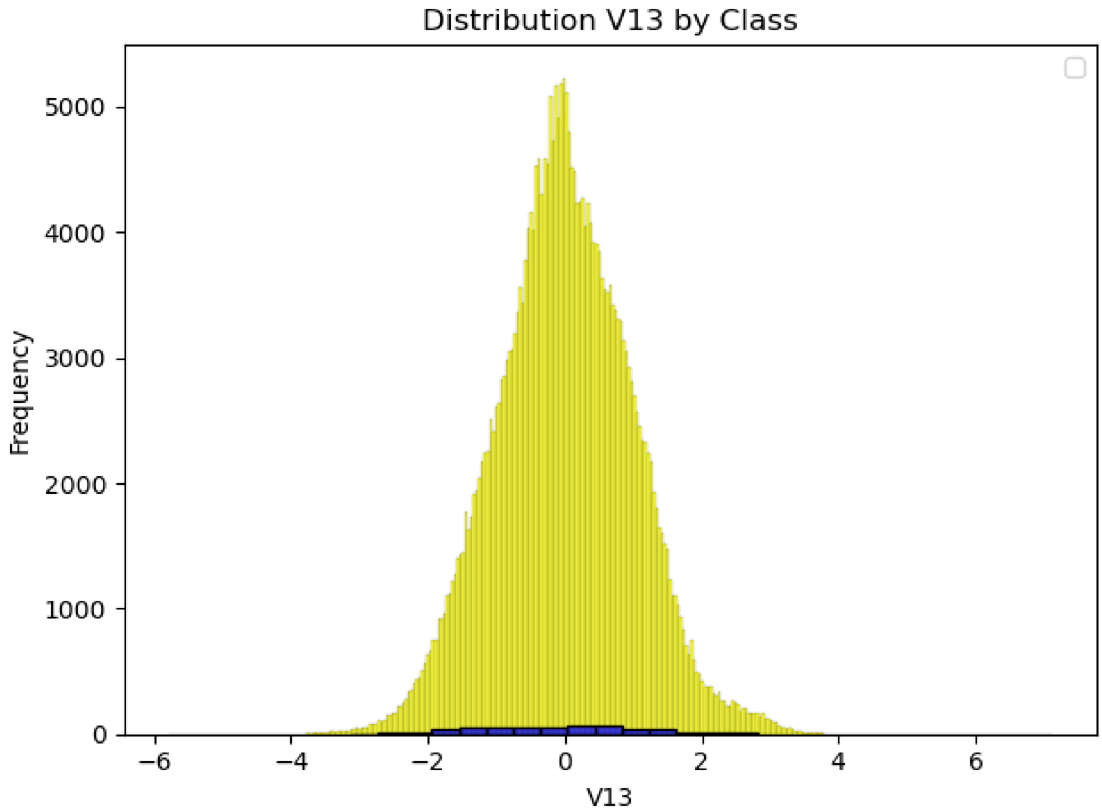

Distribution V9 by Class
Frequency
5000
4000
3000
2000
1000
0
−10
−5
0
5
10
15
V9

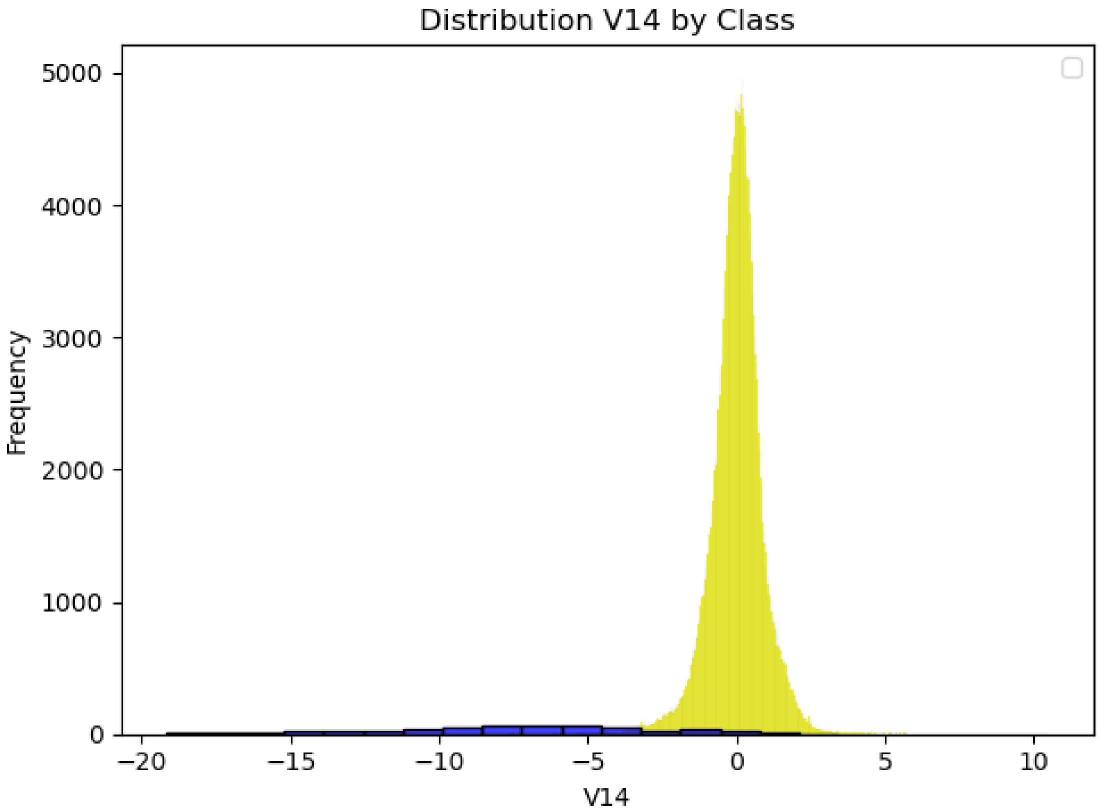


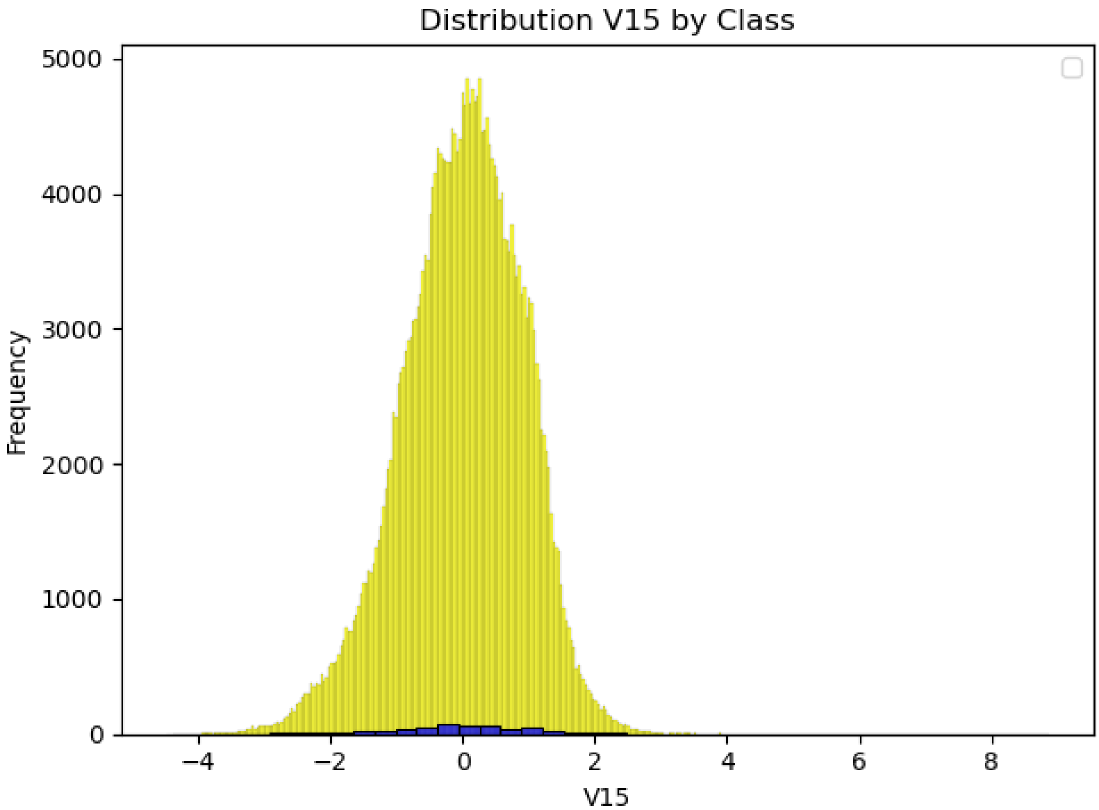
Distribution V10 by Class
6000
5000
4000
3000
2000
1000
0
Frequency
−20
−10
0
10
20
V10


Distribution V11 by Class
Frequency
5000
4000
3000
2000
1000
0
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
V11

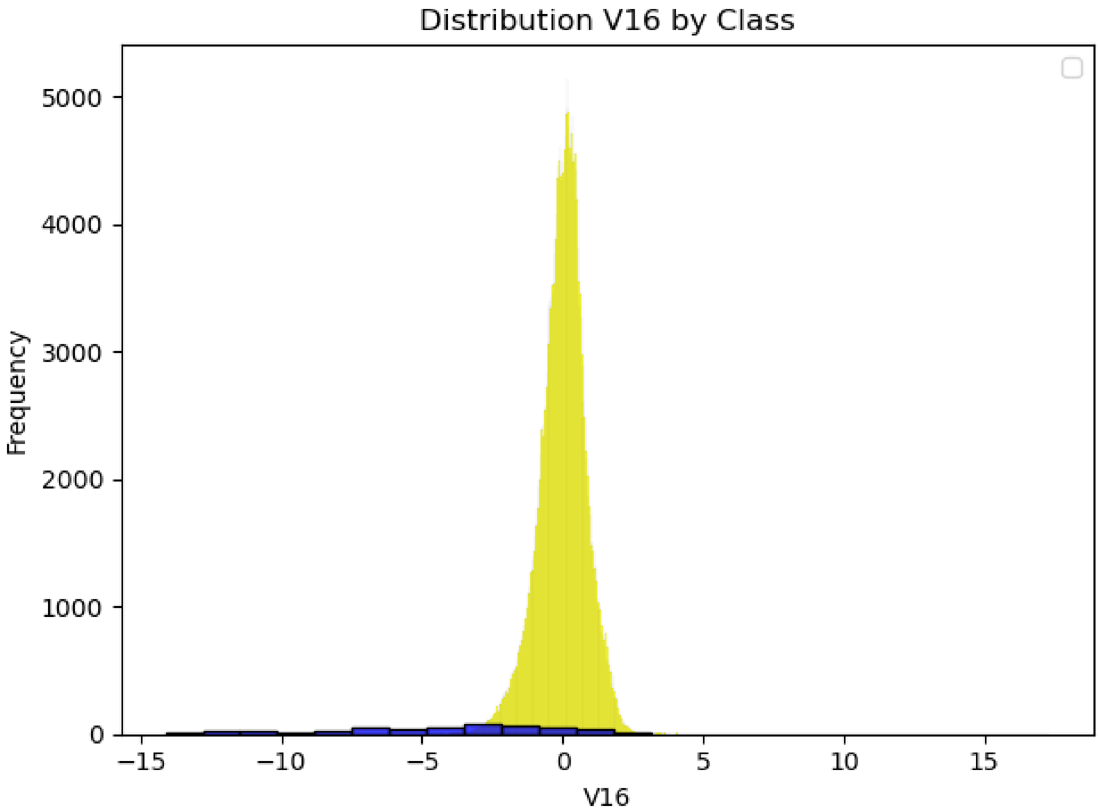

Distribution V12 by Class
5000
4000
3000
2000
1000
0
Frequency
−20
−15
−10
−5
0
5
V12

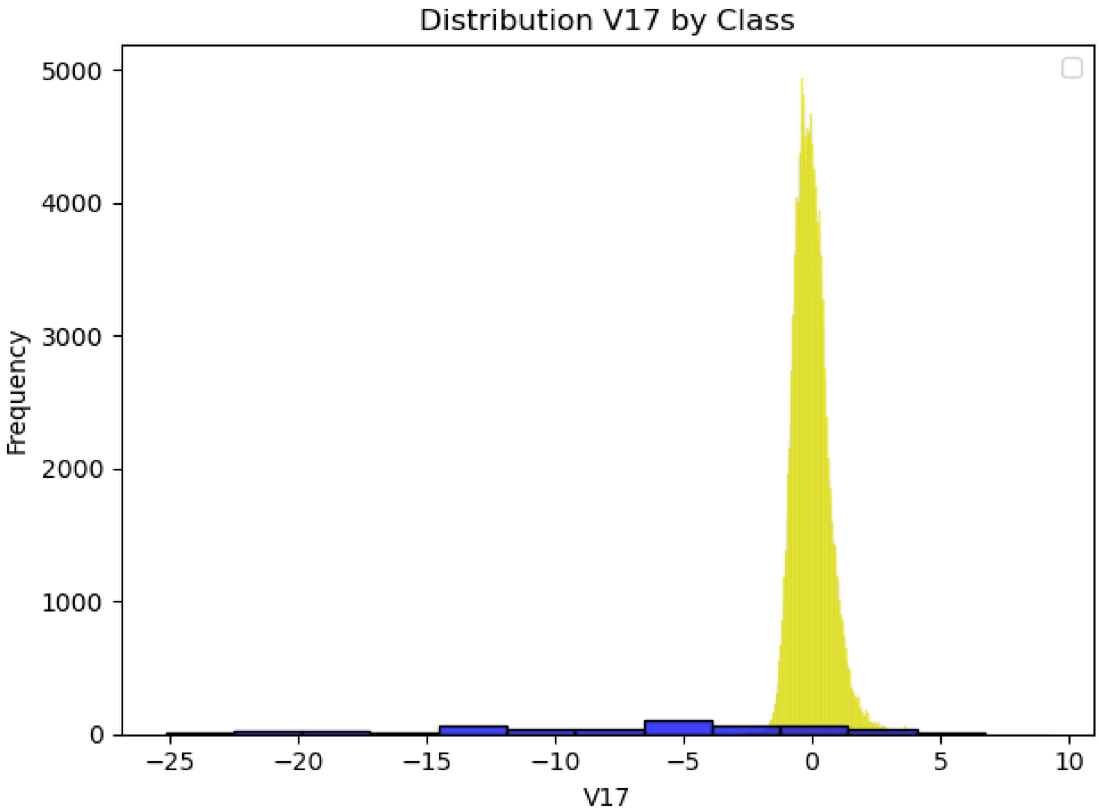

Distribution V13 by Class
Frequency
0
1000
2000
3000
4000
5000
−6
−4
−2
0
2
4
6
V13

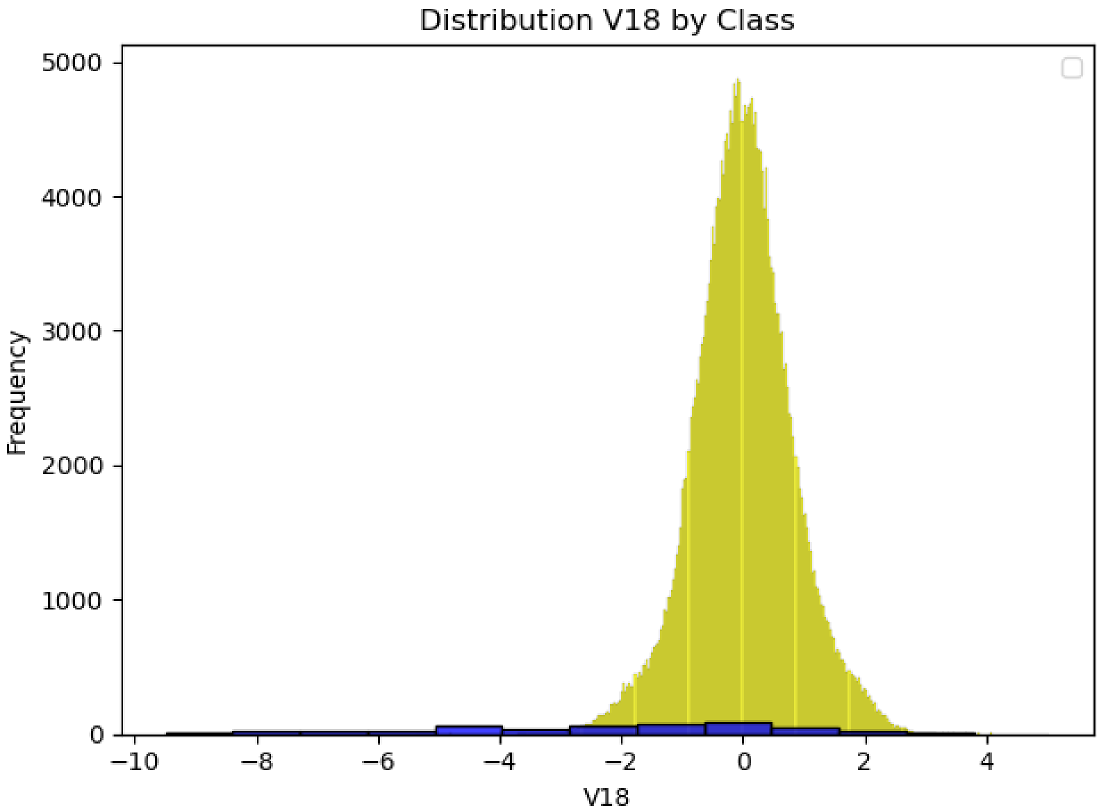

Distribution V14 by Class
5000
4000
3000
2000
1000
0
Frequency
−20
−15
−10
−5
0
5
10
V14



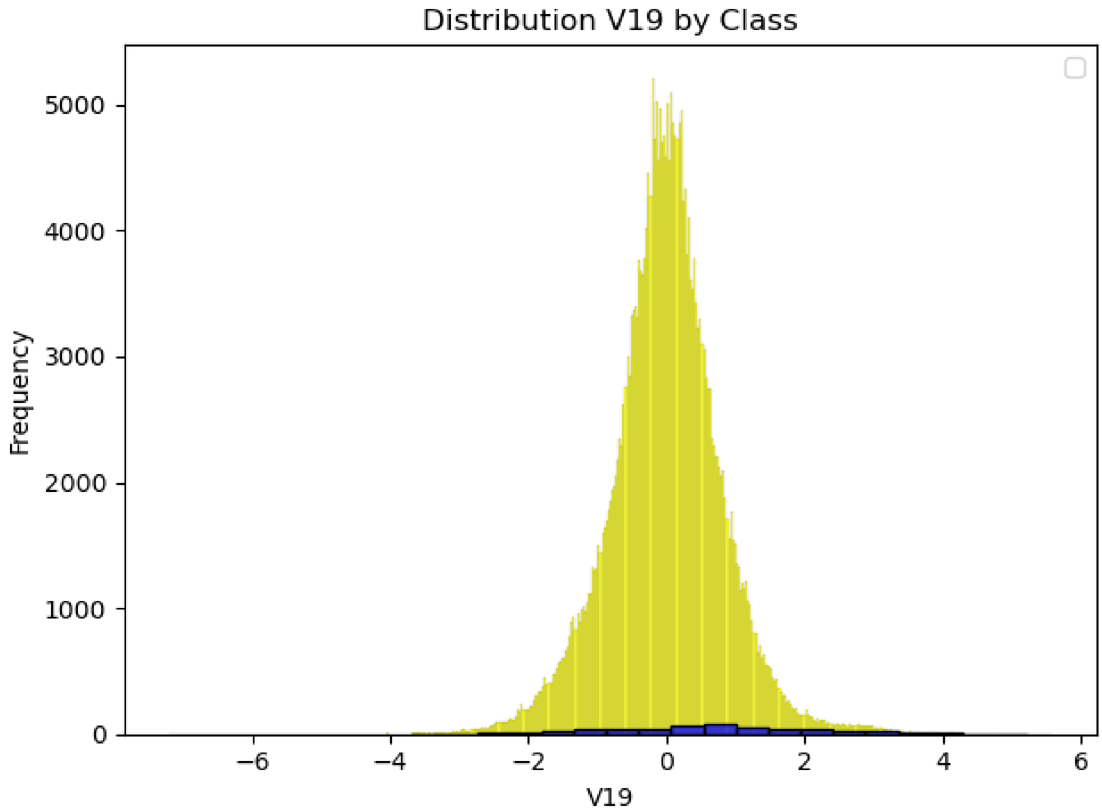
Distribution V15 by Class
Frequency
V15
5000
4000
3000
2000
1000
0
−4
−2
0
2
4
6
8

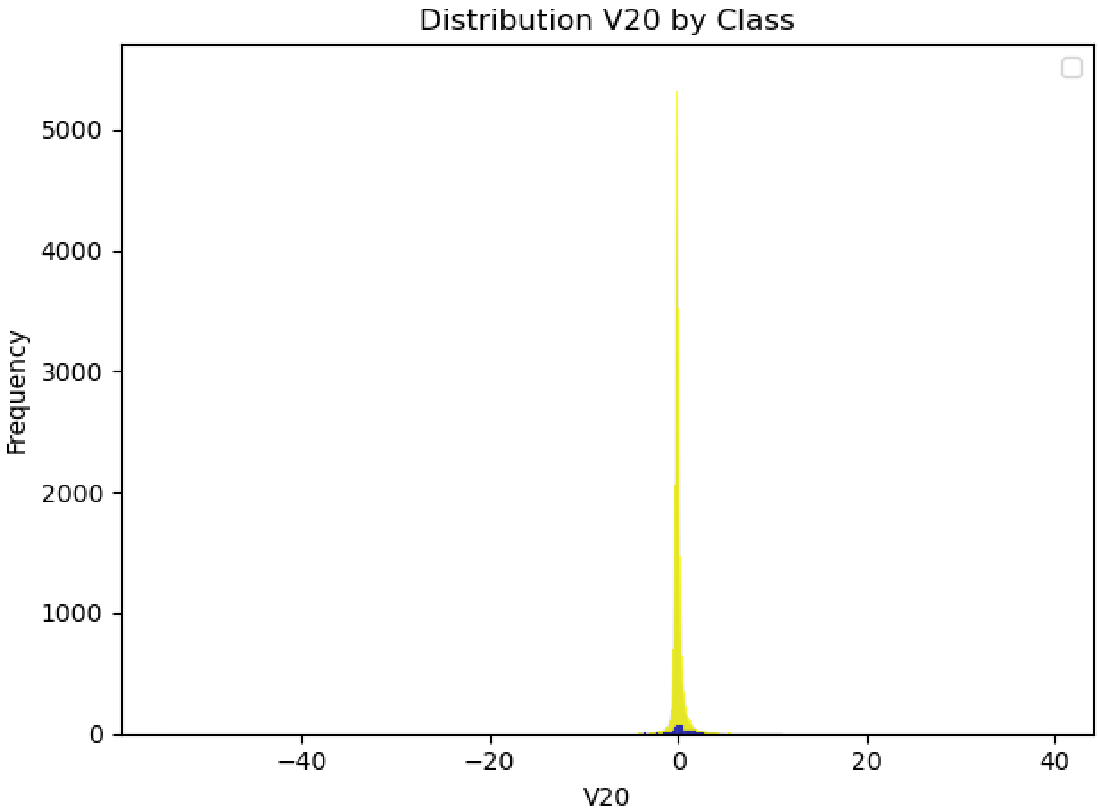

Distribution V16 by Class
Frequency
5000
4000
3000
2000
1000
0
−15
−10
−5
0
5
10
15
V16

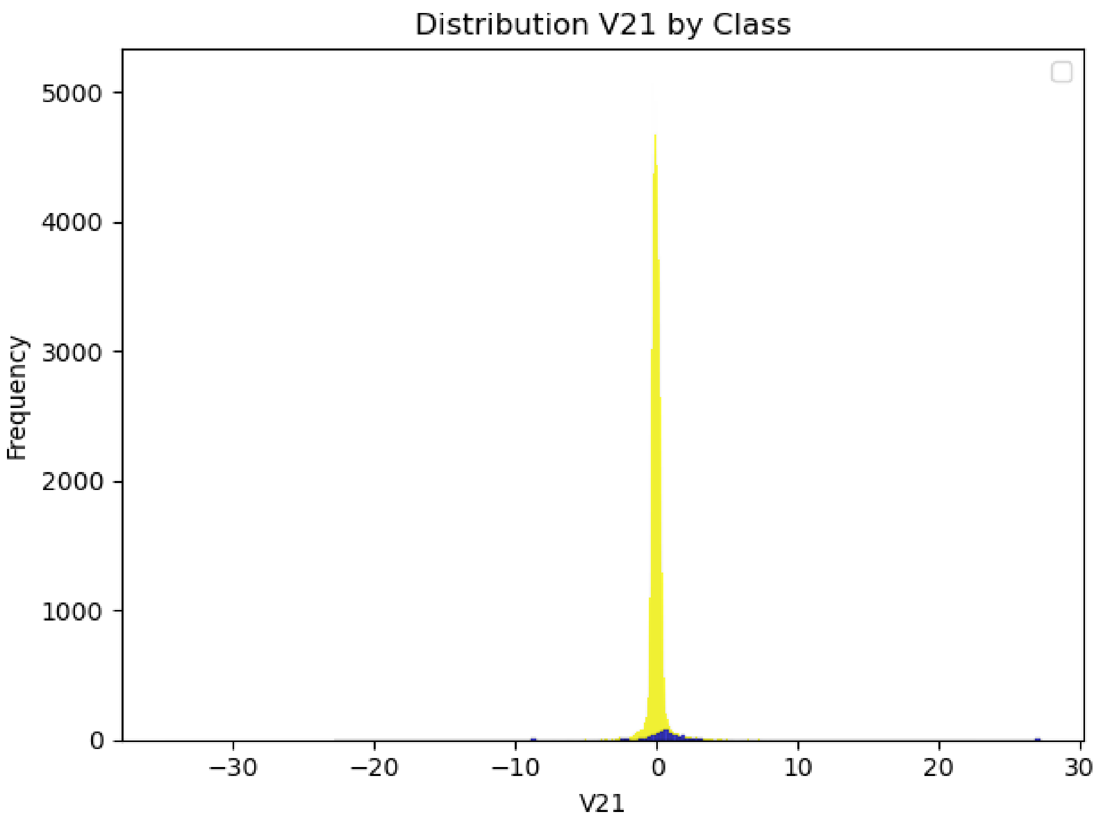

Distribution V17 by Class
Frequency
V17
0
1000
2000
3000
4000
5000
−25
−20
−15
−10
−5
0
5
10



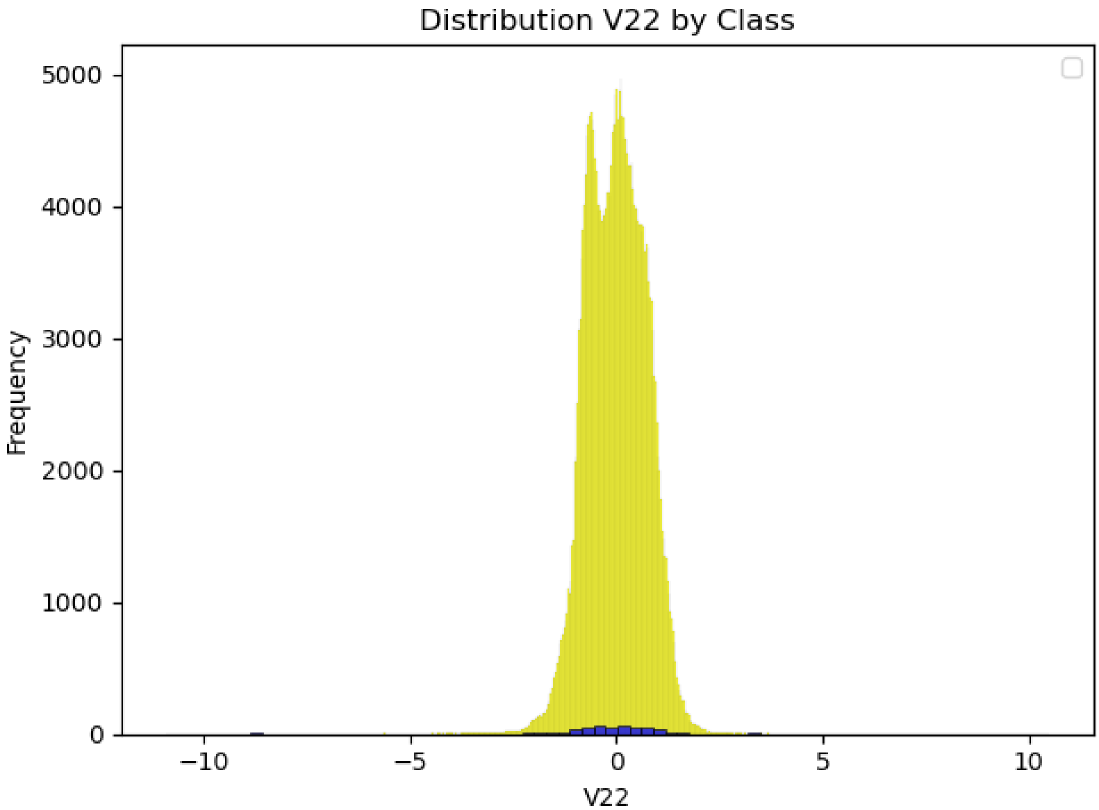
Distribution V18 by Class
5000
4000
3000
2000
1000
0
Frequency
−10
−8
−6
−4
−2
0
2
4
V18

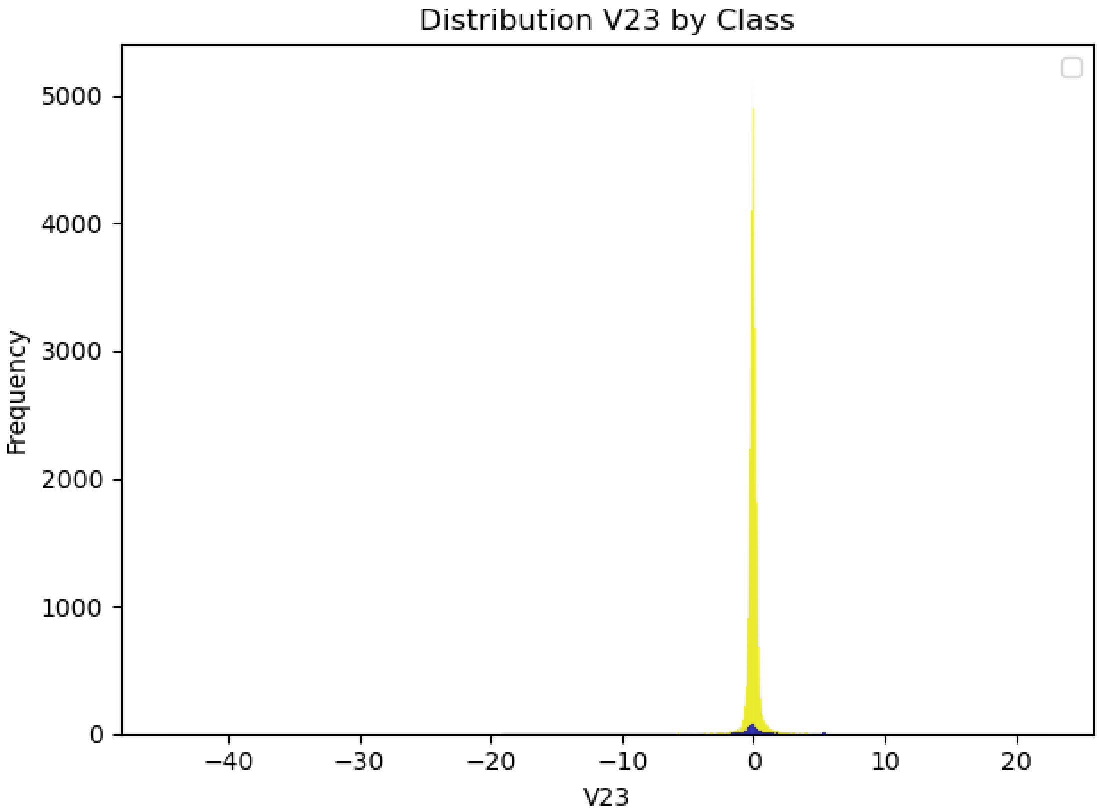

Distribution V19 by Class
Frequency
V19
5000
4000
3000
2000
1000
0
−6
−4
−2
0
2
4
6

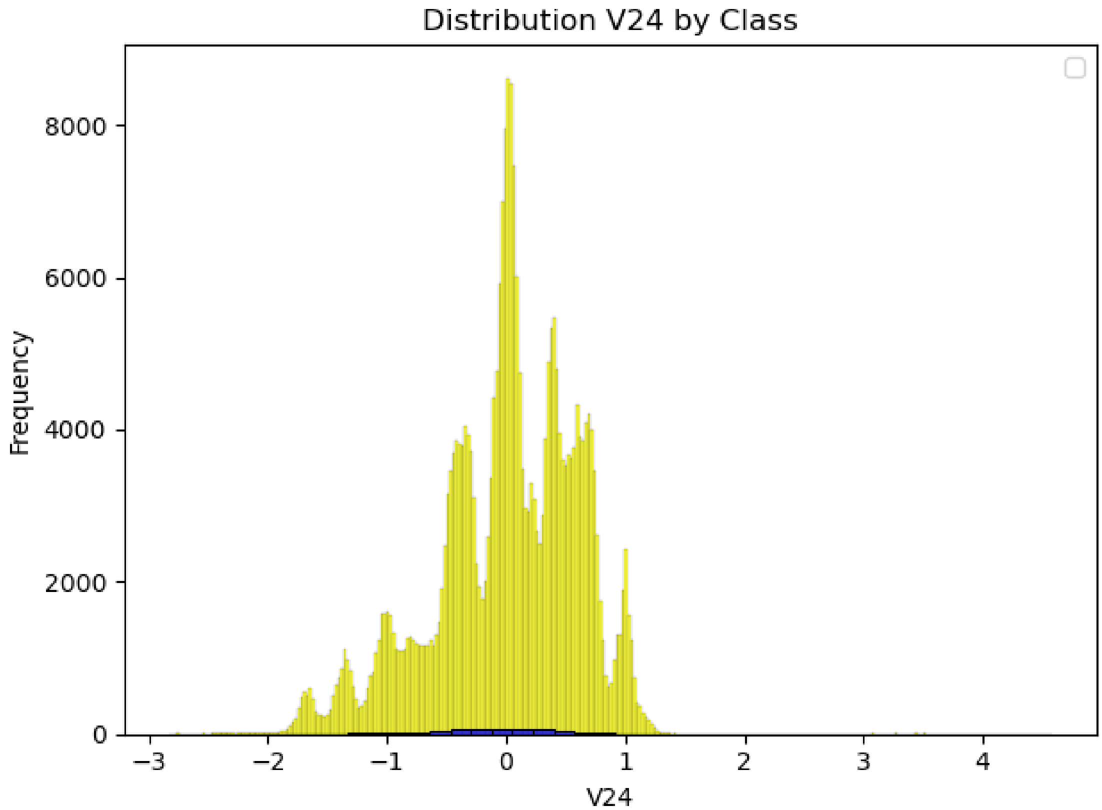

Distribution V20 by Class
Frequency
5000
4000
3000
2000
1000
0
−40
−20
0
20
40
V20

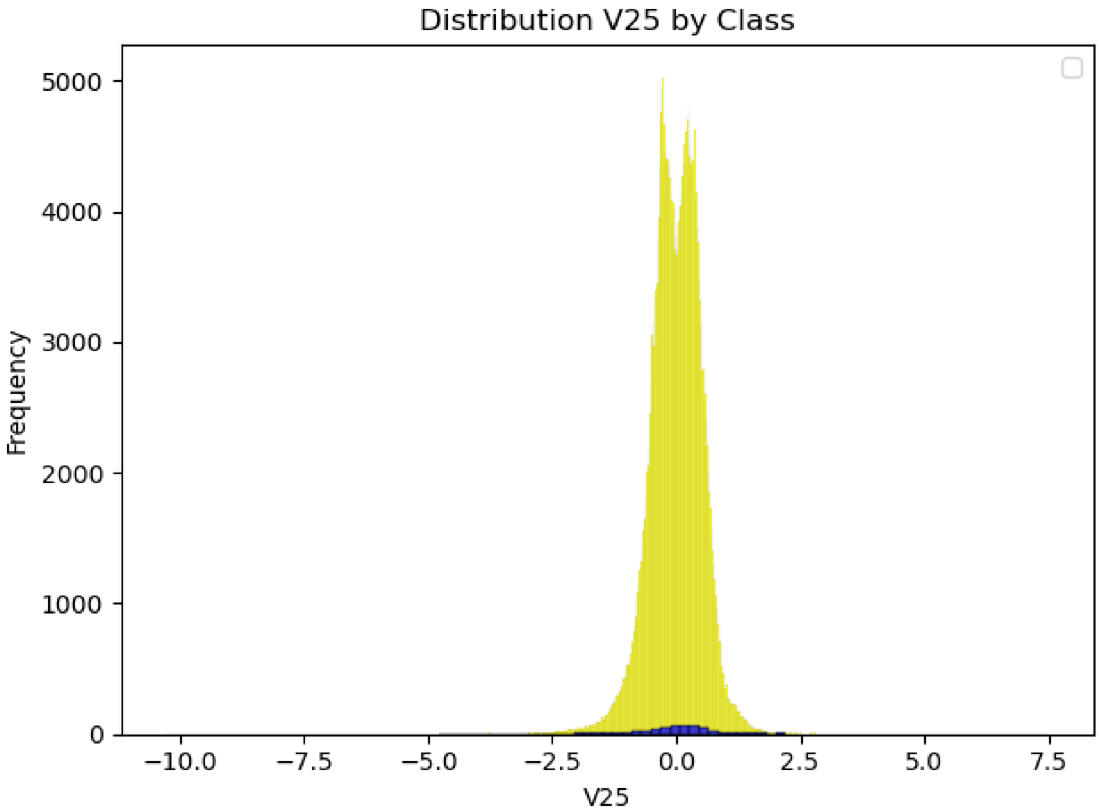


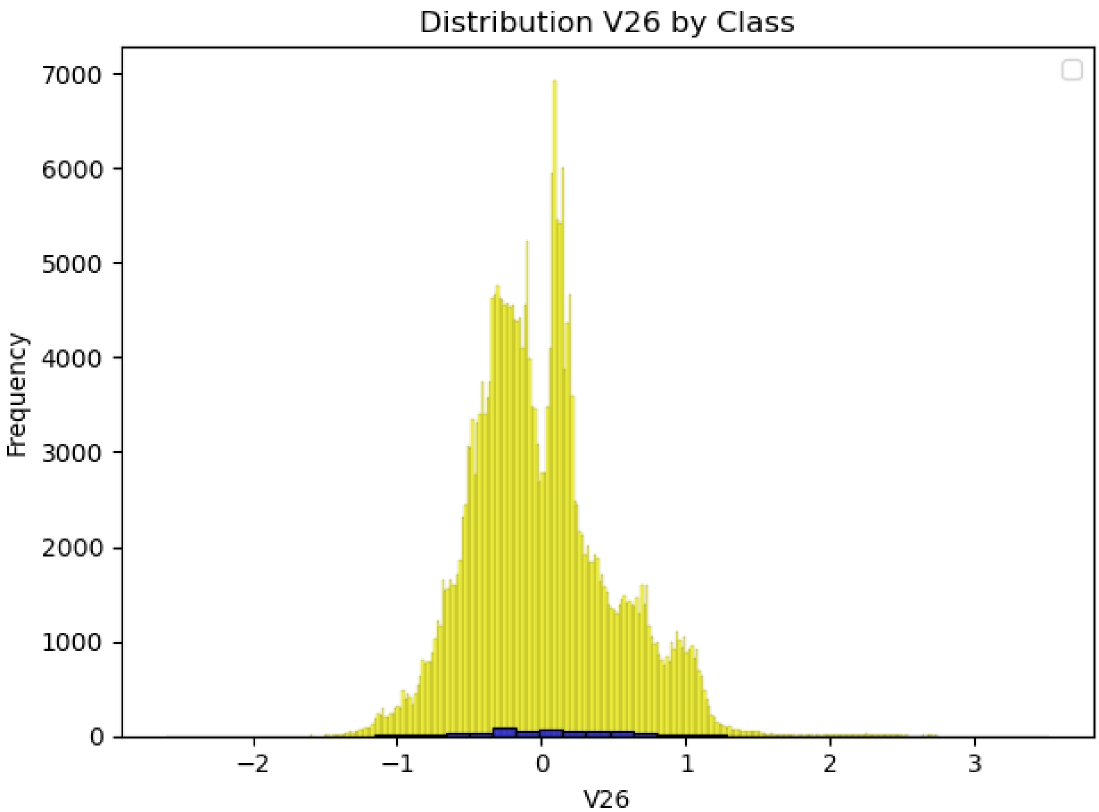
Distribution V21 by Class
5000
4000
3000
2000
1000
0
Frequency
−30
−20
−10
0
10
20
30
V21


Distribution V22 by Class
5000
4000
3000
2000
1000
0
Frequency
−10
−5
0
5
10
V22

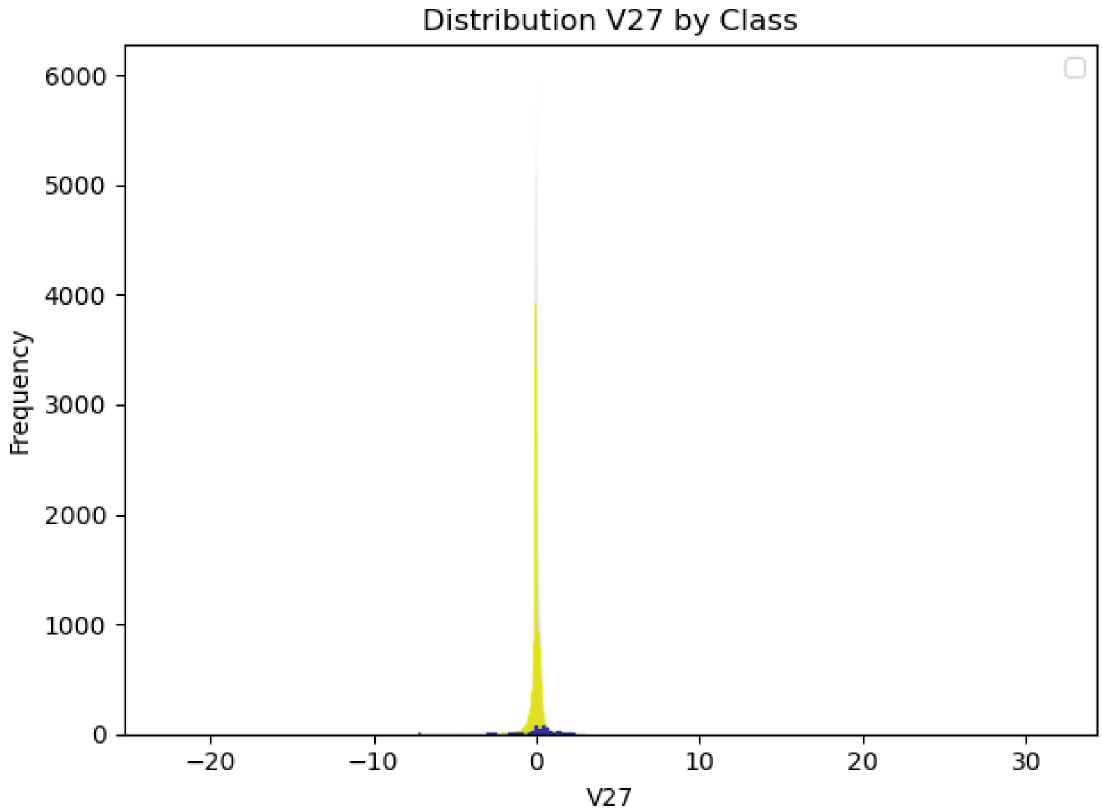

Distribution V23 by Class
Frequency
V23
5000
4000
3000
2000
1000
0
−40
−30
−20
−10
0
10
20


Distribution V24 by Class
8000
6000
4000
2000
0
Frequency
−3
−2
−1
0
1
2
3
4
V24

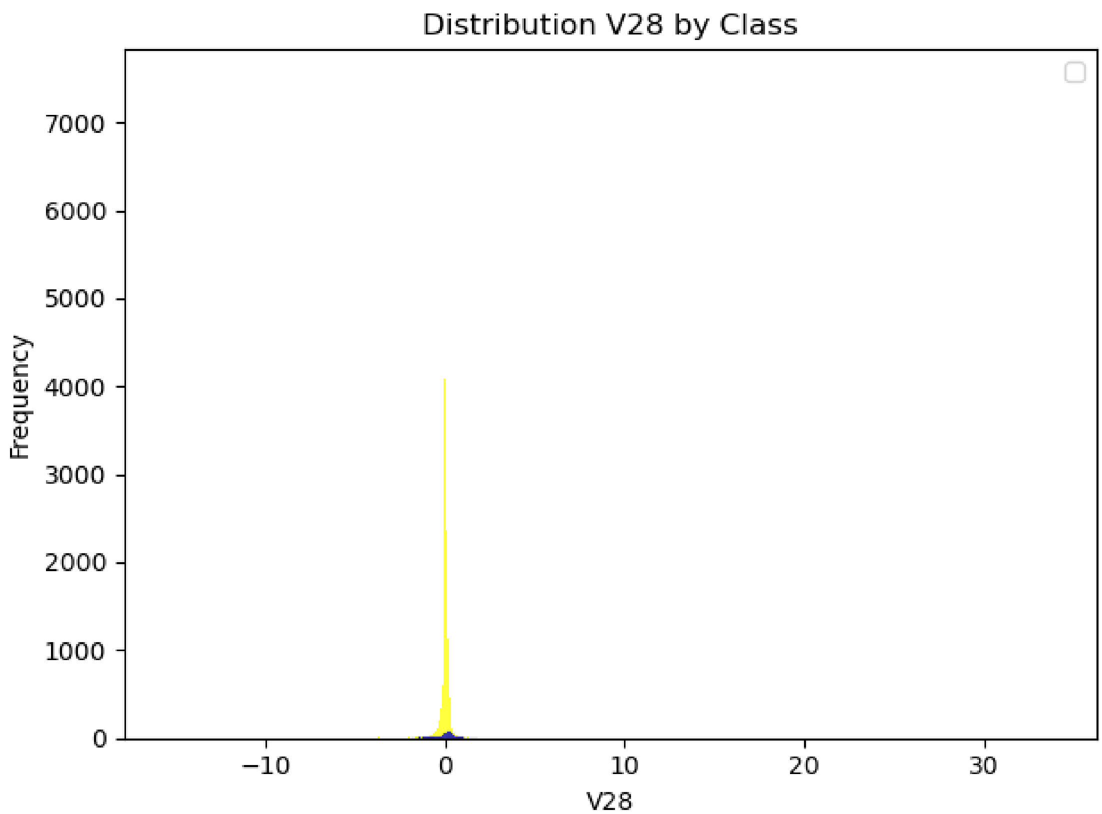

Distribution V25 by Class
5000
4000
3000
2000
1000
0
Frequency
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V25

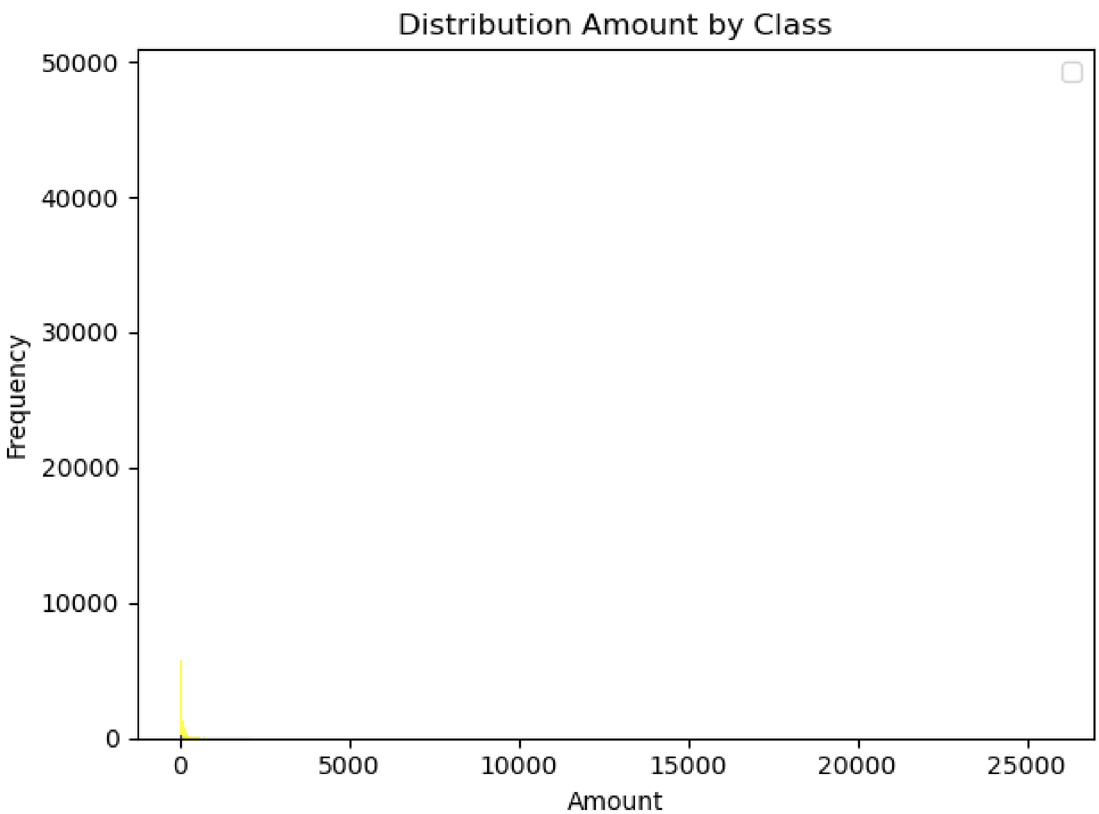


Distribution V26 by Class
Frequency
7000
6000
5000
4000
3000
2000
1000
0
−2
−1
0
1
2
3
V26



Distribution V27 by Class
Frequency
V27
6000
5000
4000
3000
2000
1000
0
−20
−10
0
10
20
30


Distribution V28 by Class
Frequency
7000
6000
5000
4000
3000
2000
1000
0
−10
0
10
20
30
V28



Insights of correlation matrix and histogram distributions of each feature for Class :

1 **Time:** The distribution is similar for both classes, indicating that the time of transaction is not a strong indicator of fraud.

2 **V1 to V28:** These features are PCA components, so their distributions are not directly interpretable. However, some features show distinct differences:

- **V4, V11, V17:** These features have noticeable differences between the two classes, suggesting they may be important for distinguishing fraud.
- **V10, V12, V14, V16:** These features also show some separation, indicating potential usefulness in fraud detection. 3 **Amount:** Fraudulent transactions tend to have lower amounts compared to non-fraudulent ones, which might be useful for prediction.

Overall, features with distinct differences in distribution between the two classes are likely more informative for predicting fraudulent transactions

Data Preprocessing

[12]:
```
from sklearn.preprocessing import StandardScaler
#Scale 'Amount' and 'Time' (other features are already standardized)
scaler = StandardScaler()
A[['Time', 'Amount']] = scaler.fit_transform(A[['Time', 'Amount']])
```

```
# A is now scaled and ready for splitting
print(A.head())
```

```
       Time        V1        V2        V3        V4        V5        V6  \
0 -1.996583 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
1 -1.996583  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361
2 -1.996562 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
3 -1.996562 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
4 -1.996541 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921

         V7        V8        V9  …       V20       V21       V22       V23  \
0  0.239599  0.098698  0.363787  …  0.251412 -0.018307  0.277838 -0.110474
1 -0.078803  0.085102 -0.255425  … -0.069083 -0.225775 -0.638672  0.101288
2  0.791461  0.247676 -1.514654  …  0.524980  0.247998  0.771679  0.909412
3  0.237609  0.377436 -1.387024  … -0.208038 -0.108300  0.005274 -0.190321
4  0.592941 -0.270533  0.817739  …  0.408542 -0.009431  0.798278 -0.137458

        V24       V25       V26       V27       V28    Amount
0  0.066928  0.128539 -0.189115  0.133558 -0.021053  0.244964
1 -0.339846  0.167170  0.125895 -0.008983  0.014724 -0.342475
2 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  1.160686
3 -1.175575  0.647376 -0.221929  0.062723  0.061458  0.140534
4  0.141267 -0.206010  0.502292  0.219422  0.215153 -0.073403

[5 rows x 30 columns]
```

Train-Test Split

[14]:
```
from sklearn.model_selection import train_test_split

# Split into training and testing sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    A, B, test_size=0.2, random_state=42, stratify=B)  # maintain class␣
 ↪distribution
```

Handle Class Imbalance with SMOTE Fraud cases are extremely rare, so let's balance the training set using SMOTE (Synthetic Minority Over-sampling Technique):

[19]:
```
from imblearn.over_sampling import SMOTE

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Check class distribution
from collections import Counter
print("Before SMOTE:", Counter(y_train))
print("After SMOTE:", Counter(y_train_resampled))
```

```
Before SMOTE: Counter({0: 227451, 1: 394})
After SMOTE: Counter({0: 227451, 1: 227451})
```

Evaluate multiple Classification models to compare performance

Model 1 - Logistic Regression

[21]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
↪roc_auc_score

# Initialize and train the model
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train_resampled, y_train_resampled)

# Predict on the test set
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]
```

[23]:
```
# Classification report
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# ROC AUC score
print("ROC AUC Score:", roc_auc_score(y_test, y_proba))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.97      0.99     56864
           1       0.06      0.92      0.11        98

    accuracy                           0.97     56962
   macro avg       0.53      0.95      0.55     56962
weighted avg       1.00      0.97      0.99     56962

Confusion Matrix:
 [[55406  1458]
 [    8    90]]
ROC AUC Score: 0.9698482164390798
```

Model 2 - Random Forest

[34]:
```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_resampled, y_train_resampled)
```

```
rf_pred = rf_model.predict(X_test)
rf_proba = rf_model.predict_proba(X_test)[:, 1]

print("=== Random Forest ===")
print("Confusion Matrix:\n", confusion_matrix(y_test, rf_pred))
print("Classification Report:\n", classification_report(y_test, rf_pred))
print("ROC AUC Score:", roc_auc_score(y_test, rf_proba))
```

```
=== Random Forest ===
Confusion Matrix:
 [[56849    15]
 [   16    82]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.85      0.84      0.84        98

    accuracy                           1.00     56962
   macro avg       0.92      0.92      0.92     56962
weighted avg       1.00      1.00      1.00     56962

ROC AUC Score: 0.9731024901519414
```

Model 3 - K-Nearest Neighbors

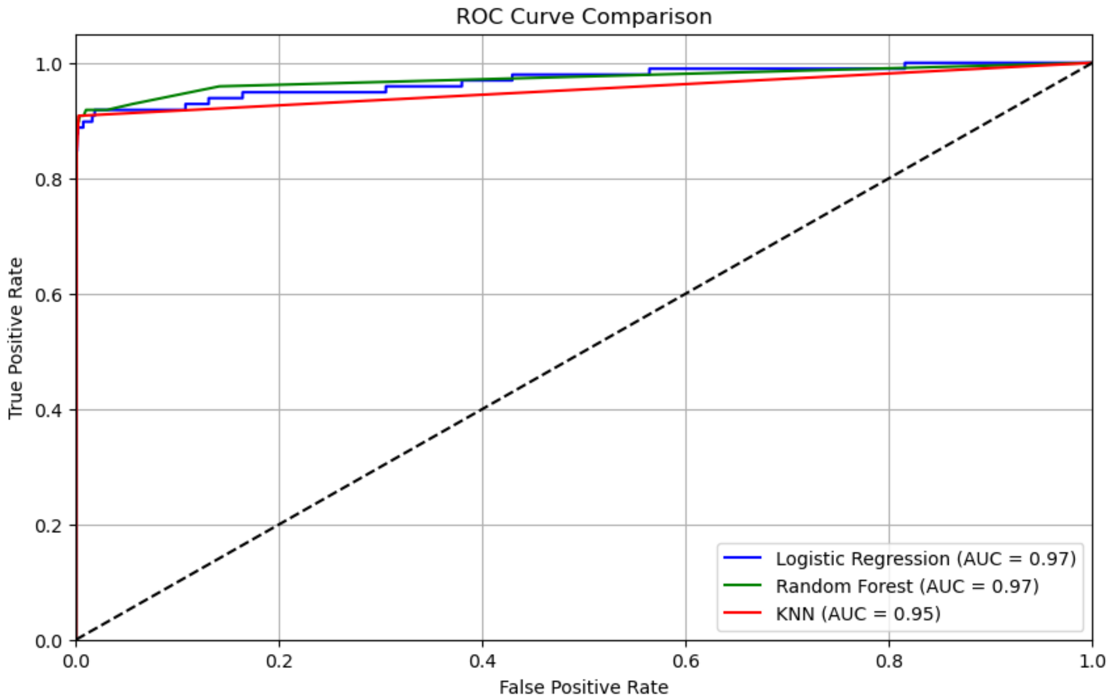
[27]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train_resampled, y_train_resampled)

knn_pred = knn_model.predict(X_test)
knn_proba = knn_model.predict_proba(X_test)[:, 1]

print("\n=== K-Nearest Neighbors ===")
print("Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))
print("Classification Report:\n", classification_report(y_test, knn_pred))
print("ROC AUC Score:", roc_auc_score(y_test, knn_proba))
```

```
=== K-Nearest Neighbors ===
Confusion Matrix:
 [[56765    99]
 [   12    86]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.46      0.88      0.61        98
```

```
    accuracy                           1.00     56962
   macro avg       0.73      0.94      0.80     56962
weighted avg       1.00      1.00      1.00     56962

ROC AUC Score: 0.9535882427675628
```

Visualization of ROC curves

[36]:
```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# we have already calculated Predict probabilities for all models above

# Compute ROC curve and AUC
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_proba)
roc_auc_lr = auc(fpr_lr, tpr_lr)

fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_proba)
roc_auc_rf = auc(fpr_rf, tpr_rf)

fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_proba)
roc_auc_knn = auc(fpr_knn, tpr_knn)

# Plot all ROC curves
plt.figure(figsize=(10, 6))
plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression (AUC = {roc_auc_lr:.2f})',␣
↪color='blue')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.2f})',␣
↪color='green')
plt.plot(fpr_knn, tpr_knn, label=f'KNN (AUC = {roc_auc_knn:.2f})', color='red')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



Model Performance Summary:

Logistic Regression - Accuracy= 0.97% , ROC AUC= 0.96

Random Forest - Accuracy= 1.00% , ROC AUC= 0.97

KNN - Accuracy= 1.00% , ROC AUC= 0.95

Conclusion

Based on the evaluation metrics and visual analysis, Random Forest outperformed other models in every critical category—achieving high accuracy, and ROC AUC. This indicates that it can reliably detect fraudulent transactions while minimizing false positives and false negatives. Logistic Regression also performed decently and can serve as a lightweight alternative. KNN, while simpler, lagged behind in key metrics, making it less suitable for highly imbalanced fraud detection tasks. Overall, this project demonstrates that combining robust preprocessing techniques with a powerful ensemble model like Random Forest is an effective strategy for fraud detection