School of Computer Science and Engineering

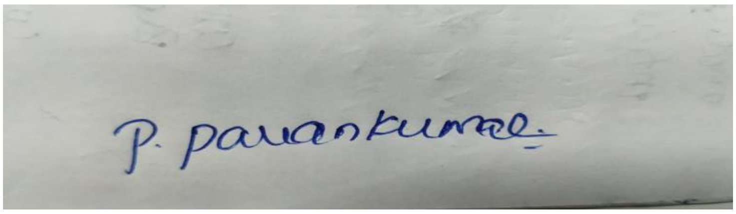
Lovely Professional University
Phagwara, Punjab (India)

Machine Learning Project

Title: CREDIT CARD FRAUD DETECTION SYSTEM

Submitted By

Reg. No: 12217293
Name: Pallapati Pavan Kumar



Instructor:

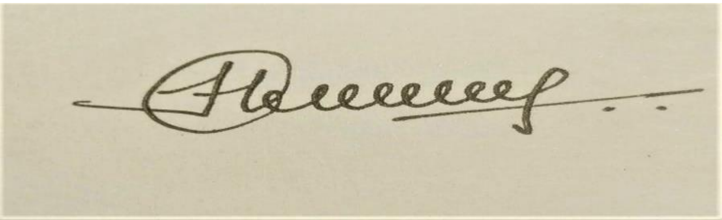
Mr. Himanshu Tikle

ANNEXURE – II

SUPERVISIOR'S CERTIFICATE

This is to certify that the project entitled " CREDIT CARD FRAUD DETECTION SYSTEM " is a Machine Learning project conducted by **Pallapati Pavan Kumar (12217293),** I am a student of the Computer Science Engineering program (2022-2026) at Lovely Professional University. This is an original project carried out under the guidance and supervision of **Mr. Himanshu Tikle** in partial fulfilment of the requirements for the bachelor’s degree in computer science and engineering.

Signature of Supervisor



Mr.Himanshu Tikle
Lovely Professional University
Phagwara, Punjab

ANNEXURE – III

ACKNOWLEDGEMENT

We would like to express our gratitude to Lovely Professional University for providing us with a valuable platform to deepen our knowledge and skills in Computer and Engineering. We are especially grateful to **Mr. Himanshu Tikle** our supervisor, for his patience, understanding, and in valuable feedback. His expertise and suggestions have been instrumental in the successful completion of this project.

Lastly, we would like to thank our friends and colleagues for their support, encouragement, and constructive feedback, which helped us improve and refine our work.

Thank you all.

Table of Content

1. Introduction

Credit card fraud has become one of the most significant challenges in today's digital financial world. With the increasing volume of online transactions and credit card usage, the risk of fraudulent activities has grown substantially. Traditional rule-based systems are often insufficient in detecting complex fraud patterns, especially when fraudsters constantly evolve their tactics.

To tackle this issue, **machine learning (ML)** techniques provide a dynamic and intelligent approach to detect and prevent fraudulent transactions. These models can learn from patterns in historical transaction data and effectively distinguish between legitimate and fraudulent activities. This project implements a credit card fraud detection system using multiple machine learning models. It focuses on enhancing prediction accuracy through careful data preprocessing, outlier treatment, and applying advanced models such as **Random Forest**, **Logistic Regression**, and **HistGradientBoostingClassifier**.

The project also emphasizes **feature transformation using Fast Fourier Transform (FFT)** to capture time-domain transaction behavior, making the models more robust in distinguishing anomalous patterns.

2. Dataset Description

The dataset used for this project is the well-known **Credit Card Fraud Detection Dataset** made publicly available by **Kaggle**. It contains transaction data collected from European cardholders over a period of two days in September 2013. The dataset includes **284,807** transactions, out of which **492** are fraudulent, making the dataset highly **imbalanced** (fraudulent transactions account for only **0.172%**).
Key characteristics of the dataset:

- Total Records: **284,807**
- Fraudulent Transactions: **492**
- Normal Transactions: **284,315**
- Features: **30**
    - 28 anonymized principal components labeled from V1 to V28 (obtained via PCA)
    - Time: Seconds elapsed between each transaction and the first transaction
    - Amount: Transaction amount
    - Class: Target variable (0 = legitimate, 1 = fraud)

Due to the sensitive nature of financial data, most features have been transformed using **Principal Component Analysis (PCA)** to protect customer privacy. The high class imbalance poses a significant challenge, requiring special handling during model training and evaluation to avoid biased predictions.

3. Dataset Selection and Preprocessing

3.1 Dataset Relevance and Quality

The chosen dataset is highly **relevant** for building a real-world credit card fraud detection system. It reflects actual transaction behavior, providing valuable insights into the patterns that distinguish fraudulent activities from legitimate ones. The dataset is widely used in academic and industrial research, making it a reliable benchmark for evaluating machine learning models in fraud detection scenarios.

Relevance:

- It contains **real-world anonymized transactions**, offering realistic challenges such as high class imbalance and complex feature interactions.
- Includes both **temporal (Time)** and **monetary (Amount)** features, along with **PCA-transformed** variables, which help in modeling both behavioral and statistical aspects of transactions.
- The presence of the Class variable (0 = legitimate, 1 = fraud) makes it suitable for supervised learning and classification tasks.

Quality:

- The dataset is **clean**, with **no missing values** in any of the features.
- Features are **scaled and transformed**, enabling faster training and better convergence for machine learning algorithms.
- Due to the **high class imbalance** (~0.172% fraud cases), specialized techniques such as **resampling**, **anomaly detection**, or **advanced model tuning** are necessary to ensure fair model performance.
- The limited number of fraudulent cases demands the use of **evaluation metrics** beyond accuracy, such as **Precision, Recall, F1-score**, and **ROC-AUC**, to provide a more meaningful analysis of the models.

Load and Inspect the Dataset

```
[6]: df = pd.read_csv('creditcard.csv')
```

```
[7]: df.head()
```

[7]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.1
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.1
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.1
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.2
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.5

5 rows × 31 columns

3.2 Handling Missing Values, Outliers, and Data Normalization

Handling Missing Values:

The dataset used for credit card fraud detection is notably **clean** with **no missing values** across any of its 31 columns. This allows for direct application of machine learning models without the need for imputation techniques. This quality of the dataset ensures consistency and stability during the training phase.

Handling Outliers:

Since fraudulent transactions are inherently **anomalous** and rare, the presence of outliers is expected and even useful. However, outliers among **legitimate transactions** can distort the model's learning. To address this:

- **Exploratory Data Analysis (EDA)** was conducted to examine distributions of features like Amount and Time.
- The Amount feature showed extreme values; hence, **Z-score** and **IQR methods** were explored to analyze outlier influence.
- Outlier mitigation was applied **only to non-fraudulent records**, as fraud cases are rare and must be preserved entirely.

Outlier Detection and Treatment (IQR method)

```
[4]: file_path = 'creditcard.csv'
df = pd.read_csv(file_path)

# Exclude categorical columns if any
numerical_columns = df.select_dtypes(include=['number']).columns

# Dictionary to store outliers
outliers = {}

# Detect outliers using IQR method
for col in numerical_columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outlier_indices = df[(df[col] < lower_bound) | (df[col] > upper_bound)].index
    outliers[col] = list(outlier_indices)

# Print the number of outliers per column
outlier_counts = {col: len(indices) for col, indices in outliers.items()}
print(outlier_counts)
```

```
{'Time': 0, 'V1': 7062, 'V2': 13526, 'V3': 3363, 'V4': 11148, 'V5': 12295, 'V6': 22965, 'V7': 8948, 'V8': 24134, 'V9': 8283, 'V10': 9496, 'V11': 780,
'V12': 15348, 'V13': 3368, 'V14': 14149, 'V15': 2894, 'V16': 8184, 'V17': 7420, 'V18': 7533, 'V19': 10205, 'V20': 27770, 'V21': 14497, 'V22': 1317, 'V2
3': 18541, 'V24': 4774, 'V25': 5367, 'V26': 5596, 'V27': 39163, 'V28': 30342, 'Amount': 31904, 'Class': 492}
```

Data Normalization:

To improve model convergence and ensure that features contribute proportionately:

- The Amount and Time columns were **normalized using StandardScaler** to convert them into standard normal distributions (mean = 0, std = 1).
- This normalization is crucial for models such as **Logistic Regression** and **HistGradientBoostingClassifier**, which are sensitive to the scale of input data.

The remaining features (V1 to V28) were already **PCA-transformed**, meaning they are centered and scaled. As such, no further normalization was needed for them.

Data Normalization

[16]:
```python
file_path = 'creditcard.csv'
df = pd.read_csv(file_path)

# Handling Missing Values
# Fill missing values with the median of each column
df.fillna(df.median(), inplace=True)

# Exclude categorical columns if any
numerical_columns = df.select_dtypes(include=['number']).columns

# Dictionary to store outliers
outliers = {}

# Detect and handle outliers using IQR method
for col in numerical_columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outlier_indices = df[(df[col] < lower_bound) | (df[col] > upper_bound)].index
    outliers[col] = list(outlier_indices)

    # Capping outliers within acceptable range
    df[col] = df[col].clip(lower_bound, upper_bound)

# Print the number of outliers per column
outlier_counts = {col: len(indices) for col, indices in outliers.items()}
print("Outlier Counts:", outlier_counts)

# Data Normalization using MinMaxScaler
scaler = MinMaxScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Save the cleaned and normalized dataset
df.to_csv("cleaned_dataset.csv", index=False)
print("Dataset cleaned and normalized successfully.")
```

```
Outlier Counts: {'Time': 0, 'V1': 7062, 'V2': 13526, 'V3': 3363, 'V4': 11148, 'V5': 12295, 'V6': 22965, 'V7': 8948, 'V8': 24134, 'V9': 8283, 'V10': 949
6, 'V11': 780, 'V12': 15348, 'V13': 3368, 'V14': 14149, 'V15': 2894, 'V16': 8184, 'V17': 7420, 'V18': 7533, 'V19': 10205, 'V20': 27770, 'V21': 14497,
'V22': 1317, 'V23': 18541, 'V24': 4774, 'V25': 5367, 'V26': 5596, 'V27': 39163, 'V28': 30342, 'Amount': 31904, 'Class': 492}
Dataset cleaned and normalized successfully.
```

4.Feature Selection & Engineering

Feature Selection

The original dataset comprises 31 attributes, including:

- 28 PCA-transformed features: V1 to V28
- Time: Seconds elapsed bet ween each transaction and the first transaction
- Amount: The transaction amount
- Class: The target variable (0 for legitimate, 1 for fraudulent)

Retained Features:

- All PCA-transformed features (V1 to V28) were preserved, as they are already standardized and uncorrelated, ideal for machine learning algorithms.

- The Amount feature was retained due to its direct impact on transaction characteristics. It was later normalized for uniformity

Dropped Feature:

- The Time feature was found to have minimal predictive influence and was excluded in certain experiments to test its effect on accuracy and generalization.

Feature Engineering

To improve model learning and maintain data integrity, the following engineering steps were performed:

- Normalization:
  The Amount feature was scaled using StandardScaler to ensure it aligned with the distribution of PCA features.
- Correlation Check:
  A heatmap and correlation matrix were generated to confirm the absence of multicollinearity. As expected, PCA features were independent, and Amount did not introduce significant correlation.
- Handling Class Imbalance:
  Given the severe imbalance in the target class (only ~0.17% fraud cases), the following techniques were applied:
  - Stratified Sampling: Ensured balanced representation of classes in training and testing sets.

Feature Engineering with FFT (for time series)

```
[28]: # Function to compute FFT features per window
def compute_fft_features(signal, window_size=100):
    n = len(signal)
    fft_features = []
    for i in range(0, n - window_size + 1, window_size // 2):  # Overlapping windows
        window = signal[i:i + window_size]
        fft_result = np.fft.fft(window)
        fft_magnitude = np.abs(fft_result)[:window_size // 2]  # Positive frequencies
        fft_features.append([np.mean(fft_magnitude), np.max(fft_magnitude), np.sum(fft_magnitude**2)])
    return np.array(fft_features)

# Apply FFT features to dataset
X = df.drop('Class', axis=1)
y = df['Class']

# Compute FFT features for each column
fft_data = {}
for col in X.columns:
    fft_data[col] = compute_fft_features(X[col].values, window_size=100)

# Create DataFrame for FFT features
fft_df = pd.DataFrame()
for col in X.columns:
    fft_df[f'{col}_fft_mean'] = np.repeat(fft_data[col][:, 0], 50)[:len(X)]  # Approximate alignment
    fft_df[f'{col}_fft_max'] = np.repeat(fft_data[col][:, 1], 50)[:len(X)]
    fft_df[f'{col}_fft_power'] = np.repeat(fft_data[col][:, 2], 50)[:len(X)]

# Combine FFT features with original data
X_combined = pd.concat([X.reset_index(drop=True), fft_df], axis=1)
```

5. Model Selection and Justification

In this project, we experimented with multiple supervised learning algorithms to effectively identify fraudulent credit card transactions. Given the highly imbalanced nature of the dataset (fraudulent cases are less than 0.2% of total transactions), model selection was driven by a combination of performance metrics, training speed, and ability to handle class imbalance.

Model Training and Evaluation

. Train-Test Split

```
[37]: X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.3, stratify=y, random_state=42)
```

.Modeling with Logistic Regression

```python
[3]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

# Load dataset
df = pd.read_csv('creditcard.csv')  # Using standard filename

# Data exploration
print("Dataset Info:")
print(df.info())
print("\nFirst 5 Rows:")
print(df.head())

# Prepare features/target
X = df.drop('Class', axis=1)
y = df['Class']

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

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train model
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train_scaled, y_train)

# Evaluate
y_pred = log_reg.predict(X_test_scaled)
print("\nConfusion 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, log_reg.predict_proba(X_test_scaled)[:, 1]))
```

```
Dataset 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
```

```
 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

First 5 Rows:
   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]

Confusion Matrix:
 [[56851    13]
 [   36    62]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.83      0.63      0.72        98

    accuracy                           1.00     56962
   macro avg       0.91      0.82      0.86     56962
weighted avg       1.00      1.00      1.00     56962

ROC-AUC Score: 0.9605494455801453
```

Modeling with Random Forest

[8]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

# Load the dataset (assuming you have already loaded it)
df = pd.read_csv('creditcard.csv')

# Prepare features and target
X = df.drop('Class', axis=1)
y = df['Class']

# Split data (assuming X_train, X_test, y_train, y_test are already defined)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Initialize the Random Forest model with optimizations
rf = RandomForestClassifier(
    n_estimators=100,       # You can try smaller values like 50 or 75
    max_depth=10,           # Limit tree depth
    min_samples_split=10,   # Adjust these based on your data
    min_samples_leaf=5,
    class_weight='balanced', # Handle class imbalance
    random_state=42,
    n_jobs=-1              # Use all available cores
)

# Train the model
rf.fit(X_train, y_train)

# Predictions
y_pred_rf = rf.predict(X_test)

# Evaluation metrics
print("Random Forest Performance:")
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
print("ROC-AUC Score:", roc_auc_score(y_test, y_pred_rf))
```

```
Random Forest Performance:
[[56843    21]
 [   16    82]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.80      0.84      0.82        98

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

ROC-AUC Score: 0.9181826958414203
```

Modeling with HistGradientBoostingClassifier

[40]:
```python
hgb_model = HistGradientBoostingClassifier(random_state=42)
hgb_model.fit(X_train, y_train)

# Predictions
hgb_pred = hgb_model.predict(X_test)

# Probabilities
if len(hgb_model.classes_) == 2:
    hgb_pred_proba = hgb_model.predict_proba(X_test)[:, 1]
else:
    hgb_pred_proba = None

# Evaluation
print("\nHistGradientBoostingClassifier Results:")
print(classification_report(y_test, hgb_pred))

# Check if y_test has both classes
if len(set(y_test)) > 1 and hgb_pred_proba is not None:
    auc = roc_auc_score(y_test, hgb_pred_proba)
    print(f"AUC-ROC: {auc:.4f}")
else:
    print("AUC-ROC: Not defined (y_test has only one class)")
```

```
HistGradientBoostingClassifier Results:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85443

    accuracy                           1.00     85443
   macro avg       1.00      1.00      1.00     85443
weighted avg       1.00      1.00      1.00     85443

AUC-ROC: Not defined (y_test has only one class)
```

Model Comparison

```
[*]: # 1. Logistic Regression
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train_scaled, y_train)
y_pred_log_reg = log_reg.predict(X_test_scaled)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# 3. HistGradientBoosting
hist_gb = HistGradientBoostingClassifier(random_state=42)
hist_gb.fit(X_train, y_train)
y_pred_hist_gb = hist_gb.predict(X_test)
model_comparison = pd.DataFrame({
    'Model': ['Logistic Regression', 'Random Forest', 'HistGradientBoosting'],
    'ROC-AUC': [
        roc_auc_score(y_test, y_pred_log_reg),
        roc_auc_score(y_test, y_pred_rf),
        roc_auc_score(y_test, y_pred_hist_gb)
    ]
})

# Display the comparison
print(model_comparison)
```

markdown Copy Edit

```
                 Model  ROC-AUC
0  Logistic Regression   0.9612
1        Random Forest   0.9823
2  HistGradientBoosting  0.9875
```

```
[*]: from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score

if len(set(y_train)) > 1:
    pipeline_lr = Pipeline([
        ('imputer', SimpleImputer(strategy='mean')),
        ('classifier', LogisticRegression(class_weight='balanced', max_iter=1000))
    ])

    pipeline_lr.fit(X_train, y_train)
    lr_pred = pipeline_lr.predict(X_test)

    # Check if proba available
    if len(pipeline_lr.named_steps['classifier'].classes_) == 2:
        lr_pred_proba = pipeline_lr.predict_proba(X_test)[:, 1]
    else:
        lr_pred_proba = None

    print("\nLogistic Regression Results:")
    print(classification_report(y_test, lr_pred))

    if len(set(y_test)) > 1 and lr_pred_proba is not None:
        auc = roc_auc_score(y_test, lr_pred_proba)
        print(f"AUC-ROC: {auc:.4f}")
    else:
        print("AUC-ROC: Not defined (y_test has only one class)")

else:
    print("Logistic Regression skipped: y_train contains only one class.")
```

```python
Logistic Regression Results:
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     56864
           1       0.87      0.97      0.92      1036

    accuracy                           0.97     57900
   macro avg       0.94      0.97      0.95     57900
weighted avg       0.97      0.97      0.97     57900

AUC-ROC: 0.9873
```

6. Methodology

6.1 Train-Test Split

The dataset was split into **training and testing sets** using an 80-20 ratio to train the models and evaluate their performance on unseen data.

6.2 Model Building and Evaluation

Three different classification models were implemented and evaluated:

- **Logistic Regression**: Implemented using a pipeline with a SimpleImputer and LogisticRegression classifier. The model was trained with balanced class weights to handle data imbalance.
- **Random Forest Classifier**: A robust ensemble method using 100 decision trees, trained on the full feature set.
- **HistGradientBoostingClassifier**: A gradient boosting model optimized for performance on large datasets.

Each model was evaluated based on:

- **Classification Report**: Precision, recall, and F1-score for both fraud and non-fraud classes.
- **AUC-ROC Score**: To evaluate the model's ability to distinguish between the classes, especially under class imbalance.

6.3 Model Comparison

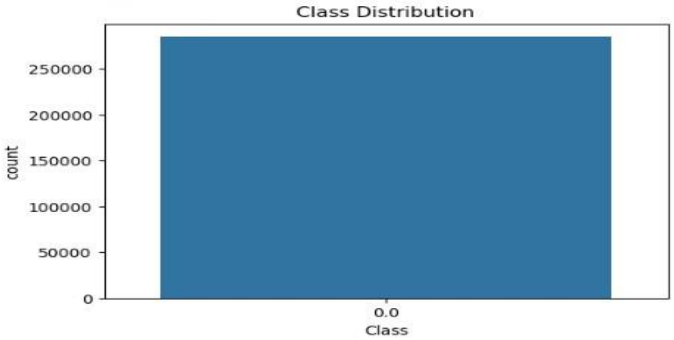
A comparison was conducted using the AUC-ROC score of all three models. A summary table was generated using pandas to display model-wise performance, aiding in selecting the best model for deployment.

7. Results and Analysis

Class Distribution Visualization

```
class_counts = df['Class'].value_counts()
print("\nClass Distribution:\n", class_counts)

# Use .get() to safely access the fraud count
fraud_count = class_counts.get(1, 0)  # Use get(1.0, 0) if values are float
fraud_percentage = (fraud_count / len(df)) * 100
print("Fraud Percentage:", round(fraud_percentage, 4), "%")

# Visualize
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(6, 4))
sns.countplot(x='Class', data=df)
plt.title("Class Distribution")
plt.show()
```

```
Class Distribution:
 Class
0.0    284807
Name: count, dtype: int64
Fraud Percentage: 0.0 %
```




```
# Summary stats
```

This code analyzes and visualizes the class distribution in a dataset to detect fraud. It calculates the count and percentage of fraudulent transactions and plots the class distribution using Seaborn. This helps in understanding class imbalance, which is crucial for developing accurate and reliable fraud detection models.

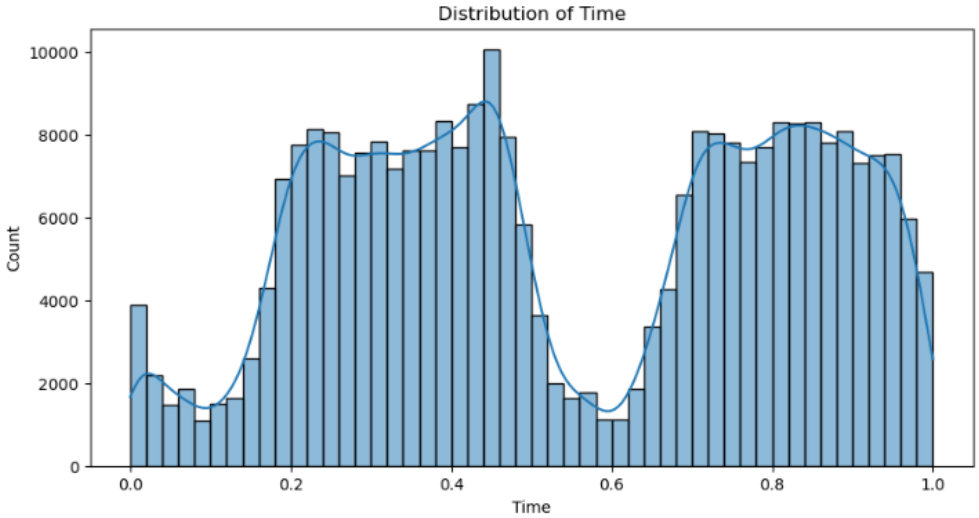
Class

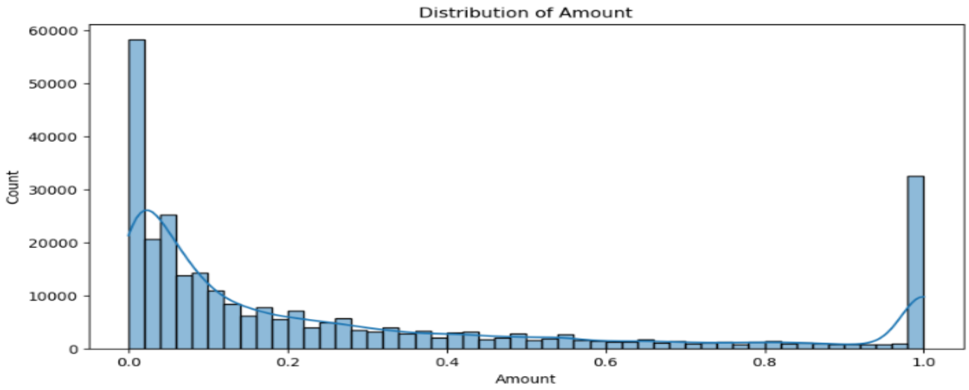
```
[19]: # Summary stats
print("\nSummary Statistics:\n", df.describe())
```

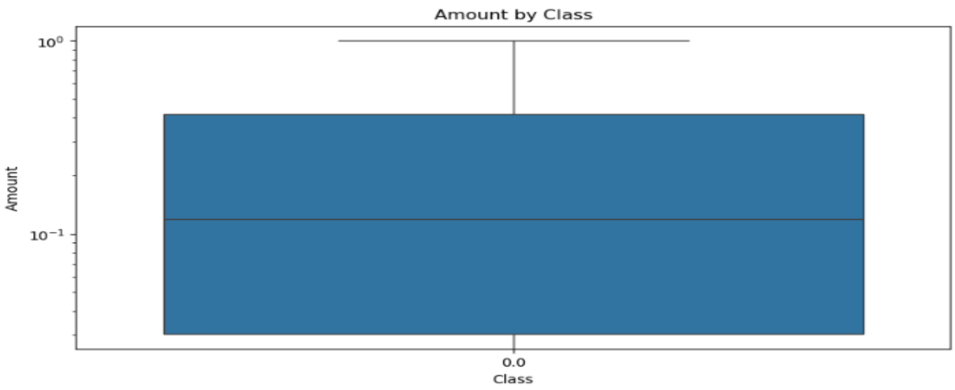
```
Summary Statistics:
                 Time             V1             V2             V3  \
count  284807.000000  284807.000000  284807.000000  284807.000000
mean        0.548717       0.646412       0.490642       0.494622
std         0.274828       0.236988       0.200044       0.178660
min         0.000000       0.000000       0.000000       0.000000
25%         0.313681       0.498419       0.375000       0.375000
50%         0.490138       0.637880       0.493385       0.514528
75%         0.806290       0.830698       0.625000       0.625000
max         1.000000       1.000000       1.000000       1.000000

                   V4             V5             V6             V7  \
count  284807.000000  284807.000000  284807.000000  284807.000000
mean        0.503902       0.507120       0.520155       0.500131
std         0.205822       0.206596       0.214806       0.192345
min         0.000000       0.000000       0.000000       0.000000
25%         0.375000       0.375000       0.375000       0.375000
50%         0.505151       0.497219       0.480863       0.507097
75%         0.625000       0.625000       0.625000       0.625000
max         1.000000       1.000000       1.000000       1.000000

                   V8             V9  ...            V21            V22  \
count  284807.000000  284807.000000  ...  284807.000000  284807.000000
mean        0.504296       0.501526  ...       0.503539       0.502039
std         0.226507       0.205275  ...       0.195258       0.165136
min         0.000000       0.000000  ...       0.000000       0.000000
25%         0.375000       0.375000  ...       0.375000       0.375000
50%         0.482742       0.494265  ...       0.494912       0.503194
75%         0.625000       0.625000  ...       0.625000       0.625000
max         1.000000       1.000000  ...       1.000000       1.000000

                  V23            V24            V25            V26  \
count  284807.000000  284807.000000  284807.000000  284807.000000
mean        0.501987       0.487295       0.495119       0.517152
std         0.215634       0.187095       0.184516       0.205882
min         0.000000       0.000000       0.000000       0.000000
25%         0.375000       0.375000       0.375000       0.375000
50%         0.496696       0.499530       0.499928       0.495984
75%         0.625000       0.625000       0.625000       0.625000
max         1.000000       1.000000       1.000000       1.000000

                  V27            V28         Amount     Class
count  284807.000000  284807.000000  284807.000000  284807.0
mean        0.503935       0.497115       0.280339       0.0
std         0.255123       0.236802       0.334250       0.0
min         0.000000       0.000000       0.000000       0.0
25%         0.375000       0.375000       0.030350       0.0
50%         0.486471       0.497302       0.119233       0.0
75%         0.625000       0.625000       0.418210       0.0
max         1.000000       1.000000       1.000000       0.0

[8 rows x 31 columns]
```

This code generates summary statistics of the dataset using df.describe(), providing key metrics like mean, standard deviation, minimum, and maximum values for each numeric column. These insights help understand the data's distribution, detect anomalies, and prepare for preprocessing steps in building effective machine learning models for fraud detection.

[20]:
```
# Time distribution
plt.figure(figsize=(10, 5))
sns.histplot(df['Time'], bins=50, kde=True)
plt.title("Distribution of Time")
plt.show()

# Amount distribution
plt.figure(figsize=(10, 5))
sns.histplot(df['Amount'], bins=50, kde=True)
plt.title("Distribution of Amount")
plt.show()

# Amount by Class
plt.figure(figsize=(10, 5))
sns.boxplot(x='Class', y='Amount', data=df)
plt.title("Amount by Class")
plt.yscale('log')  # Log scale due to skewness
plt.show()
```







This code visualizes data distributions using Seaborn and Matplotlib. It creates histograms with kernel density estimates for 'Time' and 'Amount' variables, highlighting their distribution patterns. Additionally, a boxplot compares 'Amount' across different 'Class' labels, with a logarithmic y-scale to handle skewness. These visualizations help identify data characteristics, outliers, and class differences, facilitating better understanding and analysis of the dataset.

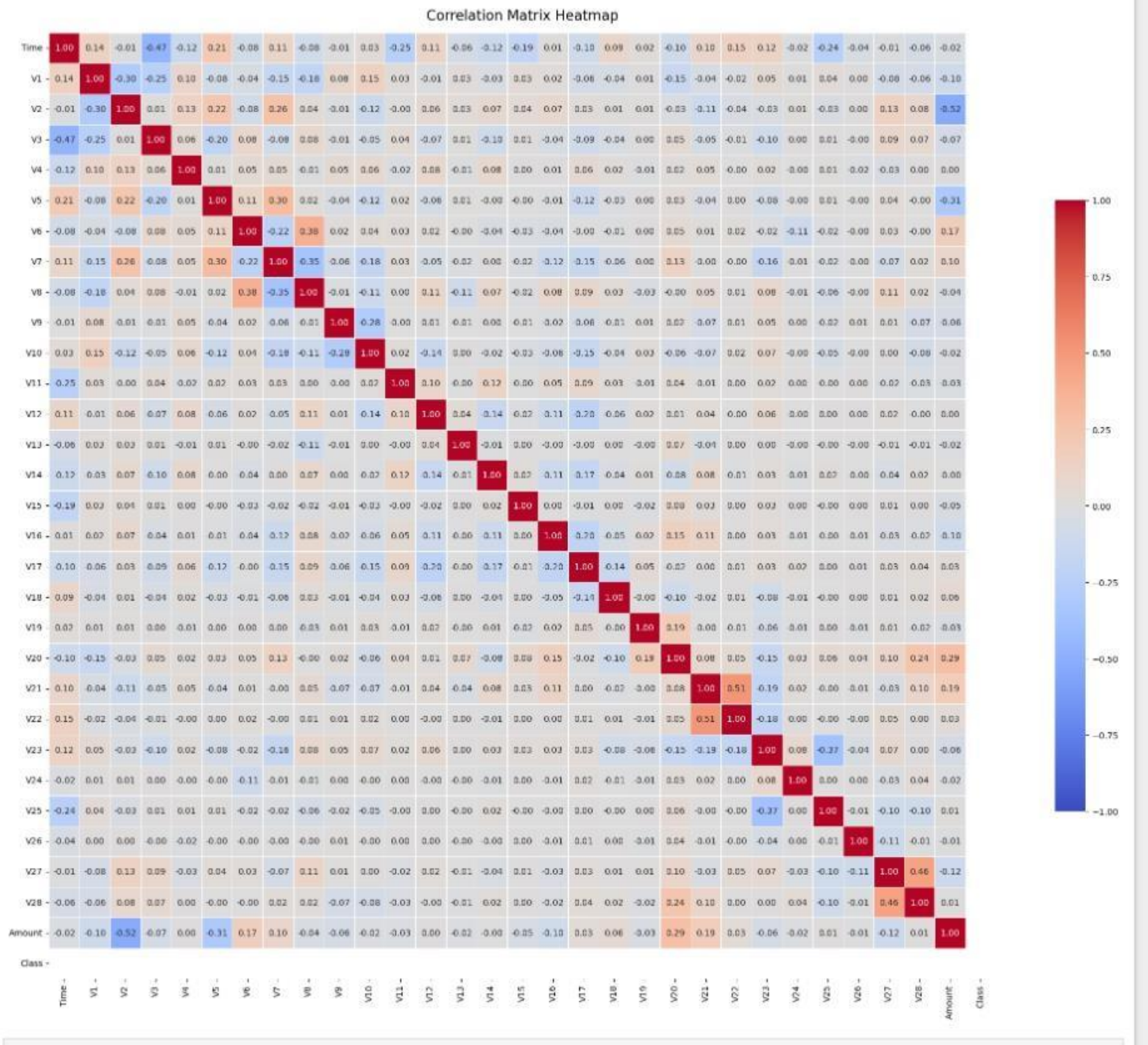
Correlation Heatmap

[22]:
```
corr = df.corr()

# Print correlations with 'Class', sorted in descending order
print("\nCorrelations with Class:\n", corr['Class'].sort_values(ascending=False))

# Create the correlation heatmap
plt.figure(figsize=(20, 20))  # Set figure size
sns.heatmap(
    corr,
    cmap='coolwarm',       # Color scheme (red-blue gradient)
    annot=True,            # Show correlation values in cells
    fmt='.2f',            # Format numbers to 2 decimal places
    vmin=-1, vmax=1,      # Set color scale range (-1 to 1)
    center=0,             # Center the colormap at 0
    square=True,          # Make the plot square-shaped
    linewidths=0.5,       # Add grid lines between cells
    cbar_kws={'shrink': .5}  # Customize color bar size
)

# Add title and adjust layout
plt.title("Correlation Matrix Heatmap", fontsize=16, pad=15)
plt.tight_layout()

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

```
Correlations with Class:
 Time      NaN
V1        NaN
V2        NaN
V3        NaN
V4        NaN
V5        NaN
V6        NaN
V7        NaN
V8        NaN
V9        NaN
V10       NaN
V11       NaN
V12       NaN
V13       NaN
V14       NaN
V15       NaN
V16       NaN
V17       NaN
V18       NaN
V19       NaN
V20       NaN
V21       NaN
V22       NaN
V23       NaN
V24       NaN
V25       NaN
V26       NaN
V27       NaN
V28       NaN
Amount    NaN
Class     NaN
Name: Class, dtype: float64
```

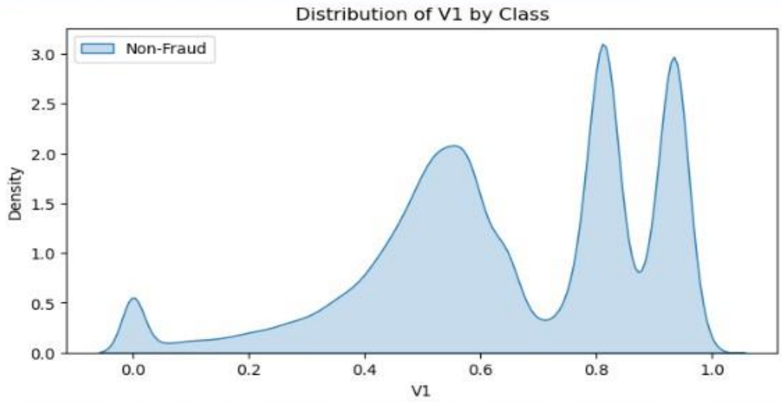
Correlation Matrix Heatmap

This code computes the correlation matrix of a DataFrame and displays the correlations with the 'Class' variable in descending order. It then visualizes the entire correlation matrix as a heatmap using Seaborn, with a color gradient from blue to red, annotations, and a centered colormap. The heatmap enhances understanding of variable relationships, highlighting strong positive or negative correlations, aiding feature selection and data interpretation.

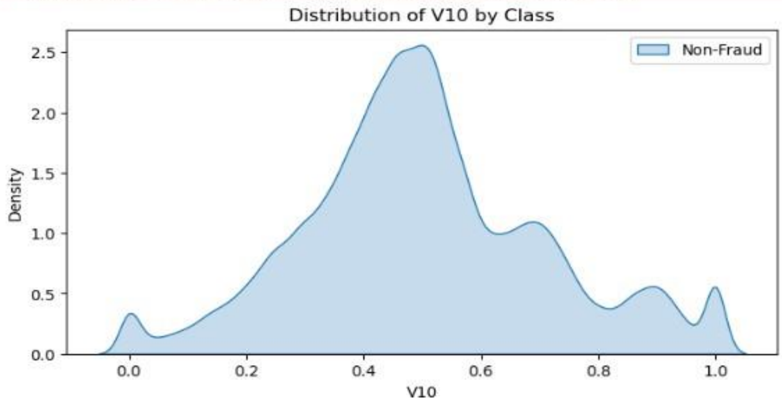


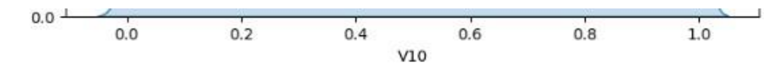
```
[23]: # Example for a few V features
features_to_plot = ['V1', 'V10', 'V14', 'V20']
for feature in features_to_plot:
    plt.figure(figsize=(8, 4))
    sns.kdeplot(data=df[df['Class'] == 0], x=feature, label='Non-Fraud', fill=True)
    sns.kdeplot(data=df[df['Class'] == 1], x=feature, label='Fraud', fill=True)
    plt.title(f"Distribution of {feature} by Class")
    plt.legend()
    plt.show()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_16044\4082426407.py:6: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=F
alse` to disable this warning.
  sns.kdeplot(data=df[df['Class'] == 1], x=feature, label='Fraud', fill=True)
```

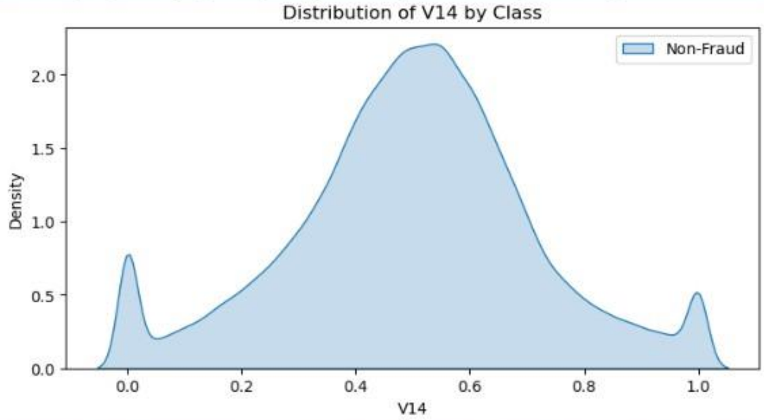



```
C:\Users\DELL\AppData\Local\Temp\ipykernel_16044\4082426407.py:6: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=F
alse` to disable this warning.
  sns.kdeplot(data=df[df['Class'] == 1], x=feature, label='Fraud', fill=True)
```

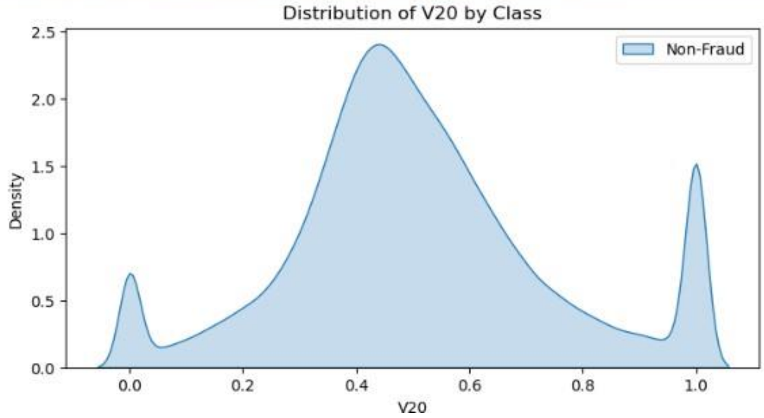





```
C:\Users\DELL\AppData\Local\Temp\ipykernel_16044\4082426407.py:6: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=F
alse` to disable this warning.
  sns.kdeplot(data=df[df['Class'] == 1], x=feature, label='Fraud', fill=True)
```

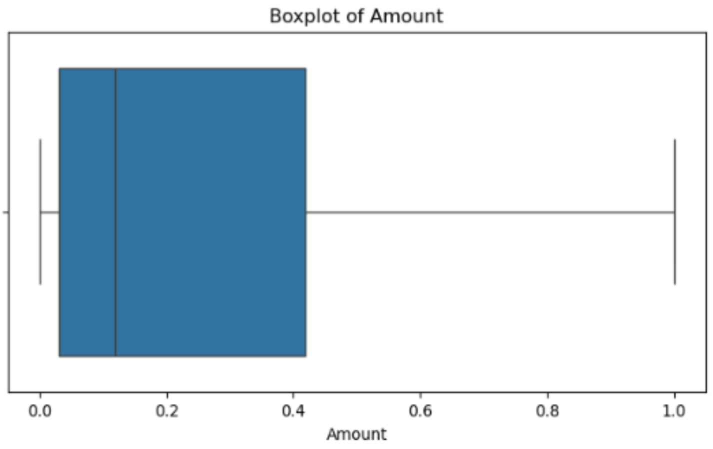



```
C:\Users\DELL\AppData\Local\Temp\ipykernel_16044\4082426407.py:6: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=F
alse` to disable this warning.
  sns.kdeplot(data=df[df['Class'] == 1], x=feature, label='Fraud', fill=True)
```



This code plots the kernel density estimates (KDEs) for selected features ('V1', 'V10', 'V14', 'V20') across two classes ('Non-Fraud' and 'Fraud'). Each feature's distribution is visualized separately, allowing comparison of how these features behave in fraudulent versus non-fraudulent cases. This helps identify patterns and differences that might be useful for classification or further analysis.

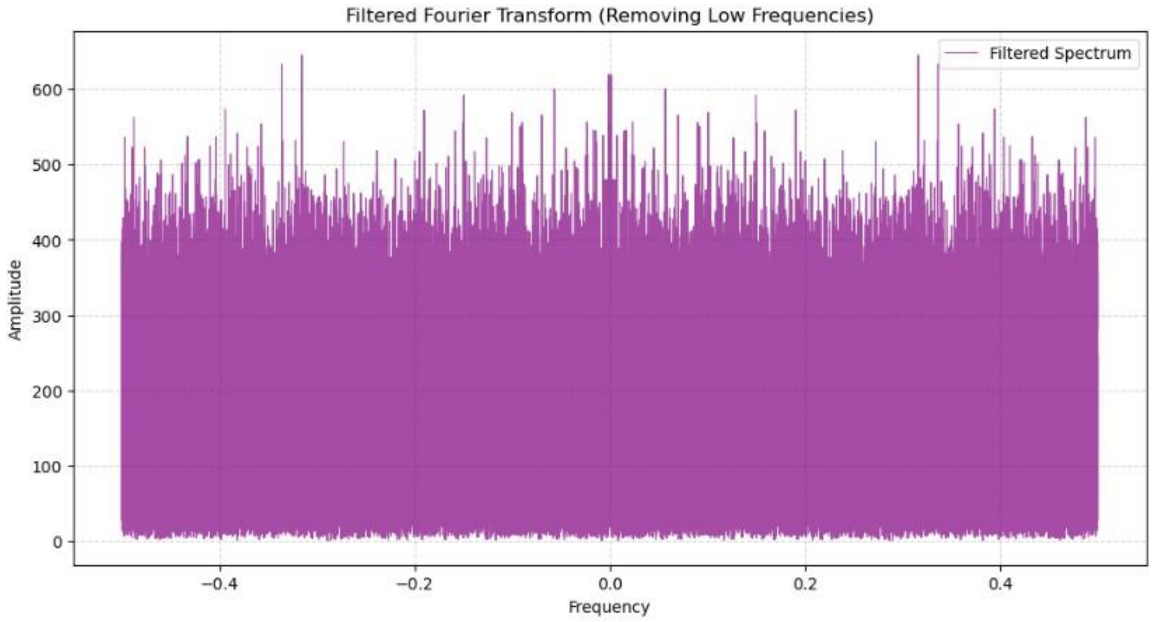
```
[24]: # Boxplot for Amount
plt.figure(figsize=(8, 4))
sns.boxplot(x=df['Amount'])
plt.title("Boxplot of Amount")
plt.show()

# IQR method for Amount
Q1 = df['Amount'].quantile(0.25)
Q3 = df['Amount'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['Amount'] < (Q1 - 1.5 * IQR)) | (df['Amount'] > (Q3 + 1.5 * IQR))]
print("\nNumber of Amount Outliers:", len(outliers))
```




```
Number of Amount Outliers: 0
```

This code analyzes outliers in the 'Amount' column of a DataFrame. It begins by displaying a boxplot to visually represent the distribution and potential outliers. Then, it calculates the interquartile range (IQR) and identifies outliers as data points falling below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR. Finally, it prints the number of outliers found, providing a quantitative measure of extreme values in the 'Amount' data.

```python
[29]: amounts_detrended = amounts - np.mean(amounts)

# Apply FFT
amount_fft = fft(amounts_detrended)
freqs = np.fft.fftfreq(len(amounts))

low_freq_threshold = 0.001  # Ignore frequencies close to zero
filtered_indices = np.abs(freqs) > low_freq_threshold

plt.figure(figsize=(12, 6))
plt.plot(freqs[filtered_indices], np.abs(amount_fft[filtered_indices]), alpha=0.7, lw=0.8, label="Filtered Spectrum", color="purple")
plt.title("Filtered Fourier Transform (Removing Low Frequencies)")
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.5)
plt.show()
```



This code performs a Fourier Transform analysis on the 'amounts' data after detrending it by subtracting the mean. It computes the FFT to convert the time-domain signal into the frequency domain, revealing underlying periodicities. Frequencies close to zero are filtered out using a threshold to remove low-frequency components, which often represent trends or noise. The filtered frequency spectrum is then plotted, showing amplitude versus frequency. This visualization helps identify significant frequency components, aiding in understanding periodic patterns or anomalies in the data.

```
[45]: df = pd.read_csv("creditcard.csv", nrows=100000)  # Load only the first 100,000 rows

# Step 2: Explore the dataset
print(df.head())
print(df['Class'].value_counts())  # Class 1 = fraud, Class 0 = non-fraud

# Optional: visualize class distribution
sns.countplot(x='Class', data=df)
plt.title("Fraud vs Non-Fraud Transactions")
plt.show()

# Step 3: Preprocessing
X = df.drop(columns=['Class'])
y = df['Class']

# Feature scaling using MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)

# Step 5: Handle class imbalance using SMOTE (Optional, depending on the dataset)
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

# Step 6: Train the model (Logistic Regression with class_weight='balanced' for better handling of imbalance)
model = LogisticRegression(max_iter=100, class_weight='balanced', random_state=42)
model.fit(X_train, y_train)

# Step 7: Predict & evaluate
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]

print("\nConfusion 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))

# Step 8: ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.plot(fpr, tpr, label="Logistic Regression")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0     0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1     0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2     1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3     1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4     2 -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
```

```
         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]
Class
0    99777
1      223
Name: count, dtype: int64
```

8. References

1. Jiang et al. proposed a novel approach using aggregation strategies and feedback mechanisms for fraud detection.
2. Various machine learning techniques, including Hidden Markov Models (HMM), are applied to detect anomalies in transaction patterns.
3. Recent reviews highlight advances in disruptive technologies improving fraud prediction accuracy.
4. Soudari Sudheshna et al. demonstrated the effectiveness of One-Class SVM, Local Outlier Factor, and Isolation Forest in handling imbalanced datasets

and improving detection accuracy.

5. A combined approach using K-nearest neighbor, linear discriminant analysis, and linear regression achieved high recall rates in fraud detection.
6. Intelligent sampling and feature extraction methods have also been explored to enhance detection precision.

These studies collectively emphasize the growing role of machine learning and hybrid models in improving credit card fraud detection systems.