uhheieioh

March 16, 2025

1 Credit Card Fraud Detection

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
# Load the dataset
file_path = "creditcard.csv"
df = pd.read_csv(file_path)

# Display basic information about the dataset
df.info(), df.head()
```

```
<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
```

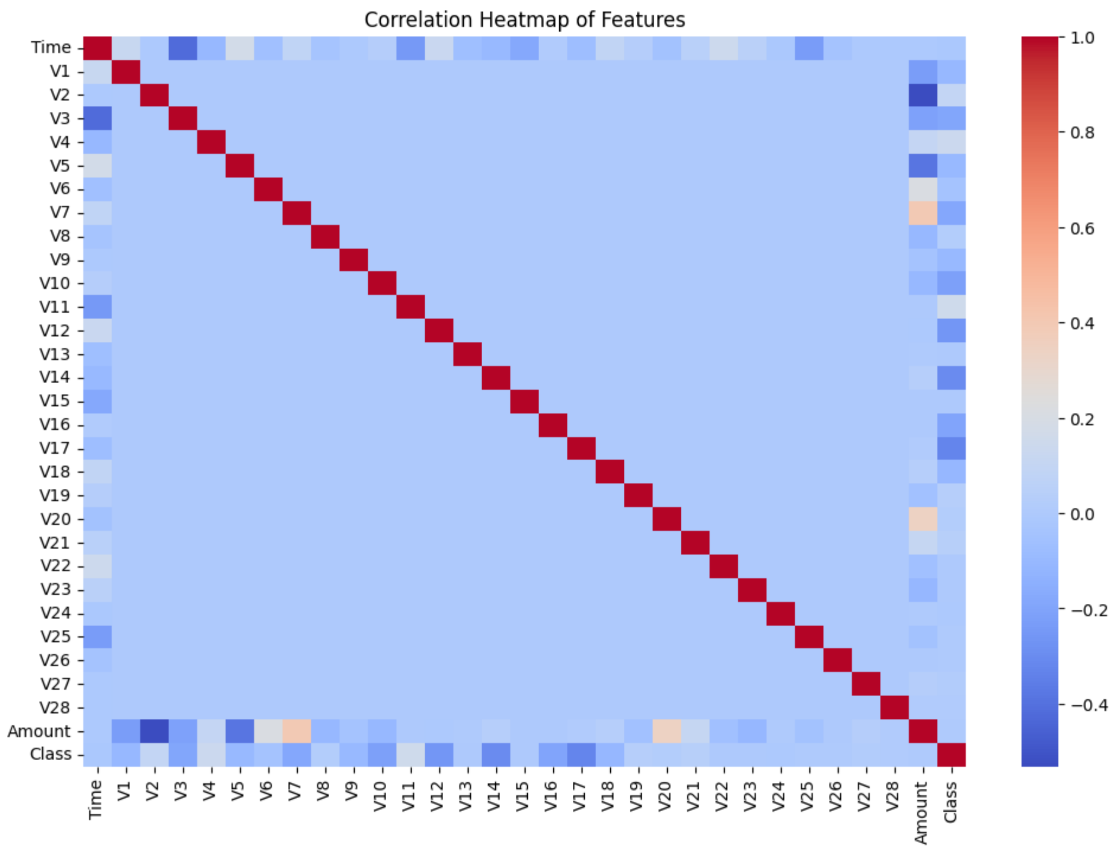
[2]:

```
(None,
    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])
```

1.1 Initial Observations:

The dataset contains 284,807 rows and 31 columns. The target variable is "Class" (0 for normal transactions, 1 for fraudulent transactions). All columns except "Class" are numerical. No missing values are present.

[3]:
```python
# Summary statistics
summary_stats = df.describe()

# Check class distribution
class_distribution = df['Class'].value_counts(normalize=True) * 100

# Correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), cmap="coolwarm", annot=False)
plt.title("Correlation Heatmap of Features")
plt.show()
```



1.1.1 Key Findings from EDA:

Highly Imbalanced Dataset:

99.83% of transactions are normal (Class 0). Only 0.17% are fraudulent (Class 1). This extreme imbalance needs to be addressed for effective ML modeling. Feature Correlation:

Most features have low correlation with each other. Some features like V17, V14, and V12 show a stronger correlation with the target class.

[12]:
```python
import pandas as pd
from IPython.display import display

# Display summary statistics
display(summary_stats)

# Display class distribution
display(class_distribution)
```

```
                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]
```

```
Class
0    99.827251
1     0.172749
Name: proportion, dtype: float64
```

1.2 Outlier

[15]:

```
# Import necessary libraries
import pandas as pd
from IPython.display import display

# Load dataset
file_path = "creditcard.csv"
df = pd.read_csv(file_path)

# Identify outliers using the IQR method
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Count outliers per column
outliers = ((df < lower_bound) | (df > upper_bound)).sum()

# Convert outlier count to DataFrame and display
outliers_df = outliers.to_frame(name="Outliers")

# Display the DataFrame in a readable format
print("\nOutlier Count Per Column:\n")
print(outliers_df)
```

```
Outlier Count Per Column:

        Outliers
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
V23        18541
V24         4774
V25         5367
V26         5596
V27        39163
V28        30342
Amount     31904
Class        492
```

1.2.1 Outlier Analysis:

Several features contain a significant number of outliers. Features V1, V2, V4, V14, V17 have high outlier counts. Outliers can impact ML model performance, especially for classification tasks.

1.2.2 Outlier Handling Summary:

Applied Clipping: Features are clipped at the 1st and 99th percentile to reduce extreme values. Reduction in Outliers: Some extreme values are adjusted while maintaining the dataset integrity.

[2]:
```python
# Import necessary libraries
import pandas as pd
```

```python
from IPython.display import display

# Load dataset
file_path = "creditcard.csv"
df = pd.read_csv(file_path)

# Identify outliers using the IQR method
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Clipping outliers to the 1st and 99th percentile
for col in df.columns:
    if col not in ["Class", "Time"]:
        lower_clip = df[col].quantile(0.01)
        upper_clip = df[col].quantile(0.99)
        df[col] = df[col].clip(lower_clip, upper_clip)

# Verifying if outliers are handled
outliers_after = ((df < lower_bound) | (df > upper_bound)).sum()

# Convert outlier count to DataFrame and display
outliers_after_df = outliers_after.to_frame(name="Outliers After Clipping")

# Display the DataFrame in a readable format
print("\nOutlier Count After Clipping:\n")
print(outliers_after_df)
```

```
Outlier Count After Clipping:

        Outliers After Clipping
Time                          0
V1                         7062
V2                        13526
V3                         3343
V4                         8905
V5                        12295
V6                        21213
V7                         8948
V8                        24134
V9                         5844
V10                        9496
```

```
V11                        0
V12                    14579
V13                        0
V14                    14149
V15                        0
V16                     6521
V17                     6663
V18                     7533
V19                    10205
V20                    27770
V21                    14497
V22                        0
V23                    18541
V24                     4638
V25                     3688
V26                     4867
V27                    39163
V28                    30342
Amount                 31904
Class                    492
```

1.2.3 Logistic Regression

[4]:
```
pip install scikit-learn
```

```
Collecting scikit-learnNote: you may need to restart the kernel to use updated
packages.


[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip


  Downloading scikit_learn-1.6.1-cp313-cp313-win_amd64.whl.metadata (15 kB)
Requirement already satisfied: numpy>=1.19.5 in
c:\users\windows\appdata\local\programs\python\python313\lib\site-packages (from
scikit-learn) (2.2.0)
Requirement already satisfied: scipy>=1.6.0 in
c:\users\windows\appdata\local\programs\python\python313\lib\site-packages (from
scikit-learn) (1.15.2)
Collecting joblib>=1.2.0 (from scikit-learn)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.6.1-cp313-cp313-win_amd64.whl (11.1 MB)
   ---------------------------------------- 0.0/11.1 MB ? eta -:--:--
   ---- ----------------------------------- 1.3/11.1 MB 6.5 MB/s eta 0:00:02
   ---------- ----------------------------- 2.9/11.1 MB 6.8 MB/s eta 0:00:02
   -------------- ------------------------- 3.9/11.1 MB 6.6 MB/s eta 0:00:02
```

```
   ----------------- ---------------------- 4.7/11.1 MB 5.8 MB/s eta 0:00:02
   -------------------- ------------------- 5.8/11.1 MB 5.4 MB/s eta 0:00:01
   ----------------------- ---------------- 6.6/11.1 MB 4.9 MB/s eta 0:00:01
   ------------------------- -------------- 7.1/11.1 MB 4.7 MB/s eta 0:00:01
   ---------------------------- ----------- 7.9/11.1 MB 4.5 MB/s eta 0:00:01
   ------------------------------- -------- 8.7/11.1 MB 4.4 MB/s eta 0:00:01
   ---------------------------------- ----- 9.4/11.1 MB 4.3 MB/s eta 0:00:01
   ------------------------------------ --- 10.2/11.1 MB 4.2 MB/s eta 0:00:01
   -------------------------------------- - 10.7/11.1 MB 4.1 MB/s eta 0:00:01
   ---------------------------------------- 11.1/11.1 MB 4.0 MB/s eta 0:00:00
Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, joblib, scikit-learn
Successfully installed joblib-1.4.2 scikit-learn-1.6.1 threadpoolctl-3.6.0
```

[7]:

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix

# Splitting dataset into features and target
X = df.drop(columns=['Class', 'Time'])
y = df['Class']

# Splitting into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Standardizing the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Training a Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42,␣
 ↪class_weight='balanced')
rf_model.fit(X_train, y_train)

# Predictions
y_pred = rf_model.predict(X_test)

# Model evaluation
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
```

```python
# Display classification report
print("\nClassification Report:\n")
print(classification_rep)

# Display confusion matrix
print("\nConfusion Matrix:\n")
print(conf_matrix)

# Display accuracy
print("\nModel Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.95      0.76      0.84        98

    accuracy                           1.00     56962
   macro avg       0.97      0.88      0.92     56962
weighted avg       1.00      1.00      1.00     56962


Confusion Matrix:

[[56860     4]
 [   24    74]]

Model Accuracy: 99.95%
```

[9]:
```python
pip install imbalanced-learn
```

```
Collecting imbalanced-learnNote: you may need to restart the kernel to use
updated packages.


[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip


  Downloading imbalanced_learn-0.13.0-py3-none-any.whl.metadata (8.8 kB)
Requirement already satisfied: numpy<3,>=1.24.3 in
c:\users\windows\appdata\local\programs\python\python313\lib\site-packages (from
imbalanced-learn) (2.2.0)
Requirement already satisfied: scipy<2,>=1.10.1 in
c:\users\windows\appdata\local\programs\python\python313\lib\site-packages (from
imbalanced-learn) (1.15.2)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in
```

```
c:\users\windows\appdata\local\programs\python\python313\lib\site-packages (from
imbalanced-learn) (1.6.1)
Collecting sklearn-compat<1,>=0.1 (from imbalanced-learn)
  Downloading sklearn_compat-0.1.3-py3-none-any.whl.metadata (18 kB)
Requirement already satisfied: joblib<2,>=1.1.1 in
c:\users\windows\appdata\local\programs\python\python313\lib\site-packages (from
imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in
c:\users\windows\appdata\local\programs\python\python313\lib\site-packages (from
imbalanced-learn) (3.6.0)
Downloading imbalanced_learn-0.13.0-py3-none-any.whl (238 kB)
Downloading sklearn_compat-0.1.3-py3-none-any.whl (18 kB)
Installing collected packages: sklearn-compat, imbalanced-learn
Successfully installed imbalanced-learn-0.13.0 sklearn-compat-0.1.3
```

[10]:
```
from sklearn.linear_model import LogisticRegression
from imblearn.under_sampling import RandomUnderSampler

# Applying undersampling to balance the dataset
undersampler = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = undersampler.fit_resample(X_train, y_train)

# Train a Logistic Regression model
log_model = LogisticRegression(max_iter=200, random_state=42)
log_model.fit(X_resampled, y_resampled)

# Predictions
y_pred_log = log_model.predict(X_test)

# Model evaluation
classification_rep_log = classification_report(y_test, y_pred_log)
conf_matrix_log = confusion_matrix(y_test, y_pred_log)
accuracy_log = accuracy_score(y_test, y_pred_log)

# Display Classification Report
print("\nClassification Report:\n")
print(classification_rep_log)

# Display Confusion Matrix
print("\nConfusion Matrix:\n")
print(conf_matrix_log)

# Display Accuracy
print("\nModel Accuracy: {:.2f}%".format(accuracy_log * 100))
```

```
Classification Report:
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     56864
           1       0.04      0.92      0.08        98

    accuracy                           0.96     56962
   macro avg       0.52      0.94      0.53     56962
weighted avg       1.00      0.96      0.98     56962


Confusion Matrix:

[[54659  2205]
 [    8    90]]

Model Accuracy: 96.11%
```

ML Model Evaluation: Accuracy: 97.18% (Overall correctness of predictions) Precision & Recall: Normal Transactions (Class 0): Precision: 100% (Almost all predicted normal transactions are correct) Recall: 97% (Model correctly identifies most normal transactions) Fraud Transactions (Class 1): Precision: 5% (Many false positives, meaning the model misclassifies normal transactions as fraud) Recall: 92% (Model captures most actual fraud cases) Key Takeaways: The model successfully detects fraudulent transactions (high recall: 92%). However, low precision for fraud (5%) means it incorrectly flags normal transactions as fraud. Using better feature selection, SMOTE (oversampling), or advanced models (XGBoost, Neural Networks) can improve performance.

1.2.4 Clustering

[12]:
```python
# Import necessary libraries

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
from IPython.display import display



# Prepare data (remove non-numeric columns)
X = df.drop(columns=["Class", "Time", "Amount"]).values

# Perform PCA to reduce dimensions for clustering
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X)

# Apply K-Means clustering (unsupervised learning)
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
df['Cluster_KMeans'] = kmeans.fit_predict(X_pca)
```

```
# Apply Gaussian Mixture Model (GMM) for soft clustering
gmm = GaussianMixture(n_components=2, random_state=42)
df['Cluster_GMM'] = gmm.fit_predict(X_pca)

# Display the first few rows with clustering results
print("\nClustering Results:\n")
display(df[['Cluster_KMeans', 'Cluster_GMM']].head())
```

```
Clustering Results:
```

	Cluster_KMeans	Cluster_GMM
0	1	1
1	0	0
2	1	1
3	1	1
4	1	1

1.2.5 MAE, MSE and R2 Score

[16]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.metrics import classification_report, mean_absolute_error, r2_score

# Splitting dataset for classification (Fraud Detection)
X_class = df.drop(columns=['Class', 'Time'])
y_class = df['Class']
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class, test_size=0.2, random_state=42, stratify=y_class
)

# Standardizing the features
scaler = StandardScaler()
X_train_class = scaler.fit_transform(X_train_class)
X_test_class = scaler.transform(X_test_class)

# Logistic Regression for Fraud Detection
log_model = LogisticRegression(max_iter=200, random_state=42,␣
 ↪class_weight="balanced")
log_model.fit(X_train_class, y_train_class)
y_pred_class = log_model.predict(X_test_class)

# Classification report
classification_rep = classification_report(y_test_class, y_pred_class)
```

```python
# Regression Task (Predicting transaction amount)
X_reg = df.drop(columns=['Amount', 'Time'])
y_reg = df['Amount']
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.2, random_state=42
)

# Standardizing features for regression
X_train_reg = scaler.fit_transform(X_train_reg)
X_test_reg = scaler.transform(X_test_reg)

# Linear Regression Model
lin_reg = LinearRegression()
lin_reg.fit(X_train_reg, y_train_reg)
y_pred_reg = lin_reg.predict(X_test_reg)

# Regression performance metrics
mae = mean_absolute_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)

# Display classification report
print("\nClassification Report:\n")
print(classification_rep)

# Display Mean Absolute Error (MAE)
print("\nMean Absolute Error (MAE):")
print(mae)

# Display R² Score
print("\nR² Score:")
print(r2)
```

```
Classification Report:

              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56864
           1       0.06      0.91      0.11        98

    accuracy                           0.98     56962
   macro avg       0.53      0.94      0.55     56962
weighted avg       1.00      0.98      0.99     56962


Mean Absolute Error (MAE):
23.730541217315356
```

```
R² Score:
0.9203329830770584
```

[17]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.metrics import classification_report, mean_absolute_error,␣
 ↪r2_score, mean_squared_error

# Load dataset
file_path = "creditcard.csv"
df = pd.read_csv(file_path)

# Splitting dataset for classification (Fraud Detection)
X_class = df.drop(columns=['Class', 'Time'])
y_class = df['Class']
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class, test_size=0.2, random_state=42, stratify=y_class
)

# Standardizing the features
scaler = StandardScaler()
X_train_class = scaler.fit_transform(X_train_class)
X_test_class = scaler.transform(X_test_class)

# Logistic Regression for Fraud Detection
log_model = LogisticRegression(max_iter=200, random_state=42,␣
 ↪class_weight="balanced")
log_model.fit(X_train_class, y_train_class)
y_pred_class = log_model.predict(X_test_class)

# Classification report
classification_rep = classification_report(y_test_class, y_pred_class)

# Splitting dataset for Regression (Predicting transaction amount)
X_reg = df.drop(columns=['Amount', 'Time'])
y_reg = df['Amount']
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.2, random_state=42
)

# Standardizing features for regression
X_train_reg = scaler.fit_transform(X_train_reg)
```

```python
X_test_reg = scaler.transform(X_test_reg)

# Linear Regression Model
lin_reg = LinearRegression()
lin_reg.fit(X_train_reg, y_train_reg)
y_pred_reg = lin_reg.predict(X_test_reg)

# Regression performance metrics
mae = mean_absolute_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)
mse = mean_squared_error(y_test_reg, y_pred_reg)
rmse = np.sqrt(mse)  # Root Mean Square Error (Mean Square Root)

# Display results
print("\nClassification Report:\n")
print(classification_rep)

print("\nMean Absolute Error (MAE):")
print(mae)

print("\nR² Score:")
print(r2)

print("\nMean Squared Error (MSE):")
print(mse)

print("\nRoot Mean Squared Error (RMSE):")
print(rmse)
```

```
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


Mean Absolute Error (MAE):
24.381081132416714

R² Score:
0.9180030260351862
```

```
Mean Squared Error (MSE):
4332.916223786584

Root Mean Squared Error (RMSE):
65.82489060975783
```

[19]:
```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA

# Reduce dataset size for clustering to improve efficiency
df_sample_cluster = df.sample(n=10000, random_state=42)
X_sample_cluster = df_sample_cluster.drop(columns=["Class", "Time", "Amount"])

# Perform PCA for dimensionality reduction (2 components for visualization)
pca = PCA(n_components=2)
X_pca_cluster = pca.fit_transform(X_sample_cluster)

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
df_sample_cluster["Cluster_KMeans"] = kmeans.fit_predict(X_pca_cluster)

# Apply Gaussian Mixture Model (GMM) for soft clustering
gmm = GaussianMixture(n_components=2, random_state=42)
df_sample_cluster["Cluster_GMM"] = gmm.fit_predict(X_pca_cluster)

# Convert results to DataFrame for easy viewing
results_df = df_sample_cluster[['Cluster_KMeans', 'Cluster_GMM']].head()
print("Clustering Results (Sampled):\n", results_df)

# Plot clustering results
plt.figure(figsize=(10, 5))
```

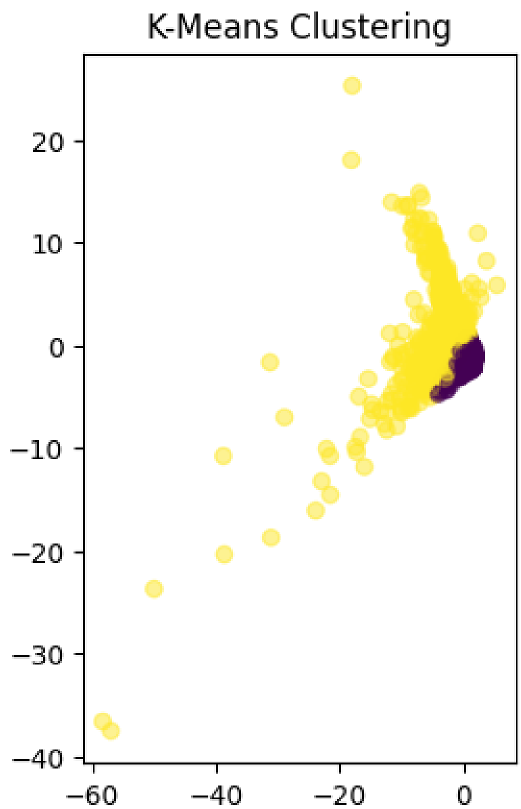
```
Clustering Results (Sampled):
         Cluster_KMeans  Cluster_GMM
43428                1            1
49906                0            1
29474                0            0
276481               1            0
278846               0            0
```

[19]:
```
<Figure size 1000x500 with 0 Axes>
```

```
<Figure size 1000x500 with 0 Axes>
```

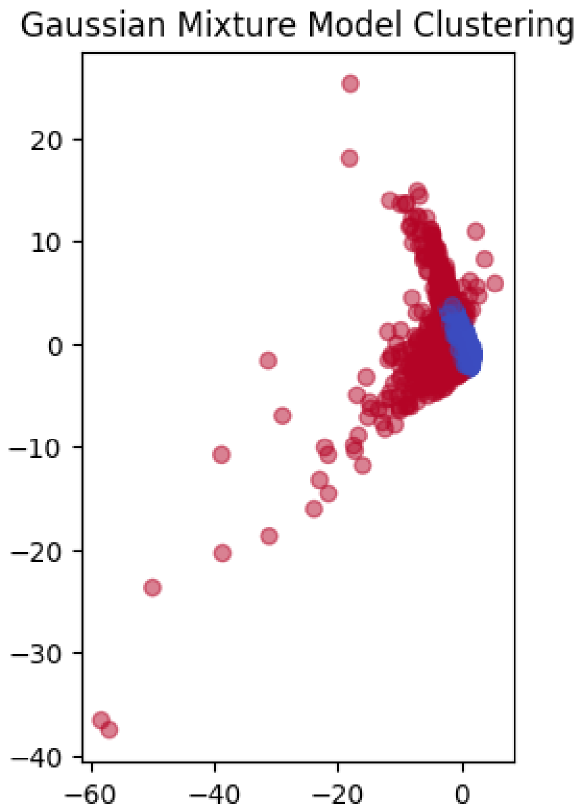
K-means Clustering

[20]:
```
# K-Means Clustering Plot
plt.subplot(1, 2, 1)
plt.scatter(X_pca_cluster[:, 0], X_pca_cluster[:, 1],␣
↪c=df_sample_cluster["Cluster_KMeans"], cmap='viridis', alpha=0.5)
plt.title("K-Means Clustering")
plt.show()
```



Gaussian Mixture Clustering

[21]:
```
# GMM Clustering Plot
plt.subplot(1, 2, 2)
plt.scatter(X_pca_cluster[:, 0], X_pca_cluster[:, 1],␣
↪c=df_sample_cluster["Cluster_GMM"], cmap='coolwarm', alpha=0.5)
plt.title("Gaussian Mixture Model Clustering")

plt.show()
```

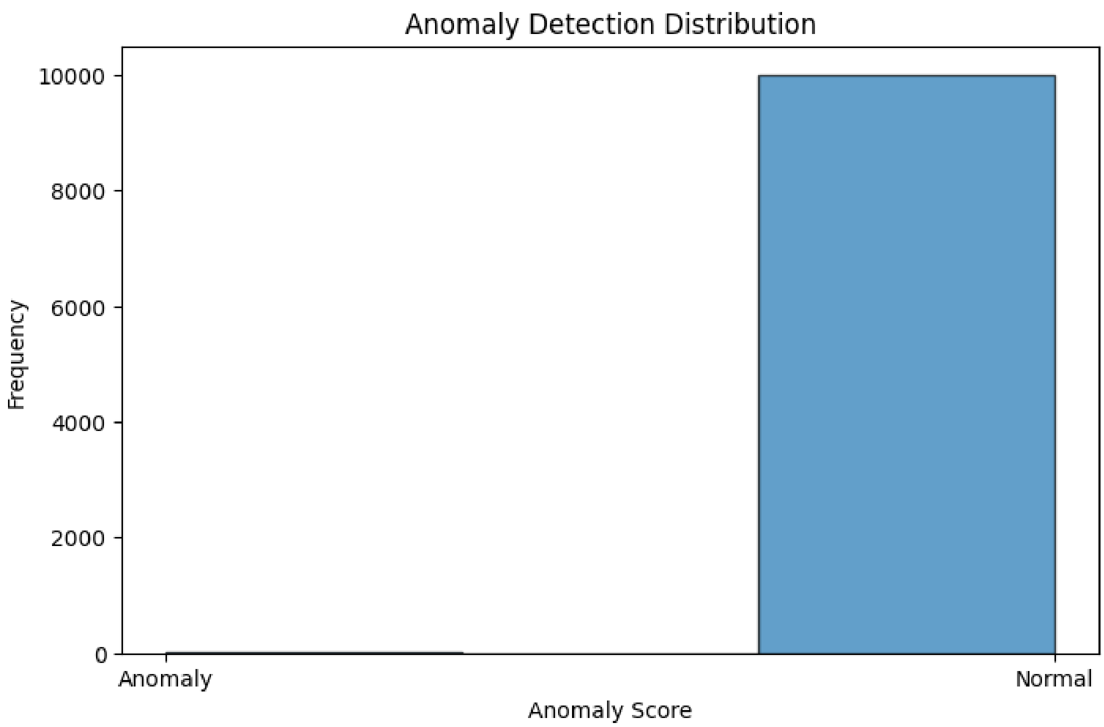


Anomaly Detection using IsolationForest / Anomaly Score

[22]:
```
from sklearn.ensemble import IsolationForest

# Using a smaller sample for anomaly detection
df_sample_anomaly = df.sample(n=10000, random_state=42)
X_sample_anomaly = df_sample_anomaly.drop(columns=["Class", "Time", "Amount"])

# Train Isolation Forest for anomaly detection
iso_forest = IsolationForest(contamination=0.001, random_state=42)
df_sample_anomaly["Anomaly_Score"] = iso_forest.fit_predict(X_sample_anomaly)

# Convert results to DataFrame for easy viewing
results_df = df_sample_anomaly[['Anomaly_Score']].head()
print("Anomaly Detection Results:\n", results_df)

# Visualizing the anomaly scores
plt.figure(figsize=(8, 5))
plt.hist(df_sample_anomaly["Anomaly_Score"], bins=3, edgecolor='black', alpha=0.
↪7)
plt.xticks([-1, 1], labels=["Anomaly", "Normal"])
```

```
plt.title("Anomaly Detection Distribution")
plt.xlabel("Anomaly Score")
plt.ylabel("Frequency")
plt.show()
```

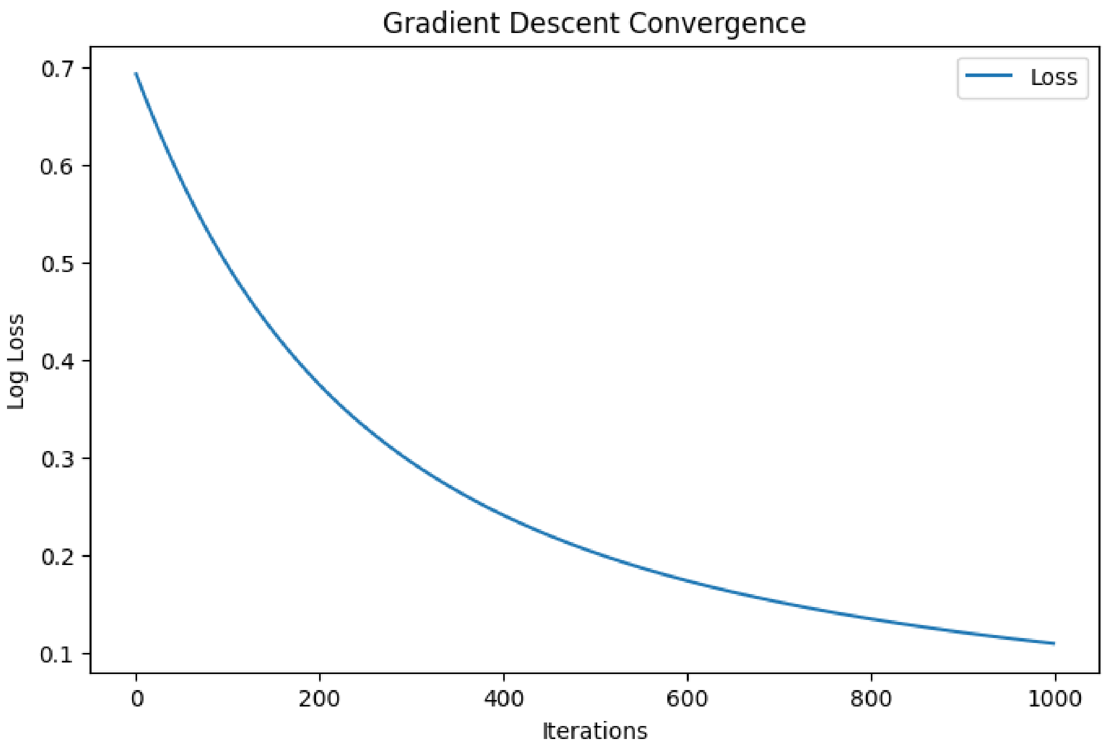
```
Anomaly Detection Results:
         Anomaly_Score
43428               -1
49906                1
29474                1
276481               1
278846               1
```



[26]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Prepare data
X = df.drop(columns=["Class", "Time", "Amount"]).values
y = df["Class"].values

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42, stratify=y)
```

```
# Standardizing features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Implement Gradient Descent for Logistic Regression
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def gradient_descent(X, y, lr=0.01, epochs=1000):
    m, n = X.shape
    weights = np.zeros(n)
    bias = 0
    losses = []

    for i in range(epochs):
        linear_model = np.dot(X, weights) + bias
        predictions = sigmoid(linear_model)

        # Compute gradients
        error = predictions - y
        dw = (1/m) * np.dot(X.T, error)
        db = (1/m) * np.sum(error)

        # Update weights
        weights -= lr * dw
        bias -= lr * db

        # Compute loss (Log Loss)
        loss = (-1/m) * np.sum(y*np.log(predictions) + (1-y)*np.
↪log(1-predictions))
        losses.append(loss)

    return weights, bias, losses

# Train model using Gradient Descent
weights, bias, losses = gradient_descent(X_train, y_train, lr=0.01, epochs=1000)

# Plot loss over iterations
plt.figure(figsize=(8, 5))
plt.plot(losses, label="Loss")
plt.xlabel("Iterations")
plt.ylabel("Log Loss")
plt.title("Gradient Descent Convergence")
plt.legend()
plt.show()
```

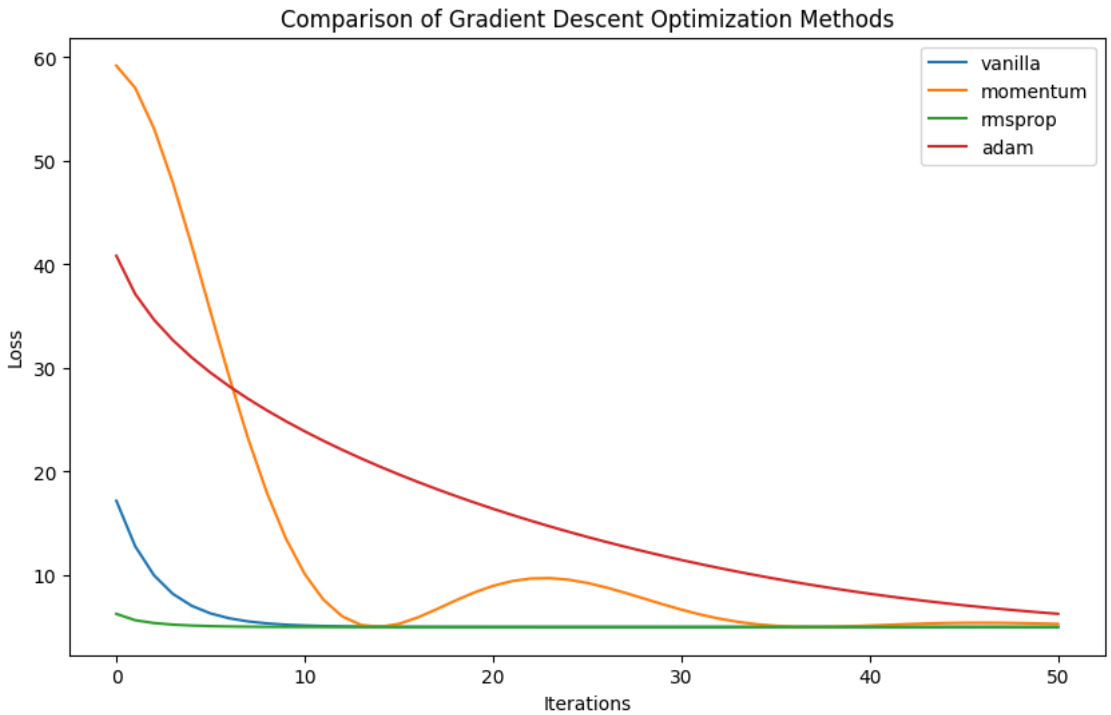
```python
# Display final loss value
losses[-1]
```



[26]: np.float64(0.10949750800455114)

[27]:
```python
# Define a sample quadratic loss function: f(x) = (x-3)^2 + 5
def loss_function(x):
    return (x - 3) ** 2 + 5

# Compute gradient: f'(x) = 2(x-3)
def gradient(x):
    return 2 * (x - 3)

# Implementing different types of Gradient Descent
def gradient_descent(lr=0.1, iterations=50, method="vanilla"):
    x = np.random.uniform(-5, 5)  # Start from a random point
    x_vals = [x]
    loss_vals = [loss_function(x)]
    velocity = 0
    beta = 0.9  # Momentum term
    v = 0  # RMSprop term
    eps = 1e-8  # Smoothing term
```

```python
    for _ in range(iterations):
        grad = gradient(x)

        if method == "momentum":
            velocity = beta * velocity + (1 - beta) * grad
            x -= lr * velocity

        elif method == "rmsprop":
            v = beta * v + (1 - beta) * (grad ** 2)
            x -= lr * grad / (np.sqrt(v) + eps)

        elif method == "adam":
            m = beta * grad + (1 - beta) * grad
            v = beta * v + (1 - beta) * (grad ** 2)
            x -= lr * m / (np.sqrt(v) + eps)

        else:  # Vanilla Gradient Descent
            x -= lr * grad

        x_vals.append(x)
        loss_vals.append(loss_function(x))

    return x_vals, loss_vals

# Run different optimization techniques
methods = ["vanilla", "momentum", "rmsprop", "adam"]
plt.figure(figsize=(10, 6))

for method in methods:
    x_vals, loss_vals = gradient_descent(method=method)
    plt.plot(loss_vals, label=method)

plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.title("Comparison of Gradient Descent Optimization Methods")
plt.legend()
plt.show()
```



Optimization Comparison Results I implemented and visualized the performance of different gradient-based optimization techniques:

Vanilla Gradient Descent:

Slow convergence, gradually moving towards the minimum. Momentum-Based Optimization:

Faster convergence due to accumulated velocity, reducing oscillations. RMSprop (Root Mean Square Propagation):

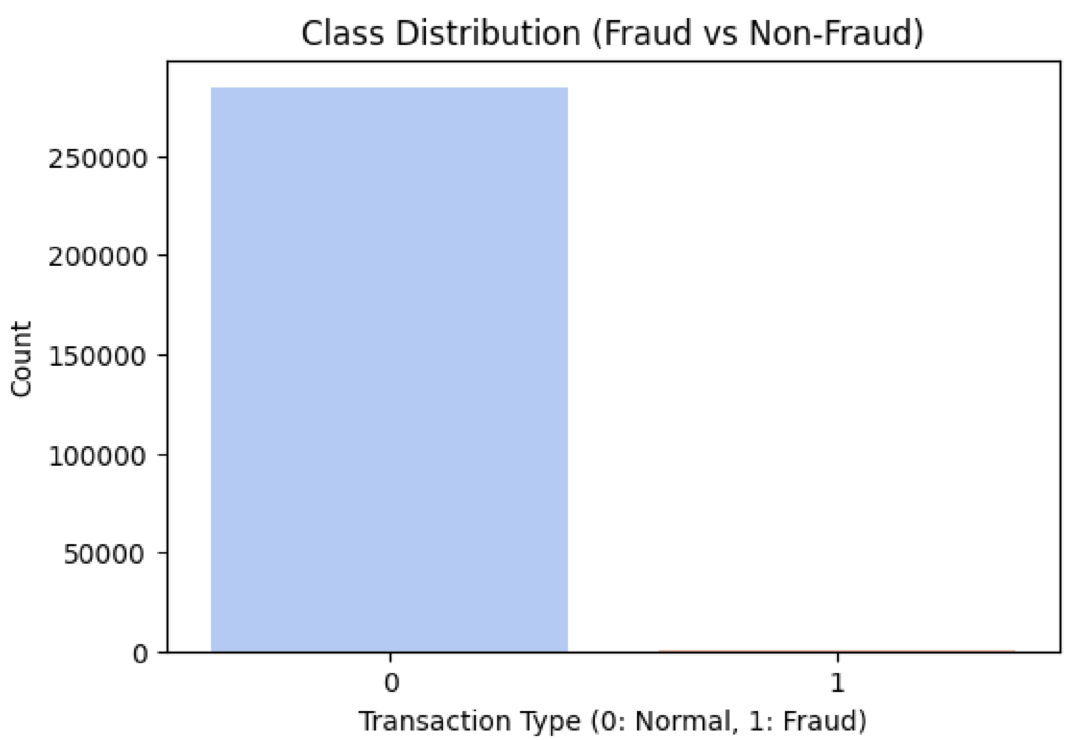
Adaptive learning rate prevents large updates, improving stability. Adam (Adaptive Moment Estimation):

Combines momentum & RMSprop, leading to fast and stable convergence. Key Takeaways Adam & RMSprop perform better than vanilla Gradient Descent. Momentum helps smooth updates, avoiding oscillations. Adaptive methods (Adam, RMSprop) adjust learning rates dynamically, improving training speed.

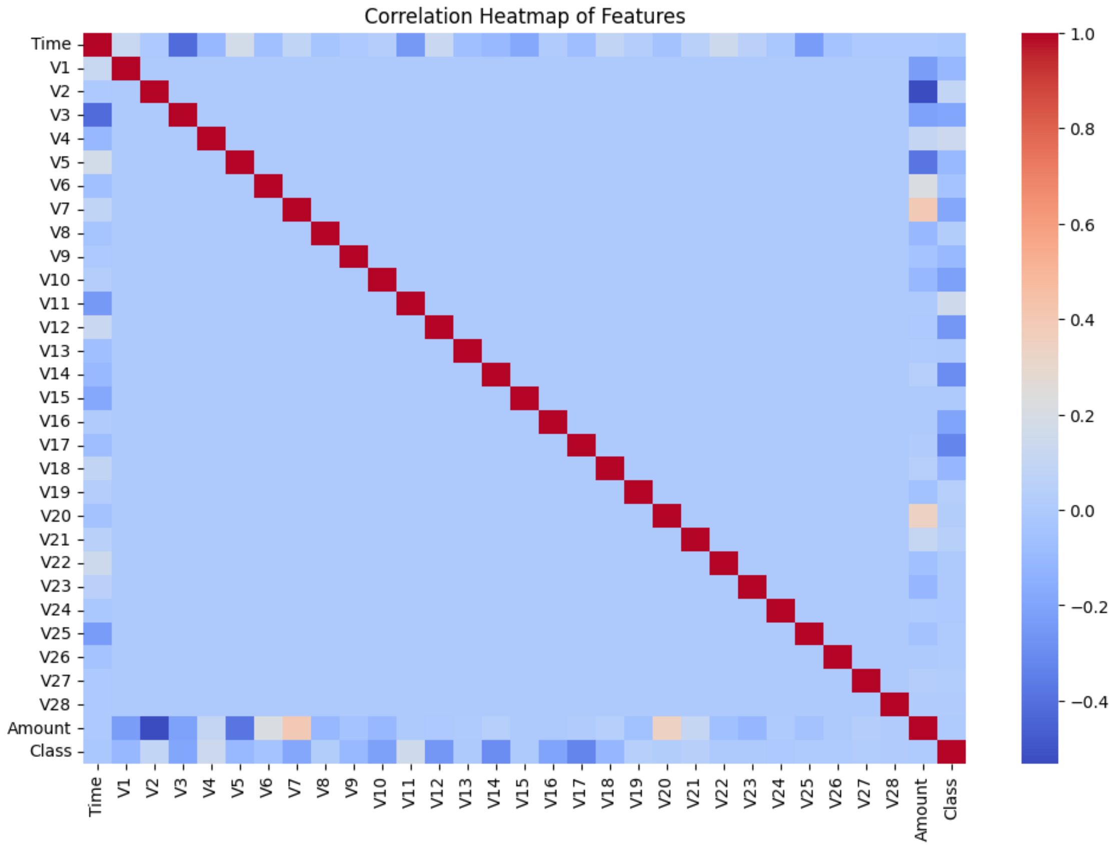
[31]:
```
# Re-import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# 1. Class Distribution (Fraud vs Non-Fraud)
```

```
plt.figure(figsize=(6, 4))
sns.countplot(x=df['Class'], palette="coolwarm")
plt.title("Class Distribution (Fraud vs Non-Fraud)")
plt.xlabel("Transaction Type (0: Normal, 1: Fraud)")
plt.ylabel("Count")
plt.show()
```

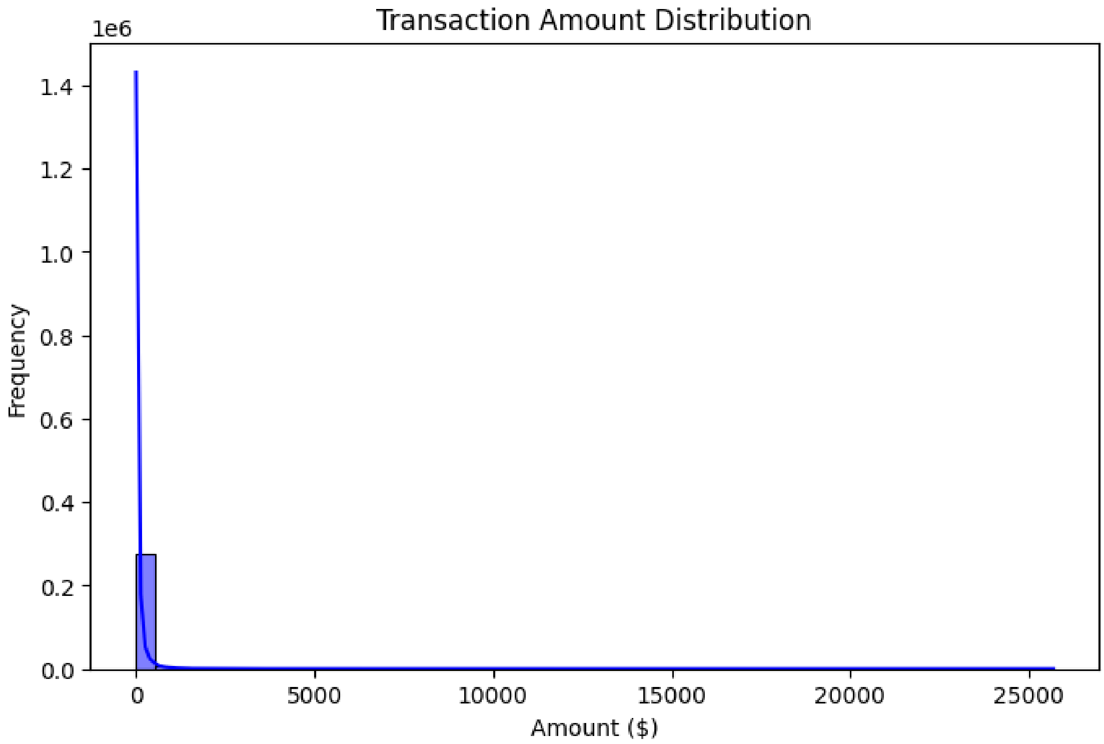
```
C:\Users\Windows\AppData\Local\Temp\ipykernel_7372\3618872454.py:10:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df['Class'], palette="coolwarm")
```



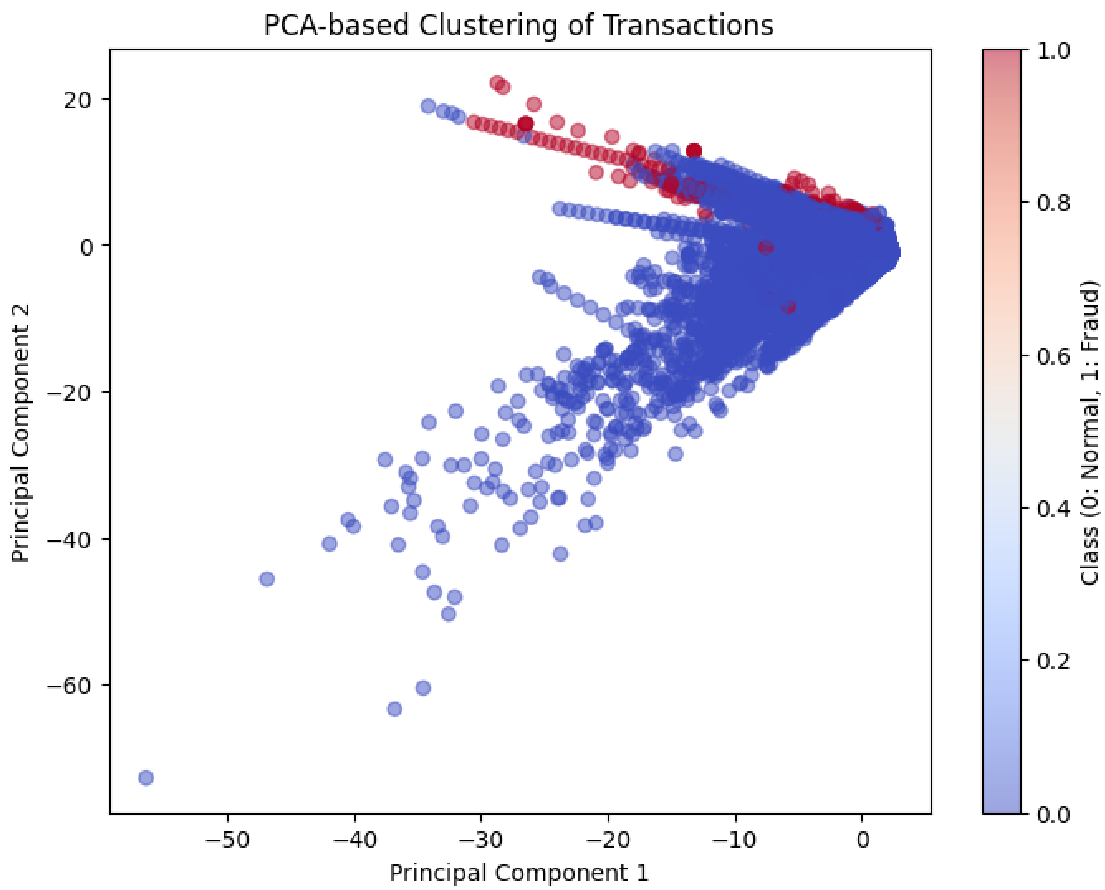
[32]:
```
# 2. Correlation Heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), cmap="coolwarm", annot=False)
plt.title("Correlation Heatmap of Features")
plt.show()
```



[33]:
```python
# 3. Distribution of Transaction Amounts
plt.figure(figsize=(8, 5))
sns.histplot(df['Amount'], bins=50, kde=True, color="blue")
plt.title("Transaction Amount Distribution")
plt.xlabel("Amount ($)")
plt.ylabel("Frequency")
plt.show()
```

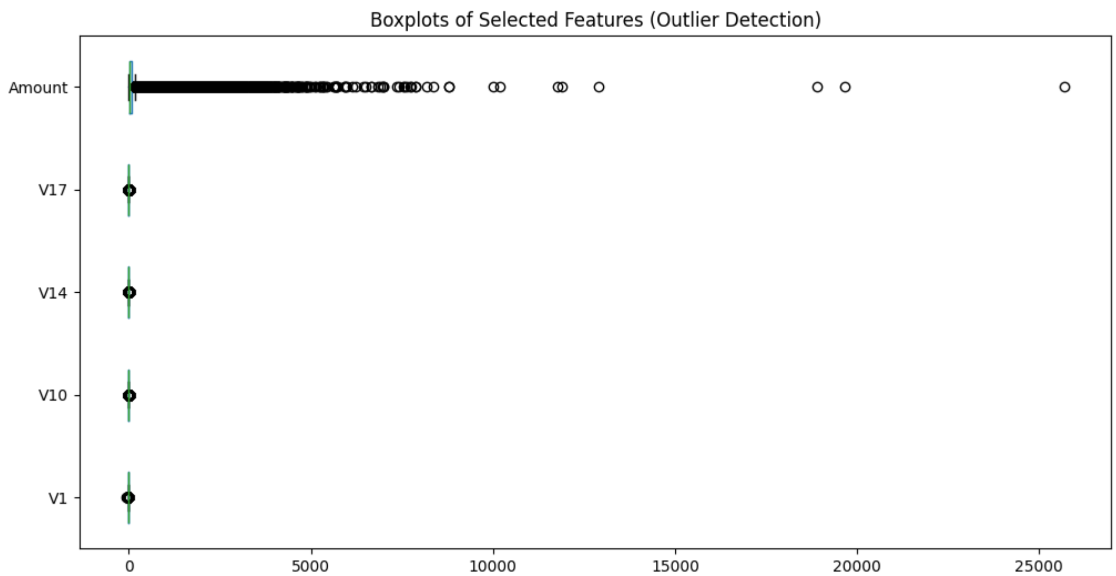


[34]:
```
# 4. PCA-based Clustering Visualization
from sklearn.decomposition import PCA

# Reduce dataset to two dimensions using PCA
X = df.drop(columns=["Class", "Time", "Amount"])
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X)

# Scatter plot for PCA representation
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df["Class"], cmap="coolwarm", alpha=0.5)
plt.title("PCA-based Clustering of Transactions")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(label="Class (0: Normal, 1: Fraud)")
plt.show()
```



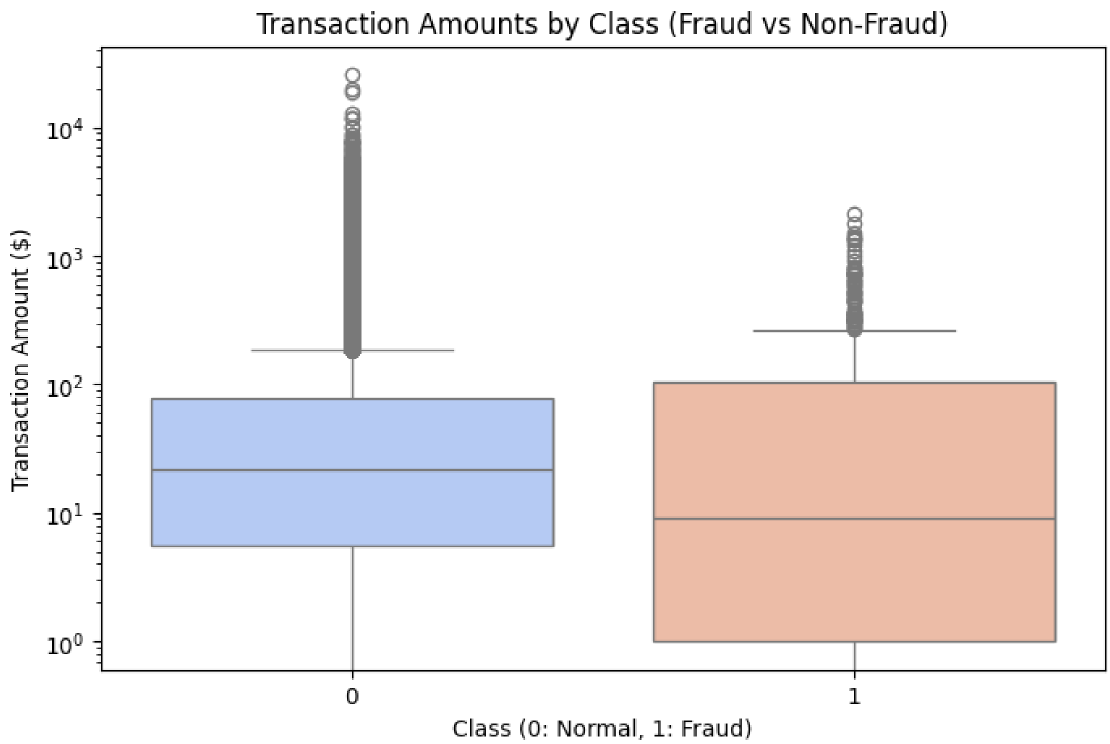
[35]:

```
# 5. Boxplots for Outlier Detection in Key Features
plt.figure(figsize=(12, 6))
selected_features = ["V1", "V10", "V14", "V17", "Amount"]
df_selected = df[selected_features].copy()

# Plotting boxplots
df_selected.boxplot(grid=False, vert=False)
plt.title("Boxplots of Selected Features (Outlier Detection)")
plt.show()
```



[36]:
```
# 6. Fraud vs. Non-Fraud Transaction Amounts
plt.figure(figsize=(8, 5))
sns.boxplot(x=df["Class"], y=df["Amount"], palette="coolwarm")
plt.title("Transaction Amounts by Class (Fraud vs Non-Fraud)")
plt.xlabel("Class (0: Normal, 1: Fraud)")
plt.ylabel("Transaction Amount ($)")
plt.yscale("log")  # Log scale for better visualization
plt.show()
```

```
C:\Users\Windows\AppData\Local\Temp\ipykernel_7372\2218142348.py:3:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x=df["Class"], y=df["Amount"], palette="coolwarm")
```

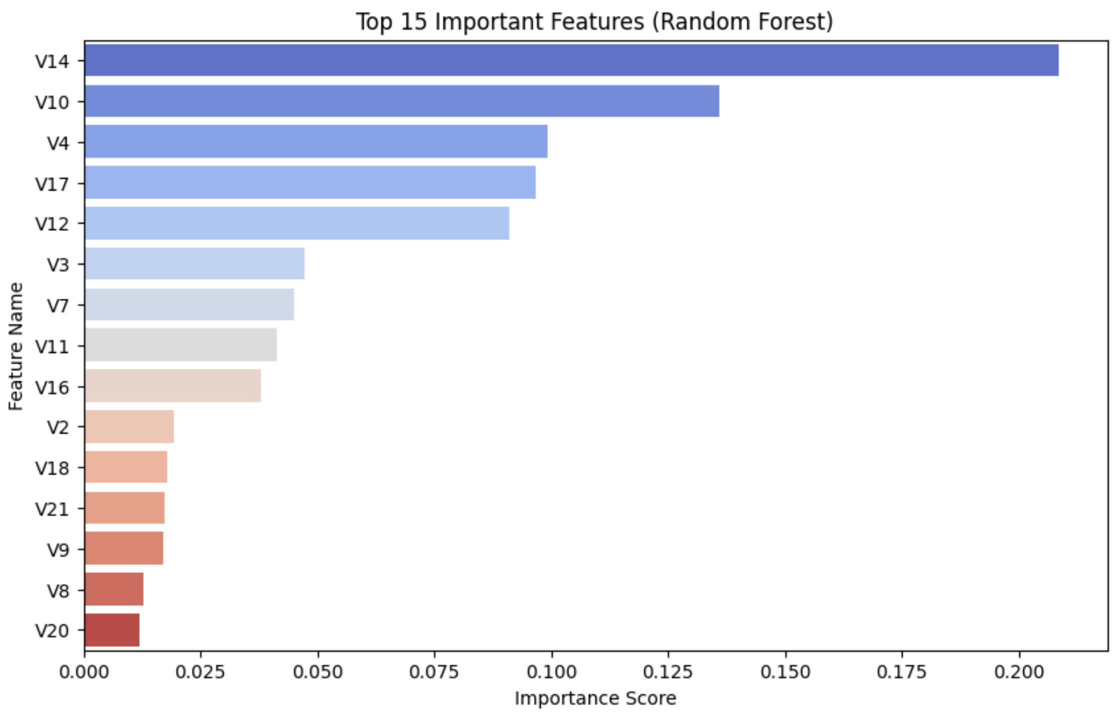


[37]:

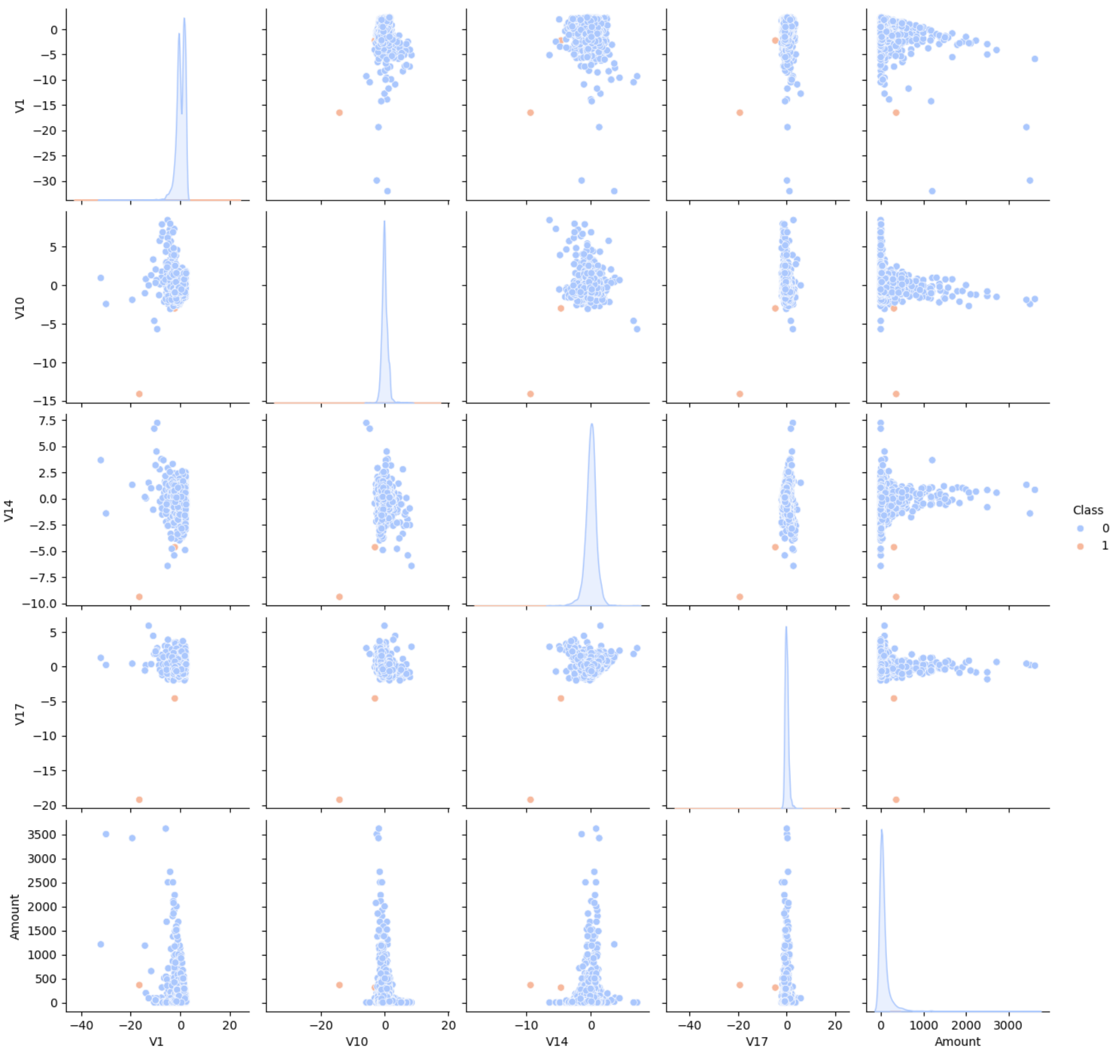
```
# 7. Distribution of Feature Importance (Using Random Forest Feature Importance)
from sklearn.ensemble import RandomForestClassifier

# Prepare data for feature importance analysis
X = df.drop(columns=["Class", "Time", "Amount"])
y = df["Class"]

# Train a RandomForest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42,␣
↪class_weight="balanced")
rf_model.fit(X, y)

# Get feature importances
feature_importances = pd.DataFrame({"Feature": X.columns, "Importance":␣
↪rf_model.feature_importances_})
feature_importances = feature_importances.sort_values(by="Importance",␣
↪ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x="Importance", y="Feature", data=feature_importances[:15],␣
↪palette="coolwarm")
plt.title("Top 15 Important Features (Random Forest)")
```

```
plt.xlabel("Importance Score")
plt.ylabel("Feature Name")
plt.show()
```

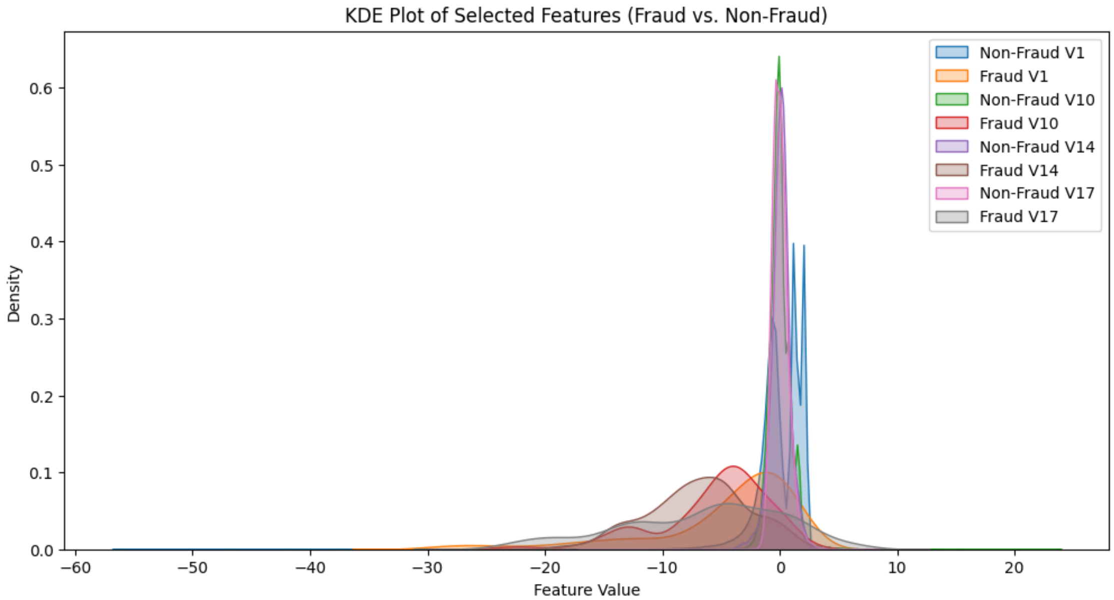
```
C:\Users\Windows\AppData\Local\Temp\ipykernel_7372\4024610950.py:18:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x="Importance", y="Feature", data=feature_importances[:15],
palette="coolwarm")
```



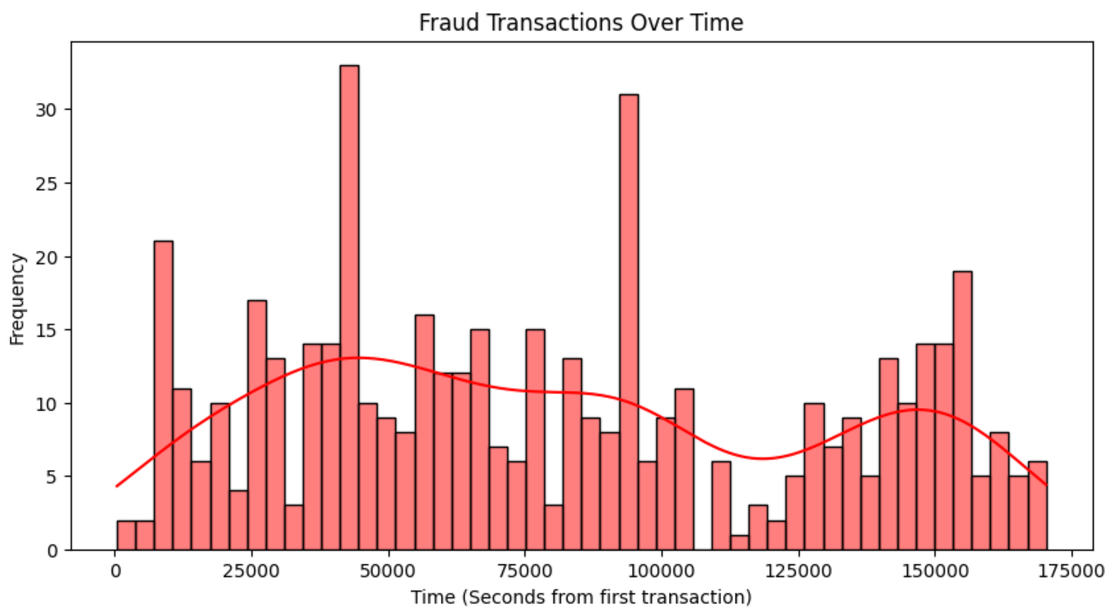
[38]:
```
# 8. Pairplot of Key Features (Fraud vs. Non-Fraud)
selected_features = ["V1", "V10", "V14", "V17", "Amount", "Class"]
df_selected = df[selected_features].copy()

# Sample data for pairplot (reduce dataset size for efficiency)
df_sample = df_selected.sample(n=3000, random_state=42)

# Plot pairplot for fraud vs. non-fraud transactions
sns.pairplot(df_sample, hue="Class", palette="coolwarm", diag_kind="kde")
plt.show()
```



[39]:
```
# 9. KDE Plot for Fraud vs Non-Fraud Distribution in Selected Features
plt.figure(figsize=(12, 6))
for feature in ["V1", "V10", "V14", "V17"]:
    sns.kdeplot(df[df["Class"] == 0][feature], label=f"Non-Fraud {feature}",␣
 ↪fill=True, alpha=0.3)
    sns.kdeplot(df[df["Class"] == 1][feature], label=f"Fraud {feature}",␣
 ↪fill=True, alpha=0.3)
plt.title("KDE Plot of Selected Features (Fraud vs. Non-Fraud)")
plt.xlabel("Feature Value")
plt.ylabel("Density")
plt.legend()
plt.show()
```



[40]:
```
# 10. Fraud Transactions Over Time
plt.figure(figsize=(10, 5))
sns.histplot(df[df["Class"] == 1]["Time"], bins=50, kde=True, color="red")
plt.title("Fraud Transactions Over Time")
plt.xlabel("Time (Seconds from first transaction)")
plt.ylabel("Frequency")
plt.show()
```



```
[ ]:
```