```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.metrics import classification_report, confusion_matrix,f1_score, make_scorer
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.decomposition import PCA
from sklearn.metrics import precision_recall_curve, auc
```

1. Data Loading and Overview

```python
path='creditcard.csv'
data=pd.read_csv(path)
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	-0.189115	0.133558	-0.021053	149.62	0
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	0.125895	-0.008983	0.014724	2.69	0
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	-0.139097	-0.055353	-0.059752	378.66	0
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	-0.221929	0.062723	0.061458	123.50	0
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	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

```python
print("=== Data Shape ===")
print(data.shape)

print("\n=== Column Names ===")
print(data.columns)
```

```
=== Data Shape ===
(284807, 31)

=== Column Names ===
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

```python
print("\n=== First 5 Rows ===")
print(data.head())
```

```
=== 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]
```

```python
print("\n=== Data Description ===")
print(data.describe())
```

```
=== Data Description ===
                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]
```

```python
print("\n=== Missing Values ===")
print(data.isnull().sum())
```

```
=== Missing Values ===
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
```

2. Exploratory Data Analysis

```python
# 2.1 Class Distribution
fraud_count = data['Class'].value_counts()
print("\n=== Class Distribution ===")
print(fraud_count)
```

```
=== Class Distribution ===
Class
0    284315
1       492
Name: count, dtype: int64
```

```python
# Fraud vs Non-Fraud in percentages
fraud_percentage = (fraud_count[1] / fraud_count.sum()) * 100
print(f"Percentage of fraudulent transactions: {fraud_percentage:.4f}%")
```

```
Percentage of fraudulent transactions: 0.1727%
```

```python
# 2.2 Basic statistics on 'Amount'

print("\n=== Amount Statistics ===")
print(data['Amount'].describe())
```

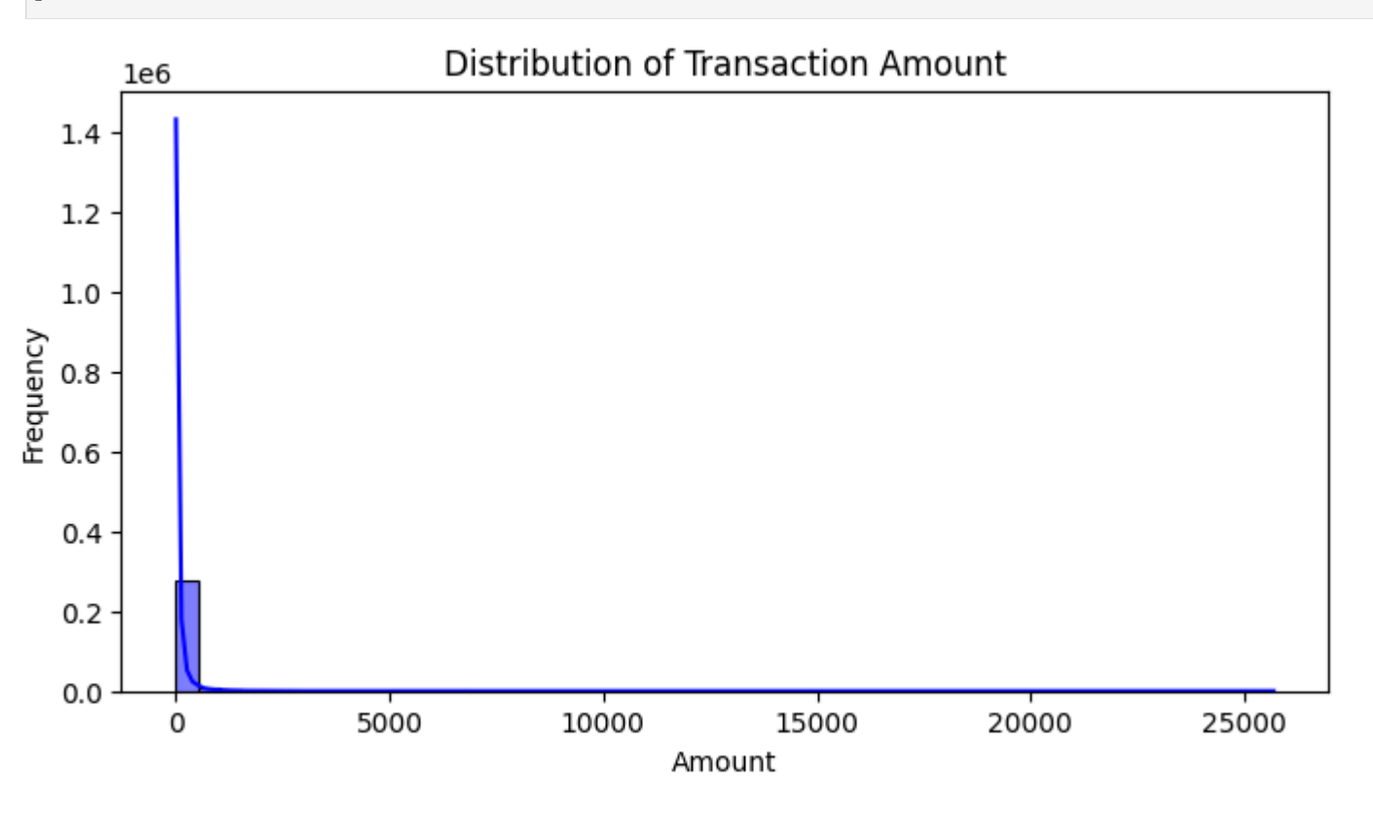
```
=== Amount Statistics ===
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64
```

```python
# 2.3 Time Feature

print("\n=== Time Statistics ===")
print(data['Time'].describe())
```

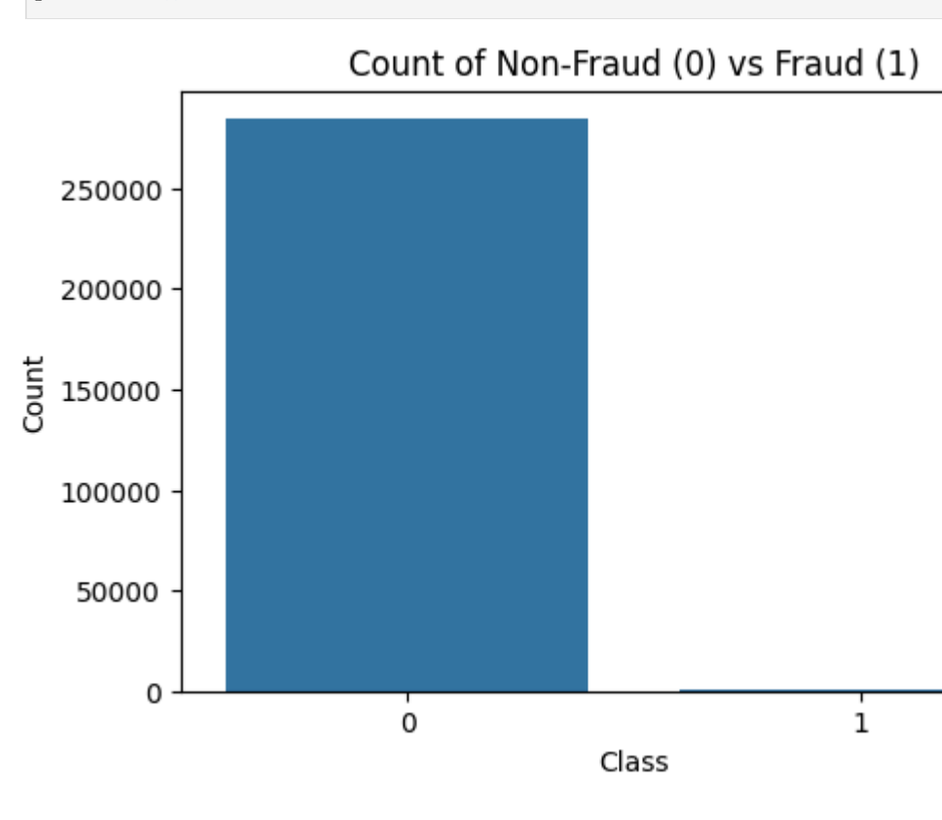
```
=== Time Statistics ===
count    284807.000000
mean      94813.859575
std       47488.145955
min           0.000000
25%       54201.500000
50%       84692.000000
75%      139320.500000
max      172792.000000
Name: Time, dtype: float64
```

3. Data Visualization

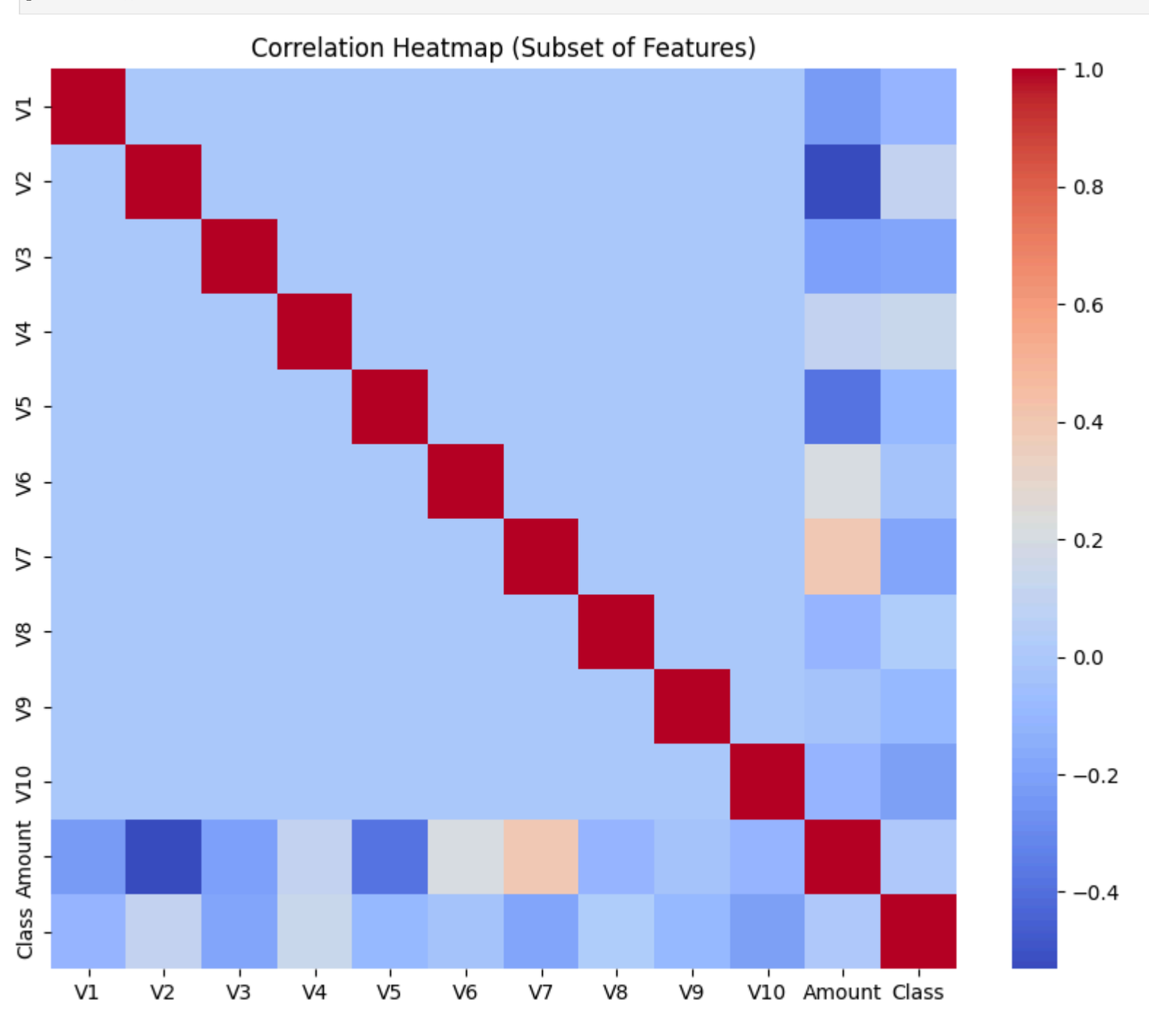
```python
# 3.1 Distribution of the 'Amount' Feature
plt.figure(figsize=(8,4))
sns.histplot(data['Amount'], bins=50, kde=True, color='blue')
plt.title("Distribution of Transaction Amount")
plt.xlabel("Amount")
plt.ylabel("Frequency")
plt.show()
```




```python
# 3.2 Countplot of Fraud vs Non-Fraud
plt.figure(figsize=(6,4))
sns.countplot(x='Class', data=data)
plt.title("Count of Non-Fraud (0) vs Fraud (1)")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```




```python
# 3.3 Correlation Heatmap

subset_cols = ['V'+str(i) for i in range(1, 11)] + ['Amount', 'Class']
corr_matrix = data[subset_cols].corr()

plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, cmap='coolwarm', annot=False)
plt.title("Correlation Heatmap (Subset of Features)")
plt.show()
```



4. Data Preprocessing

```python
# - scale 'Amount'
data['Scaled_Amount'] = StandardScaler().fit_transform(data[['Amount']])
```

```python
# Prepare feature matrix X (all V1-V28 + scaled Amount) and label y (Class)
features = [f'V{i}' for i in range(1, 29)] + ['Scaled_Amount']
X = data[features].values
y = data['Class'].values  # 0: Non-Fraud, 1: Fraud
```

```python
# Train-Test Split
X_normal_train, X_normal_test = train_test_split(X_normal, test_size=0.3, random_state=42)
# We'll combine the normal test set with all fraud data for final evaluation
X_test_combined = np.vstack([X_normal_test, X_fraud])
y_test_combined = np.hstack([np.zeros(len(X_normal_test)), np.ones(len(X_fraud))])
```

5. Model Building & Evaluation

```python
iso_forest = IsolationForest(
    n_estimators=100,
    contamination=fraud_ratio,
    random_state=42
)
iso_forest.fit(X_train)
```

```
IsolationForest
IsolationForest(contamination=0.0017254870488152324, random_state=42)
```

6. Model Evaluation

```python
# IsolationForest outputs -1 for anomalies and 1 for normal points.
y_pred = iso_forest.predict(X_test)
# Convert to 0 (normal) and 1 (anomaly/fraud)
y_pred = np.where(y_pred == -1, 1, 0)

print("\n=== Confusion Matrix ===")
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
=== Confusion Matrix ===
[[85199    96]
 [  106    42]]
```

```python
print("\n=============== Classification Report =================")
print(classification_report(y_test, y_pred))
```

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

           0       1.00      1.00      1.00     85295
           1       0.30      0.28      0.29       148

    accuracy                           1.00     85443
   macro avg       0.65      0.64      0.65     85443
weighted avg       1.00      1.00      1.00     85443
```

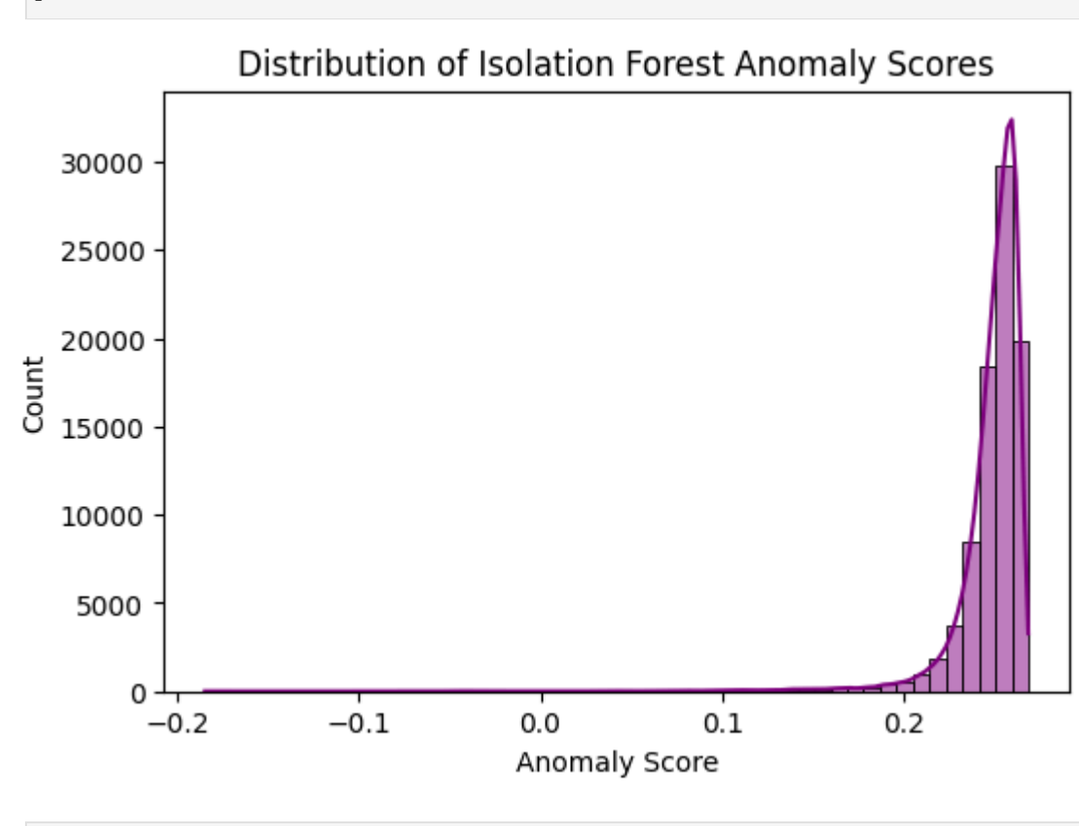
```python
# More detailed error analysis
false_positives = np.where((y_test == 0) & (y_pred == 1))[0]
false_negatives = np.where((y_test == 1) & (y_pred == 0))[0]
print(f"False Positives: {len(false_positives)}")
print(f"False Negatives: {len(false_negatives)}")
```

```
False Positives: 96
False Negatives: 106
```

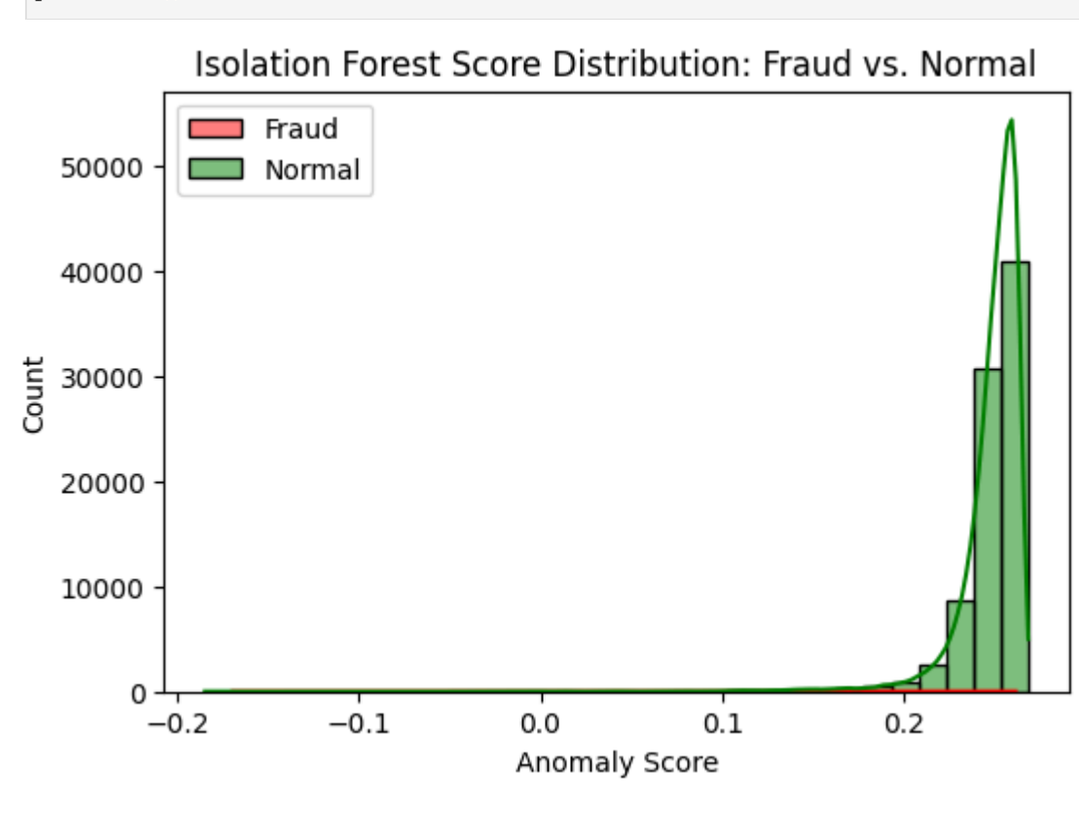
Visualization Techniques

1. Anomaly Score Distribution

```python
# For each sample in the test set, get the anomaly score
scores = best_iforest.decision_function(X_test)

plt.figure(figsize=(6,4))
sns.histplot(scores, bins=50, kde=True, color='purple')
plt.title("Distribution of Isolation Forest Anomaly Scores")
plt.xlabel("Anomaly Score")
plt.ylabel("Count")
plt.show()
```

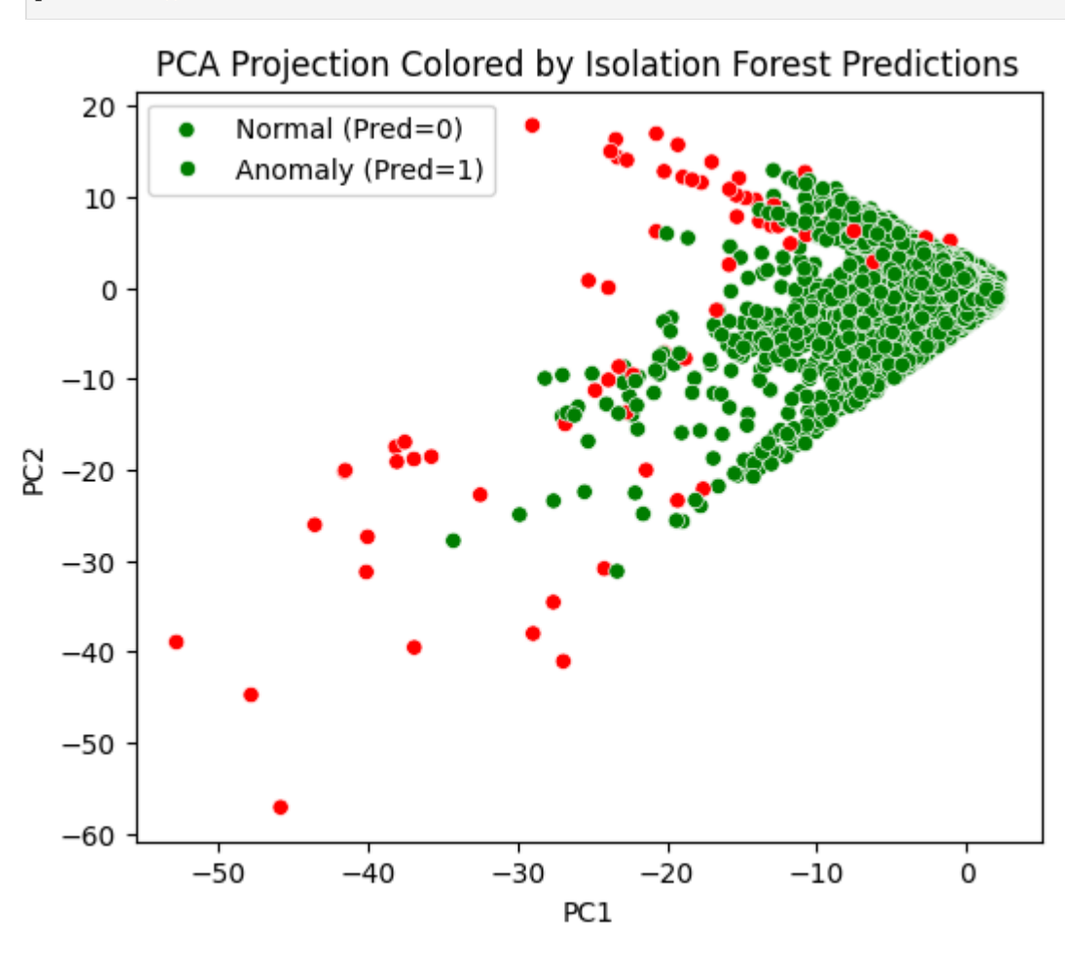



```python
fraud_scores = scores[y_test == 1]
normal_scores = scores[y_test == 0]

plt.figure(figsize=(6,4))
sns.histplot(fraud_scores, color='red', label='Fraud', kde=True, bins=30)
sns.histplot(normal_scores, color='green', label='Normal', kde=True, bins=30)
plt.title("Isolation Forest Score Distribution: Fraud vs. Normal")
plt.xlabel("Anomaly Score")
plt.legend()
plt.show()
```



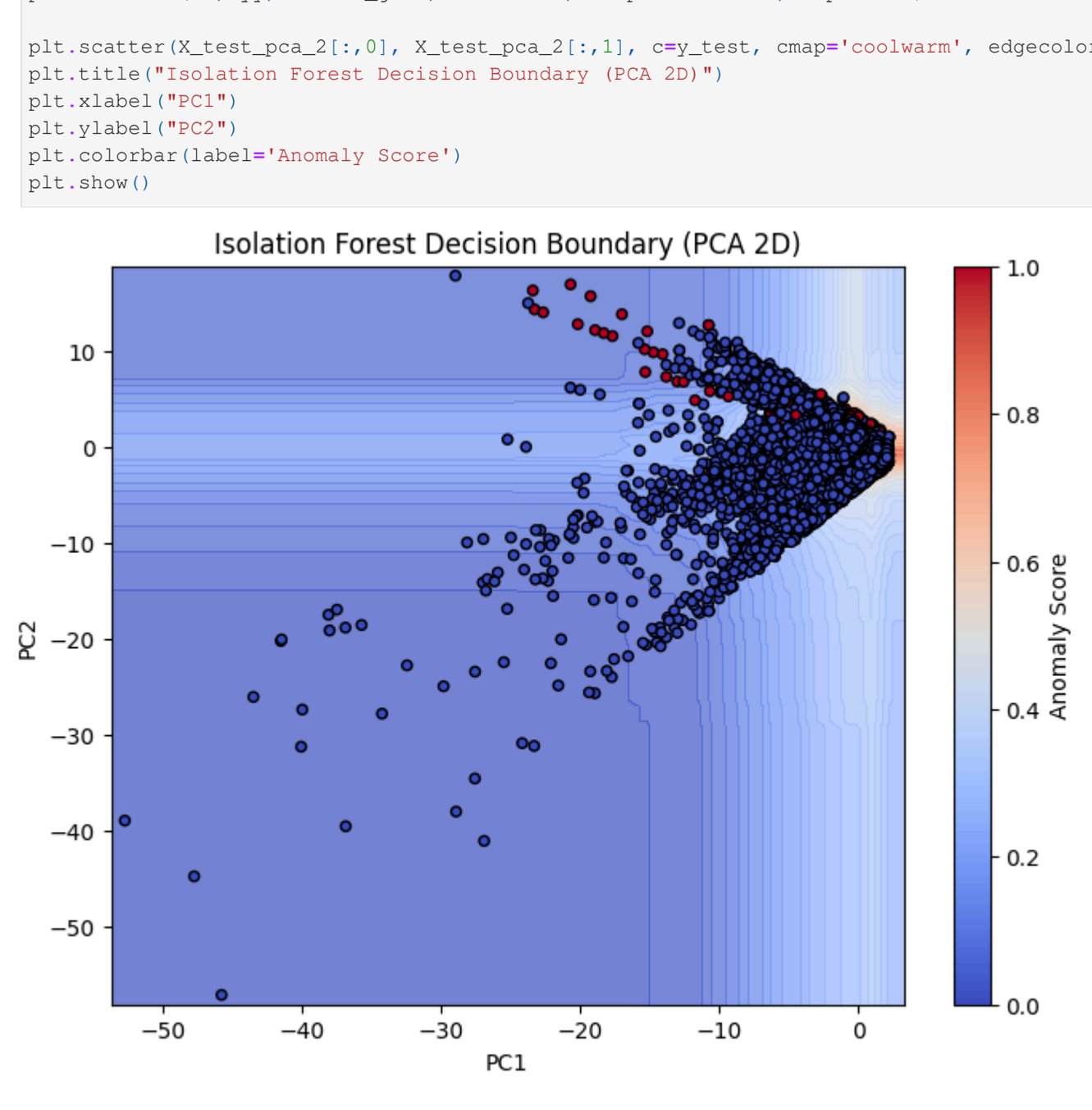
2. 2D Projection (PCA or t-SNE)

```python
y_pred_iforest = best_iforest.predict(X_test)
# Convert IsolationForest output: -1 => anomaly (1), 1 => normal (0)
y_pred_iforest = np.where(y_pred_iforest == -1, 1, 0)

# PCA Transformation
pca = PCA(n_components=2, random_state=42)

# Fit PCA on the training data to avoid data leakage
X_train_pca = pca.fit_transform(X_train)
# Transform the test set using the same PCA
X_test_pca = pca.transform(X_test)

# Scatter Plot
plt.figure(figsize=(6,5))

sns.scatterplot(
    x=X_test_pca[:, 0],
    y=X_test_pca[:, 1],
    hue=y_pred_iforest,
    palette=['green', 'red']
)
plt.title("PCA Projection Colored by Isolation Forest Predictions")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(labels=["Normal (Pred=0)", "Anomaly (Pred=1)"])
plt.show()
```



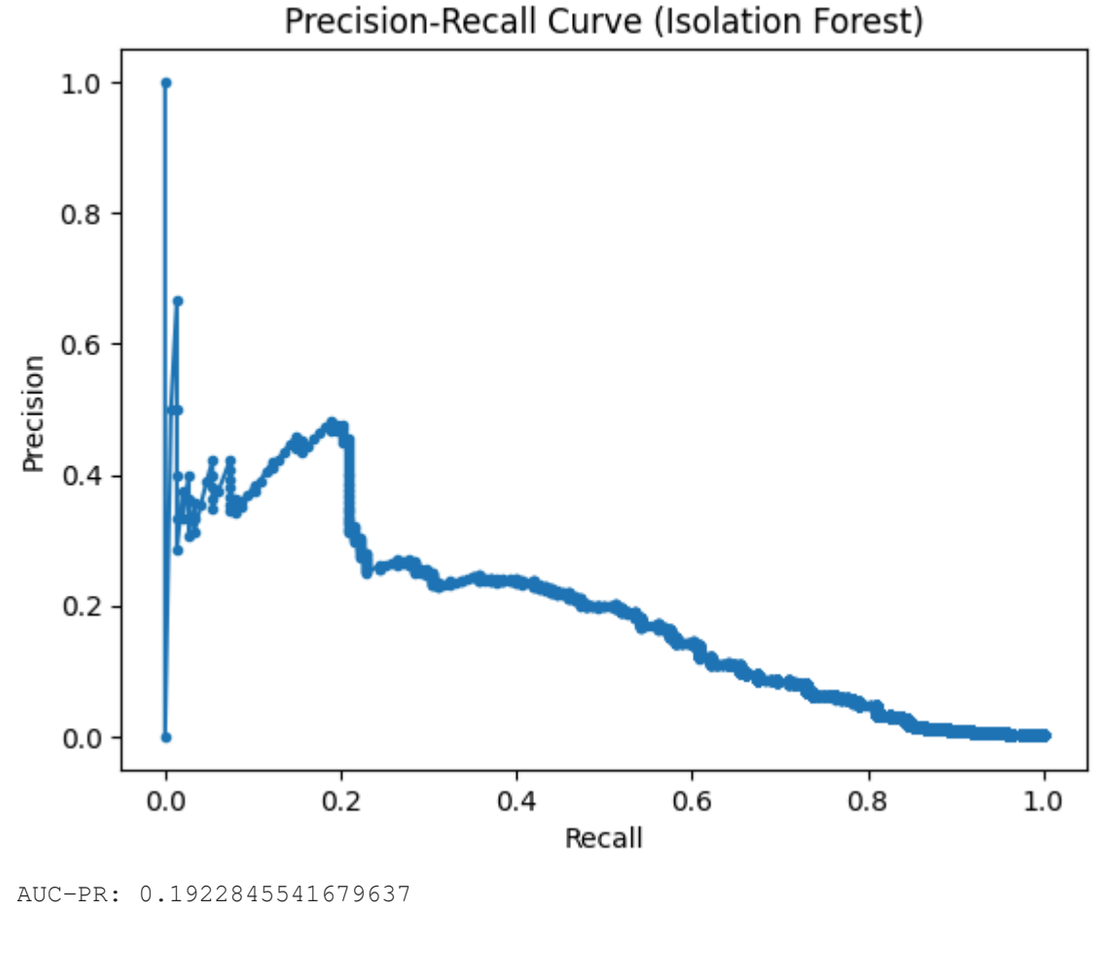
3. Decision Boundary

```python
# Suppose you do PCA to reduce to 2D
pca_2 = PCA(n_components=2, random_state=42)
X_train_pca_2 = pca_2.fit_transform(X_train)
X_test_pca_2 = pca_2.transform(X_test)

iso_2d = IsolationForest(contamination=0.01, random_state=42)
iso_2d.fit(X_train_pca_2)

# Create a meshgrid
xx, yy = np.meshgrid(
    np.linspace(X_test_pca_2[:,0].min()-1, X_test_pca_2[:,0].max()+1, 100),
    np.linspace(X_test_pca_2[:,1].min()-1, X_test_pca_2[:,1].max()+1, 100)
)

grid = np.c_[xx.ravel(), yy.ravel()]
scores_grid = iso_2d.decision_function(grid)
scores_grid = scores_grid.reshape(xx.shape)

plt.figure(figsize=(8,6))
# Contour plot for the anomaly scores
plt.contourf(xx, yy, scores_grid, levels=50, cmap='coolwarm', alpha=0.7)

plt.scatter(X_test_pca_2[:,0], X_test_pca_2[:,1], c=y_test, cmap='coolwarm', edgecolor='k', s=20)
plt.title("Isolation Forest Decision Boundary (PCA 2D)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(label='Anomaly Score')
plt.show()
```



4. Metrics Over Different Thresholds

```python
# decision_function gives anomaly scores (higher = more normal, lower = more anomalous)
scores_test = best_iforest.decision_function(X_test)

# We want to invert them so that higher = more anomalous
anomaly_scores = -scores_test

precision, recall, thresholds = precision_recall_curve(y_test, anomaly_scores)

plt.figure()
plt.plot(recall, precision, marker='.')
plt.title('Precision-Recall Curve (Isolation Forest)')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()

print("AUC-PR:", auc(recall, precision))
```




```
AUC-PR: 0.1922845541679637
```

7. Summary

```python
print("""
    Data: We analyzed a credit card transactions dataset, which was heavily imbalanced with very few fraud cases.

    EDA: Exploratory analysis revealed key distribution patterns, highlighting the importance of scaling the ‘Amount’ feature.

    Preprocessing: We applied normalization and selected relevant features (V1-V28 plus scaled Amount) for modeling.

    Unsupervised Approach: An Isolation Forest model was trained to isolate potential anomalies without relying on labels.

    Evaluation: We used a confusion matrix and classification report to measure how effectively anomalies were detected.

    Visualization: PCA projections and score distributions helped us interpret how the model separates fraud from normal transactions.

    Autoencoder Method: A deep learning approach was introduced to reconstruct normal data and flag high-error instances as fraud.

    Supervised Insight: We acknowledged that supervised models (e.g., Random Forest) can outperform unsupervised methods when labels are available.

    Challenges: Handling severe class imbalance and tuning model thresholds are critical to reducing false positives and false negatives.

    Conclusion: With careful preprocessing, model selection, and threshold optimization, anomaly detection can significantly enhance fraud prevention efforts.

""")
```


```
    Data: We analyzed a credit card transactions dataset, which was heavily imbalanced with very few fraud cases.

    EDA: Exploratory analysis revealed key distribution patterns, highlighting the importance of scaling the ‘Amount’ feature.

    Preprocessing: We applied normalization and selected relevant features (V1-V28 plus scaled Amount) for modeling.

    Unsupervised Approach: An Isolation Forest model was trained to isolate potential anomalies without relying on labels.

    Evaluation: We used a confusion matrix and classification report to measure how effectively anomalies were detected.

    Visualization: PCA projections and score distributions helped us interpret how the model separates fraud from normal transactions.

    Autoencoder Method: A deep learning approach was introduced to reconstruct normal data and flag high-error instances as fraud.

    Supervised Insight: We acknowledged that supervised models (e.g., Random Forest) can outperform unsupervised methods when labels are available.

    Challenges: Handling severe class imbalance and tuning model thresholds are critical to reducing false positives and false negatives.

    Conclusion: With careful preprocessing, model selection, and threshold optimization, anomaly detection can significantly enhance fraud prevention efforts.
```