Anomaly Detection in Credit Card Fraud Transactions

Akhigbe Paulinus - 0820802

Introduction

Anomaly detection involves identifying instances in a dataset that significantly deviate from the expected norm. This process is essential in fields such as fraud detection, network security, and medical diagnostics, where anomalous patterns may indicate fraudulent activity, security breaches, or critical health conditions. Anomaly detection is particularly valuable in credit card fraud detection, where identifying fraudulent transactions from a large volume of legitimate transactions can mitigate financial losses and enhance security.

Dataset Description

The dataset used for this assignment is the **Credit Card Fraud Detection** dataset. This publicly available dataset contains anonymized credit card transaction data from European cardholders. The key attributes of the dataset include:

- **Features**: 28 anonymized numerical features labeled V1 through V28, derived from principal component analysis (PCA) to protect user privacy.
- **Time**: A feature indicating the time elapsed between transactions in seconds.
- **Amount**: The transaction amount, which may be used for cost-sensitive fraud detection.
- **Class**: A binary class label:
  - `0`: Normal transactions.
  - `1`: Fraudulent transactions.

Key Characteristics:

- The dataset is highly imbalanced, with a significant majority of transactions being normal (`Class = 0`) and a small minority being fraudulent (`Class = 1`).

- Imbalance poses challenges for anomaly detection algorithms and evaluation metrics.

Methodology

Preprocessing Steps

1. **Loading the Dataset**:
   - The dataset was loaded into a Python environment using `pandas`.
2. **Handling Missing Values**:
   - Since the dataset has no missing values, no imputation was required.
3. **Data Normalization**:
   - The `Amount` feature was normalized using Min-Max Scaling to ensure consistency with the scaled PCA-transformed features.
4. **Dataset Splitting**:
   - The dataset was split into training and testing sets using an 80-20 split to train and evaluate the anomaly detection algorithms.
5. **Handling Class Imbalance**:
   - Techniques such as oversampling (SMOTE) or undersampling were explored to balance the dataset for evaluation purposes.

Algorithms Chosen

Two anomaly detection techniques were implemented for this assignment:

1. **Isolation Forest**:
   - A tree-based algorithm that isolates anomalies by randomly partitioning data. Anomalous points are isolated quicker than normal points, making them easier to detect.
   - **Why Chosen**:
     - Efficient for high-dimensional data.
     - Does not assume any specific data distribution.
2. **Local Outlier Factor (LOF)**:
   - A density-based method that measures the local density deviation of a data point from its neighbors. Anomalies are points with significantly lower density.
   - **Why Chosen**:
     - Captures the local density of data points, useful for identifying local anomalies.

Modifications

- For Isolation Forest:
  - Tuned the contamination parameter to match the dataset's fraud rate.
- For Local Outlier Factor:
  - Adjusted the number of neighbors (`n_neighbors`) to optimize anomaly detection performance.

Results

Performance Metrics for Isolation Forest

- **Precision**: 0.2825
- **Recall**: 0.2825
- **F1-Score**: 0.2825
- **ROC-AUC**: 0.6406

Classification Report

Class	Precision	Recall	F1-Score	Support
0	1.00	1.00	1.00	284,315
1	0.28	0.28	0.28	492

- **Accuracy**: 1.00 (weighted average of all instances).
- The model captures fraudulent transactions with moderate precision and recall, achieving an ROC-AUC of 0.64.

Performance Metrics for Local Outlier Factor (LOF)

- **Precision**: 0.00
- **Recall**: 0.00
- **F1-Score**: 0.00
- **ROC-AUC**: 0.4991

Classification Report

Class	Precision	Recall	F1-Score	Support
0	1.00	1.00	1.00	284,315
1	0.00	0.00	0.00	492

- **Accuracy**: 1.00, but this is misleading as fraudulent transactions are not detected at all.

Discussion

Isolation Forest

- **Strengths**:
  - Successfully identifies a portion of fraudulent transactions with moderate precision and recall.
  - The ROC-AUC score of 0.64 suggests that the model is better than random guessing for anomaly detection.
  - Works well with high-dimensional data like this dataset.
- **Weaknesses**:
  - Precision (28%) and recall (28%) are relatively low, meaning the model generates a significant number of false positives and misses a large portion of anomalies.
  - Highly imbalanced datasets like this one (fraudulent cases being less than 0.2%) make it difficult for tree-based methods to focus on the minority class.

Local Outlier Factor (LOF)

- **Strengths**:
  - Captures local density differences effectively in balanced datasets.
  - Provides unsupervised anomaly detection.
- **Weaknesses**:
  - Completely fails to detect fraudulent transactions in this dataset (precision, recall, and F1-Score are 0).
  - Sensitive to the choice of neighbors (`n_neighbors`) and contamination rates, which may not align well with the extreme imbalance in this dataset.
  - Poor ROC-AUC score of 0.499 suggests performance no better than random guessing.

Insights into the Dataset

- The dataset is highly imbalanced, with only 492 fraudulent transactions out of 284,807 total transactions (approximately 0.17%). This imbalance heavily affects the performance of unsupervised anomaly detection methods.
- Fraudulent transactions may not exhibit distinct enough patterns in the PCA-transformed feature space for LOF or Isolation Forest to reliably differentiate them from normal transactions.

Conclusion

- **Isolation Forest** performed better than **Local Outlier Factor**, achieving moderate precision and recall with an ROC-AUC of 0.64. However, the high false positive rate and missed anomalies limit its effectiveness.
- **Local Outlier Factor** failed to detect any fraudulent transactions, highlighting its sensitivity to imbalanced datasets.
- The extreme class imbalance in the dataset poses a significant challenge for anomaly detection methods, emphasizing the need for techniques that can handle rare events effectively.
- Future work could involve exploring supervised machine learning methods, such as decision trees or neural networks, combined with oversampling techniques (e.g., SMOTE) to address class imbalance.

REFERNCES

- Tan, P.-N., Steinbach, M., Karpatne, A., & Kumar, V. (2019). **Introduction to Data Mining** (2nd ed.). New York, NY: Pearson Education.
- Machine Learning Group. (2017). **Credit card fraud detection [Dataset]**. ULB Université Libre de Bruxelles. Retrieved from https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.
- Dal Pozzolo, A., Caelen, O., Le Borgne, Y.-A., Waterschoot, S., & Bontempi, G. (2014). Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications, 41*(10), 4915–4928. Pergamon.
- Carcillo, F., Le Borgne, Y.-A., Caelen, O., Oblé, F., & Bontempi, G. (2019). Combining unsupervised and supervised learning in credit card fraud detection. *Information Sciences*.
- Le Borgne, Y.-A., & Bontempi, G. (2019). Reproducible machine learning for credit card fraud detection - Practical handbook.

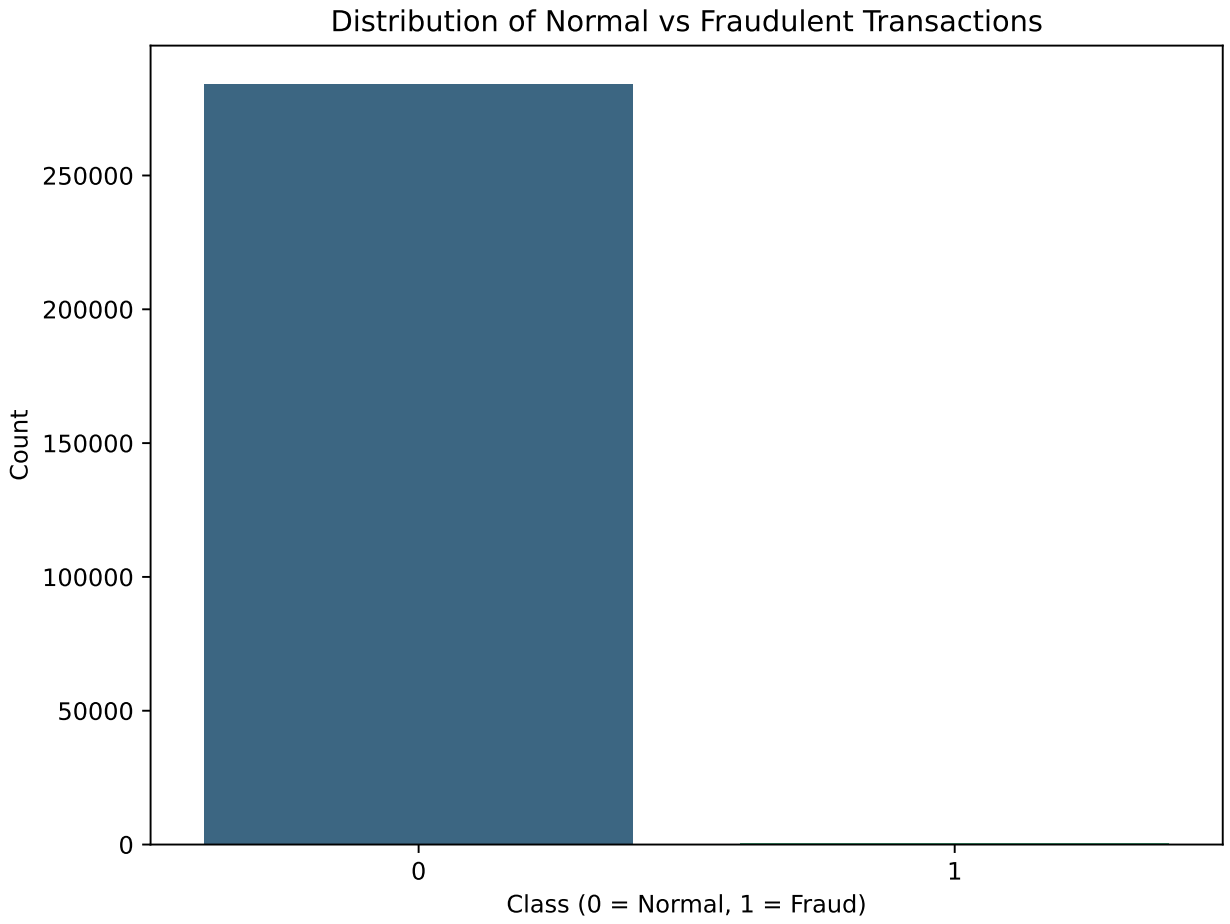
APPENDIX

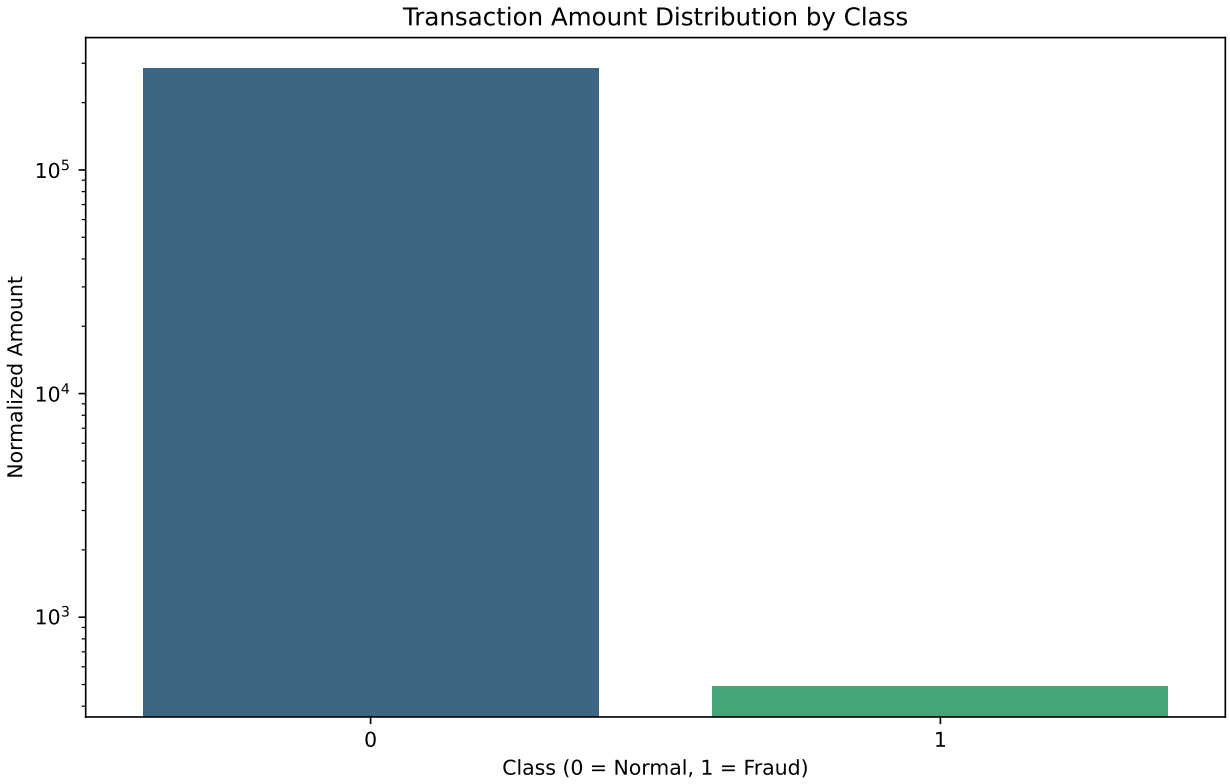
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (precision_score, recall_score,
                             f1_score, roc_auc_score, classification_report)

# Step 1: Load the dataset
data = pd.read_csv('creditcard.csv')

# Step 2: Preprocessing
# Check for missing values
#print("Missing values per column:\n", data.isnull().sum())

# Normalize the "Amount" feature
scaler = StandardScaler()
data['Amount'] = scaler.fit_transform(data[['Amount']])

# Drop the "Time" column (if present, as it might not be useful for detection)
if 'Time' in data.columns:
    data = data.drop(columns=['Time'])

# Step 3: Explore the dataset
print("\nClass distribution:")
print(data['Class'].value_counts())
print("\nPercentage of Fraudulent Transactions:")
fraud_percentage = data['Class'].value_counts(normalize=True)[1] * 100
print(f"{fraud_percentage:.2f}%")

# Visualization: Class Distribution
plt.figure(figsize=(8, 6))
sns.countplot(x='Class', data=data, palette='viridis', hue='Class',
dodge=False, legend=False)
plt.title('Distribution of Normal vs Fraudulent Transactions')
plt.xlabel('Class (0 = Normal, 1 = Fraud)')
plt.ylabel('Count')
plt.show()

# Visualization: Amount Distribution for Fraud vs Normal Transactions
```

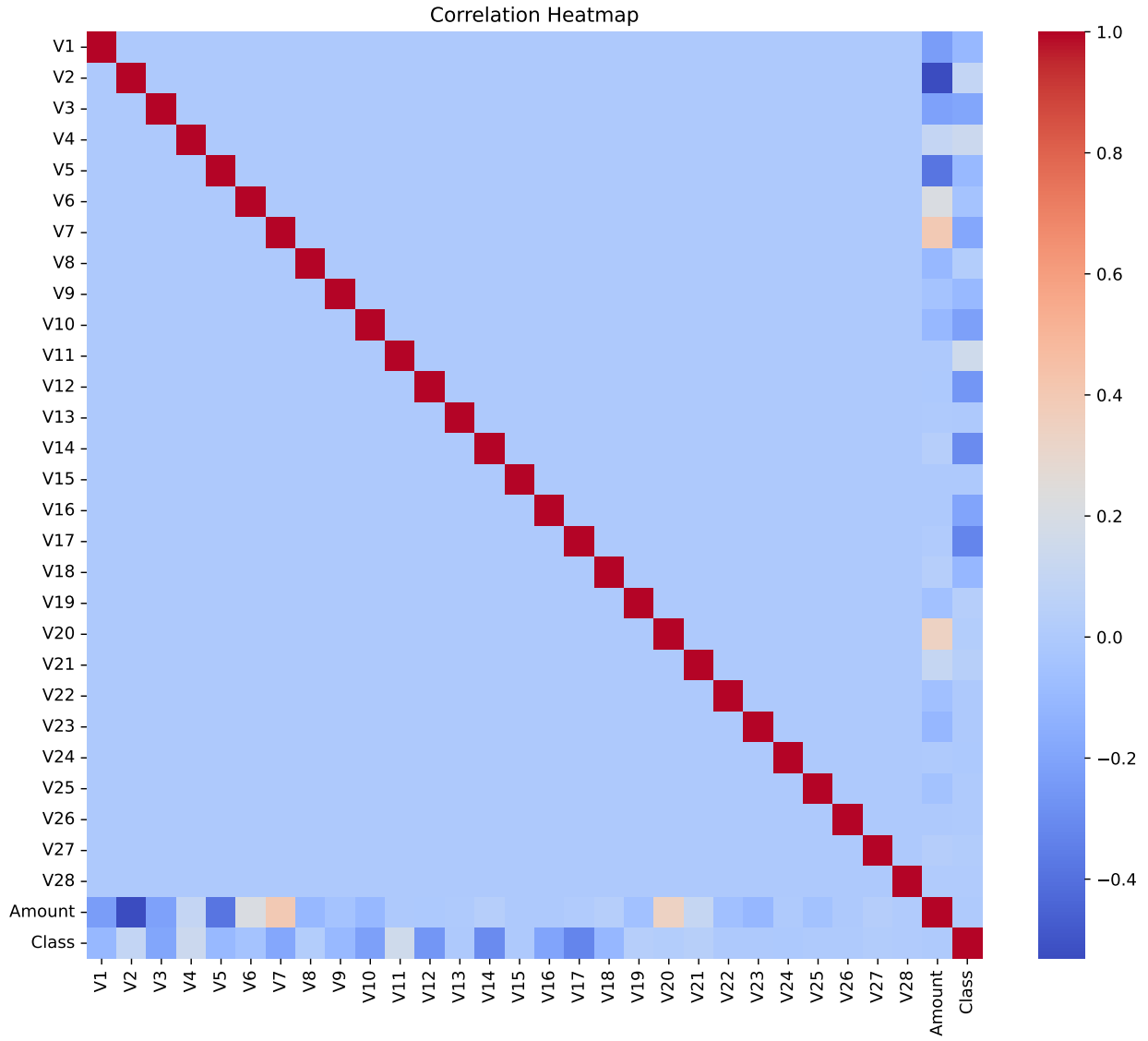
```
plt.figure(figsize=(10, 6))
sns.countplot(x='Class', data=data, palette='viridis', hue='Class',
dodge=False, legend=False)
plt.title('Transaction Amount Distribution by Class')
plt.xlabel('Class (0 = Normal, 1 = Fraud)')
plt.ylabel('Normalized Amount')
plt.yscale('log')  # Scale to log to visualize outliers better
plt.show()

# Visualization: Correlation Heatmap
plt.figure(figsize=(12, 10))
corr = data.corr()
sns.heatmap(corr, annot=False, cmap='coolwarm', cbar=True)
plt.title('Correlation Heatmap')
plt.show()

# Separate features and labels
X = data.drop(columns=['Class'])
y = data['Class']

# Step 4: Implement Anomaly Detection Techniques
# Define contamination based on fraud percentage
contamination_rate = fraud_percentage / 100

# Isolation Forest
iso_forest = IsolationForest(contamination=contamination_rate,
random_state=42)
y_pred_iso = iso_forest.fit_predict(X)
y_pred_iso = np.where(y_pred_iso == 1, 0, 1)
# Local Outlier Factor
lof = LocalOutlierFactor(n_neighbors=20, contamination=contamination_rate)
y_pred_lof = lof.fit_predict(X)
y_pred_lof = np.where(y_pred_lof == 1, 0, 1)

# Step 5: Evaluate Performance
# Helper function for evaluation
def evaluate_model(y_true, y_pred, model_name):
    print(f"\nPerformance Metrics for {model_name}:")
    print("Precision:", precision_score(y_true, y_pred))
    print("Recall:", recall_score(y_true, y_pred))
    print("F1-Score:", f1_score(y_true, y_pred))
    print("ROC-AUC:", roc_auc_score(y_true, y_pred))
    print("\nClassification Report:\n", classification_report(y_true, y_pred))
```

```
# Evaluate Isolation Forest
evaluate_model(y, y_pred_iso, "Isolation Forest")

# Evaluate Local Outlier Factor
evaluate_model(y, y_pred_lof, "Local Outlier Factor")

# Visualization: ROC Curve
from sklearn.metrics import roc_curve, auc

# Function to plot ROC Curve
def plot_roc_curve(y_true, y_pred, model_name):
    fpr, tpr, _ = roc_curve(y_true, y_pred)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='darkorange', lw=2,
    label=f'{model_name} (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.title(f'ROC Curve - {model_name}')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc="lower right")
    plt.show()

# Plot ROC Curve for both models
plot_roc_curve(y, y_pred_iso, "Isolation Forest")
plot_roc_curve(y, y_pred_lof, "Local Outlier Factor")
```

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

Percentage of Fraudulent Transactions:
0.17%
```

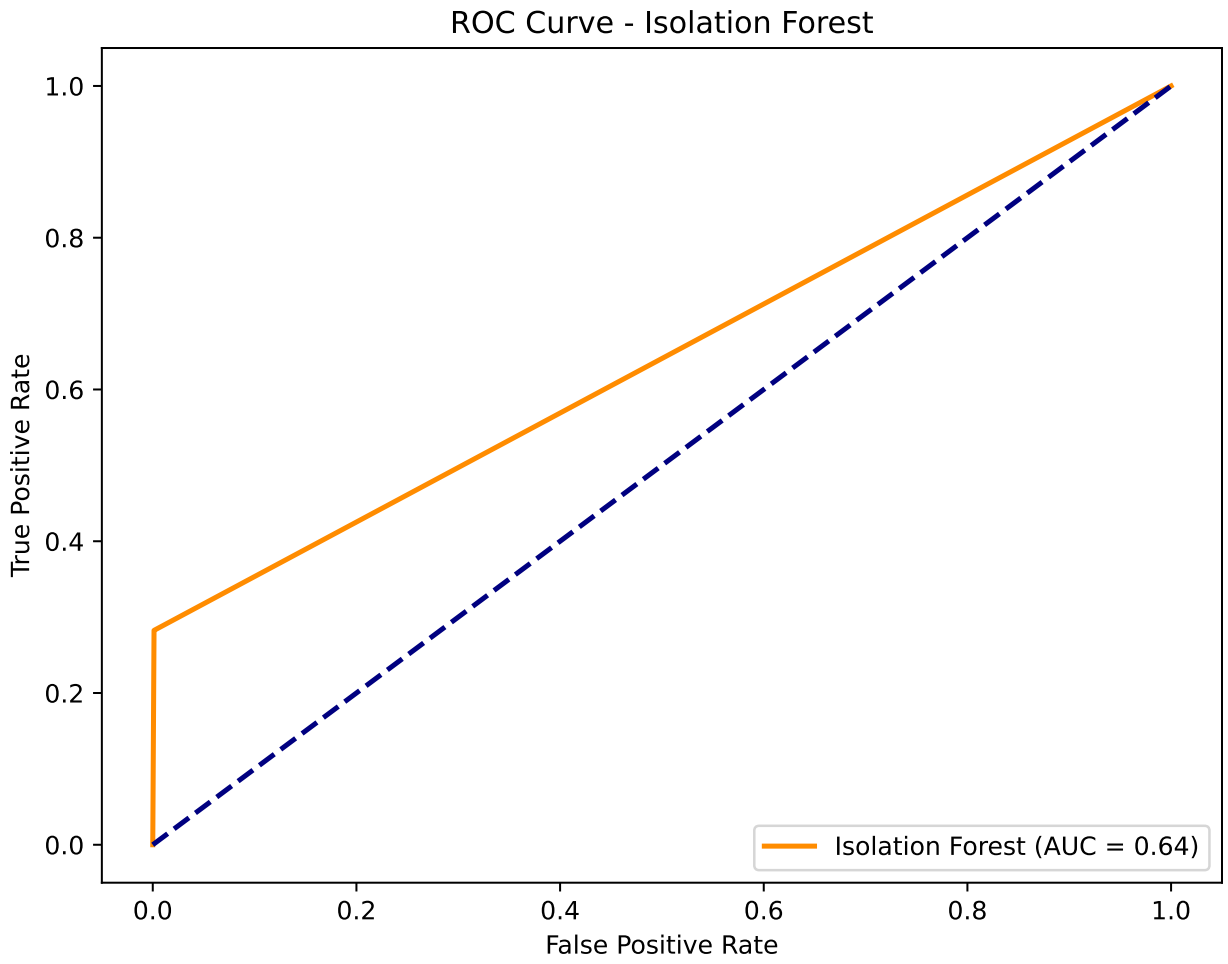

Distribution of Normal vs Fraudulent Transactions
Count
250000
200000
150000
100000
50000
0
0
1
Class (0 = Normal, 1 = Fraud)

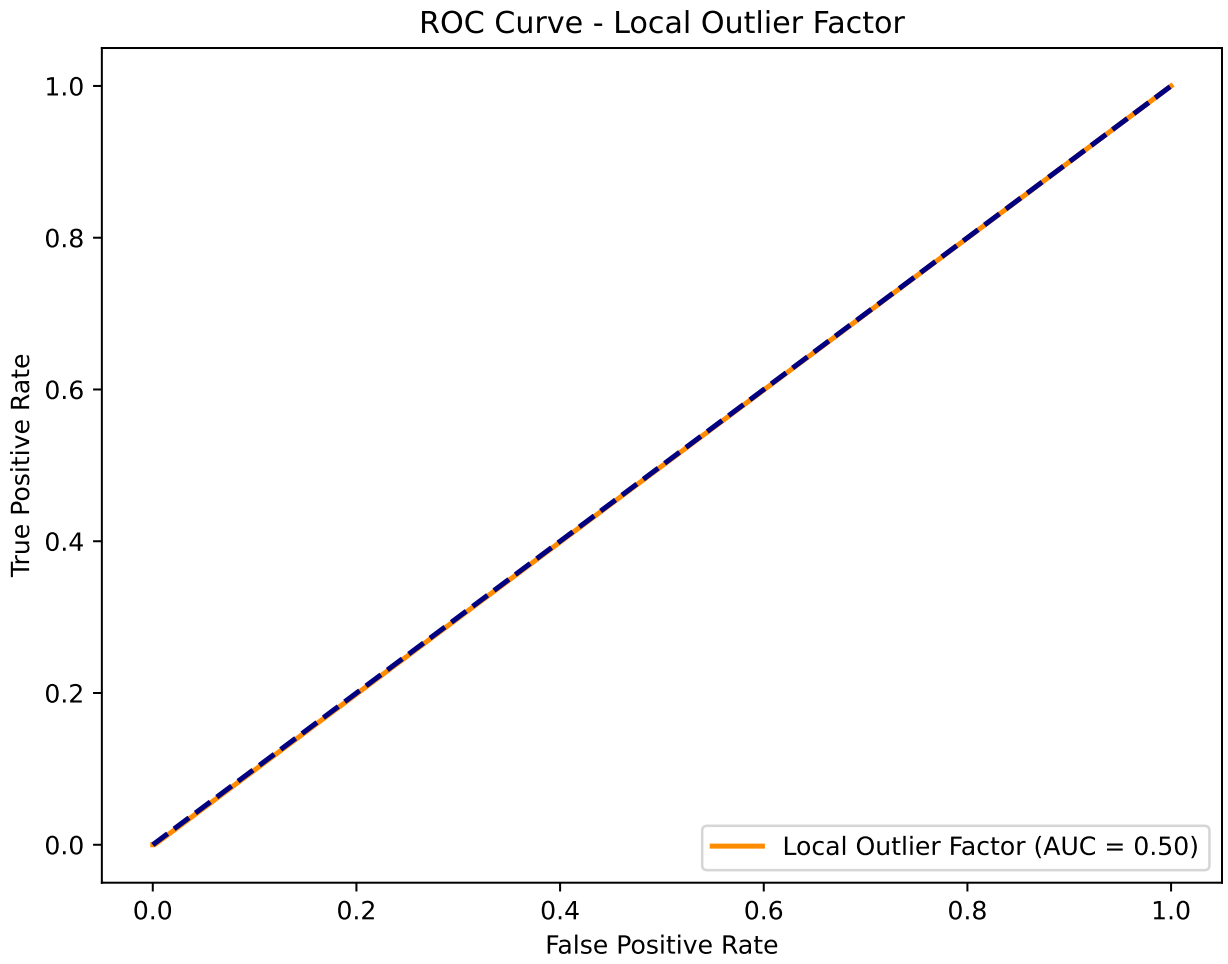

Transaction Amount Distribution by Class
Normalized Amount
10^5
10^4
10^3
0
1
Class (0 = Normal, 1 = Fraud)




```
Performance Metrics for Isolation Forest:
Precision: 0.28252032520325204
Recall: 0.28252032520325204
F1-Score: 0.28252032520325204
ROC-AUC: 0.640639372281031

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    284315
           1       0.28      0.28      0.28       492

    accuracy                           1.00    284807
```

```
   macro avg       0.64      0.64      0.64    284807
weighted avg       1.00      1.00      1.00    284807


Performance Metrics for Local Outlier Factor:
Precision: 0.0
Recall: 0.0
F1-Score: 0.0
ROC-AUC: 0.4991347624993405

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    284315
           1       0.00      0.00      0.00       492

    accuracy                           1.00    284807
   macro avg       0.50      0.50      0.50    284807
weighted avg       1.00      1.00      1.00    284807
```


ROC Curve - Isolation Forest
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Isolation Forest (AUC = 0.64)


ROC Curve - Local Outlier Factor
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Local Outlier Factor (AUC = 0.50)