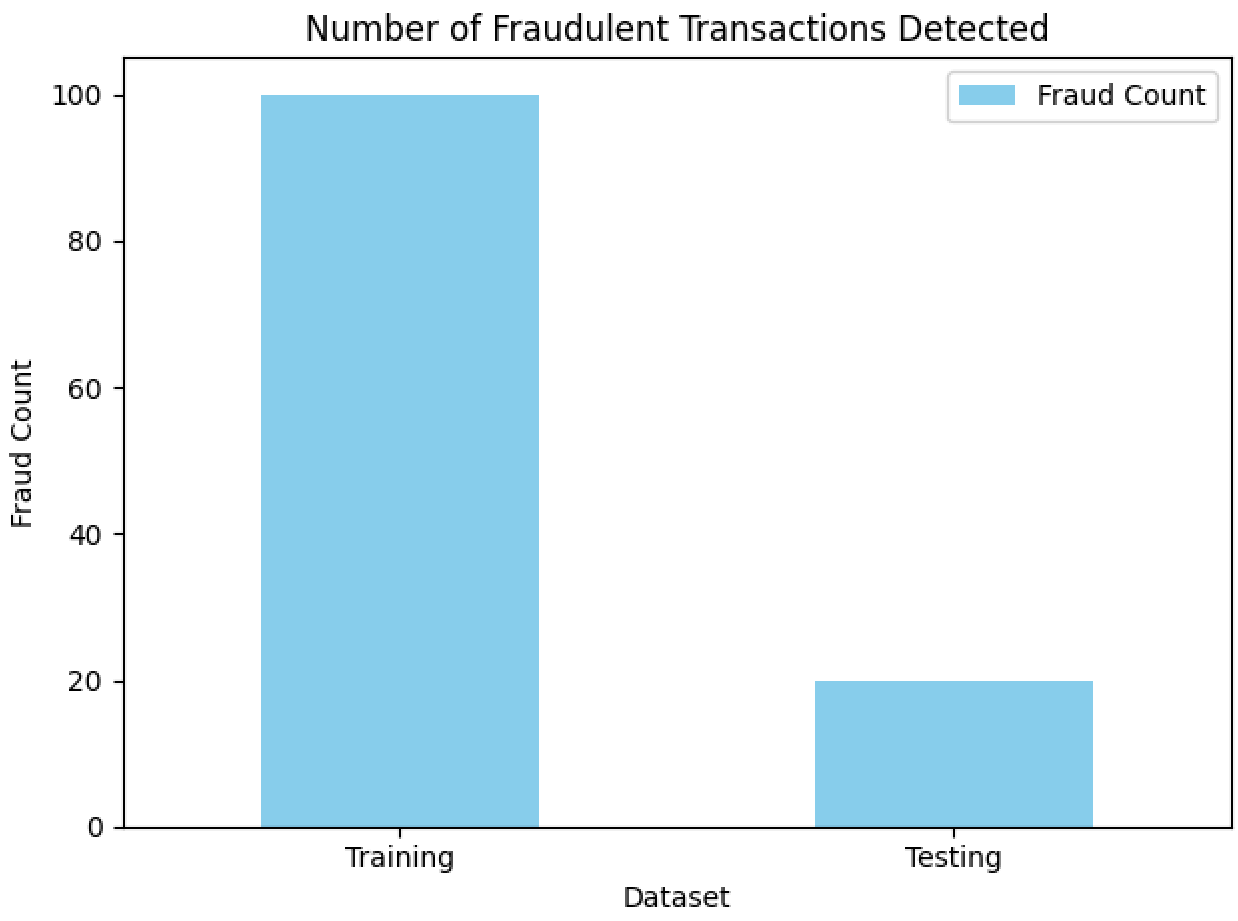
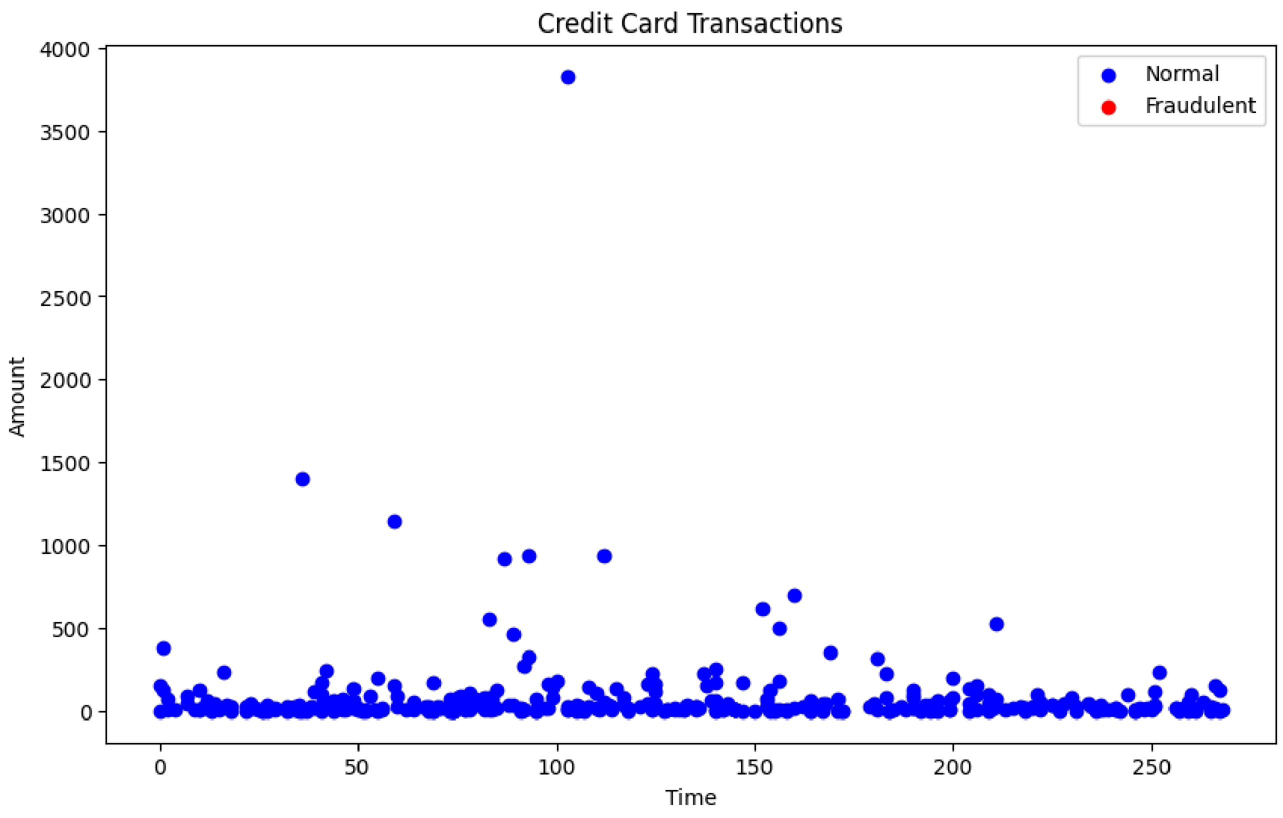
```
import pandas as pd
import matplotlib.pyplot as plt



  data = pd.read_csv('Book1.csv')


fraud_counts = {
    'Training': 100,  # Replace with the actual count of fraudulent transactions in the
    'Testing': 20     # Replace with the actual count of fraudulent transactions in the
}

# Convert the fraud counts dictionary to a pandas DataFrame
fraud_df = pd.DataFrame.from_dict(fraud_counts, orient='index', columns=['Fraud Count'])

# Plot the bar chart
fraud_df.plot(kind='bar', color='skyblue')
plt.title('Number of Fraudulent Transactions Detected')
plt.xlabel('Dataset')
plt.ylabel('Fraud Count')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```



```
# Load the dataset (replace 'credit_card_data.csv' with your dataset)
data = pd.read_csv('Book1.csv')

# Assuming 'Class' column indicates fraud (1) or normal (0)
fraudulent = data[data['Class'] == 1]
normal = data[data['Class'] == 0]

# Plot fraudulent transactions in red and normal transactions in blue
plt.figure(figsize=(10, 6))
plt.scatter(normal['Time'], normal['Amount'], color='blue', label='Normal')
plt.scatter(fraudulent['Time'], fraudulent['Amount'], color='red', label='Fraudulent')
plt.title('Credit Card Transactions')
plt.xlabel('Time')
plt.ylabel('Amount')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x78471d3ed8a0>
```



```
import pandas as pd
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix


data = pd.read_csv('Book1.csv')
```

```
X = data.drop('Class', axis=1)
y = data['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
clf = IsolationForest(n_estimators=100, max_samples='auto', contamination=0.01, random_s
clf.fit(X_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_train[y_pred_train == 1] = 0
y_pred_train[y_pred_train == -1] = 1
y_pred_test[y_pred_test == 1] = 0
y_pred_test[y_pred_test == -1] = 1

print("Training Classification Report:")
print(classification_report(y_train, y_pred_train))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
  warnings.warn(
Training Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       294
           1       0.00      0.00      0.00         0

    accuracy                           0.99       294
   macro avg       0.50      0.49      0.50       294
weighted avg       1.00      0.99      0.99       294

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
  _warn_prf(average, modifier, msg_start, len(result))
```

```
print("\nConfusion Matrix for Testing:")
print(confusion_matrix(y_test, y_pred_test))
```

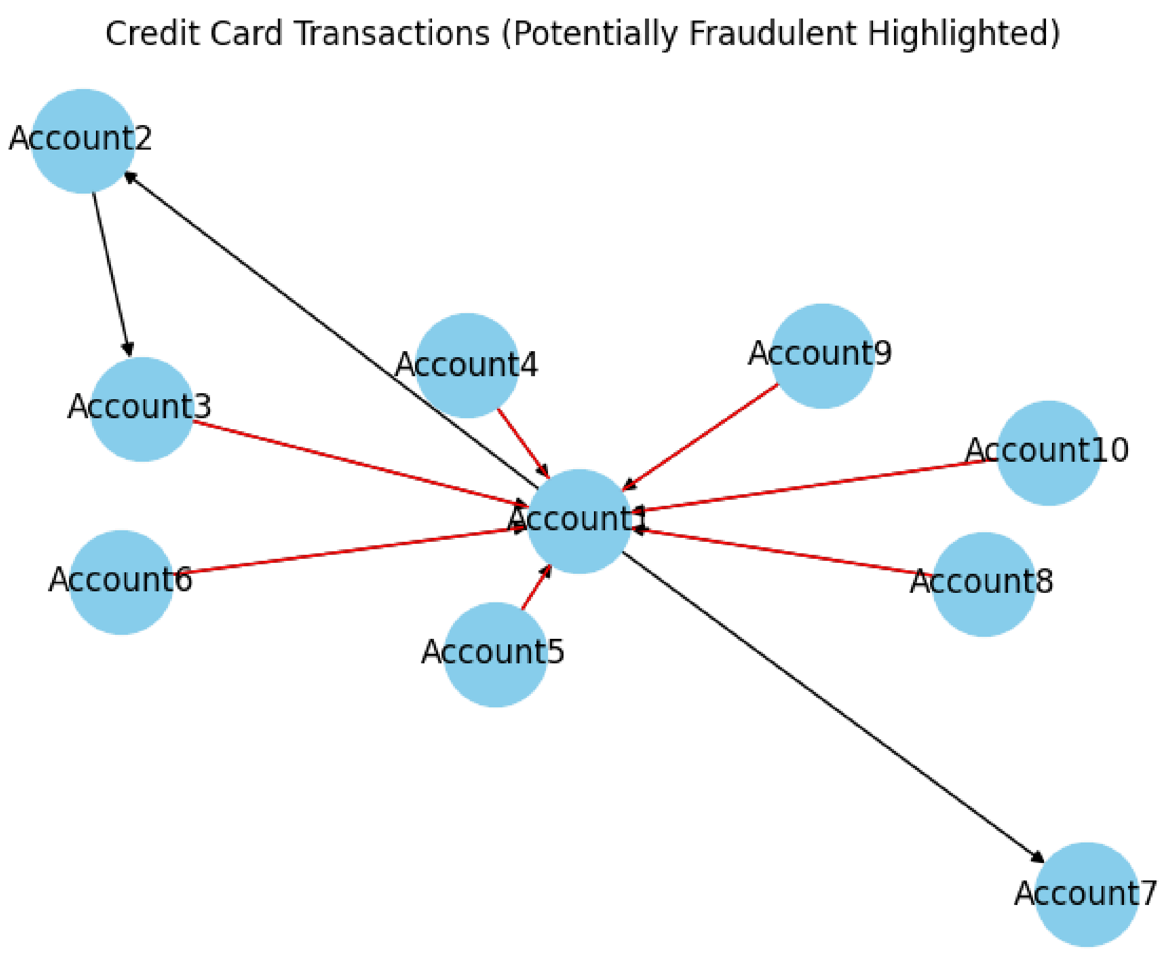
```
Confusion Matrix for Testing:
[[ 0 74]
 [ 0  0]]
```

```
print("\nTesting Classification Report:")
print(classification_report(y_test, y_pred_test))
```

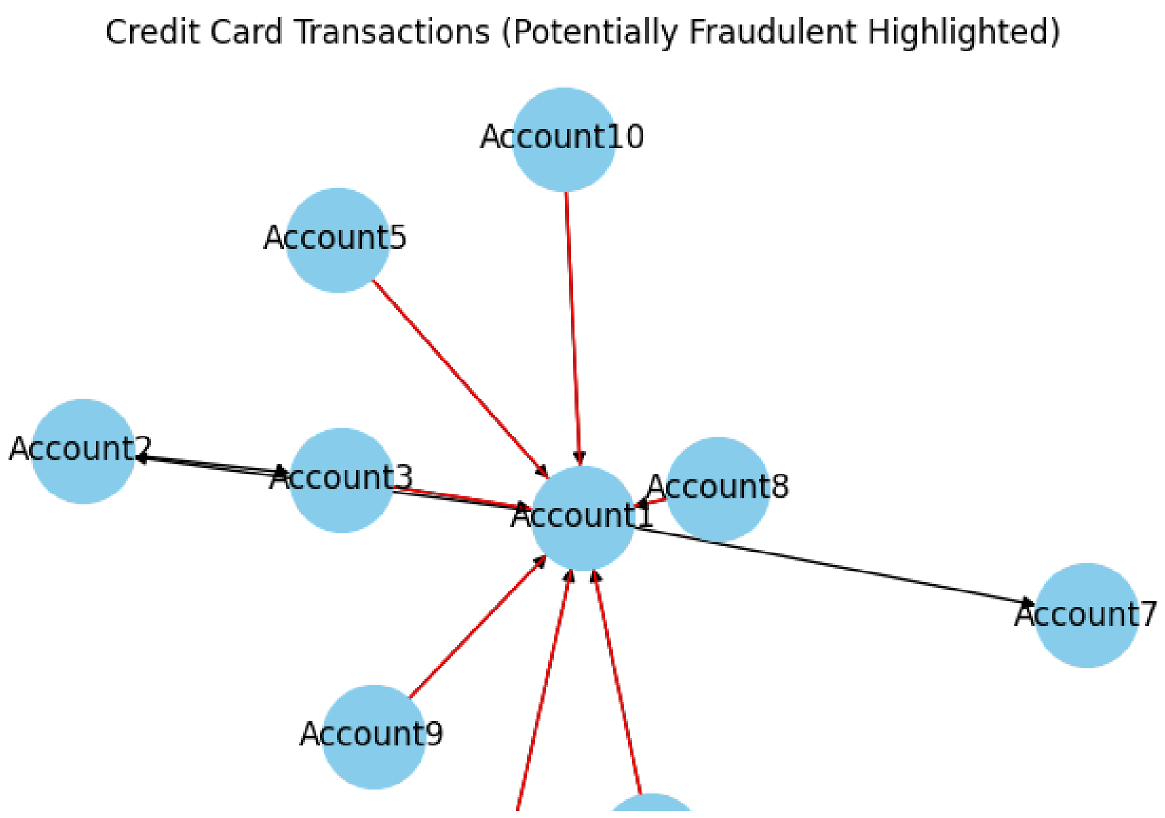
```
Testing Classification Report:
              precision    recall  f1-score   support
```

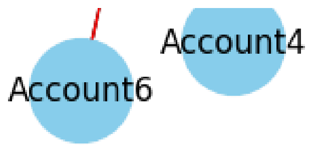
```
           0       0.00      0.00      0.00      74.0
           1       0.00      0.00      0.00       0.0

    accuracy                           0.00      74.0
   macro avg       0.00      0.00      0.00      74.0
weighted avg       0.00      0.00      0.00      74.0

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
  _warn_prf(average, modifier, msg_start, len(result))
```

```
import networkx as nx
import matplotlib.pyplot as plt

# Sample credit card transaction data (edges: transactions, nodes: accounts)
transactions = [
    ('Account1', 'Account2', {'amount': 100}),
    ('Account2', 'Account3', {'amount': 50}),
    ('Account3', 'Account1', {'amount': 200}),
    ('Account4', 'Account1', {'amount': 150}),
    ('Account5', 'Account1', {'amount': 300}),
    ('Account6', 'Account1', {'amount': 500}),
    ('Account1', 'Account7', {'amount': 400}),
    ('Account8', 'Account1', {'amount': 200}),
    ('Account9', 'Account1', {'amount': 100}),
    ('Account10', 'Account1', {'amount': 300}),
]

# Create a directed graph
G = nx.DiGraph()

# Add transactions as edges with amount attribute
G.add_edges_from(transactions)

# Calculate node centrality (e.g., degree centrality)
centrality = nx.degree_centrality(G)

# Set threshold for detecting anomalous transactions (e.g., above 75th percentile)
threshold = sorted(centrality.values())[int(0.75 * len(centrality))]

# Identify potentially fraudulent transactions
fraudulent_transactions = [(u, v) for (u, v) in G.edges() if centrality[v] > threshold]

# Visualize the graph with potentially fraudulent transactions highlighted
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=1500)
nx.draw_networkx_edges(G, pos, edgelist=fraudulent_transactions, edge_color='red', arrow
plt.title('Credit Card Transactions (Potentially Fraudulent Highlighted)')
plt.show()

# Output potentially fraudulent transactions
print("Potentially fraudulent transactions:")
for transaction in fraudulent_transactions:
    print(transaction)
```



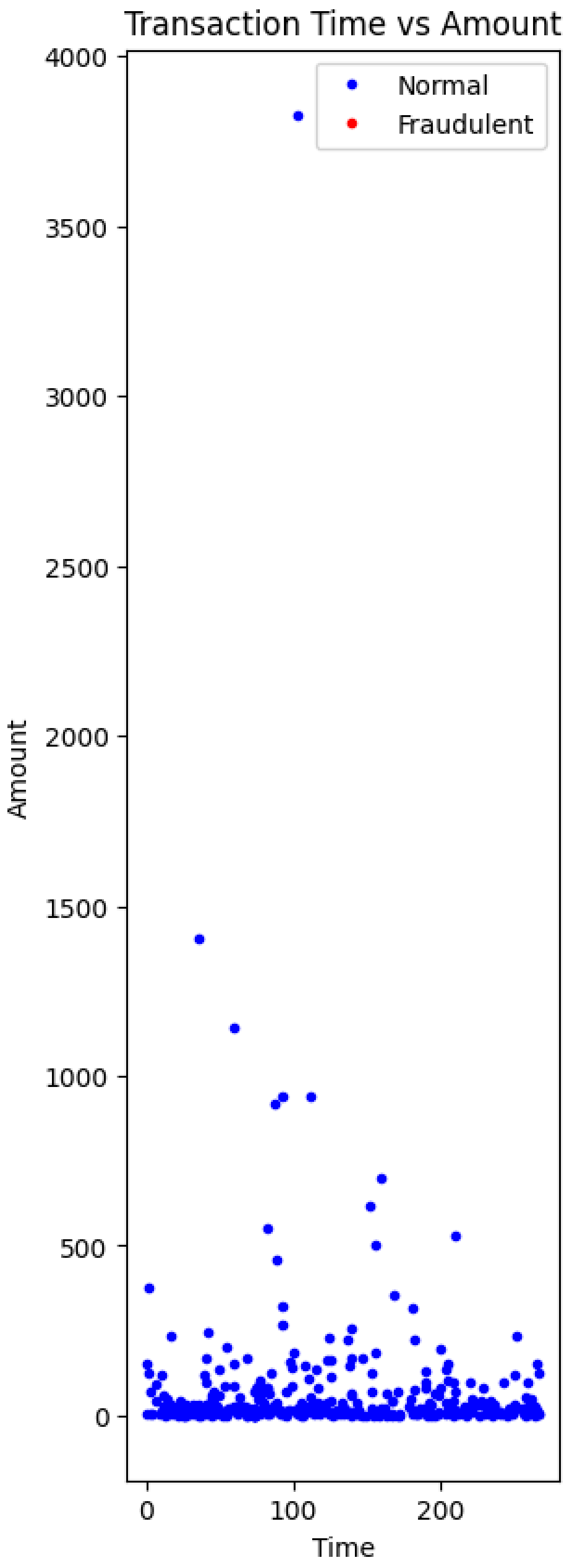
```
Potentially fraudulent transactions:
('Account3', 'Account1')
('Account4', 'Account1')
('Account5', 'Account1')
('Account6', 'Account1')
('Account8', 'Account1')
('Account9', 'Account1')
('Account10', 'Account1')
```



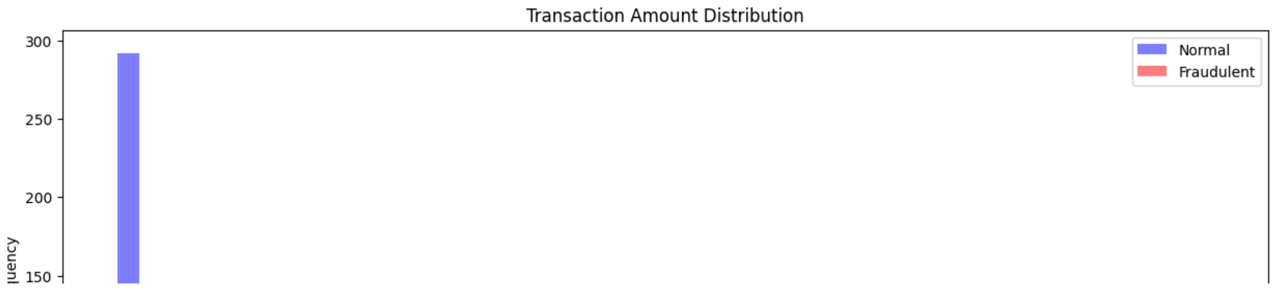


```
Potentially fraudulent transactions:
('Account3', 'Account1')
('Account4', 'Account1')
('Account5', 'Account1')
('Account6', 'Account1')
('Account8', 'Account1')
('Account9', 'Account1')
```

```
data = pd.read_csv('Book1.csv')
plt.figure(figsize=(3, 10))
plt.plot(normal['Time'], normal['Amount'], linestyle='None', marker='o', markersi
plt.plot(fraudulent['Time'], fraudulent['Amount'], linestyle='None', marker='o',
plt.title('Transaction Time vs Amount')
plt.xlabel('Time')
plt.ylabel('Amount')
plt.legend()
plt.show()
```


Transaction Time vs Amount
Normal
Fraudulent
Amount
Time
4000
3500
3000
2500
2000
1500
1000
500
0
0
100
200

```
data = pd.read_csv('Book1.csv')
fraudulent = data[data['Class'] == 1]
normal = data[data['Class'] == 0]

# Plot histograms for transaction amount
plt.figure(figsize=(15, 6))
plt.hist(normal['Amount'], bins=50, color='blue', alpha=0.5, label='Normal')
plt.hist(fraudulent['Amount'], bins=50, color='red', alpha=0.5, label='Fraudulent')
plt.title('Transaction Amount Distribution')
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



```
data = pd.read_csv('Book1.csv')

# Assuming 'Class' column indicates fraud (1) or normal (0)
fraudulent = data[data['Class'] == 1]
normal = data[data['Class'] == 0]

# Create a figure and two subplots (one horizontal and one vertical)
```