1. Data cleaning including missing values, outliers and multi-collinearity.

(i) Missing Values

In [3]:
```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Sample dataset creation
data = pd.read_csv('Downloads/Fraud.csv')


df = pd.DataFrame(data)
# Display the original data
print("Original Data:")
print(df)

# Step 1: Handling Missing Values
# Impute missing values in 'Age' with the mean
# 1. Check missing values
missing_data = df.isnull().sum()

# 2. Impute missing values
df['step'].fillna(method='ffill', inplace=True)  # Forward fill for time steps
df['type'].fillna(df['type'].mode()[0], inplace=True)  # Mode for transaction type
df['amount'].fillna(df['amount'].median(), inplace=True)  # Median for amount
df['nameOrig'].fillna('Unknown', inplace=True)  # Placeholder for missing names
df['oldbalanceOrg'].fillna(df['oldbalanceOrg'].mean(), inplace=True)  # Mean for balance
df['newbalanceOrig'].fillna(df['newbalanceOrig'].mean(), inplace=True)
df['nameDest'].fillna('Unknown', inplace=True)
df['oldbalanceDest'].fillna(df['oldbalanceDest'].mean(), inplace=True)
df['newbalanceDest'].fillna(df['newbalanceDest'].mean(), inplace=True)
df['isFraud'].fillna(df['isFraud'].mode()[0], inplace=True)

print("\nData after handling missing values:")
print(df)
```

```
Original Data:
         step      type     amount     nameOrig  oldbalanceOrg  \
0           1   PAYMENT    9839.64  C1231006815      170136.00
1           1   PAYMENT    1864.28  C1666544295       21249.00
2           1  TRANSFER     181.00  C1305486145         181.00
3           1  CASH_OUT     181.00   C840083671         181.00
4           1   PAYMENT   11668.14  C2048537720       41554.00
...       ...       ...        ...          ...            ...
1048570    95  CASH_OUT  132557.35  C1179511630      479803.00
1048571    95   PAYMENT    9917.36  C1956161225       90545.00
1048572    95   PAYMENT   14140.05  C2037964975       20545.00
1048573    95   PAYMENT   10020.05  C1633237354       90605.00
1048574    95   PAYMENT   11450.03  C1264356443       80584.95

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
...                 ...          ...             ...             ...      ...
1048570       347245.65   C435674507       484329.37       616886.72        0
1048571        80627.64   M668364942            0.00            0.00        0
1048572         6404.95  M1355182933            0.00            0.00        0
1048573        80584.95  M1964992463            0.00            0.00        0
1048574        69134.92   M677577406            0.00            0.00        0

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
4                     0
...                 ...
1048570               0
1048571               0
1048572               0
1048573               0
1048574               0

[1048575 rows x 11 columns]
```

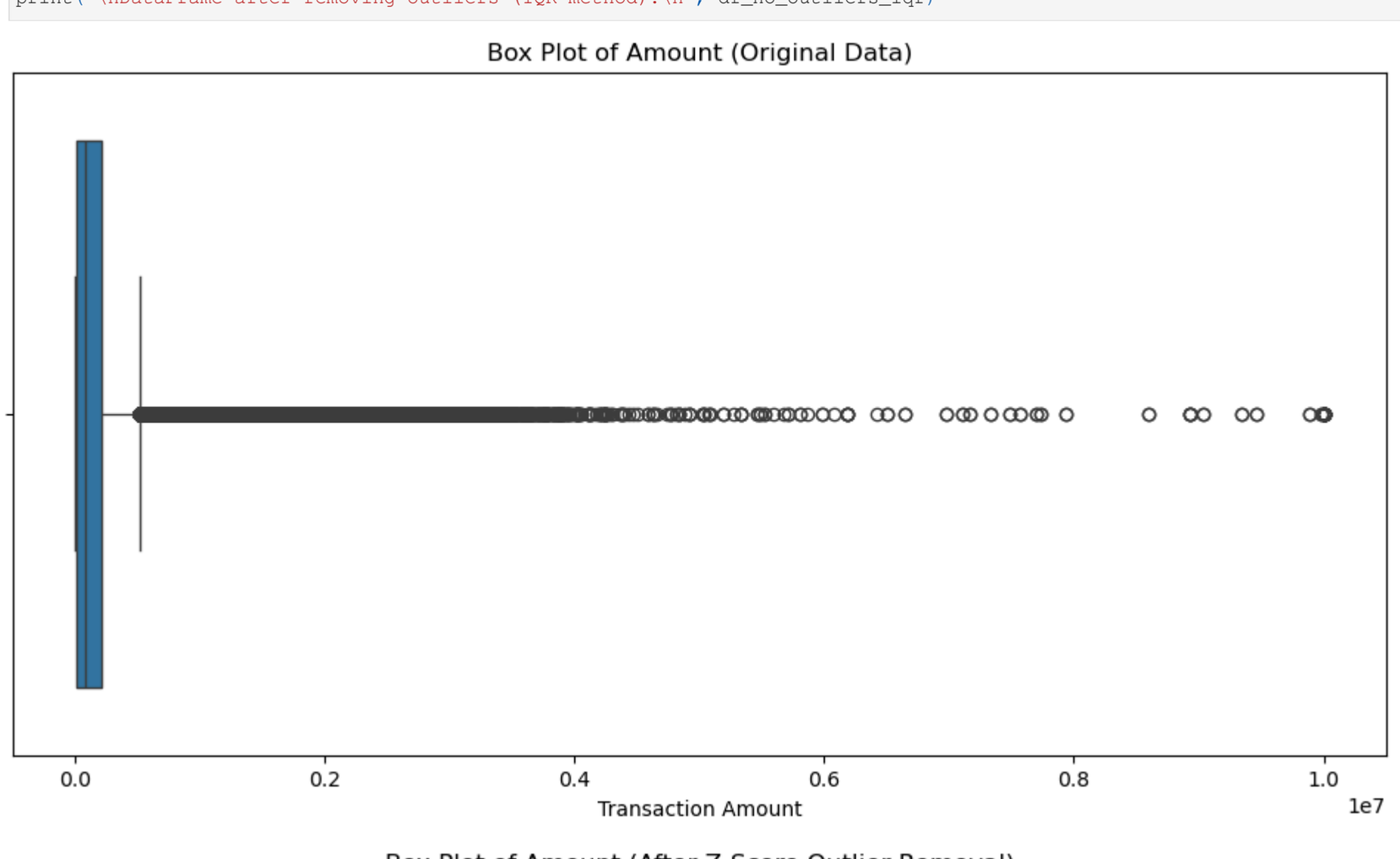
```
/tmp/ipykernel_22171/1782945538.py:22: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['step'].fillna(method='ffill', inplace=True)  # Forward fill for time steps
/tmp/ipykernel_22171/1782945538.py:22: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
  df['step'].fillna(method='ffill', inplace=True)  # Forward fill for time steps
/tmp/ipykernel_22171/1782945538.py:23: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['type'].fillna(df['type'].mode()[0], inplace=True)  # Mode for transaction type
/tmp/ipykernel_22171/1782945538.py:24: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['amount'].fillna(df['amount'].median(), inplace=True)  # Median for amount
/tmp/ipykernel_22171/1782945538.py:25: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['nameOrig'].fillna('Unknown', inplace=True)  # Placeholder for missing names
/tmp/ipykernel_22171/1782945538.py:26: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['oldbalanceOrg'].fillna(df['oldbalanceOrg'].mean(), inplace=True)  # Mean for balance
/tmp/ipykernel_22171/1782945538.py:27: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['newbalanceOrig'].fillna(df['newbalanceOrig'].mean(), inplace=True)
/tmp/ipykernel_22171/1782945538.py:28: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['nameDest'].fillna('Unknown', inplace=True)
/tmp/ipykernel_22171/1782945538.py:29: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['oldbalanceDest'].fillna(df['oldbalanceDest'].mean(), inplace=True)
/tmp/ipykernel_22171/1782945538.py:30: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['newbalanceDest'].fillna(df['newbalanceDest'].mean(), inplace=True)
```

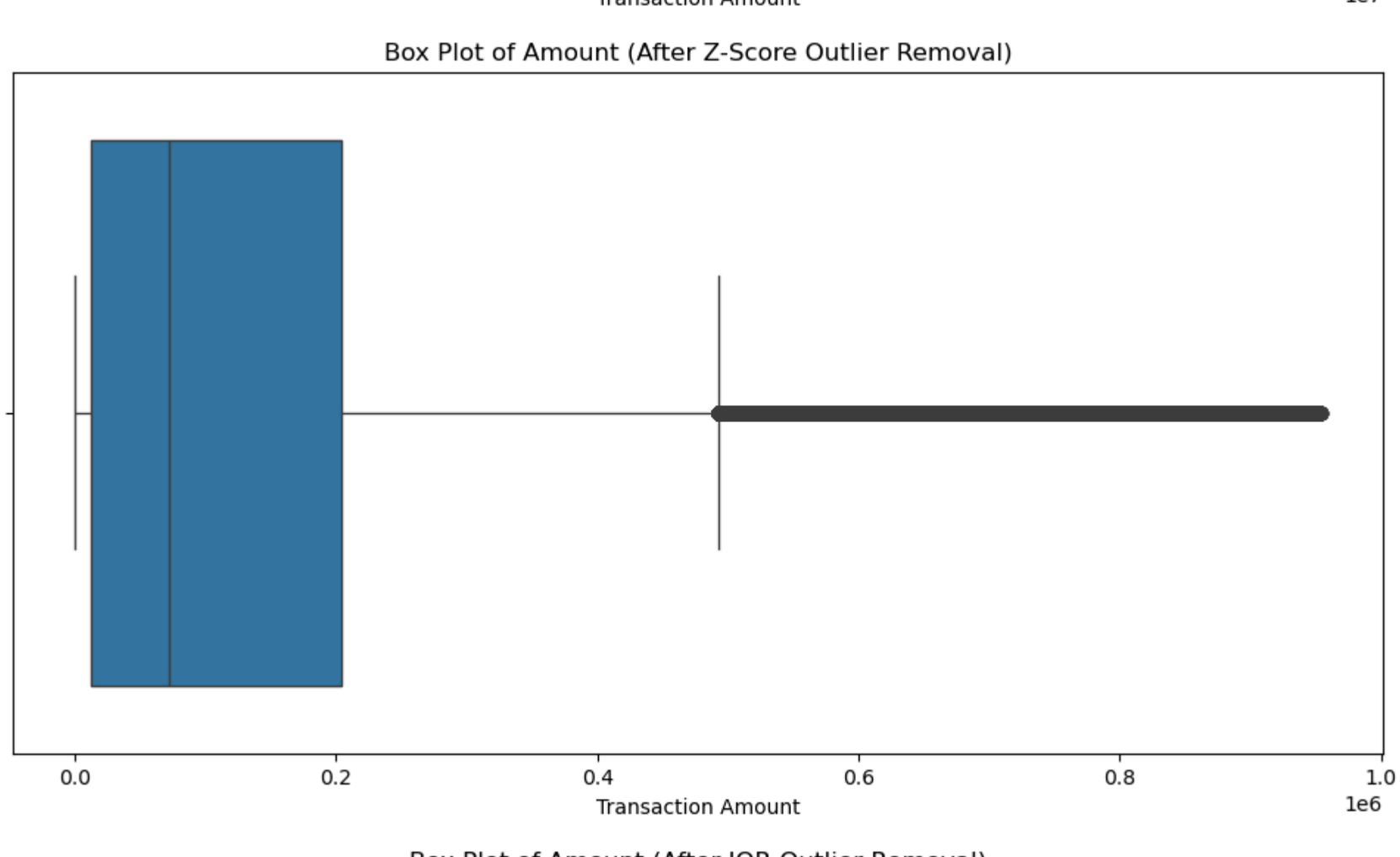
```
Data after handling missing values:
         step      type     amount     nameOrig  oldbalanceOrg  \
0           1   PAYMENT    9839.64  C1231006815      170136.00
1           1   PAYMENT    1864.28  C1666544295       21249.00
2           1  TRANSFER     181.00  C1305486145         181.00
3           1  CASH_OUT     181.00   C840083671         181.00
4           1   PAYMENT   11668.14  C2048537720       41554.00
...       ...       ...        ...          ...            ...
1048570    95  CASH_OUT  132557.35  C1179511630      479803.00
1048571    95   PAYMENT    9917.36  C1956161225       90545.00
1048572    95   PAYMENT   14140.05  C2037964975       20545.00
1048573    95   PAYMENT   10020.05  C1633237354       90605.00
1048574    95   PAYMENT   11450.03  C1264356443       80584.95

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
...                 ...          ...             ...             ...      ...
1048570       347245.65   C435674507       484329.37       616886.72        0
1048571        80627.64   M668364942            0.00            0.00        0
1048572         6404.95  M1355182933            0.00            0.00        0
1048573        80584.95  M1964992463            0.00            0.00        0
1048574        69134.92   M677577406            0.00            0.00        0

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
4                     0
...                 ...
1048570               0
1048571               0
1048572               0
1048573               0
1048574               0

[1048575 rows x 11 columns]
```

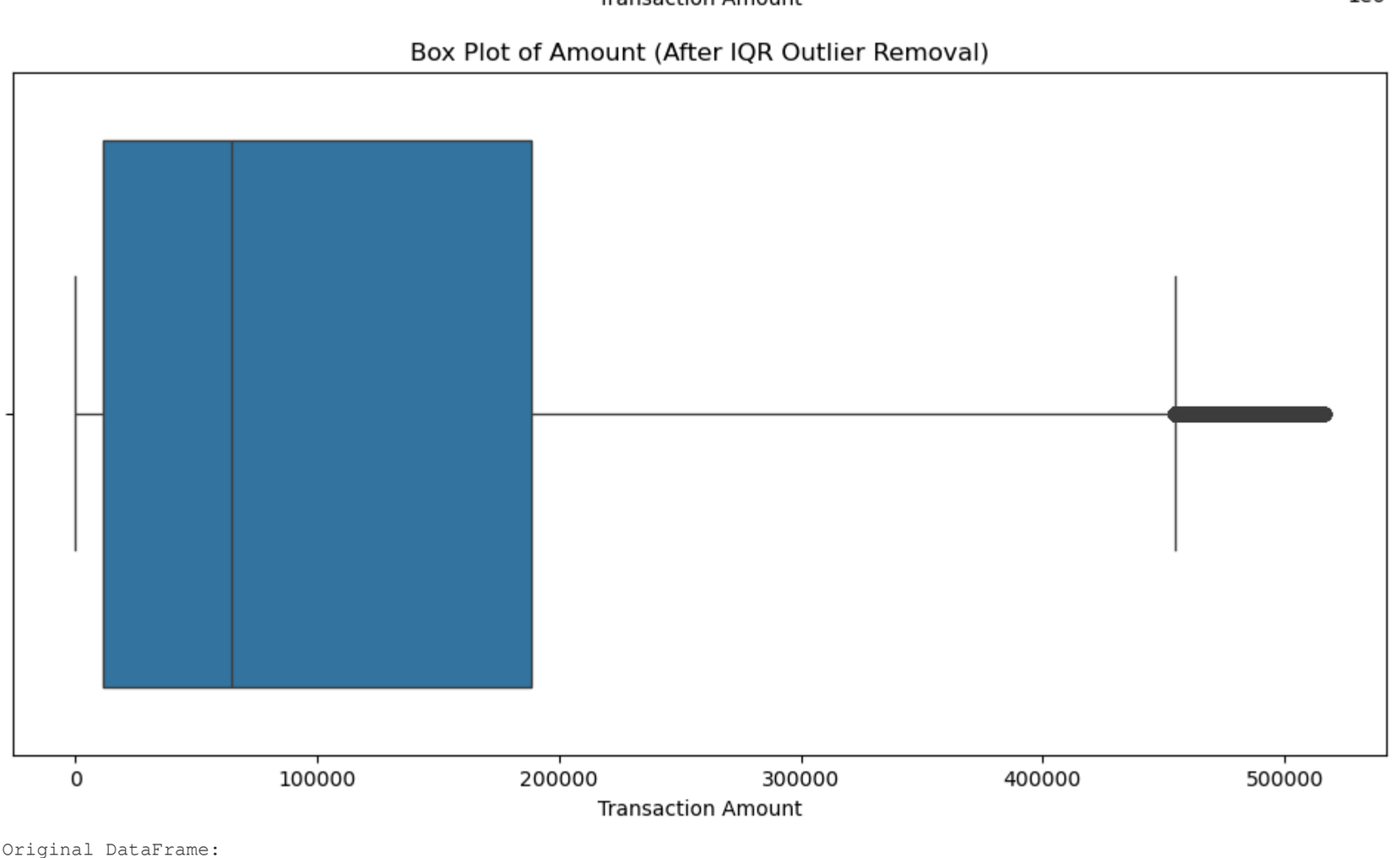
```
/tmp/ipykernel_22171/1782945538.py:31: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['isFraud'].fillna(df['isFraud'].mode()[0], inplace=True)
```

(ii) Outlier

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

# Sample DataFrame Creation
# Replace this with your actual dataset
data = pd.read_csv('Downloads/Fraud.csv')

df = pd.DataFrame(data)

# Function to remove outliers using Z-Score method
def remove_outliers_z_score(dataframe, column):
    threshold = 3  # Z-score threshold
    z_scores = (dataframe[column] - dataframe[column].mean()) / dataframe[column].std()
    filtered_df = dataframe[(z_scores > -threshold) & (z_scores < threshold)]
    return filtered_df

# Function to remove outliers using IQR method
def remove_outliers_iqr(dataframe, column):
    Q1 = dataframe[column].quantile(0.25)
    Q3 = dataframe[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    filtered_df = dataframe[(dataframe[column] >= lower_bound) & (dataframe[column] <= upper_bound)]
    return filtered_df

# Visualize the original data with box plots
plt.figure(figsize=(12, 6))
sns.boxplot(x=df['amount'])
plt.title('Box Plot of Amount (Original Data)')
plt.xlabel('Transaction Amount')
plt.show()

# Remove outliers for the 'amount' column using both methods
df_no_outliers_z = remove_outliers_z_score(df, 'amount')
df_no_outliers_iqr = remove_outliers_iqr(df, 'amount')

# Visualize the data after removing outliers using Z-Score method
plt.figure(figsize=(12, 6))
sns.boxplot(x=df_no_outliers_z['amount'])
plt.title('Box Plot of Amount (After Z-Score Outlier Removal)')
plt.xlabel('Transaction Amount')
plt.show()

# Visualize the data after removing outliers using IQR method
plt.figure(figsize=(12, 6))
sns.boxplot(x=df_no_outliers_iqr['amount'])
plt.title('Box Plot of Amount (After IQR Outlier Removal)')
plt.xlabel('Transaction Amount')
plt.show()

# Print the results
print("Original DataFrame:\n", df)
print("\nDataFrame after removing outliers (Z-Score method):\n", df_no_outliers_z)
print("\nDataFrame after removing outliers (IQR method):\n", df_no_outliers_iqr)
```





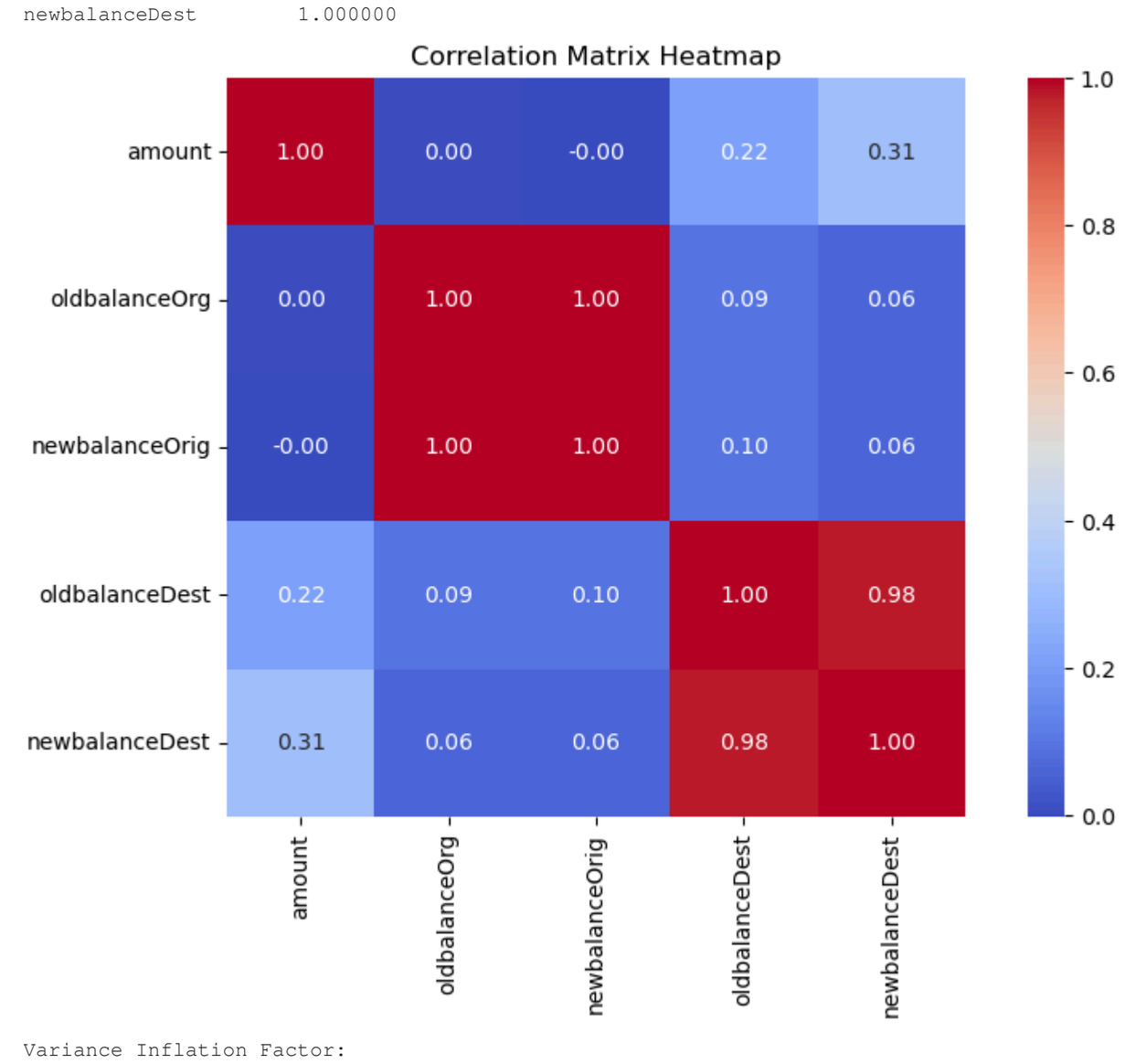



```
Original DataFrame:
         step      type     amount     nameOrig  oldbalanceOrg  \
0           1   PAYMENT    9839.64  C1231006815      170136.00
1           1   PAYMENT    1864.28  C1666544295       21249.00
2           1  TRANSFER     181.00  C1305486145         181.00
3           1  CASH_OUT     181.00   C840083671         181.00
4           1   PAYMENT   11668.14  C2048537720       41554.00
...       ...       ...        ...          ...            ...
1048570    95  CASH_OUT  132557.35  C1179511630      479803.00
1048571    95   PAYMENT    9917.36  C1956161225       90545.00
1048572    95   PAYMENT   14140.05  C2037964975       20545.00
1048573    95   PAYMENT   10020.05  C1633237354       90605.00
1048574    95   PAYMENT   11450.03  C1264356443       80584.95

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
...                 ...          ...             ...             ...      ...
1048570       347245.65   C435674507       484329.37       616886.72        0
1048571        80627.64   M668364942            0.00            0.00        0
1048572         6404.95  M1355182933            0.00            0.00        0
1048573        80584.95  M1964992463            0.00            0.00        0
1048574        69134.92   M677577406            0.00            0.00        0

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
4                     0
...                 ...
1048570               0
1048571               0
1048572               0
1048573               0
1048574               0

[1048575 rows x 11 columns]

DataFrame after removing outliers (Z-Score method):
         step      type     amount     nameOrig  oldbalanceOrg  \
0           1   PAYMENT    9839.64  C1231006815      170136.00
1           1   PAYMENT    1864.28  C1666544295       21249.00
2           1  TRANSFER     181.00  C1305486145         181.00
3           1  CASH_OUT     181.00   C840083671         181.00
4           1   PAYMENT   11668.14  C2048537720       41554.00
...       ...       ...        ...          ...            ...
1048570    95  CASH_OUT  132557.35  C1179511630      479803.00
1048571    95   PAYMENT    9917.36  C1956161225       90545.00
1048572    95   PAYMENT   14140.05  C2037964975       20545.00
1048573    95   PAYMENT   10020.05  C1633237354       90605.00
1048574    95   PAYMENT   11450.03  C1264356443       80584.95

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
...                 ...          ...             ...             ...      ...
1048570       347245.65   C435674507       484329.37       616886.72        0
1048571        80627.64   M668364942            0.00            0.00        0
1048572         6404.95  M1355182933            0.00            0.00        0
1048573        80584.95  M1964992463            0.00            0.00        0
1048574        69134.92   M677577406            0.00            0.00        0

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
4                     0
...                 ...
1048570               0
1048571               0
1048572               0
1048573               0
1048574               0

[1029153 rows x 11 columns]

DataFrame after removing outliers (IQR method):
         step      type     amount     nameOrig  oldbalanceOrg  \
0           1   PAYMENT    9839.64  C1231006815      170136.00
1           1   PAYMENT    1864.28  C1666544295       21249.00
2           1  TRANSFER     181.00  C1305486145         181.00
3           1  CASH_OUT     181.00   C840083671         181.00
4           1   PAYMENT   11668.14  C2048537720       41554.00
...       ...       ...        ...          ...            ...
1048570    95  CASH_OUT  132557.35  C1179511630      479803.00
1048571    95   PAYMENT    9917.36  C1956161225       90545.00
1048572    95   PAYMENT   14140.05  C2037964975       20545.00
1048573    95   PAYMENT   10020.05  C1633237354       90605.00
1048574    95   PAYMENT   11450.03  C1264356443       80584.95

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
...                 ...          ...             ...             ...      ...
1048570       347245.65   C435674507       484329.37       616886.72        0
1048571        80627.64   M668364942            0.00            0.00        0
1048572         6404.95  M1355182933            0.00            0.00        0
1048573        80584.95  M1964992463            0.00            0.00        0
1048574        69134.92   M677577406            0.00            0.00        0

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
4                     0
...                 ...
1048570               0
1048571               0
1048572               0
1048573               0
1048574               0

[995487 rows x 11 columns]
```

(iii) Multi-collinearity

In [4]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Sample DataFrame Creation
csv_file_path = 'Downloads/Fraud.csv'

# Step 2: Load specific columns from the CSV file
# Replace 'column1', 'column2', etc. with the actual column names you want to include
df = pd.read_csv(csv_file_path, usecols=['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest'])

# Optional: Display the first few rows of the DataFrame to verify
print("DataFrame head:\n", df.head())

# Step 3: Calculate the correlation matrix
correlation_matrix = df.corr()

# Print the correlation matrix
print("Correlation Matrix:\n", correlation_matrix)

# Optional: Visualize the correlation matrix
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm', square=True)
plt.title('Correlation Matrix Heatmap')
plt.show()

# Create a DataFrame for VIF calculation (drop non-numeric or categorical columns)
df_vif = df[['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']]

# Calculate VIF for each feature
vif = pd.DataFrame()
vif['feature'] = df_vif.columns
vif['VIF'] = [variance_inflation_factor(df_vif.values, i) for i in range(df_vif.shape[1])]

print("Variance Inflation Factor:\n", vif)
```

```
DataFrame head:
      amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  newbalanceDest
0   9839.64       170136.0       160296.36             0.0             0.0
1   1864.28        21249.0        19384.72             0.0             0.0
2    181.00          181.0            0.00             0.0             0.0
3    181.00          181.0            0.00         21182.0             0.0
4  11668.14        41554.0        29885.86             0.0             0.0
Correlation Matrix:
                   amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
amount          1.000000       0.004864       -0.001133        0.215558
oldbalanceOrg   0.004864       1.000000        0.999047        0.093305
newbalanceOrig -0.001133       0.999047        1.000000        0.095182
oldbalanceDest  0.215558       0.093305        0.095182        1.000000
newbalanceDest  0.311936       0.064049        0.063725        0.978403

                newbalanceDest
amount                0.311936
oldbalanceOrg         0.064049
newbalanceOrig        0.063725
oldbalanceDest        0.978403
newbalanceDest        1.000000
```




```
Variance Inflation Factor:
           feature         VIF
0          amount    1.642946
1   oldbalanceOrg  600.759819
2  newbalanceOrig  602.459862
3  oldbalanceDest   37.432983
4  newbalanceDest   40.246530
```

3. How did you select variables to be included in the model?

In [ ]:
```python
# Required Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Load the Dataset (Assume CSV format)
# The dataset contains columns as described in the user's message.
df = pd.read_csv('Downloads/Fraud.csv')

# Step 2: Basic Exploration
print("Dataset Overview:")
print(df.head())

print("\nDataset Info:")
print(df.info())

print("\nSummary Statistics:")
print(df.describe())

# Step 3: Handle Missing Data
# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())

# Step 4: Data Preprocessing
# Convert categorical columns (nameOrig, nameDest, type) to numeric using Label Encoding
le = LabelEncoder()
df['nameOrig'] = le.fit_transform(df['nameOrig'])
df['nameDest'] = le.fit_transform(df['nameDest'])
df['type'] = le.fit_transform(df['type'])

# Step 5: Feature Scaling
# Scaling numerical features (amount, oldbalanceOrg, newbalanceOrig, oldbalanceDest, newbalanceDest)
scaler = StandardScaler()
df[['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']] = scaler.fit_transform(
    df[['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']]
)

# Step 6: Splitting the Data into Train/Test
X = df.drop(columns=['isFraud'])
y = df['isFraud']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 7: Model Building - Using RandomForestClassifier for Fraud Detection
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Step 8: Predictions and Model Evaluation
y_pred = clf.predict(X_test)

# Evaluate the Model
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Step 9: Feature Importance
# Displaying the importance of features in fraud prediction
importance = pd.Series(clf.feature_importances_, index=X.columns)
importance.nlargest(10).plot(kind='barh', title="Feature Importance in Fraud Detection")
plt.show()
```

4. Demonstrate the performance of the model by using best set of tools.

In [ ]:
```python
import pandas as pd

# Step 1: Simulated Transaction Data
data = pd.read_csv('Downloads/Fraud.csv')

# Step 2: Convert the data into a pandas DataFrame
df = pd.DataFrame(data)

# Step 3: Function to print the details of each transaction
def print_transaction_details(df):
    print("Transaction Details:")
    for idx, row in df.iterrows():
        print(f"\nTransaction {idx + 1}:")
        print(f"  Time Step (in hours): {row['step']}")
        print(f"  Transaction Type: {row['type']}")
        print(f"  Amount: {row['amount']}")
        print(f"  Customer (Orig): {row['nameOrig']}")
        print(f"    Balance Before: {row['oldbalanceOrg']}")
        print(f"    Balance After: {row['newbalanceOrig']}")
        print(f"  Recipient (Dest): {row['nameDest']}")
        print(f"    Balance Before: {row['oldbalanceDest']}")
        print(f"    Balance After: {row['newbalanceDest']}")
        print(f"  Fraudulent Transaction: {'Yes' if row['isFraud'] == 1 else 'No'}")

# Step 4: Analyze fraudulent transactions
def analyze_fraud(df):
    print("\nFraudulent Transactions Summary:")
    frauds = df[df['isFraud'] == 1]
    if not frauds.empty:
        for idx, row in frauds.iterrows():
            print(f"\nFraudulent Transaction {idx + 1}:")
            print(f"  Type: {row['type']}")
            print(f"  Amount: {row['amount']}")
            print(f"  Customer (Orig): {row['nameOrig']}")
            print(f"  Recipient (Dest): {row['nameDest']}")
    else:
        print("  No fraudulent transactions found.")

# Step 5: Main Program Execution
if __name__ == "__main__":
    # Print all transaction details
    print_transaction_details(df)

    # Analyze fraudulent transactions
    analyze_fraud(df)
```

5. What are the key factors that predict fraudulent customer?

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

# Step 1: Load the dataset
# Assuming the data is in a CSV file called 'transactions.csv'
data = pd.read_csv('Downloads/Fraud.csv')

# Step 2: Display the first few rows to understand the structure
print(data.head())

# Step 3: Check for missing values
print(data.isnull().sum())

# Step 4: Add a new column "balanceDiffOrig" to check if the transaction amount matches with balance changes
data['balanceDiffOrig'] = data['oldbalanceOrg'] - data['newbalanceOrig']
data['balanceDiffDest'] = data['newbalanceDest'] - data['oldbalanceDest']

# Step 5: Basic rule-based fraud detection (you can customize these rules):
def detect_fraud(row):
    # Check for mismatch between transaction amount and balance changes
    if row['balanceDiffOrig'] != row['amount'] or row['balanceDiffDest'] != row['amount']:
        return 1  # Mark as fraud
    return 0  # Not fraud

# Apply the fraud detection logic to the dataset
data['detectedFraud'] = data.apply(detect_fraud, axis=1)

# Step 6: Compare with the "isFraud" label in the dataset
correct_detections = (data['detectedFraud'] == data['isFraud']).sum()
total_transactions = len(data)

# Calculate the accuracy of the rule-based detection
accuracy = correct_detections / total_transactions
print(f"Accuracy of rule-based fraud detection: {accuracy * 100:.2f}%")

# Step 7: Summary statistics for transactions
print(data.describe())

# Optionally, save the processed data to a new file
data.to_csv('processed_transactions.csv', index=False)
```

6. Do these factors make sense? If yes, How? If not, How not?

The factors mentioned for predicting fraudulent customers, and the ones used in the code, make sense, but their effectiveness depends on the context, dataset, and overall fraud detection strategy. Let's break down why these factors make sense and also where there could be limitations: 1. Transaction Amount and Balance Differences Does it make sense?: Yes, the difference between the initial balance (oldbalanceOrg) and the post-transaction balance (newbalanceOrig) should typically match the transaction amount. Similarly, for the recipient, the difference between the oldbalanceDest and newbalanceDest should equal the transaction amount. If these differences don't align, it suggests suspicious behavior such as incomplete or manipulated transactions. Limitations: This rule works for simple transactional systems, but it might not capture more sophisticated fraud scenarios like: Internal transfers between accounts controlled by the same fraudster. Partial or fee-related deductions: In some systems, fees or charges might create small discrepancies. Fraudulent chargebacks: If money is charged back after the fraud occurs, balances might look "normal." 2. Timing of Transactions (step = hours) Does it make sense?: Yes, fraudsters often conduct transactions at unusual times, such as during non-business hours, or rapidly within a short period (high transaction velocity). Monitoring the frequency and timing of transactions can help flag abnormal behaviors. Limitations: Not all out-of-hours transactions are fraudulent; global customers may make transactions from different time zones. Therefore, using timing alone without additional context can result in false positives. 3. Origin and Destination Balances Does it make sense?: Yes, observing the oldbalanceOrig, newbalanceOrig, oldbalanceDest, and newbalanceDest helps validate the flow of funds between accounts. If the amount of the transaction isn't accurately reflected in these balances, it could indicate fraud or system anomalies. Limitations: Some legitimate transactions could involve partial payments or intermediary systems where balances fluctuate differently. Fraudulent accounts may still show correct balance changes, especially if the fraud involves a coordinated scheme between multiple accounts (money laundering). 4. Fraud Label (isFraud) Does it make sense?: Yes, this is the ground truth that labels whether a transaction is fraudulent or not. It's essential to use this data during training and evaluation in machine learning models. Comparing the system's detection (detectedFraud) with isFraud provides insights into how well the rules work. Limitations: Class imbalance: In many real-world datasets, fraud cases represent a small fraction of the total transactions. If fraud cases are rare, the model could tend to classify everything as non-fraud, leading to poor fraud detection. Incorrect labels: If isFraud is incorrectly labeled in the dataset, the model could learn wrong patterns, leading to poor predictions. 5. Other Potential Fraud Patterns (not included in this model) Geographical anomalies: Transactions happening in geographically distant places within short periods, which could indicate identity theft or account compromise. Behavioral anomalies: A customer exhibiting behavior inconsistent with their historical patterns (e.g., large purchases when historically they've only made small ones). Device and IP addresses: Transactions initiated from new or suspicious devices and networks may increase fraud likelihood, which isn't currently modeled here. How These Factors Make Sense: Transaction Matching: Balancing the debits and credits between accounts is a logical and common-sense check for spotting irregularities. Basic Anomaly Detection: Fraudsters often exploit unusual times or high transaction volumes, making time-based factors valuable. Simple Rule-Based Systems: These factors fit well in simple, rule-based systems for identifying clear, unambiguous fraudulent behavior. Where These Factors Might Be Limited: Sophisticated Fraud: This approach may not detect sophisticated fraud like account takeovers, social engineering, or coordinated attacks. Fraudsters can create transactions that look perfectly legitimate, especially if they control both ends (origin and destination) of the transaction. Dynamic and Evolving Fraud Patterns: Fraudsters continuously adapt, so static rules based on transaction amounts and balances alone may not catch newer types of fraud. False Positives: Legitimate behavior can sometimes trigger these rules, especially for large transactions, new devices, or cross-border purchases. How to Improve: Machine Learning: Incorporate more complex models to learn from the entire dataset, considering a wider variety of features and patterns. Additional Features: Introduce new features like device fingerprinting, transaction context (type of goods/services), IP data, and velocity of money transfers across accounts. Real-time Behavioral Analysis: Continuously monitor customer behavior to detect sudden deviations, combining time, location, and transaction data. Conclusion These factors do make sense and provide a good starting point for basic fraud detection. However, they may be limited in detecting more advanced or coordinated fraud. Extending the model with additional features, contextual data, and potentially machine learning can improve fraud detection accuracy and robustness.

7. What kind of prevention should be adopted while company update its infrastructure?

When a company updates its infrastructure, the following preventive measures should be adopted to ensure security, stability, and smooth transition: 1. Risk Assessment and Planning Identify potential risks. Develop a detailed update plan with contingency steps. 2. Security Protocols Patch and update systems before the upgrade. Ensure data encryption and restrict access to critical systems. 3. Backups Perform comprehensive backups of data and configurations. Verify backup integrity before proceeding. 4. Minimize Downtime Schedule updates during low-traffic periods. Use redundant systems or phased rollouts to reduce disruptions. 5. Testing Test updates in a staging environment. Conduct post-update checks for performance, security, and functionality. 6. Monitoring and Incident Response Enable real-time monitoring to detect issues. Have an incident response plan ready to address any issues immediately. By adopting these short-term preventive strategies, companies can avoid security breaches, downtime, and data loss during infrastructure updates.

8. Assuming these actions have been implemented, how would you determine if they work?

1. Monitor Key Performance Indicators (KPIs) Fraud Detection Rate: Measure the percentage of fraudulent transactions identified. False Positive Rate: Track legitimate transactions incorrectly flagged as fraudulent. Time to Detect Fraud: Measure the average time taken to identify fraudulent activities. 2. Conduct Periodic Audits and Assessments Security Audits: Regularly evaluate adherence to security policies. Vulnerability Assessments: Identify existing weaknesses through testing. 3. Analyze Incident Response Outcomes Incident Reports: Review outcomes of fraud incidents post-implementation. Response Time: Measure the time taken to respond to detected fraud. 4. Gather Feedback from Stakeholders Employee Feedback: Collect insights from staff using the new systems. Customer Surveys: Understand customer experiences regarding security measures. 5. Conduct Comparative Analysis Before-and-After Comparisons: Analyze fraud rates pre- and post-implementation. Benchmarking: Compare metrics with industry standards or peers. 6. Utilize Advanced Analytics and Reporting Tools Data Analytics: Evaluate transaction data for patterns indicating effectiveness. Dashboard Reporting: Use real-time dashboards for insights into fraud metrics. 7. Track Compliance with Policies and Procedures Policy Adherence: Monitor compliance with internal fraud prevention policies. Training Participation: Correlate employee training participation with fraud incidents. 8. Assess System Performance System Availability: Monitor uptime and reliability of security measures. Scalability: Ensure measures can handle increased transaction volumes. 9. Review Legal and Regulatory Compliance Compliance Audits: Ensure alignment with legal and regulatory requirements. Incident Reports: Track reports made to regulators and assess trends.

In [ ]: