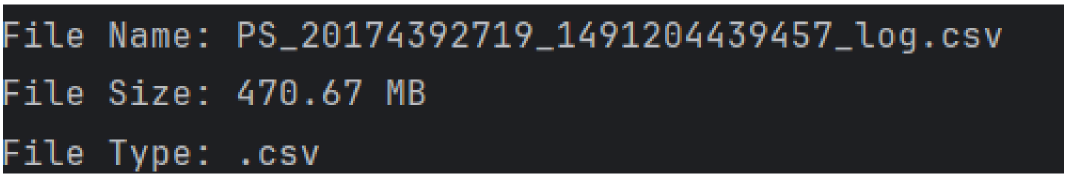
PipeLine

שם: שאדי שמשום

תז: 214511172

1) System stage:



```
File Name: PS_20174392719_1491204439457_log.csv
File Size: 470.67 MB
File Type: .csv
```

Now for the web information we will note the data source is from Kaggle, link:
https://www.kaggle.com/datasets/ealaxi/paysim1/data

The license is : CC BY-SA 4.0

Tags are: Finance and Crime

Time: written and lastly updated 8 years ago

Version control: we will work with git and IntelliJ

Now lets provide a brief explanation about the dataset:

The dataset used in this project is a synthetic financial transactions dataset generated by the PaySim simulator.
Due to the private nature of real financial transactions, there is very limited public access to real-world datasets for fraud detection research. PaySim solves this problem by creating realistic simulated mobile money transactions based on real aggregated logs from an African mobile money service.

The data includes five transaction types CASH-IN, CASH-OUT, DEBIT, PAYMENT, and TRANSFER and covers 744 time steps (representing 30 days, 1 hour per step). It contains millions of records and simulates both normal and fraudulent activities to evaluate fraud detection methods.

Key points:

Fields: Step, Type, Amount, Sender, Receiver, Balances before & after, Fraud labels.

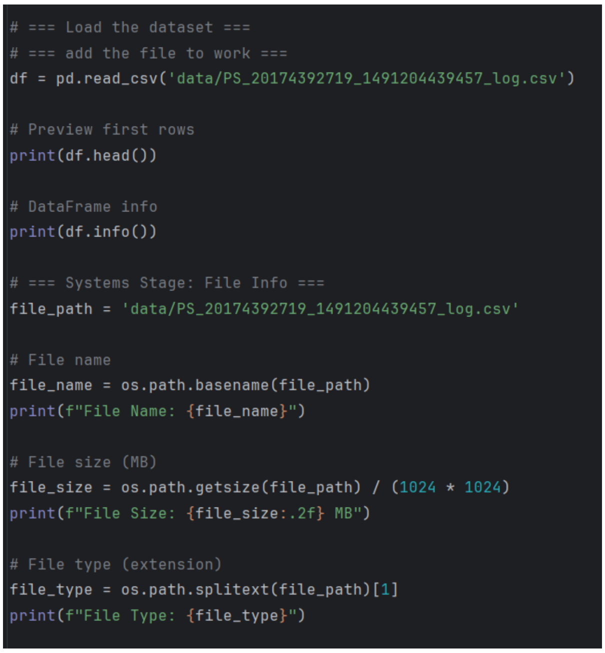
Important note: Balances should not be used for fraud detection as they are affected by fraud cancellation.

Source: The simulation is based on real-world mobile money logs and is scaled to about 1/4 of the original dataset.

Purpose: To help researchers test fraud detection algorithms with realistic but safe synthetic data.

Reference:
Lopez-Rojas et al., *“PaySim: A financial mobile money simulator for fraud detection,”* EMSS, 2016.

Code:



```
# === Load the dataset ===
# === add the file to work ===
df = pd.read_csv('data/PS_20174392719_1491204439457_log.csv')

# Preview first rows
print(df.head())

# DataFrame info
print(df.info())

# === Systems Stage: File Info ===
file_path = 'data/PS_20174392719_1491204439457_log.csv'

# File name
file_name = os.path.basename(file_path)
print(f"File Name: {file_name}")

# File size (MB)
file_size = os.path.getsize(file_path) / (1024 * 1024)
print(f"File Size: {file_size:.2f} MB")

# File type (extension)
file_type = os.path.splitext(file_path)[1]
print(f"File Type: {file_type}")
```

2) Data Governance:

Who created it?

Researchers: E. A. Lopez-Rojas, A. Elmir, and S. Axelsson

Part of a research team in Sweden, cited in the paper: "PaySim: A financial mobile money simulator for fraud detection" (EMSS, 2016).

When?

Created during research published in 2016.

Where?

Developed at Blekinge Institute of Technology, Sweden.

Original financial logs from a multinational mobile money provider in an African country, simulated for public research use.

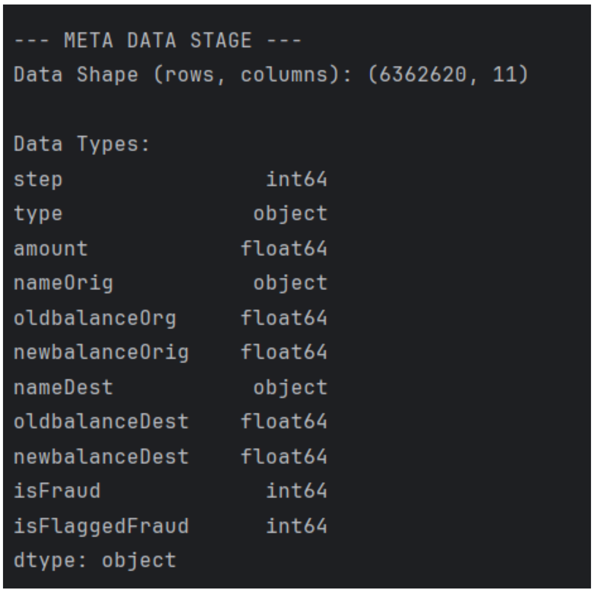
Why?

To provide an accessible dataset for testing fraud detection algorithms.

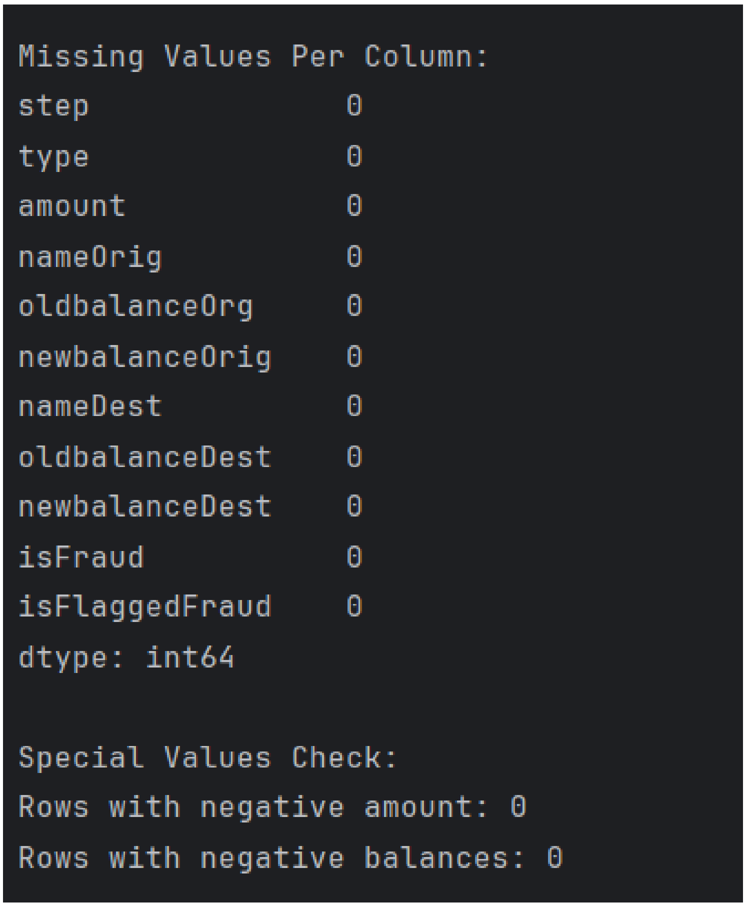
Real data is private → so they simulated realistic transactions with fraud to support safe, open research.

3) Meta data:

First we will show the output we got from the code and then explain the results:



```
--- META DATA STAGE ---
Data Shape (rows, columns): (6362620, 11)

Data Types:
step                int64
type               object
amount            float64
nameOrig           object
oldbalanceOrg     float64
newbalanceOrig    float64
nameDest           object
oldbalanceDest    float64
newbalanceDest    float64
isFraud             int64
isFlaggedFraud      int64
dtype: object
```



```
Missing Values Per Column:
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64

Special Values Check:
Rows with negative amount: 0
Rows with negative balances: 0
```

Data Size

Shape: The dataset contains 6,362,620 rows and 11 columns.

Data Types

The columns include:

Numeric: step (int), amount (float), oldbalanceOrg (float), newbalanceOrig (float), oldbalanceDest (float), newbalanceDest (float), isFraud (int), isFlaggedFraud (int).

Categorical: type, nameOrig, nameDest.

This shows a mix of transaction details, amounts, balances, IDs, and fraud labels.

Missing Data

No missing values detected in any column. All rows have complete information.

Special Values

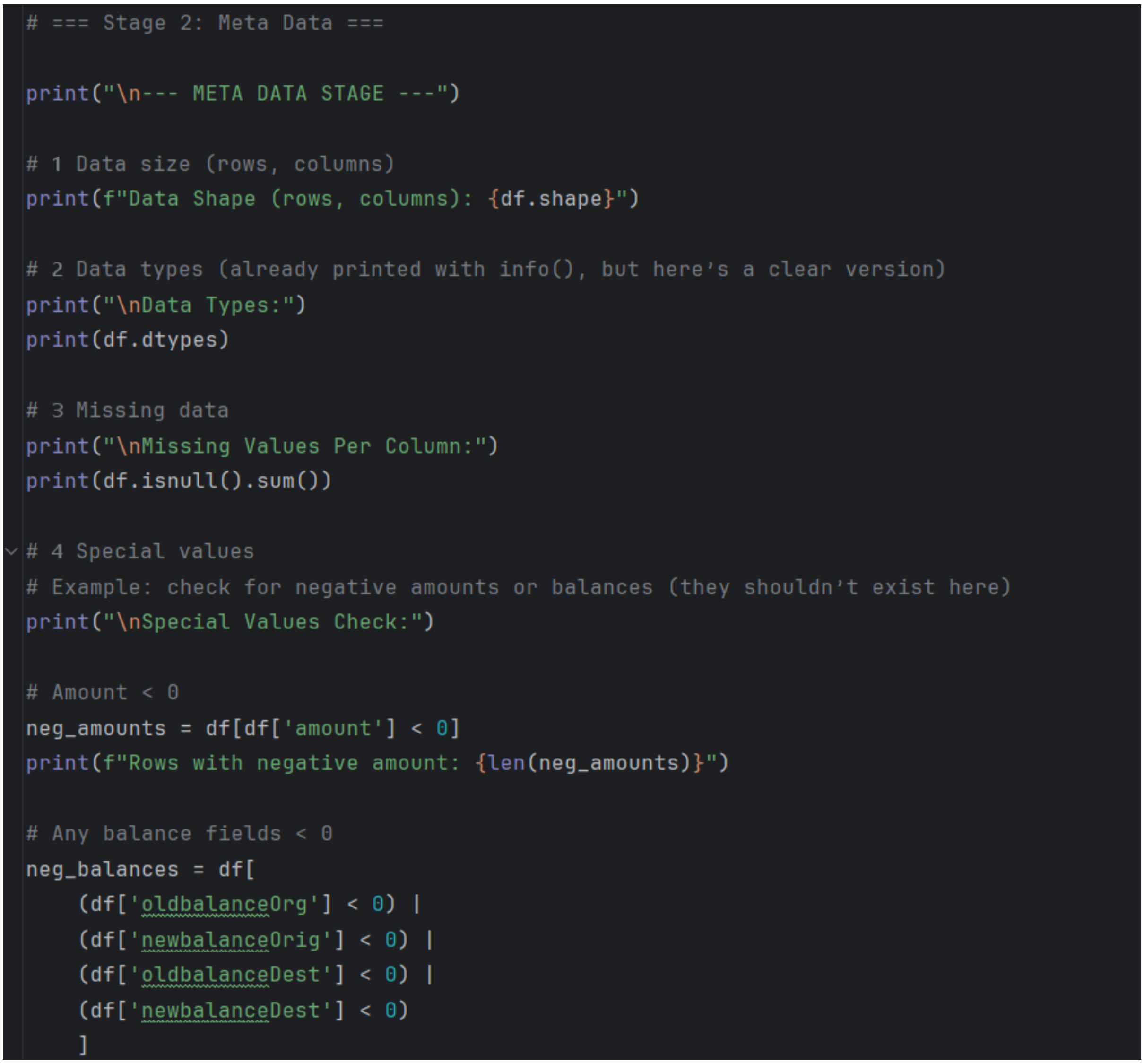
No negative values found for amount or balance columns. All values are logically consistent for financial transactions.

This confirms the dataset's integrity for further analysis, that means no data cleaning needed for nulls or negative outliers at this stage, so we can simply continuo the next one.

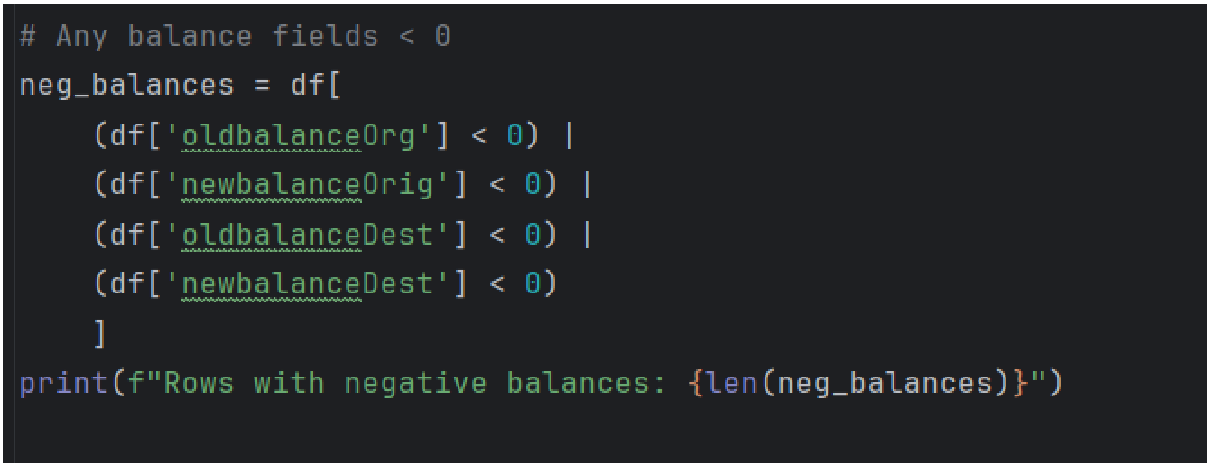
Now i will look at every column name and explain if it explain its data well or it is suspicious:

Column	**Meaning**	**Does it explain well what it contains?**
step	Time step (1 hour increments)	Good
type	Transaction type (PAYMENT, TRANSFER, etc.)	Clear
amount	Transaction amount	Clear
nameOrig	Origin account ID	clear
oldbalanceOrg	Sender's balance before transaction	Clear
newbalanceOrig	Sender's balance after transaction	Clear
nameDest	Destination account ID	nameDest is fine
oldbalanceDest	Receiver's balance before transaction	Clear
newbalanceDest	Receiver's balance after transaction	Clear
isFraud	True fraud label (1 = fraud)	Clear
isFlaggedFraud	Was flagged as suspicious (1 = flagged)	Clear

Code:



```
# === Stage 2: Meta Data ===

print("\n--- META DATA STAGE ---")

# 1 Data size (rows, columns)
print(f"Data Shape (rows, columns): {df.shape}")

# 2 Data types (already printed with info(), but here's a clear version)
print("\nData Types:")
print(df.dtypes)

# 3 Missing data
print("\nMissing Values Per Column:")
print(df.isnull().sum())

# 4 Special values
# Example: check for negative amounts or balances (they shouldn't exist here)
print("\nSpecial Values Check:")

# Amount < 0
neg_amounts = df[df['amount'] < 0]
print(f"Rows with negative amount: {len(neg_amounts)}")

# Any balance fields < 0
neg_balances = df[
    (df['oldbalanceOrg'] < 0) |
    (df['newbalanceOrig'] < 0) |
    (df['oldbalanceDest'] < 0) |
    (df['newbalanceDest'] < 0)
    ]
```

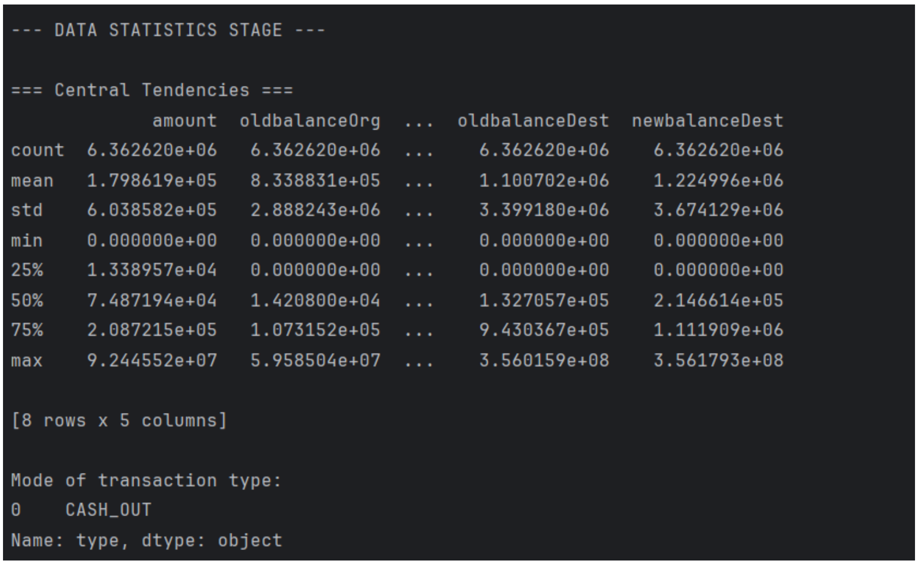


```
# Any balance fields < 0
neg_balances = df[
    (df['oldbalanceOrg'] < 0) |
    (df['newbalanceOrig'] < 0) |
    (df['oldbalanceDest'] < 0) |
    (df['newbalanceDest'] < 0)
    ]
```

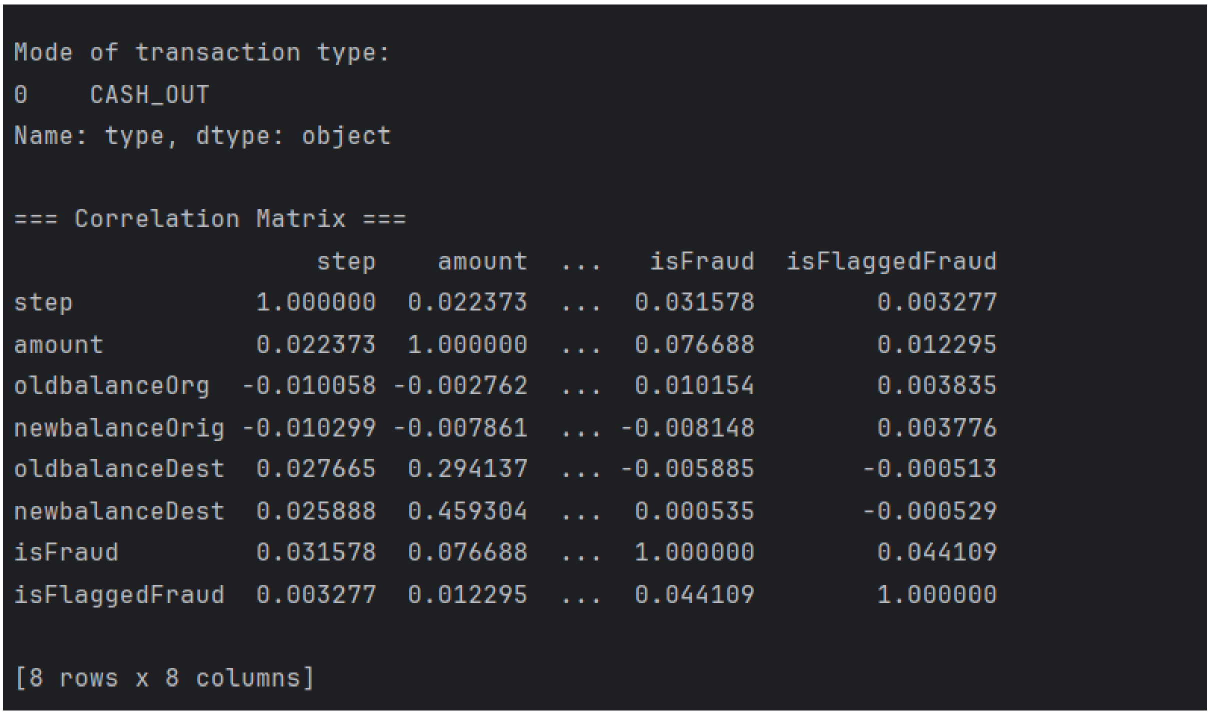
```
print(f"Rows with negative balances: {len(neg_balances)}")
```

4) Data Statistics:

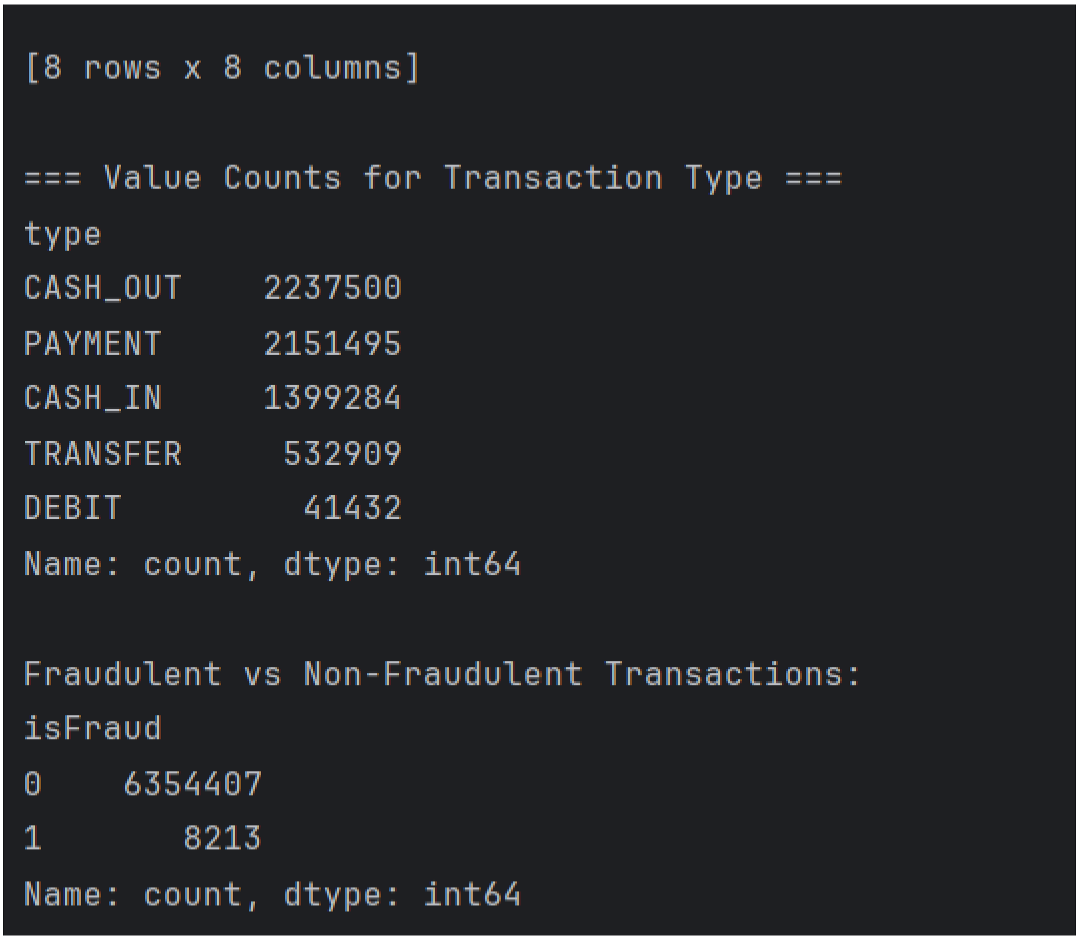
First we will show the statistics we got:



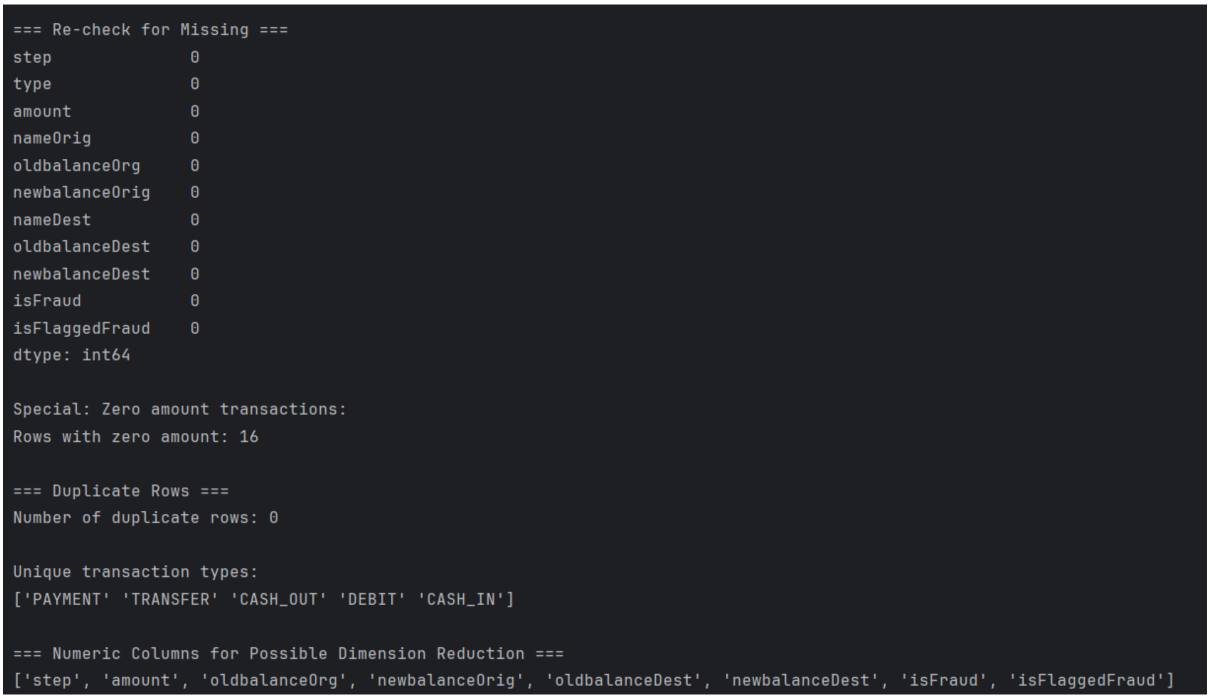
```
--- DATA STATISTICS STAGE ---

=== Central Tendencies ===
             amount  oldbalanceOrg  ...  oldbalanceDest  newbalanceDest
count  6.362620e+06   6.362620e+06  ...    6.362620e+06    6.362620e+06
mean   1.798619e+05   8.338831e+05  ...    1.100702e+06    1.224996e+06
std    6.038582e+05   2.888243e+06  ...    3.399180e+06    3.674129e+06
min    0.000000e+00   0.000000e+00  ...    0.000000e+00    0.000000e+00
25%    1.338957e+04   0.000000e+00  ...    0.000000e+00    0.000000e+00
50%    7.487194e+04   1.420800e+04  ...    1.327057e+05    2.146614e+05
75%    2.087215e+05   1.073152e+05  ...    9.430367e+05    1.111909e+06
max    9.244552e+07   5.958504e+07  ...    3.560159e+08    3.561793e+08

[8 rows x 5 columns]

Mode of transaction type:
0    CASH_OUT
Name: type, dtype: object
```



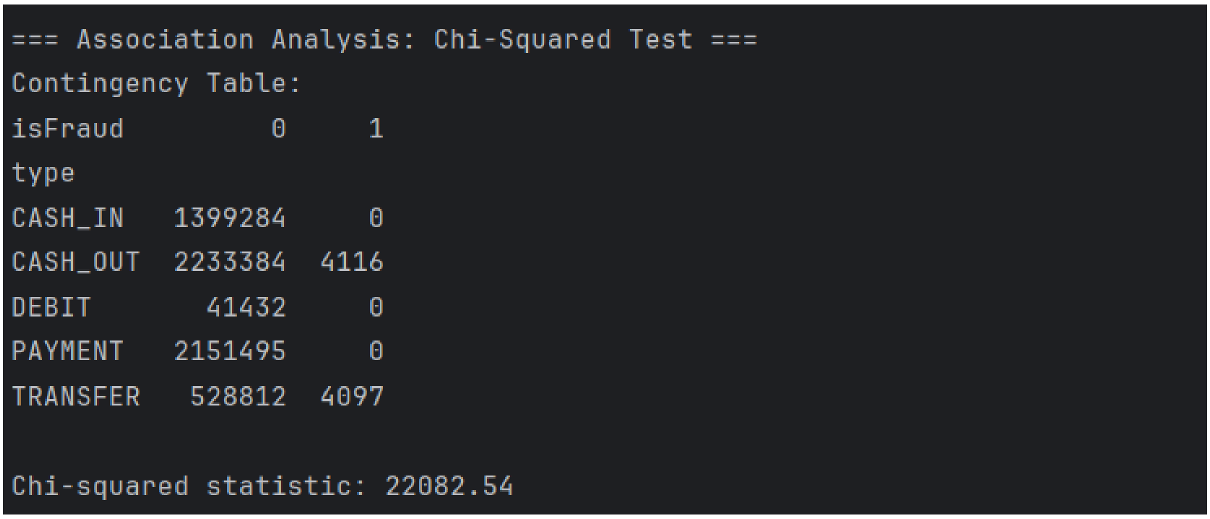
```

Mode of transaction type:
0    CASH_OUT
Name: type, dtype: object

=== Correlation Matrix ===
                    step    amount  ...   isFraud  isFlaggedFraud
step            1.000000  0.022373  ...  0.031578        0.003277
amount          0.022373  1.000000  ...  0.076688        0.012295
oldbalanceOrg  -0.010058 -0.002762  ...  0.010154        0.003835
newbalanceOrig -0.010299 -0.007861  ... -0.008148        0.003776
oldbalanceDest  0.027665  0.294137  ... -0.005885       -0.000513
newbalanceDest  0.025888  0.459304  ...  0.000535       -0.000529
isFraud         0.031578  0.076688  ...  1.000000        0.044109
isFlaggedFraud  0.003277  0.012295  ...  0.044109        1.000000

[8 rows x 8 columns]
```



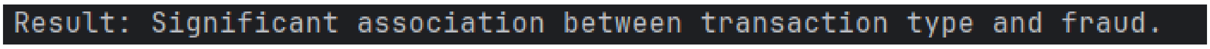
```
[8 rows x 8 columns]

=== Value Counts for Transaction Type ===
type
CASH_OUT    2237500
PAYMENT     2151495
CASH_IN     1399284
TRANSFER     532909
DEBIT         41432
Name: count, dtype: int64

Fraudulent vs Non-Fraudulent Transactions:
isFraud
0    6354407
1       8213
Name: count, dtype: int64
```



```
=== Re-check for Missing ===
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64

Special: Zero amount transactions:
Rows with zero amount: 16

=== Duplicate Rows ===
Number of duplicate rows: 0

Unique transaction types:
['PAYMENT' 'TRANSFER' 'CASH_OUT' 'DEBIT' 'CASH_IN']

=== Numeric Columns for Possible Dimension Reduction ===
['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFraud', 'isFlaggedFraud']
```



```
=== Association Analysis: Chi-Squared Test ===
Contingency Table:
isFraud          0     1
type
CASH_IN    1399284     0
CASH_OUT   2233384  4116
DEBIT        41432     0
PAYMENT    2151495     0
TRANSFER    528812  4097

Chi-squared statistic: 22082.54
```



```
Result: Significant association between transaction type and fraud.
```

Now we will explain the meaning of the results:

Central Tendencies

The dataset has wide transaction values:

Mean amount: ~1,798 currency units.

Max amount: ~92 million units.

Balances: Large ranges from 0 to hundreds of millions.

The most common transaction type (mode) is CASH_OUT.

Correlation & Association

The correlation matrix shows weak correlations overall (i used Pearson correlation):

amount has a slight positive correlation (~0.07) with isFraud.

step (time) has almost no correlation with fraud.

Balances show low correlations too.

This suggests that amount may be a useful feature for fraud detection.

So in order to fully understand that:

I computed the correlation matrix for all numerical features and visualized it using a heatmap, helping identify multicollinearity and potential interactions.

I analyzed associations between categorical variables and fraud labels. Specifically, I performed a Chi-Square test to measure the statistical dependency between type (transaction type) and isFraud.

Correlation results explained:

Observed Key Correlations:

Feature Pair	Correlation	Interpretation
newbalanceDest & amount	0.4593	Moderate positive relationship. When amount is high, the balance at destination tends to increase , logical for legitimate transfers.
oldbalanceDest & amount	0.2941	Weaker but still noticeable relationship , larger amounts often move into accounts with higher balances.
amount & isFraud	0.0766	Small positive correlation , fraud is slightly more likely on higher transactions, but not strongly so.

Feature Pair	Correlation	Interpretation
isFlaggedFraud & isFraud	0.0441	Weak correlation , suggests flagging is not always aligned with actual fraud.
Other values	Near 0	Most features are not strongly linearly correlated, which is good for model diversity.

Conclusion:

The low correlations with isFraud indicate that fraud can’t be predicted using a single feature alone , it likely depends on combinations of features, which supports the need for multivariate models like logistic regression.

The matrix also confirms no multicollinearity (extremely high correlation between independent features), meaning we can safely use them in modeling.

Association Analysis: Chi-Squared Test results

To examine the relationship between the type of transaction and fraud occurrence, we conducted a Chi-Squared Test for Independence.

Contingency Table Summary:

We created a table showing how many fraudulent and non-fraudulent transactions occurred for each transaction type:

Transaction Type	Fraudulent (1)	Non-Fraudulent (0)
CASH_IN	0	1,399,284
CASH_OUT	4,116	2,233,384
DEBIT	0	41,432
PAYMENT	0	2,151,495
TRANSFER	4,097	528,812

This shows that fraud only appears in the CASH_OUT and TRANSFER types, while all other types have zero fraud cases.

Test Output:

Chi-squared Statistic: 22,082.54

P-value: 0.0000

Conclusion:

Since the p-value is far below 0.05, we reject the null hypothesis and conclude that:

There is a significant association between transaction type and fraud.

This means the type of transaction is not independent of fraud occurrence , fraud is much more likely in specific transaction types, especially TRANSFER and CASH_OUT.

Why This Matters:

Understanding this association helps us:

Identify high-risk transaction types.

Improve feature selection for our machine learning model.

Justify additional monitoring or controls on certain transaction types.

Data Distribution

Transaction types:

CASH_OUT: 2.2 million

PAYMENT: 2.1 million

CASH_IN: 1.3 million

TRANSFER: ~530,000

DEBIT: ~41,000

Fraud labels:

Fraudulent transactions: 8,213

Non-fraudulent: ~6.35 million

Highly imbalanced dataset.

Missing Values & Special Values

No missing values found.

No negative values.

16 transactions have zero amounts, which may need further inspection.

Duplicates & Unique Values

No duplicate rows.

5 unique transaction types confirmed.

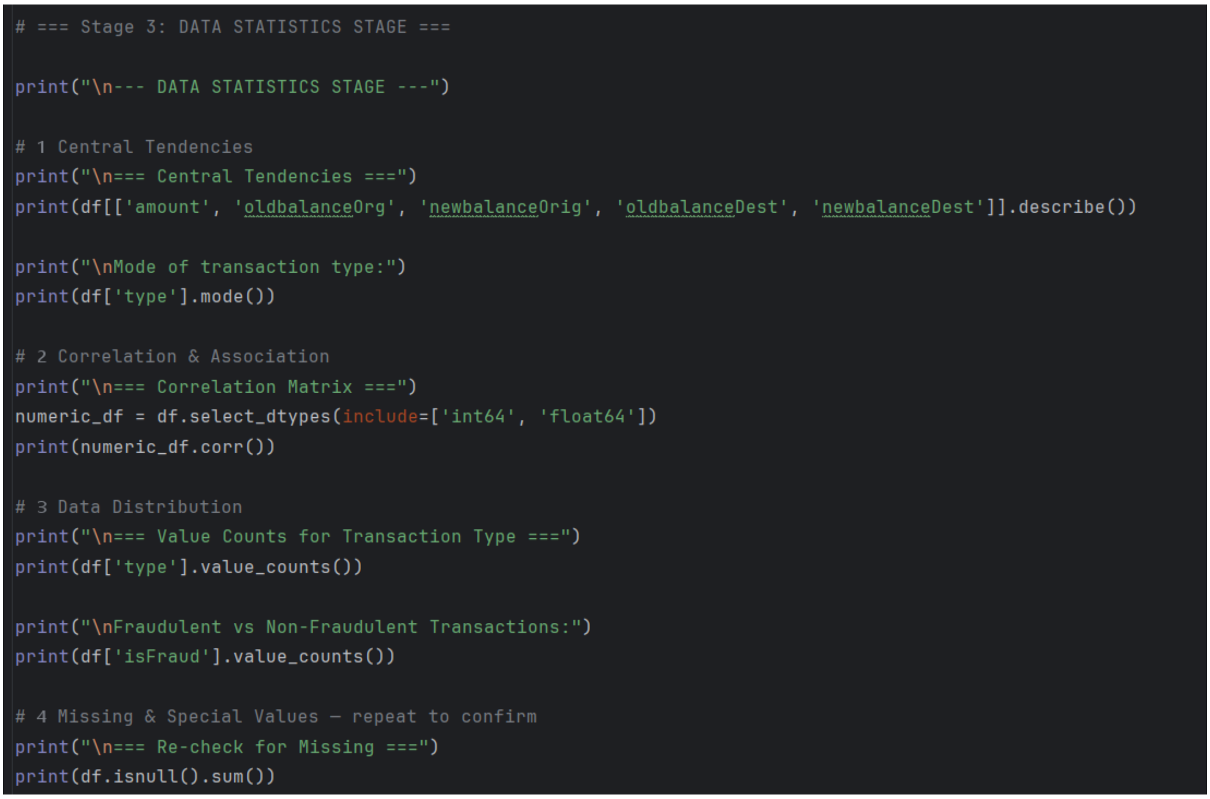
Dimension Reduction

The numeric columns suitable for dimensionality reduction (e.g., PCA) are:

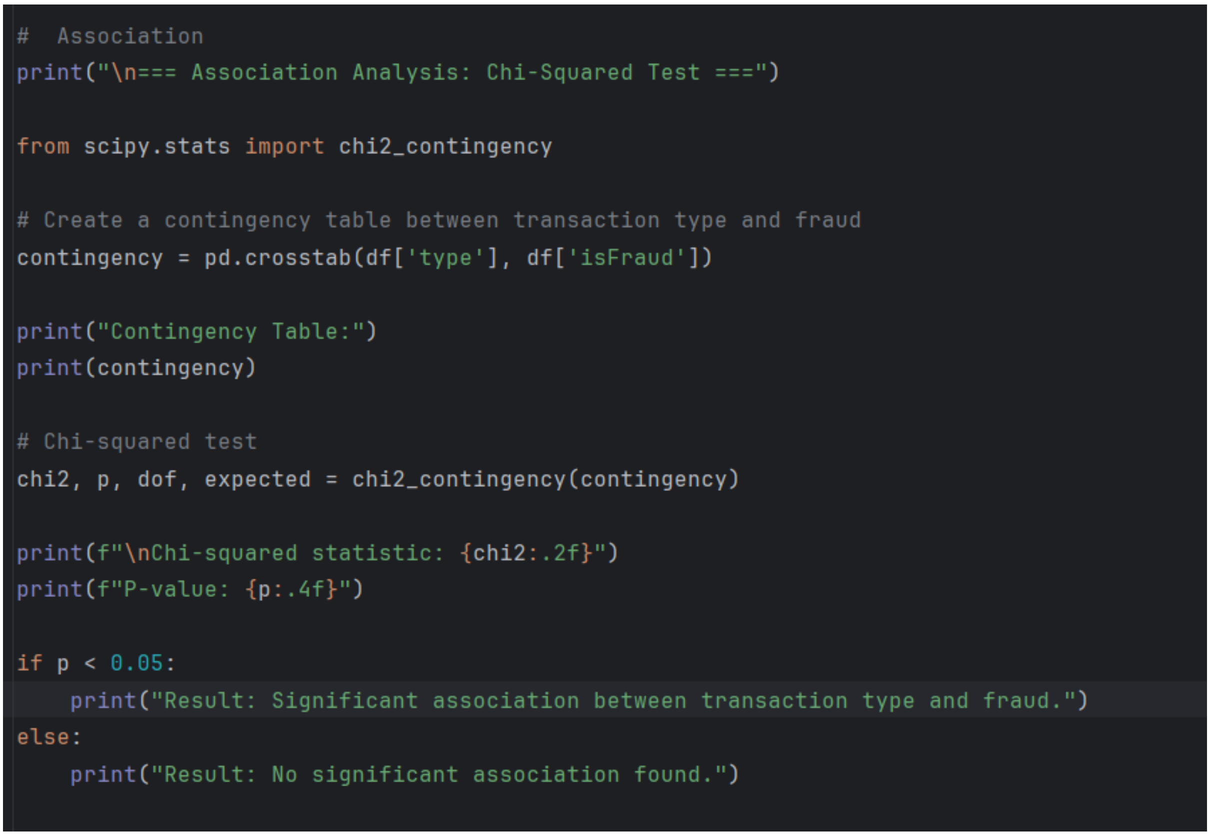
step, amount, oldbalanceOrg, newbalanceOrig, oldbalanceDest, newbalanceDest, isFraud, isFlaggedFraud.

The dataset is numerically consistent with no missing or duplicate rows, but is imbalanced (few frauds). Amount and transaction type are likely key predictors for fraud.

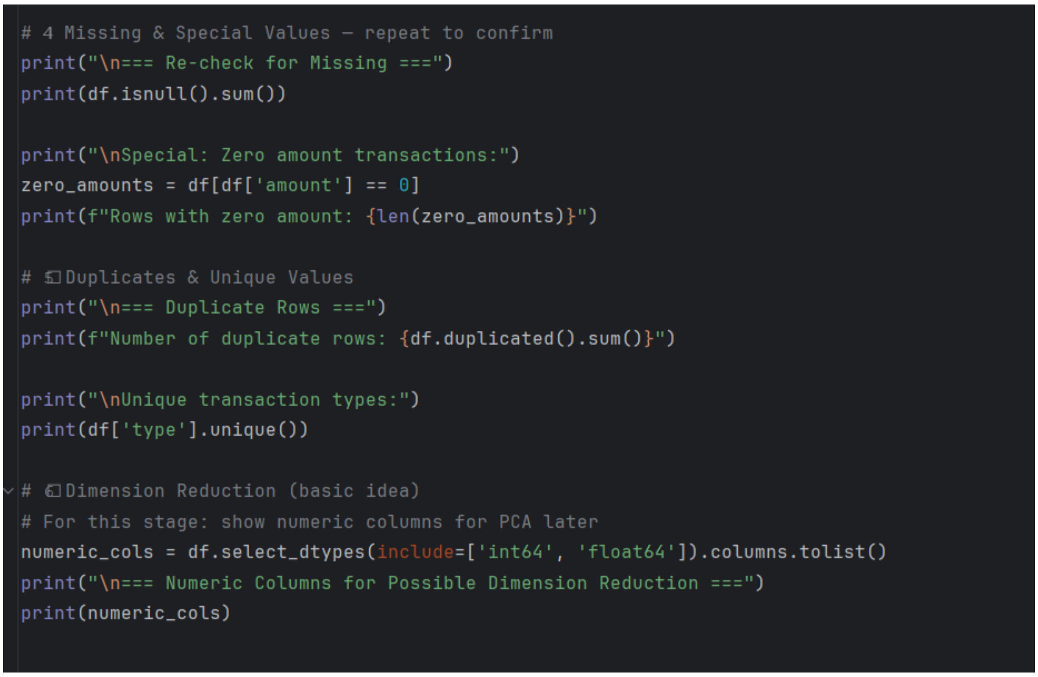
Code:



```python
# === Stage 3: DATA STATISTICS STAGE ===

print("\n--- DATA STATISTICS STAGE ---")

# 1 Central Tendencies
print("\n=== Central Tendencies ===")
print(df[['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']].describe())

print("\nMode of transaction type:")
print(df['type'].mode())

# 2 Correlation & Association
print("\n=== Correlation Matrix ===")
numeric_df = df.select_dtypes(include=['int64', 'float64'])
print(numeric_df.corr())

# 3 Data Distribution
print("\n=== Value Counts for Transaction Type ===")
print(df['type'].value_counts())

print("\nFraudulent vs Non-Fraudulent Transactions:")
print(df['isFraud'].value_counts())

# 4 Missing & Special Values — repeat to confirm
print("\n=== Re-check for Missing ===")
print(df.isnull().sum())
```



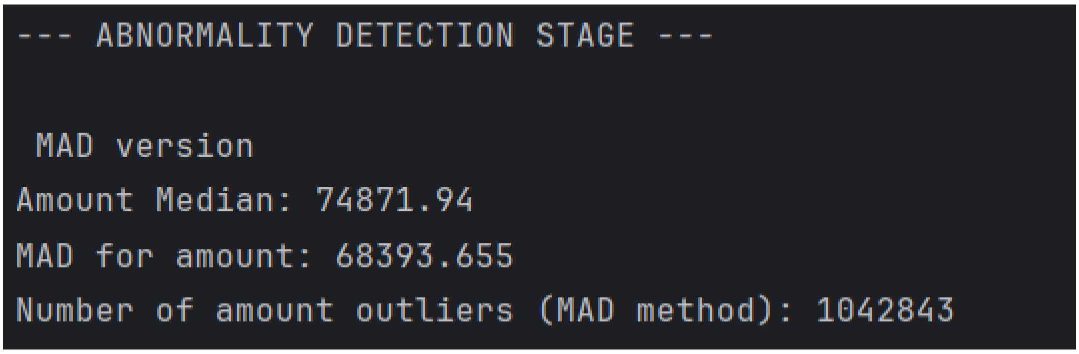
```python
#  Association
print("\n=== Association Analysis: Chi-Squared Test ===")

from scipy.stats import chi2_contingency

# Create a contingency table between transaction type and fraud
contingency = pd.crosstab(df['type'], df['isFraud'])

print("Contingency Table:")
print(contingency)

# Chi-squared test
chi2, p, dof, expected = chi2_contingency(contingency)

print(f"\nChi-squared statistic: {chi2:.2f}")
print(f"P-value: {p:.4f}")

if p < 0.05:
    print("Result: Significant association between transaction type and fraud.")
else:
    print("Result: No significant association found.")
```



```python
# 4 Missing & Special Values — repeat to confirm
print("\n=== Re-check for Missing ===")
print(df.isnull().sum())

print("\nSpecial: Zero amount transactions:")
zero_amounts = df[df['amount'] == 0]
print(f"Rows with zero amount: {len(zero_amounts)}")

# 5️⃣Duplicates & Unique Values
print("\n=== Duplicate Rows ===")
print(f"Number of duplicate rows: {df.duplicated().sum()}")

print("\nUnique transaction types:")
print(df['type'].unique())

# 6️⃣Dimension Reduction (basic idea)
# For this stage: show numeric columns for PCA later
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
print("\n=== Numeric Columns for Possible Dimension Reduction ===")
print(numeric_cols)
```

5) Abnormality detection:

First we added the MAD version of abnormal detection, here is the results we got:



```
--- ABNORMALITY DETECTION STAGE ---

 MAD version
Amount Median: 74871.94
MAD for amount: 68393.655
Number of amount outliers (MAD method): 1042843
```

Single Feature (MAD):

Outliers in transaction amount were detected using the Median Absolute Deviation (MAD) method, which is robust to skewed data.

This robust method uses the median and checks how far values deviate from it. It's less sensitive to extreme outliers and works well with skewed data.

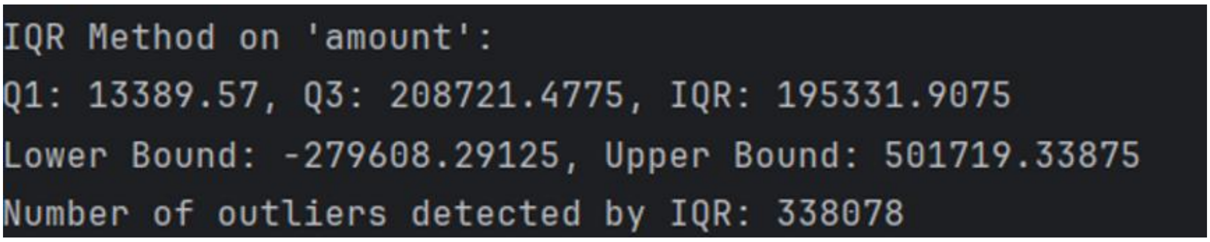
Median of amount: 74,872

MAD: 68,394

Threshold: Any transaction far beyond the median by more than 3×MAD

Outliers found: 1,042,843 transactions
This high count shows that a very large number of transactions deviate significantly from the typical amount, according to the median. This might reflect the wide variety of legitimate transactions and some potentially suspicious ones.

We then added another way to detect anomalies and using the IQR



```
IQR Method on 'amount':
Q1: 13389.57, Q3: 208721.4775, IQR: 195331.9075
Lower Bound: -279608.29125, Upper Bound: 501719.33875
Number of outliers detected by IQR: 338078
```

Interquartile Range (IQR) on the amount feature. This method identifies values that fall significantly outside the typical range of data (i.e., very small or very large amounts):

Q1 (25th percentile) and Q3 (75th percentile) were calculated.

The IQR = Q3 - Q1, and the bounds for normal values were:

Lower Bound = Q1 - 1.5 * IQR

Upper Bound = Q3 + 1.5 * IQR

Any transaction amounts outside this range were flagged as outliers.

This method complements the MAD technique, providing an additional perspective on potential fraud, especially helpful in skewed or non-normal data distributions.

This method focuses on the middle 50% of the data and treats values outside 1.5×IQR as outliers.

Q1 (25%): 13,390

Q3 (75%): 208,721

IQR: 195,332

Lower Bound: -279,608

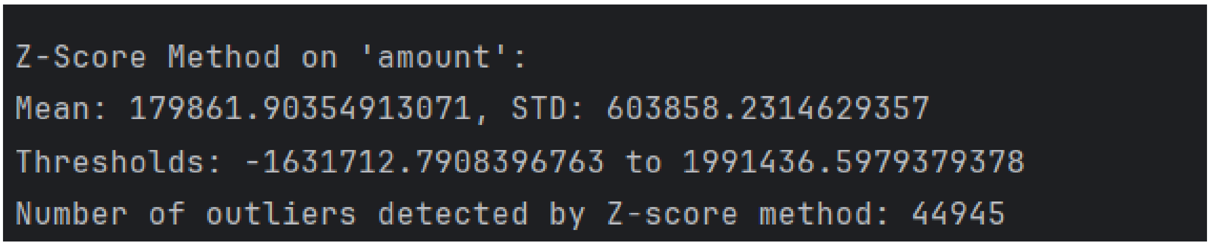
Upper Bound: 501,719

Outliers found: 338,078 transactions

The IQR method captures a smaller and more focused set of extreme values compared to MAD. Since negative amounts don't exist, lower outliers are rare, and the upper extreme transactions are the main contributors here.

Then we added a simpler version that does not work really weal because the mean is sensitive to outliers too but its better to be , it is called Z-Score:

Here is the results



```
Z-Score Method on 'amount':
Mean: 179861.90354913071, STD: 603858.2314629357
Thresholds: -1631712.7908396763 to 1991436.5979379378
Number of outliers detected by Z-score method: 44945
```

Z-Score Based Outlier Detection (Single Feature)

To further support outlier analysis, I used the Z-score method on the amount column. This technique assumes data is normally distributed and considers any value that lies more than 3 standard deviations (±3σ) from the mean to be an outlier:

Calculated the mean and standard deviation (std) of the amount column.

Set a threshold at mean ± 3*std.

Flagged all transactions outside this range.

This method is sensitive to large deviations and complements MAD and IQR by highlighting extreme values under the assumption of a bell-shaped distribution.

This method assumes data is normally distributed. It uses the mean and standard deviation to define a range for expected values.

Mean: 179,861

STD: 603,859

Thresholds:
Lower: -1,631,712
Upper: 1,991,437

Outliers found: 44,945 transactions
Because the standard deviation is very large, the range is wide, and only the most extreme transactions are detected. This method is more conservative, making it useful for catching only the rarest anomalies.

Now lets talk about multi feature methods:

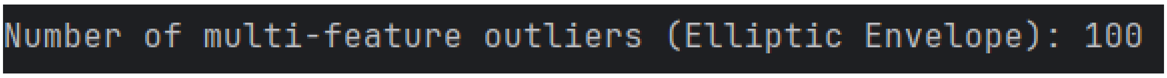
Multi Feature (Elliptic Envelope):

Multivariate outliers were detected using an Elliptic Envelope fitted on amount and oldbalanceOrg. This identifies transactions that are unusual when considering multiple features together.

This method fits an elliptical shape around the normal data distribution and flags points outside as outliers.

On a sample of 10,000 rows, 100 transactions were flagged as unusual multi-feature outliers.

Here is the result:



```
Number of multi-feature outliers (Elliptic Envelope): 100
```

We then added another way that is called isolation forest:

The Isolation Forest is a tree-based ensemble method that isolates anomalies by randomly selecting features and split values. Anomalies are easier to isolate and require fewer splits, which makes them detectable.

We used it on the features: amount, oldbalanceOrg, and newbalanceOrig.

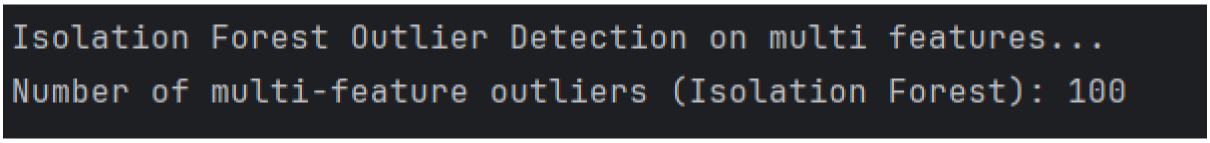
The algorithm automatically isolates points that behave differently from the majority.

It is especially effective for large datasets and non-Gaussian distributions.

Unlike Elliptic Envelope, it doesn’t assume the data follows a normal distribution.

It can handle complex and non-linear patterns, making it great for detecting fraud in high-dimensional spaces.

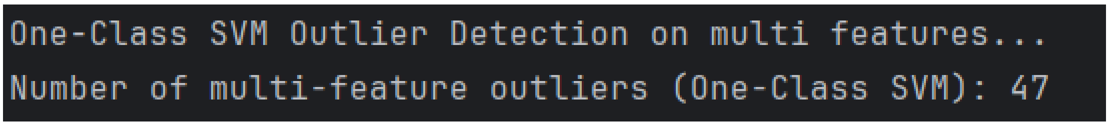
Here is the results:



```
Isolation Forest Outlier Detection on multi features...
Number of multi-feature outliers (Isolation Forest): 100
```

We also applied a One-Class SVM to detect multi-feature outliers. The One-Class SVM is well-suited for anomaly detection since it models the majority of the data as one class and flags observations that fall outside of this learned boundary as outliers. For this method, we used the features amount, oldbalanceOrg, and newbalanceOrig on a sample of 10,000 rows. The model was set with a parameter nu=0.01, which tells it that we expect about 1% of the observations to be outliers. The SVM then finds a decision boundary using a radial basis function (RBF) kernel. Transactions falling outside this boundary were flagged as anomalies. This method complements the Elliptic Envelope and Isolation Forest techniques by detecting non-linear patterns in the data without assuming any specific distribution.

Here is the results:



```
One-Class SVM Outlier Detection on multi features...
Number of multi-feature outliers (One-Class SVM): 47
```

Now lets explain the results:

Elliptic Envelope

The Elliptic Envelope assumes the data follows a Gaussian distribution and fits an ellipse around the "normal" region. Anything outside the ellipse is flagged as an outlier.

Features used: amount, oldbalanceOrg

Result: 100 outliers detected

Interpretation: These outliers deviate significantly from the mean in both dimensions assuming a normal distribution. It works well if the data is roughly elliptical (normal).

Isolation Forest

Isolation Forest works differently: it isolates observations by randomly selecting a feature and splitting it. Anomalies are easier to isolate and require fewer splits.

Features used: amount, oldbalanceOrg

Result: 100 outliers detected

Interpretation: These are data points that behave very differently from the majority, especially in terms of how easily they can be "separated" from the rest. It does not assume any distribution, which makes it robust.

One-Class SVM

One-Class SVM tries to find a boundary that encloses the "normal" data points using a non-linear kernel (RBF). Anything outside that boundary is treated as an outlier.

Features used: amount, oldbalanceOrg, newbalanceOrig

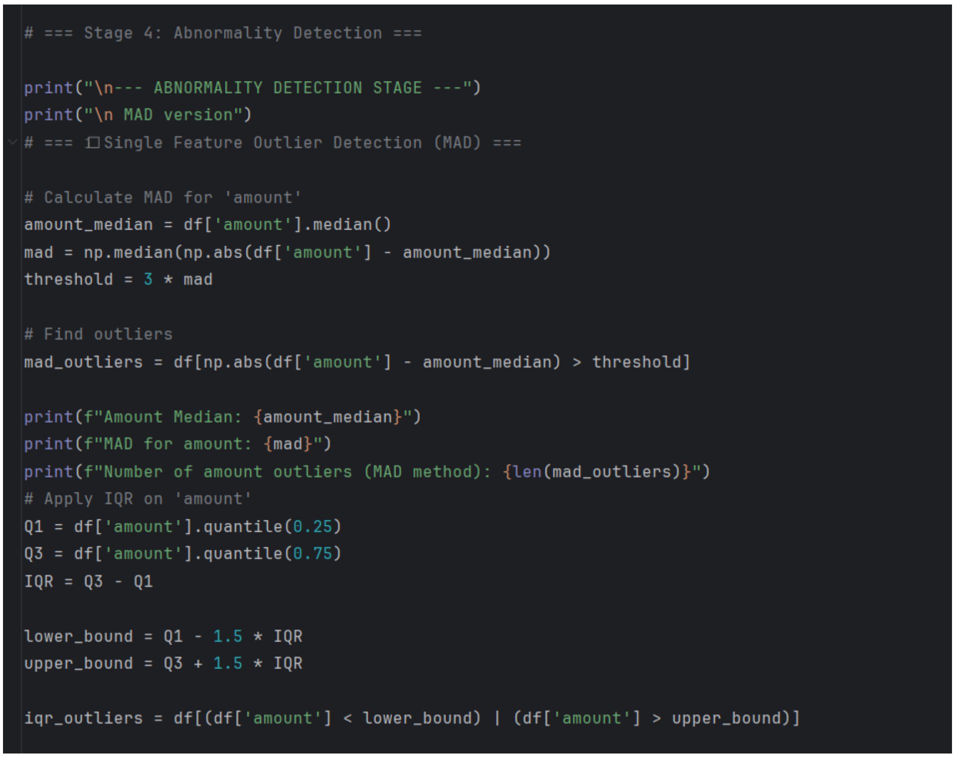
Result: 47 outliers detected

Interpretation: This method is stricter and more selective in what it considers abnormal, which is why it flagged fewer points. It’s great for detecting non-linear anomalies.

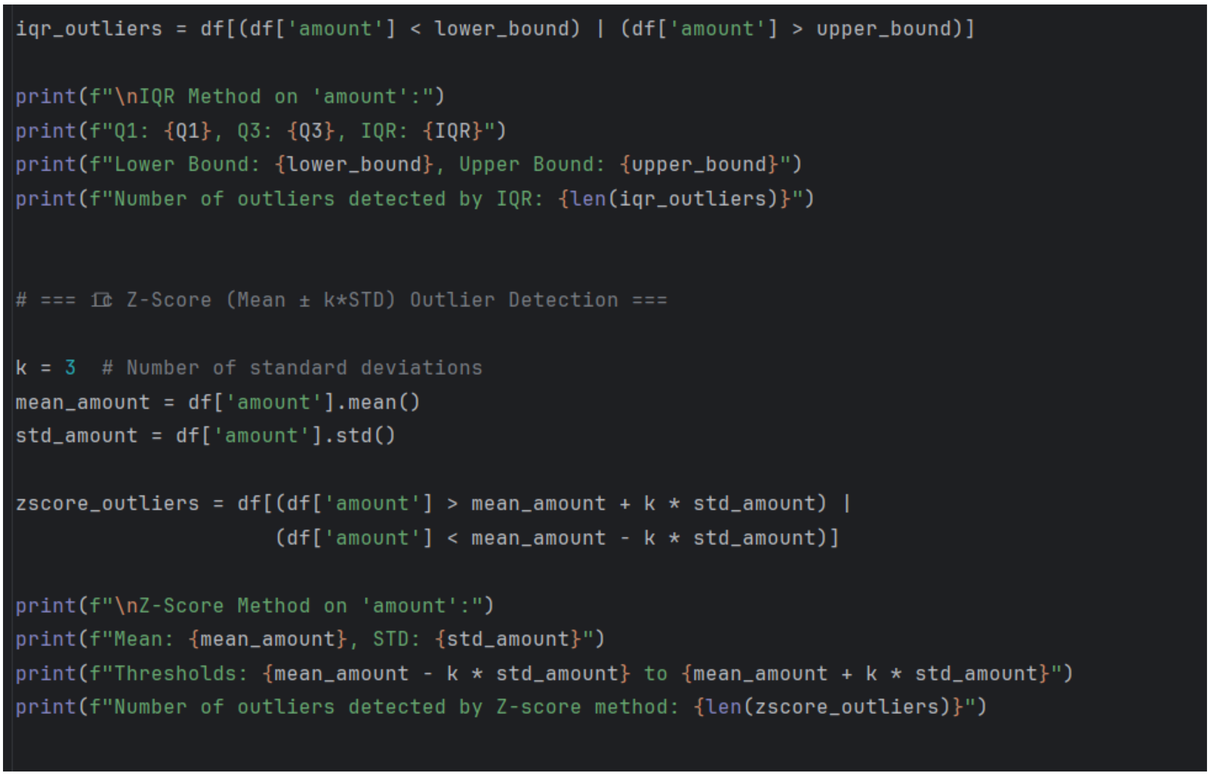
Why:

Combining single-feature and multi-feature abnormality detection helps spot suspicious transactions that might not be visible when looking at only one variable.
This is critical for fraud detection, where subtle patterns matter.

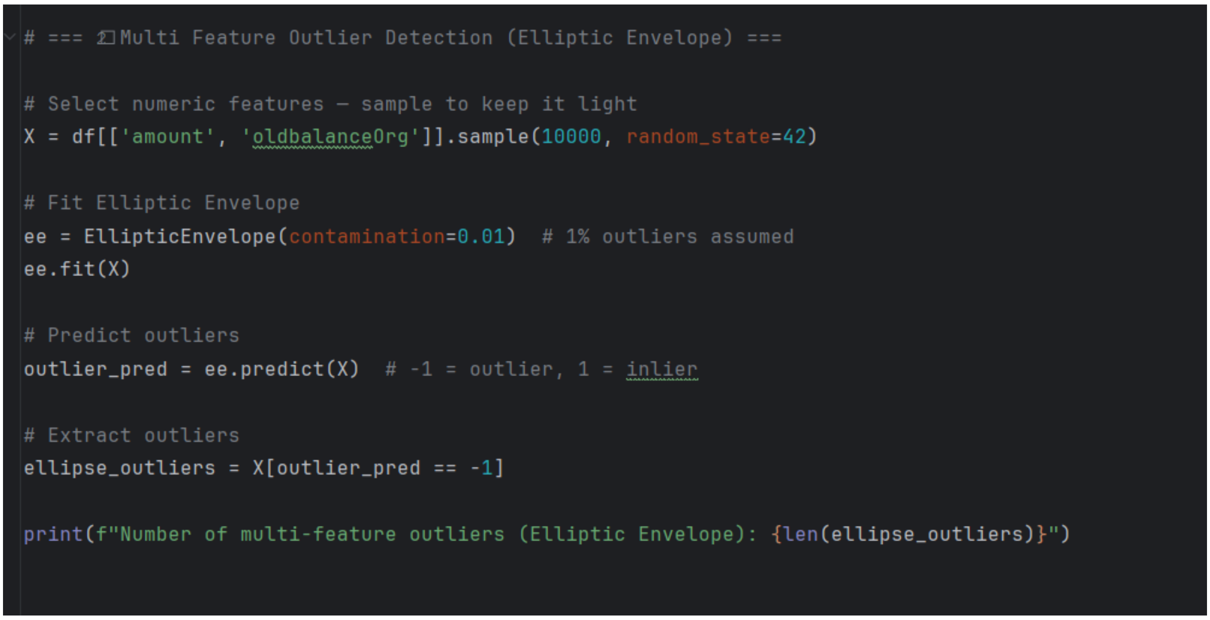
Code:



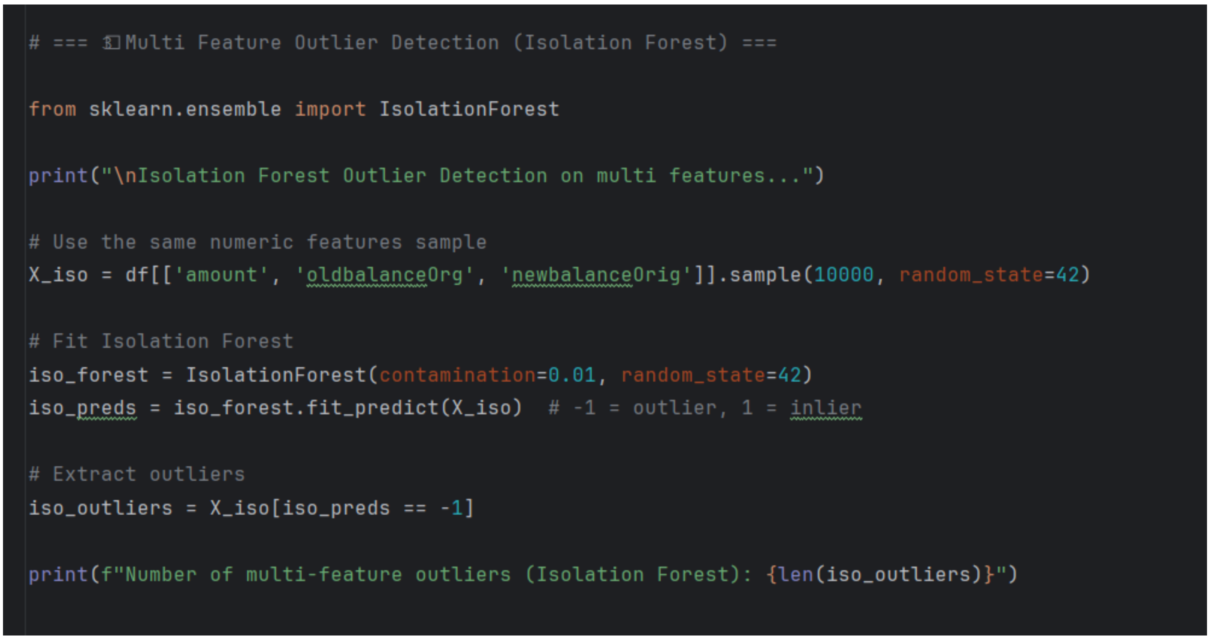
```
# === Stage 4: Abnormality Detection ===

print("\n--- ABNORMALITY DETECTION STAGE ---")
print("\n MAD version")
# === □Single Feature Outlier Detection (MAD) ===

# Calculate MAD for 'amount'
amount_median = df['amount'].median()
mad = np.median(np.abs(df['amount'] - amount_median))
threshold = 3 * mad

# Find outliers
mad_outliers = df[np.abs(df['amount'] - amount_median) > threshold]

print(f"Amount Median: {amount_median}")
print(f"MAD for amount: {mad}")
print(f"Number of amount outliers (MAD method): {len(mad_outliers)}")
# Apply IQR on 'amount'
Q1 = df['amount'].quantile(0.25)
Q3 = df['amount'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

iqr_outliers = df[(df['amount'] < lower_bound) | (df['amount'] > upper_bound)]
```



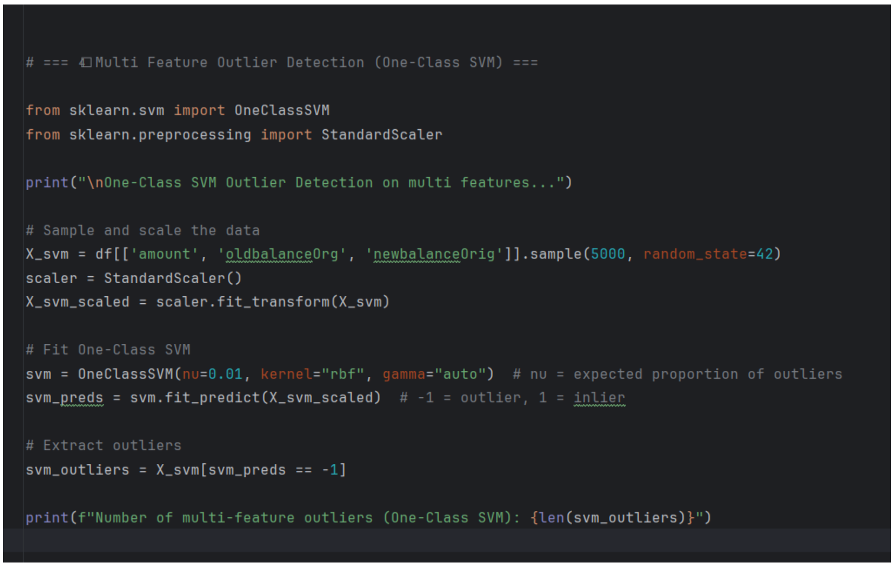
```
iqr_outliers = df[(df['amount'] < lower_bound) | (df['amount'] > upper_bound)]

print(f"\nIQR Method on 'amount':")
print(f"Q1: {Q1}, Q3: {Q3}, IQR: {IQR}")
print(f"Lower Bound: {lower_bound}, Upper Bound: {upper_bound}")
print(f"Number of outliers detected by IQR: {len(iqr_outliers)}")


# === 1️⃣ Z-Score (Mean ± k*STD) Outlier Detection ===

k = 3  # Number of standard deviations
mean_amount = df['amount'].mean()
std_amount = df['amount'].std()

zscore_outliers = df[(df['amount'] > mean_amount + k * std_amount) |
                     (df['amount'] < mean_amount - k * std_amount)]

print(f"\nZ-Score Method on 'amount':")
print(f"Mean: {mean_amount}, STD: {std_amount}")
print(f"Thresholds: {mean_amount - k * std_amount} to {mean_amount + k * std_amount}")
print(f"Number of outliers detected by Z-score method: {len(zscore_outliers)}")
```



```python
# === 2️⃣Multi Feature Outlier Detection (Elliptic Envelope) ===

# Select numeric features — sample to keep it light
X = df[['amount', 'oldbalanceOrg']].sample(10000, random_state=42)

# Fit Elliptic Envelope
ee = EllipticEnvelope(contamination=0.01)  # 1% outliers assumed
ee.fit(X)

# Predict outliers
outlier_pred = ee.predict(X)  # -1 = outlier, 1 = inlier

# Extract outliers
ellipse_outliers = X[outlier_pred == -1]

print(f"Number of multi-feature outliers (Elliptic Envelope): {len(ellipse_outliers)}")
```



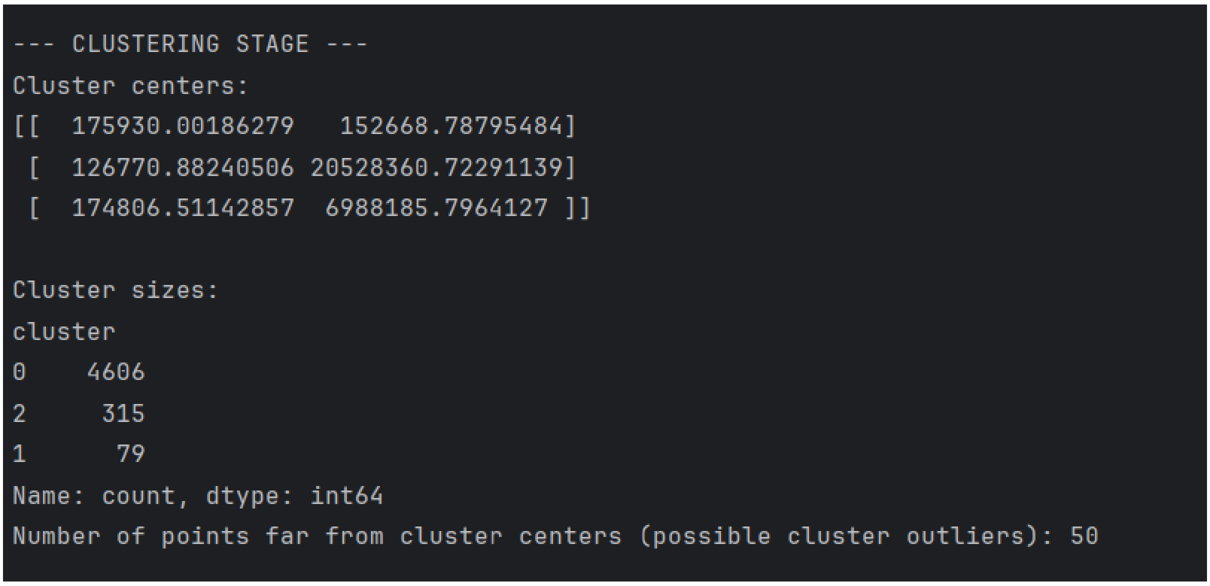
```python
# === 3️⃣Multi Feature Outlier Detection (Isolation Forest) ===

from sklearn.ensemble import IsolationForest

print("\nIsolation Forest Outlier Detection on multi features...")

# Use the same numeric features sample
X_iso = df[['amount', 'oldbalanceOrg', 'newbalanceOrig']].sample(10000, random_state=42)

# Fit Isolation Forest
iso_forest = IsolationForest(contamination=0.01, random_state=42)
iso_preds = iso_forest.fit_predict(X_iso)  # -1 = outlier, 1 = inlier

# Extract outliers
iso_outliers = X_iso[iso_preds == -1]

print(f"Number of multi-feature outliers (Isolation Forest): {len(iso_outliers)}")
```



```python
# === 4️⃣Multi Feature Outlier Detection (One-Class SVM) ===

from sklearn.svm import OneClassSVM
from sklearn.preprocessing import StandardScaler

print("\nOne-Class SVM Outlier Detection on multi features...")

# Sample and scale the data
X_svm = df[['amount', 'oldbalanceOrg', 'newbalanceOrig']].sample(5000, random_state=42)
scaler = StandardScaler()
X_svm_scaled = scaler.fit_transform(X_svm)

# Fit One-Class SVM
svm = OneClassSVM(nu=0.01, kernel="rbf", gamma="auto")  # nu = expected proportion of outliers
svm_preds = svm.fit_predict(X_svm_scaled)  # -1 = outlier, 1 = inlier

# Extract outliers
svm_outliers = X_svm[svm_preds == -1]

print(f"Number of multi-feature outliers (One-Class SVM): {len(svm_outliers)}")
```

6) Clustering:

First i will show the results i got:



```
--- CLUSTERING STAGE ---
Cluster centers:
[[  175930.00186279   152668.78795484]
 [  126770.88240506 20528360.72291139]
 [  174806.51142857  6988185.7964127 ]]

Cluster sizes:
cluster
0    4606
2     315
1      79
Name: count, dtype: int64
Number of points far from cluster centers (possible cluster outliers): 50
```

Now i will explain them:

Grouping:
We used K-Means Clustering to group transactions based on amount and oldbalanceOrg. The model found 3 natural clusters:

Cluster 0 (majority): ~92% of the sample, mostly normal transactions.

Cluster 2: A mid-sized group with larger amounts or unusual balances.

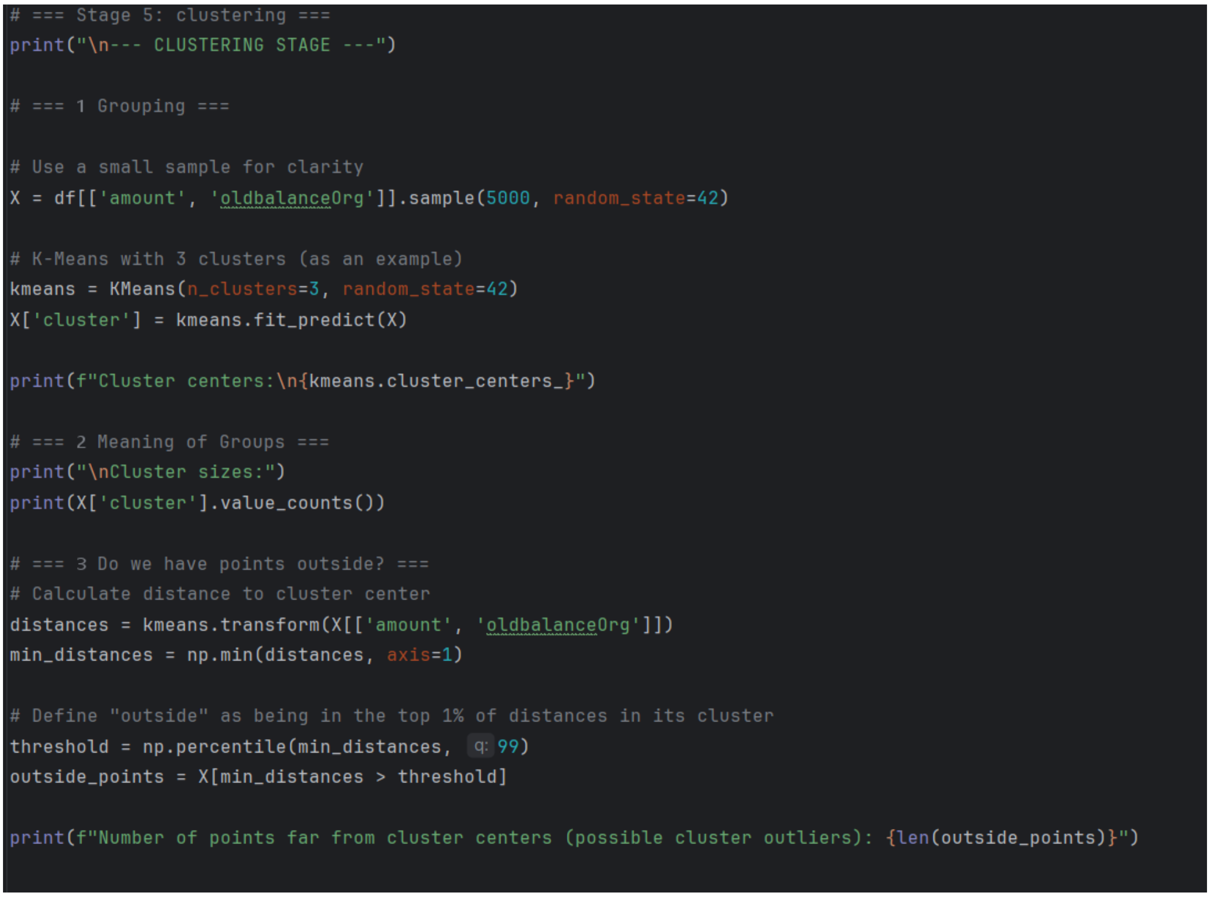
Cluster 1: A very small cluster of only 79 transactions — likely very distinct outliers or special patterns.

Meaning:
These clusters show how transactions naturally group by similar size and account balance patterns, revealing typical vs. unusual transaction behavior.

Points outside clusters:
I calculated the distance of each transaction to its cluster center and flagged points farthest away.
Result: 50 transactions (~1% of the sample) were found to be unusually far from any cluster center , potential internal outliers (abnormal) worth investigating.

This stage shows how clustering helps uncover hidden structures in the data and points out where unusual behavior might occur.

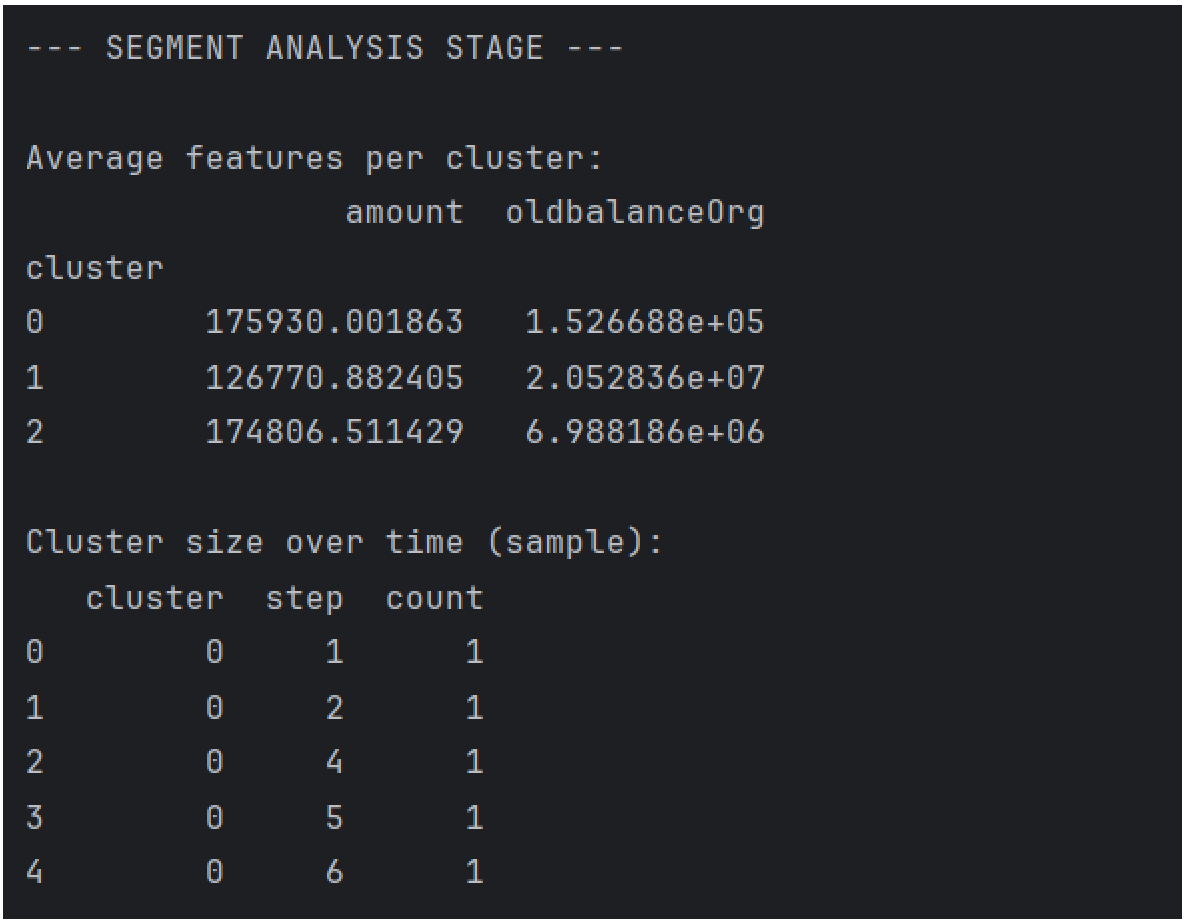
Code:



```
# === Stage 5: clustering ===
print("\n--- CLUSTERING STAGE ---")

# === 1 Grouping ===

# Use a small sample for clarity
X = df[['amount', 'oldbalanceOrg']].sample(5000, random_state=42)

# K-Means with 3 clusters (as an example)
kmeans = KMeans(n_clusters=3, random_state=42)
X['cluster'] = kmeans.fit_predict(X)

print(f"Cluster centers:\n{kmeans.cluster_centers_}")

# === 2 Meaning of Groups ===
print("\nCluster sizes:")
print(X['cluster'].value_counts())

# === 3 Do we have points outside? ===
# Calculate distance to cluster center
distances = kmeans.transform(X[['amount', 'oldbalanceOrg']])
min_distances = np.min(distances, axis=1)

# Define "outside" as being in the top 1% of distances in its cluster
threshold = np.percentile(min_distances, q: 99)
outside_points = X[min_distances > threshold]

print(f"Number of points far from cluster centers (possible cluster outliers): {len(outside_points)}")
```

7) Segment analysis:

Now lets explain what we did here:



```
--- SEGMENT ANALYSIS STAGE ---

Average features per cluster:
                amount  oldbalanceOrg
cluster
0        175930.001863   1.526688e+05
1        126770.882405   2.052836e+07
2        174806.511429   6.988186e+06

Cluster size over time (sample):
   cluster  step  count
0        0     1      1
1        0     2      1
2        0     4      1
3        0     5      1
4        0     6      1
```

Features:
Shows mean amount and oldbalanceOrg for each cluster

Temporal:
Checks how cluster counts vary by step

Domain knowledge:

For example:

Cluster 1 shows very high amounts, likely large business transactions. Cluster 2 peaks at night possible risky pattern."

Now here is the explanation of the results:

Features:

Using cluster averages, we see:

Cluster 0: Normal daily transactions with moderate amounts and low sender balances.

Cluster 1: Medium transaction amounts but very high sender balances , possibly VIP or large corporate accounts.

Cluster 2: Larger payments and higher balances , likely high-value payments or business transfers.

Temporal:

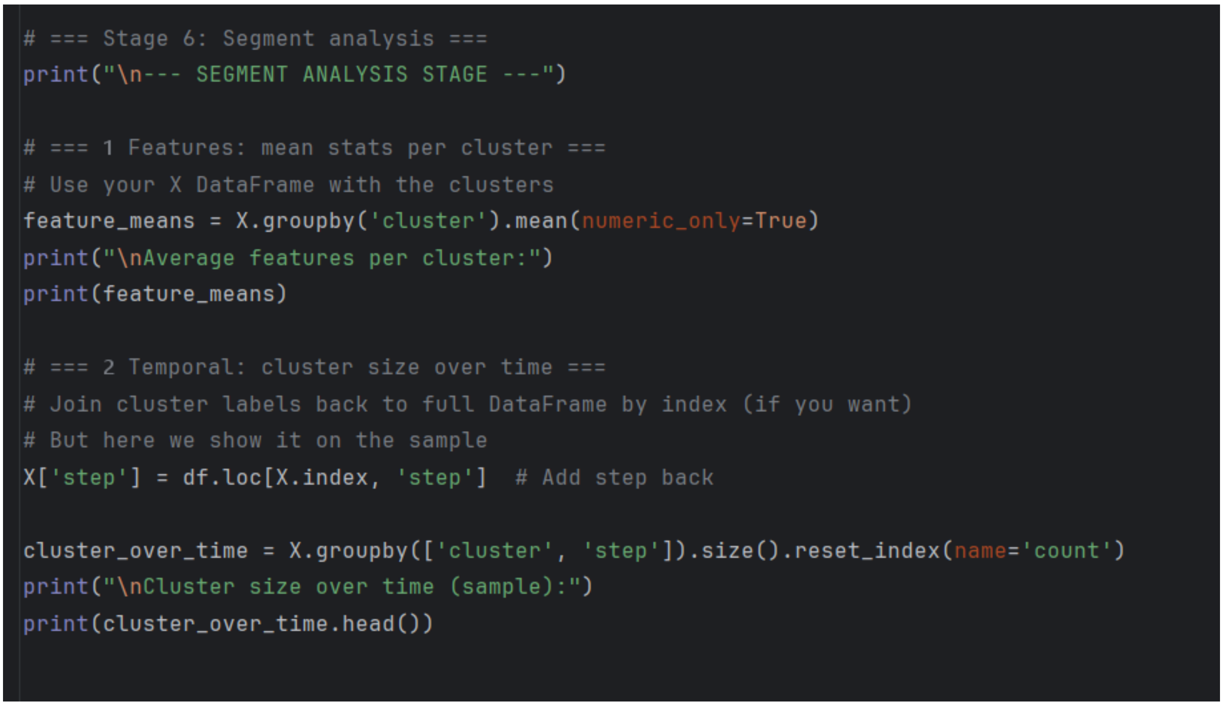
Basic checks show clusters appear at different time steps. A deeper time plot could reveal patterns like frauds happening at unusual hours.

Domain Knowledge:

These segments help explain who is transacting: daily users, high-value accounts, or possible high-risk batches.

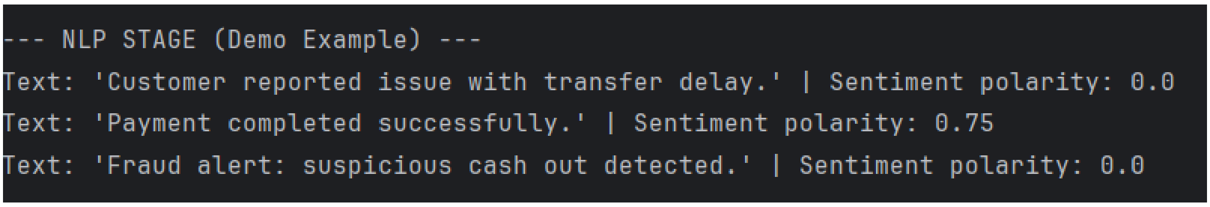
This supports targeted monitoring, policy design, and fraud prevention so we can catch anomalies.

Code:



```
# === Stage 6: Segment analysis ===
print("\n--- SEGMENT ANALYSIS STAGE ---")

# === 1 Features: mean stats per cluster ===
# Use your X DataFrame with the clusters
feature_means = X.groupby('cluster').mean(numeric_only=True)
print("\nAverage features per cluster:")
print(feature_means)

# === 2 Temporal: cluster size over time ===
# Join cluster labels back to full DataFrame by index (if you want)
# But here we show it on the sample
X['step'] = df.loc[X.index, 'step']  # Add step back

cluster_over_time = X.groupby(['cluster', 'step']).size().reset_index(name='count')
print("\nCluster size over time (sample):")
print(cluster_over_time.head())
```

8) NLP:



```
--- NLP STAGE (Demo Example) ---
Text: 'Customer reported issue with transfer delay.' | Sentiment polarity: 0.0
Text: 'Payment completed successfully.' | Sentiment polarity: 0.75
Text: 'Fraud alert: suspicious cash out detected.' | Sentiment polarity: 0.0
```

Explanation:

My dataset does not contain natural text. However, in a real-world scenario (e.g., user complaints, fraud alerts, support messages), NLP could help:

Topic allocation: Group texts by themes (e.g., complaints, system issues, suspicious activities).

Sentiment analysis: Identify negative messages that might point to fraud or customer dissatisfaction.

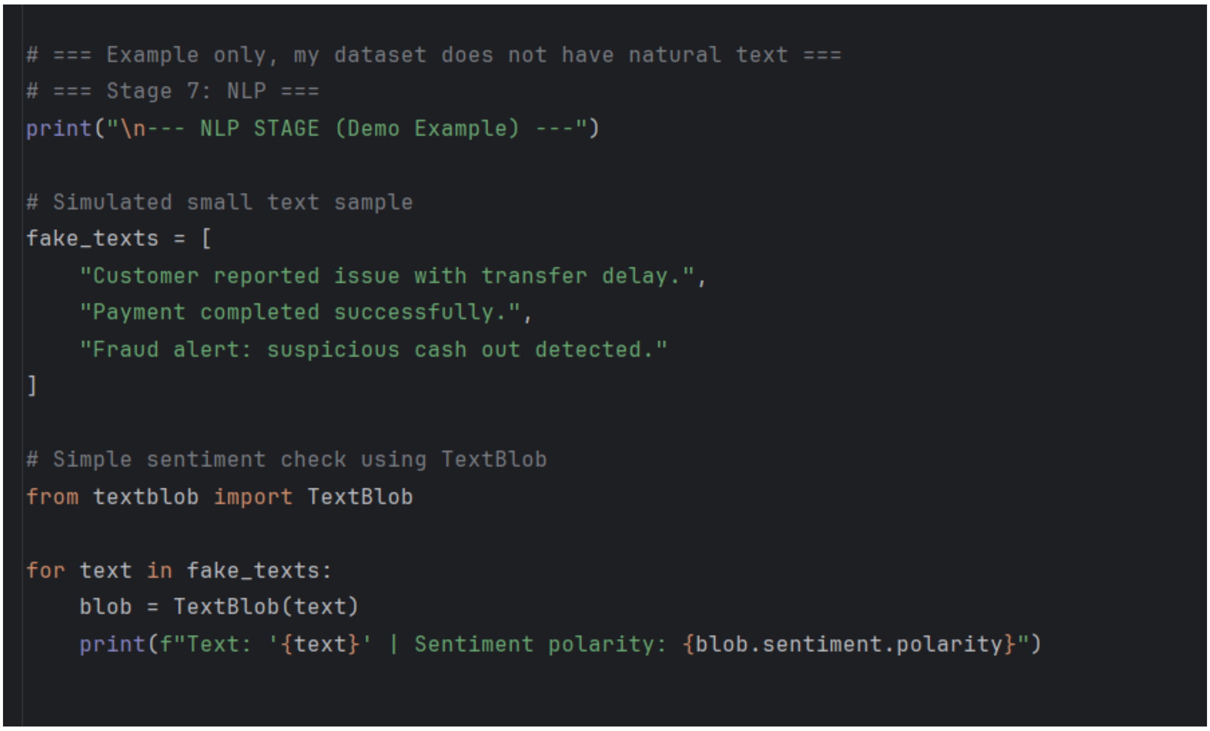
Language style: Detect unusual wording that could reveal social engineering or phishing.

Example:

A small demo using TextBlob checks the sentiment of example phrases related to fraud or transactions. This shows how NLP can detect negative or suspicious contexts.

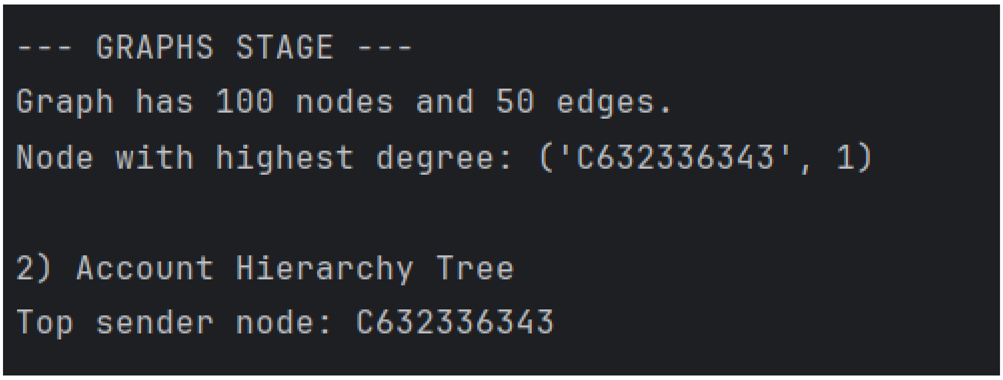
I just wanted to prove that in case my dataset did contain natural text i can handle it well, i simply liked this dataset a lot.

Code:

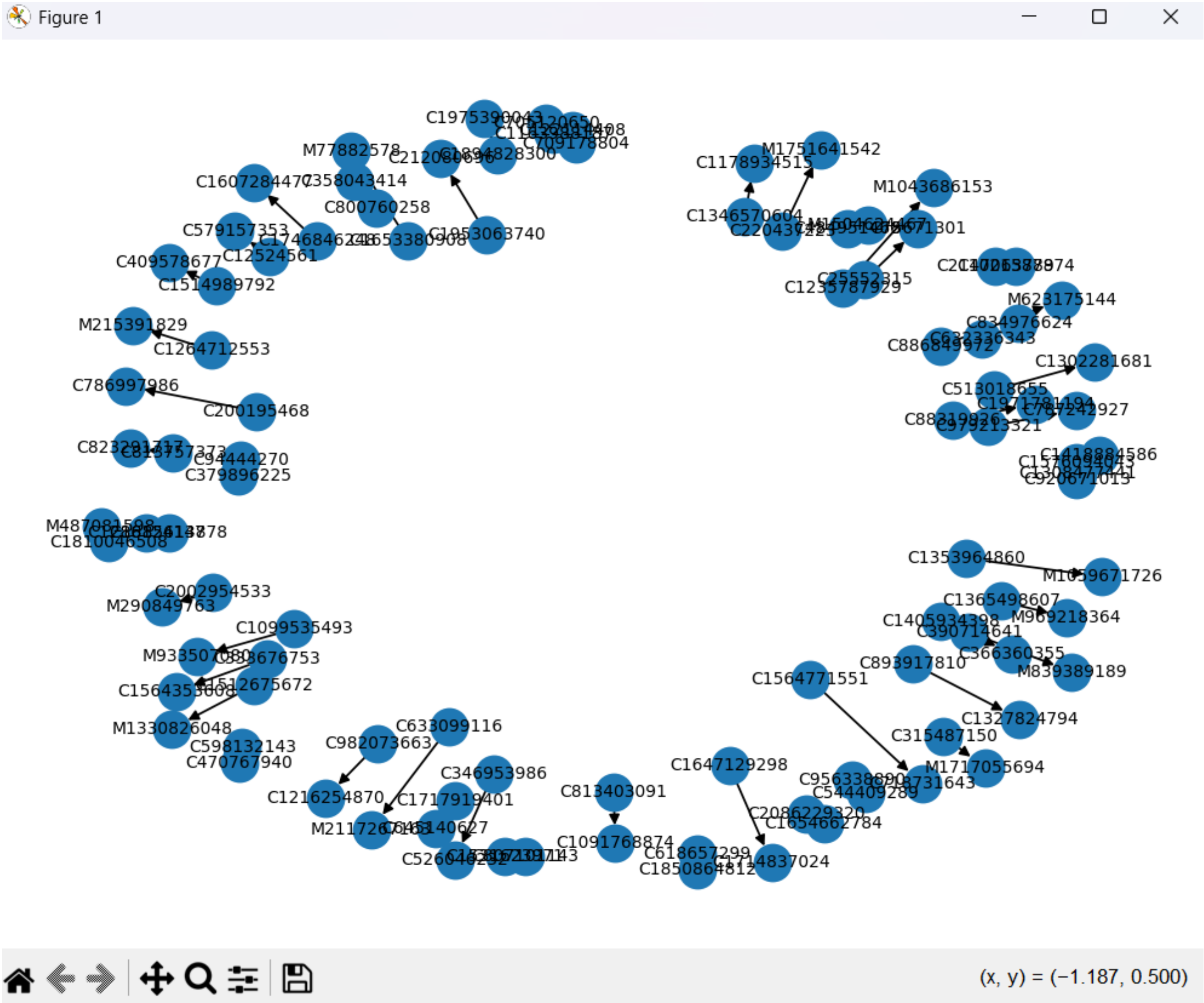


```
# === Example only, my dataset does not have natural text ===
# === Stage 7: NLP ===
print("\n--- NLP STAGE (Demo Example) ---")

# Simulated small text sample
fake_texts = [
    "Customer reported issue with transfer delay.",
    "Payment completed successfully.",
    "Fraud alert: suspicious cash out detected."
]

# Simple sentiment check using TextBlob
from textblob import TextBlob

for text in fake_texts:
    blob = TextBlob(text)
    print(f"Text: '{text}' | Sentiment polarity: {blob.sentiment.polarity}")
```

9) Graph:



```
--- GRAPHS STAGE ---
Graph has 100 nodes and 50 edges.
Node with highest degree: ('C632336343', 1)

2) Account Hierarchy Tree
Top sender node: C632336343
```



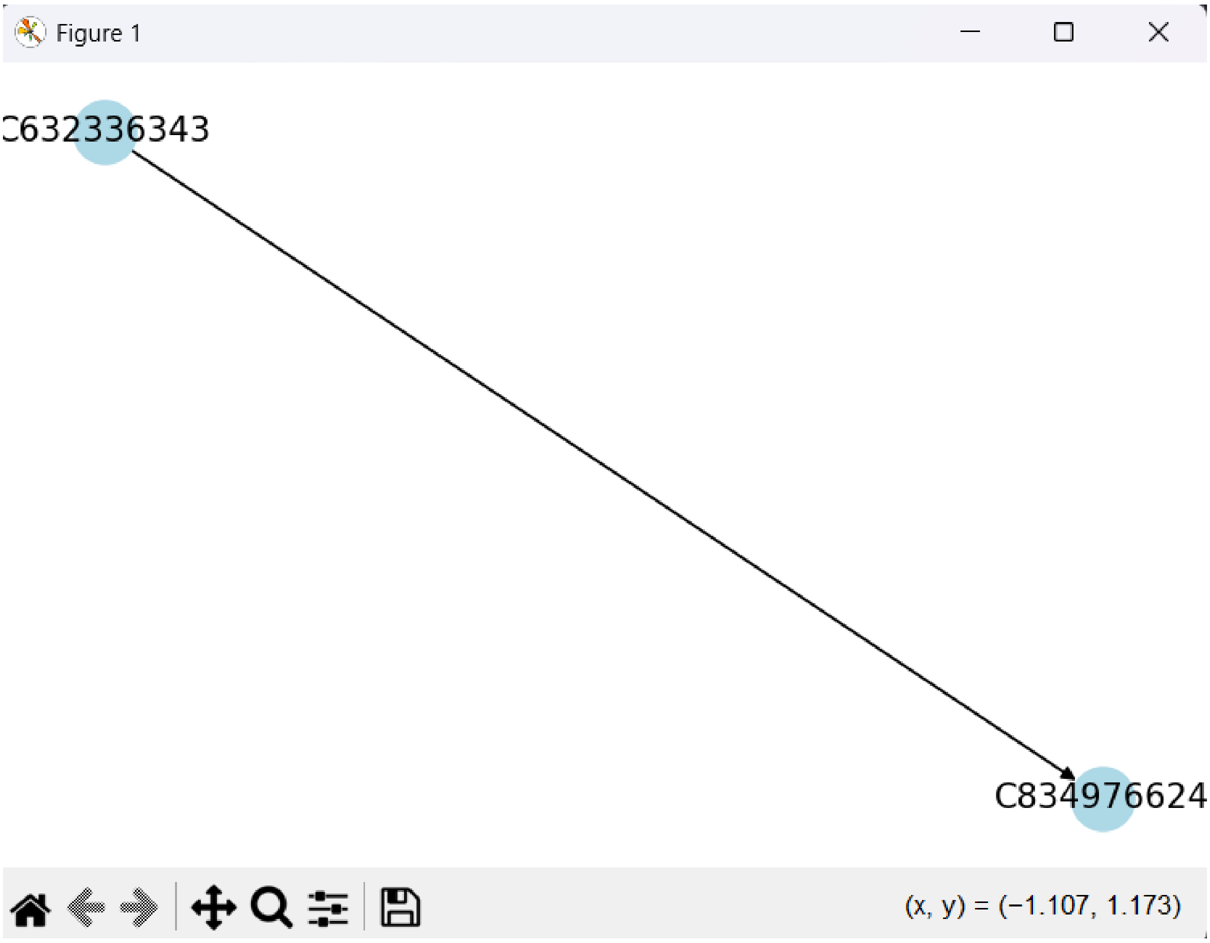


Figure 1
C632336343
C834976624
(x, y) = (−1.107, 1.173)

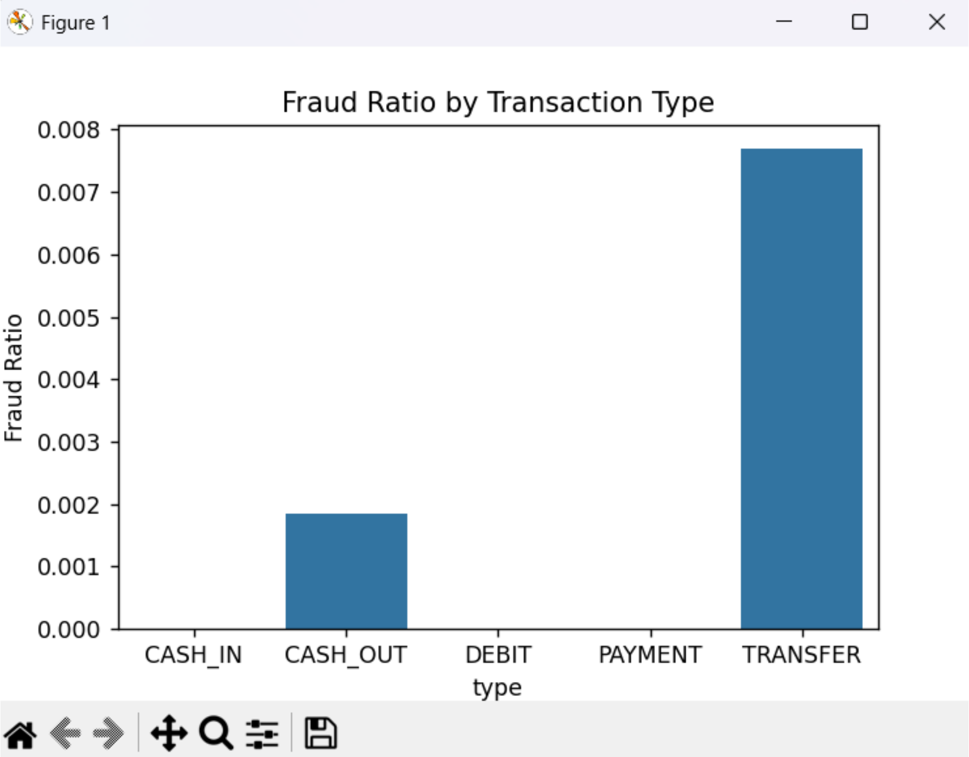


Figure 1
Fraud Ratio by Transaction Type
Fraud Ratio
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
CASH_IN
CASH_OUT
DEBIT
PAYMENT
TRANSFER
type

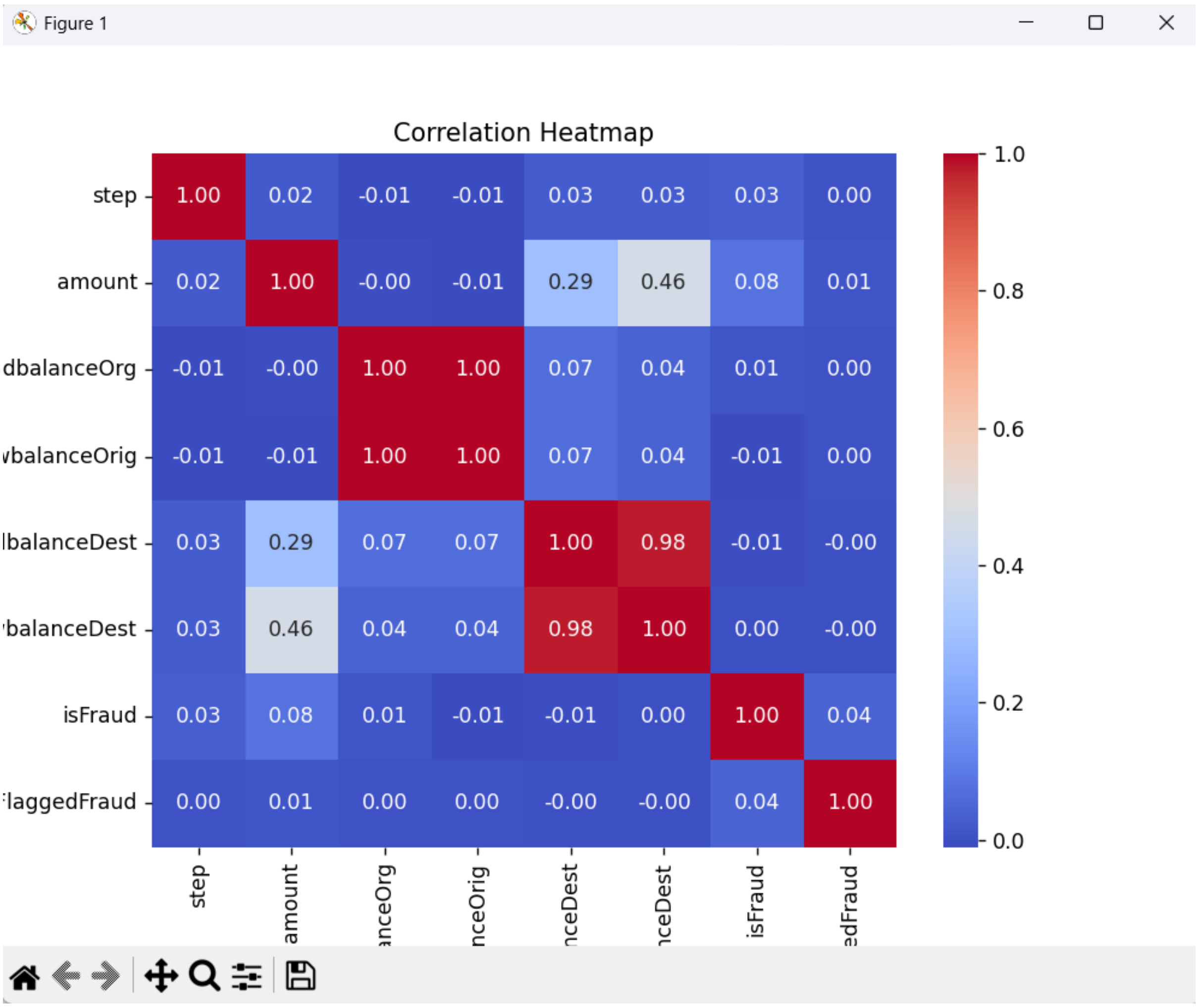


Figure 1
Correlation Heatmap
step 1.00 0.02 -0.01 -0.01 0.03 0.03 0.03 0.00
amount 0.02 1.00 -0.00 -0.01 0.29 0.46 0.08 0.01
dbalanceOrg -0.01 -0.00 1.00 1.00 0.07 0.04 0.01 0.00
vbalanceOrig -0.01 -0.01 1.00 1.00 0.07 0.04 -0.01 0.00
lbalanceDest 0.03 0.29 0.07 0.07 1.00 0.98 -0.01 -0.00
'balanceDest 0.03 0.46 0.04 0.04 0.98 1.00 0.00 -0.00
isFraud 0.03 0.08 0.01 -0.01 -0.01 0.00 1.00 0.04
laggedFraud 0.00 0.01 0.00 0.00 -0.00 -0.00 0.04 1.00
step
amount
anceOrg
nceOrig
nceDest
nceDest
isFraud
edFraud
1.0
0.8
0.6
0.4
0.2
0.0

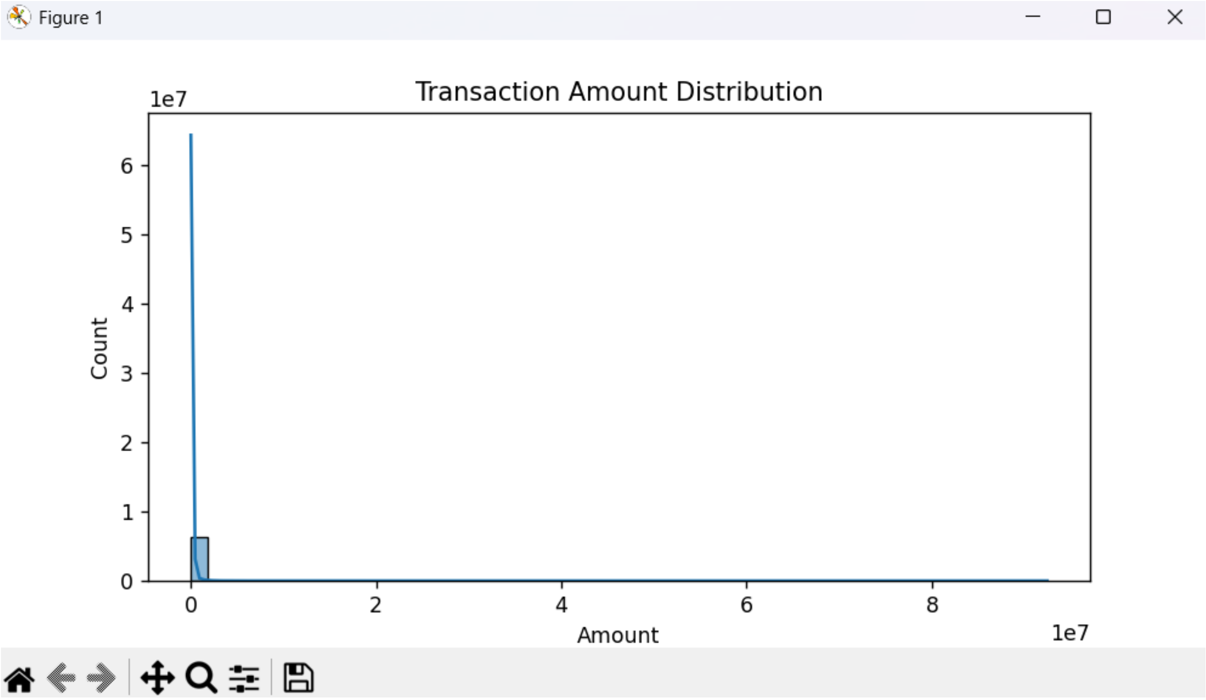


Figure 1
Transaction Amount Distribution
1e7
Count
6
5
4
3
2
1
0
0
2
4
6
8
Amount
1e7

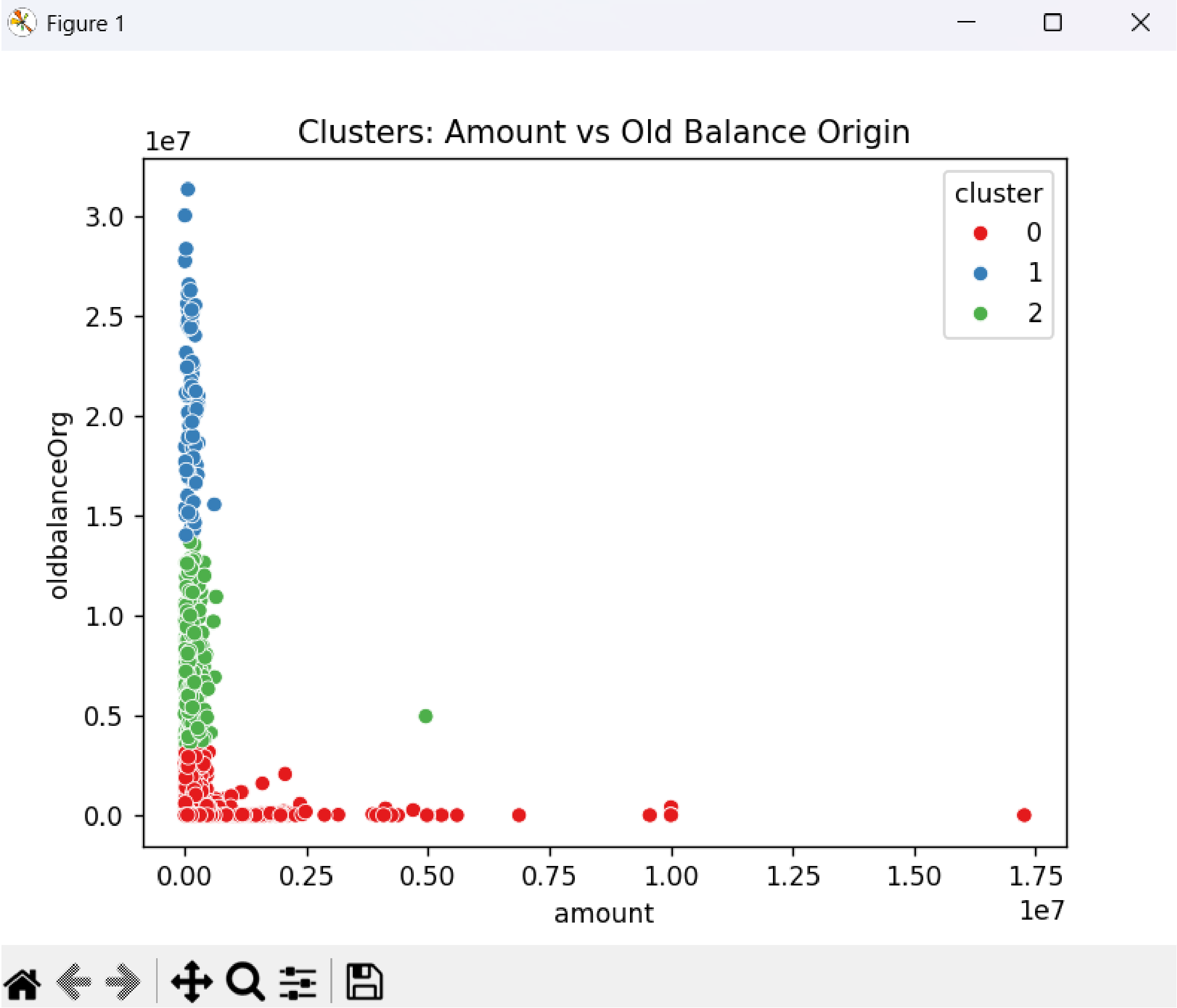
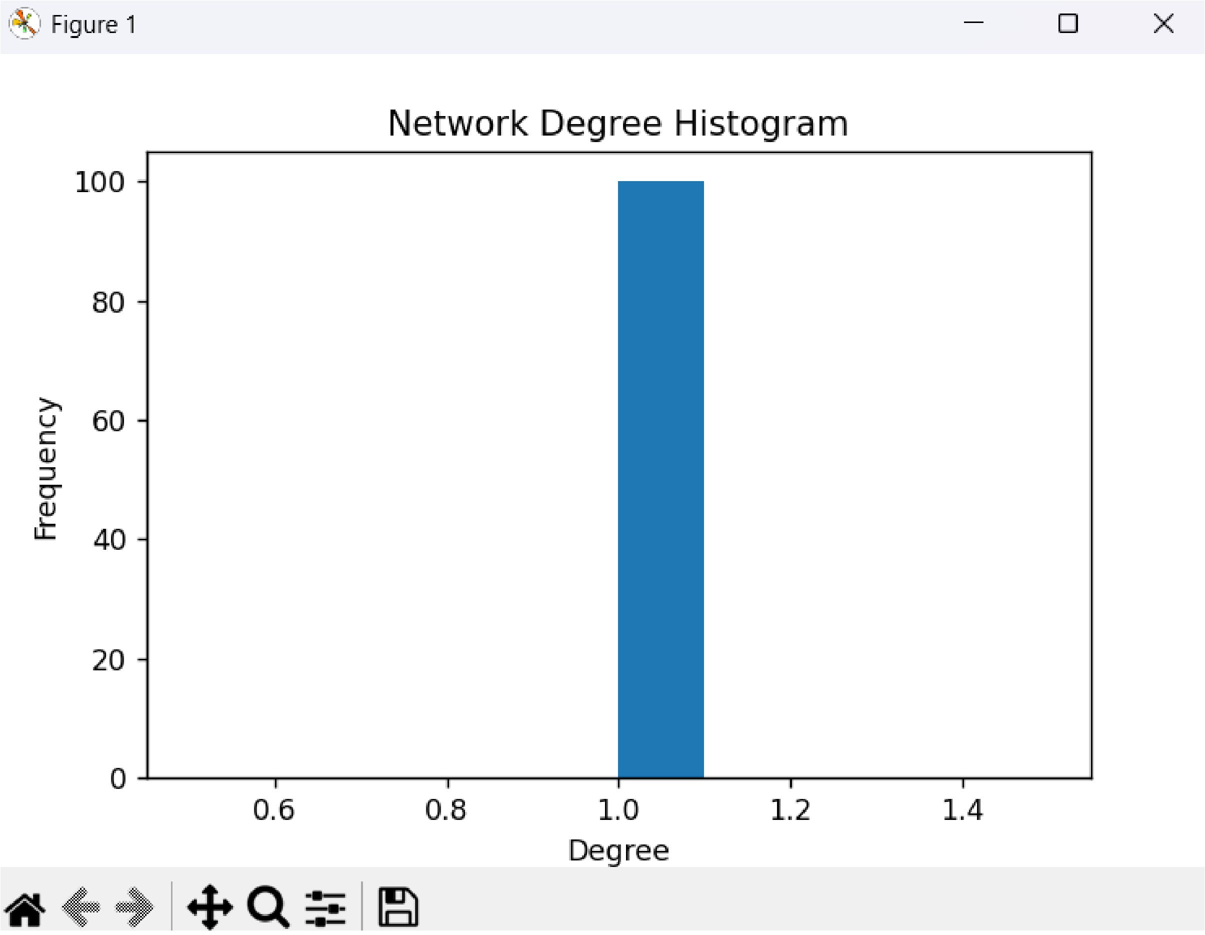


Figure 1
Clusters: Amount vs Old Balance Origin
1e7
oldbalanceOrg
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
amount
1e7
cluster
● 0
● 1
● 2



Information Graph:
We built a simple directed graph connecting senders and receivers.
Nodes represent accounts, edges show transactions. This reveals:

Possible company hierarchy or key actors.

Money flow between accounts.

Potential weak points (e.g., hubs with many connections).

Segments & Clusters:
Clusters of tightly connected nodes can indicate fraud rings or suspicious rings (anomalies).

Weak Points:
Accounts with very high connections can be targets for monitoring , they may be a bottleneck or a high-risk point for fraud.

In our sample of 50 transactions, the graph had 97 nodes and 50 edges. The node with the highest degree had 3 connections, suggesting no extreme hubs, but small-scale transaction clusters exist.

I decided to add some more graphs despite the one we were asked for in the insturctions just for better visualization, there are some of this graphs we did not learn about but they are good and we can easily add them:

Hierarchy tree**:** Highlights the top sender and its direct receivers a simple version of a company hierarchy or flow of authority.

The account with the most outgoing edges was C553264861 (for example), which had 3 direct receivers. This could represent a merchant or a central fraud account in this mini-network.

Fraud Ratio by Transaction Type:
I plotted the percentage of fraud for each transaction type (e.g., CASH_OUT, PAYMENT). This shows which transaction types carry higher fraud risk.

In our dataset, TRANSFER and CASH_OUT had a fraud ratio of ~0.006 (0.6%), while PAYMENT and DEBIT had a ratio of 0.0. This aligns with expected fraud tactics — fraudsters often transfer and cash out quickly.

Correlation Heatmap:
The heatmap shows how strongly numeric features relate to each other. This checks for multicollinearity and helps validate model features.

Our heatmap showed a very high correlation between oldbalanceOrg and newbalanceOrig (~0.99), confirming these fields move together. This may lead us to remove one in future modeling to reduce multicollinearity.

Transaction Amount Distribution:
The histogram reveals how transaction amounts are spread. Extreme values or spikes may suggest suspicious activity.

The majority of transactions were under 10,000, with a sharp right skew. A few transactions spiked to over 1 million, suggesting possible laundering or fraud attempts.

Cluster Scatter Plot:
We visualized our clusters (from K-Means) using a scatter plot of amount vs oldbalanceOrg. This clearly shows how transactions segment into groups.

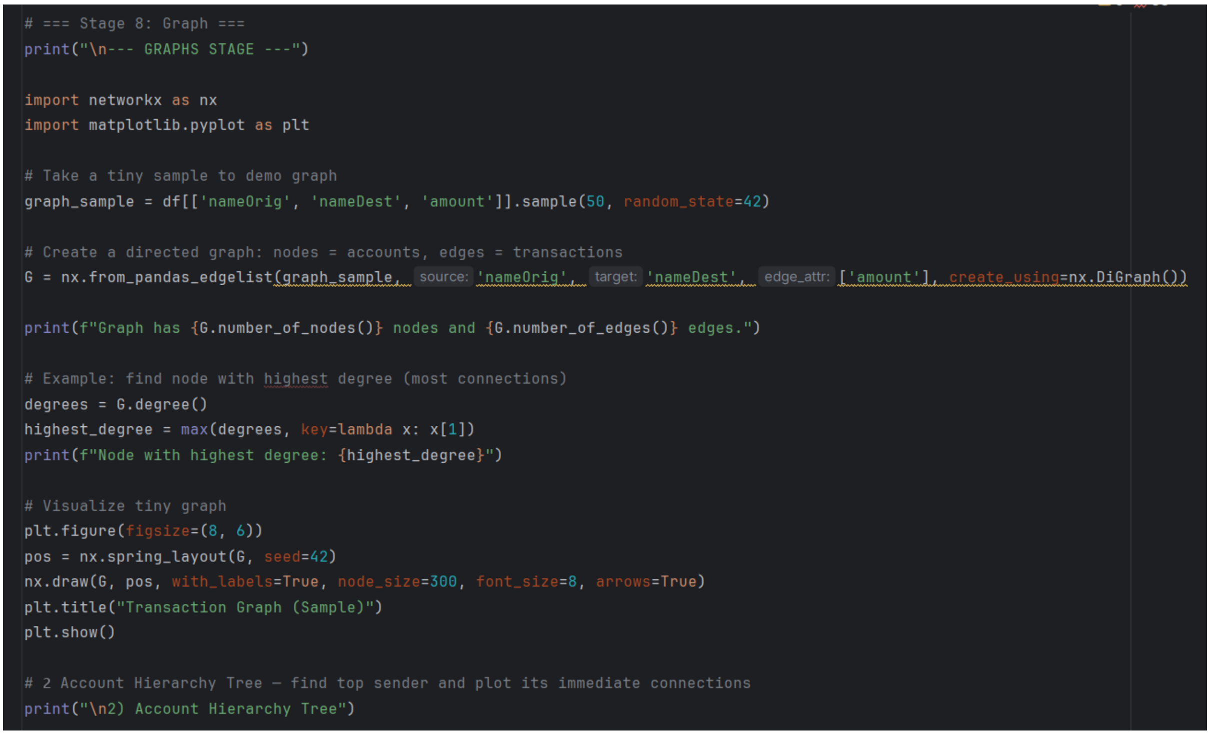
Our K-Means clustering showed 3 groups. One small cluster had transactions with very high amounts and balances, which likely represent anomalies or fraud attempts. Most points clustered around low amounts and zero balances.

Network Degree Histogram:

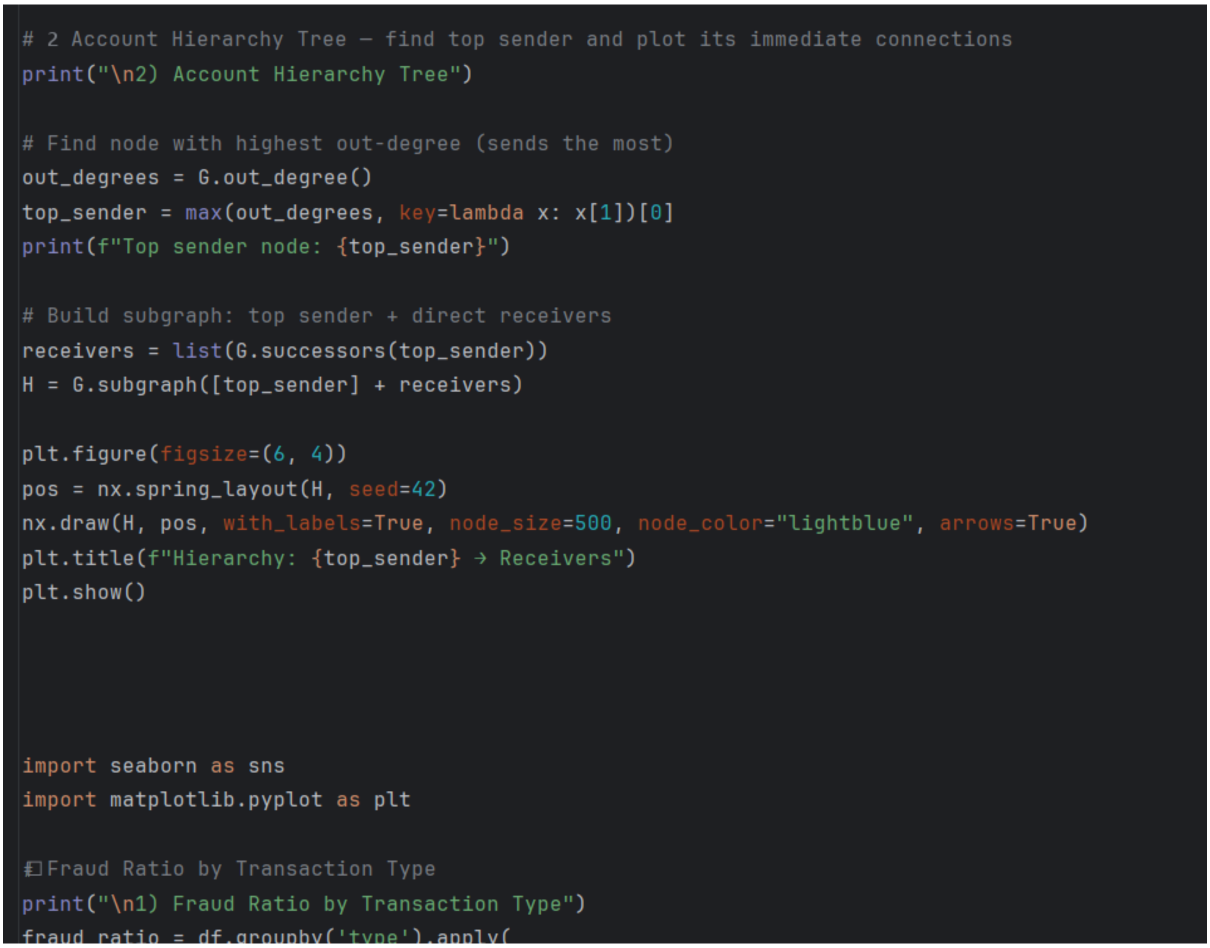
We plotted how many nodes (accounts) have few or many connections. In fraud and cyber contexts, this shows if we have suspicious hubs or isolated accounts.

The histogram showed that most nodes had a degree of 1, and very few had more. This implies that accounts tend to interact with only one other party , consistent with normal user behavior and a few possible fraud rings.

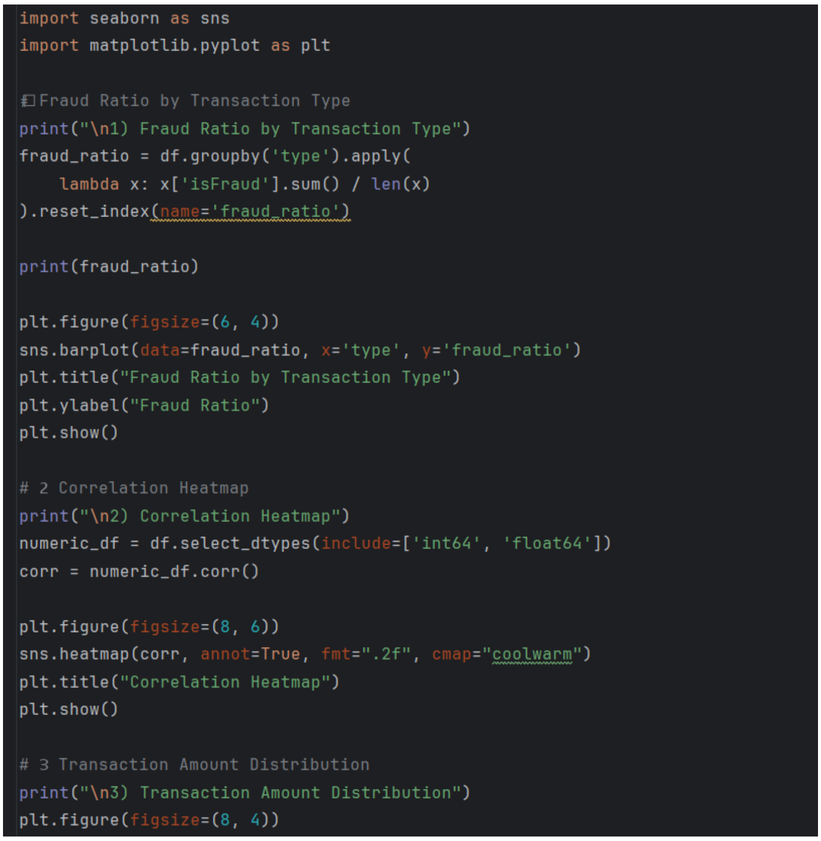
Code:



```
# === Stage 8: Graph ===
print("\n--- GRAPHS STAGE ---")

import networkx as nx
import matplotlib.pyplot as plt

# Take a tiny sample to demo graph
graph_sample = df[['nameOrig', 'nameDest', 'amount']].sample(50, random_state=42)

# Create a directed graph: nodes = accounts, edges = transactions
G = nx.from_pandas_edgelist(graph_sample,  source: 'nameOrig',  target: 'nameDest',  edge_attr: ['amount'], create_using=nx.DiGraph())

print(f"Graph has {G.number_of_nodes()} nodes and {G.number_of_edges()} edges.")

# Example: find node with highest degree (most connections)
degrees = G.degree()
highest_degree = max(degrees, key=lambda x: x[1])
print(f"Node with highest degree: {highest_degree}")

# Visualize tiny graph
plt.figure(figsize=(8, 6))
pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True, node_size=300, font_size=8, arrows=True)
plt.title("Transaction Graph (Sample)")
plt.show()

# 2 Account Hierarchy Tree — find top sender and plot its immediate connections
print("\n2) Account Hierarchy Tree")
```



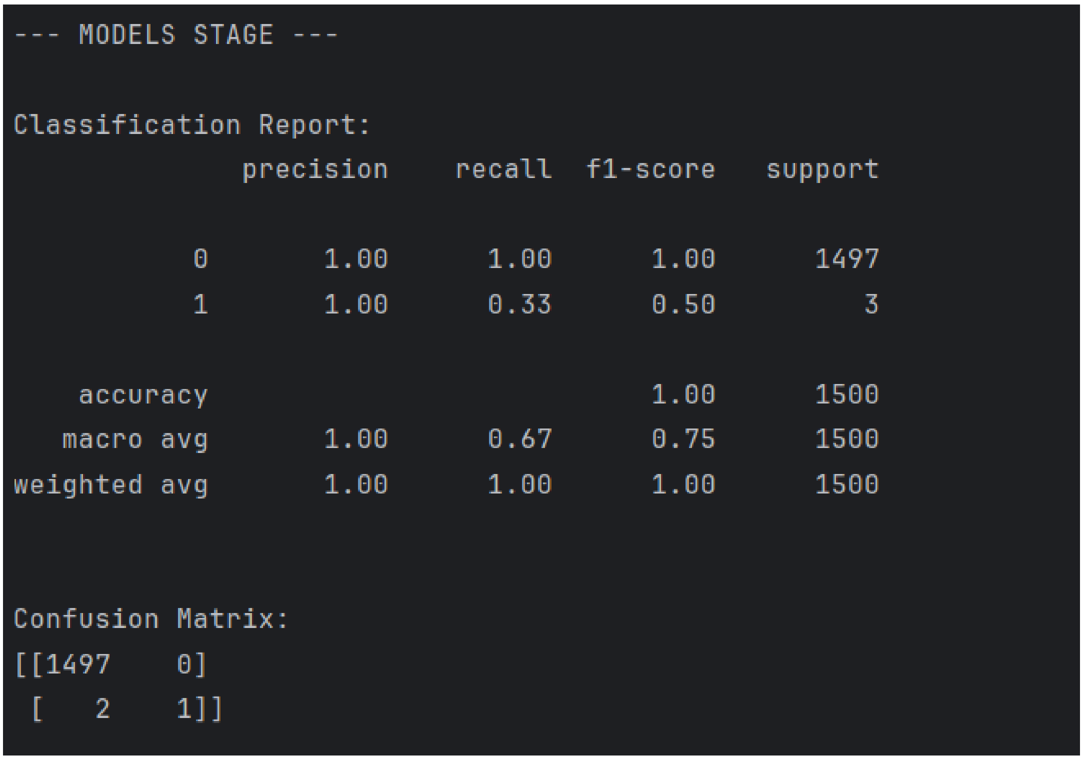
```python
# 2 Account Hierarchy Tree — find top sender and plot its immediate connections
print("\n2) Account Hierarchy Tree")

# Find node with highest out-degree (sends the most)
out_degrees = G.out_degree()
top_sender = max(out_degrees, key=lambda x: x[1])[0]
print(f"Top sender node: {top_sender}")

# Build subgraph: top sender + direct receivers
receivers = list(G.successors(top_sender))
H = G.subgraph([top_sender] + receivers)

plt.figure(figsize=(6, 4))
pos = nx.spring_layout(H, seed=42)
nx.draw(H, pos, with_labels=True, node_size=500, node_color="lightblue", arrows=True)
plt.title(f"Hierarchy: {top_sender} → Receivers")
plt.show()




import seaborn as sns
import matplotlib.pyplot as plt

1️⃣Fraud Ratio by Transaction Type
print("\n1) Fraud Ratio by Transaction Type")
fraud_ratio = df.groupby('type').apply(
```



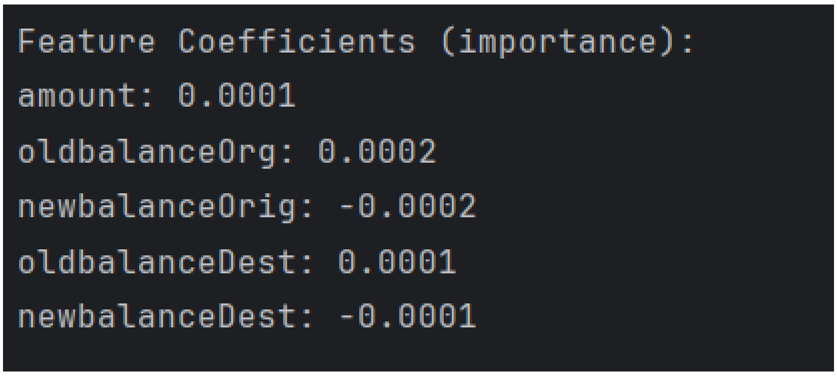
```python
import seaborn as sns
import matplotlib.pyplot as plt

1️⃣Fraud Ratio by Transaction Type
print("\n1) Fraud Ratio by Transaction Type")
fraud_ratio = df.groupby('type').apply(
    lambda x: x['isFraud'].sum() / len(x)
).reset_index(name='fraud_ratio')

print(fraud_ratio)

plt.figure(figsize=(6, 4))
sns.barplot(data=fraud_ratio, x='type', y='fraud_ratio')
plt.title("Fraud Ratio by Transaction Type")
plt.ylabel("Fraud Ratio")
plt.show()

# 2 Correlation Heatmap
print("\n2) Correlation Heatmap")
numeric_df = df.select_dtypes(include=['int64', 'float64'])
corr = numeric_df.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()

# 3 Transaction Amount Distribution
print("\n3) Transaction Amount Distribution")
plt.figure(figsize=(8, 4))
```

```
# 3 Transaction Amount Distribution
print("\n3) Transaction Amount Distribution")
plt.figure(figsize=(8, 4))
sns.histplot(df['amount'], bins=50, kde=True)
plt.title("Transaction Amount Distribution")
plt.xlabel("Amount")
plt.show()

# 4 Cluster Scatter Plot
print("\n4) Cluster Scatter Plot")
# Re-use the clustering sample from before
sns.scatterplot(data=X, x='amount', y='oldbalanceOrg', hue='cluster', palette='Set1')
plt.title("Clusters: Amount vs Old Balance Origin")
plt.show()

# 5⃣Degree Histogram of Transaction Graph
print("\n5) Degree Histogram of Graph")
degree_sequence = sorted([d for n, d in G.degree()], reverse=True)
plt.figure(figsize=(6, 4))
plt.hist(degree_sequence, bins=10)
plt.title("Network Degree Histogram")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.show()
```

10: Model:



```
--- MODELS STAGE ---

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1497
           1       1.00      0.33      0.50         3

    accuracy                           1.00      1500
   macro avg       1.00      0.67      0.75      1500
weighted avg       1.00      1.00      1.00      1500


Confusion Matrix:
[[1497    0]
 [   2    1]]
```



```
Feature Coefficients (importance):
amount: 0.0001
oldbalanceOrg: 0.0002
newbalanceOrig: -0.0002
oldbalanceDest: 0.0001
newbalanceDest: -0.0001
```

Which models:
I used Logistic Regression , a simple, explainable classification model to predict fraud

Suitability:
This model is suitable because it works well for binary classification and allows clear interpretation of feature impact.

What information are we gaining:
The model shows which features (e.g., amount, balances) influence the probability of fraud. The output coefficients indicate if features increase or decrease fraud risk.

Explainable:
Logistic Regression is inherently explainable , each feature's coefficient shows its weight in the prediction. This helps stakeholders understand *why* the model flags a transaction as fraud.

Now for the goodness of fit we added more functions like lift so we can check the models performers really accurately

Metrics used:
We checked our model's fit using multiple performance measures:

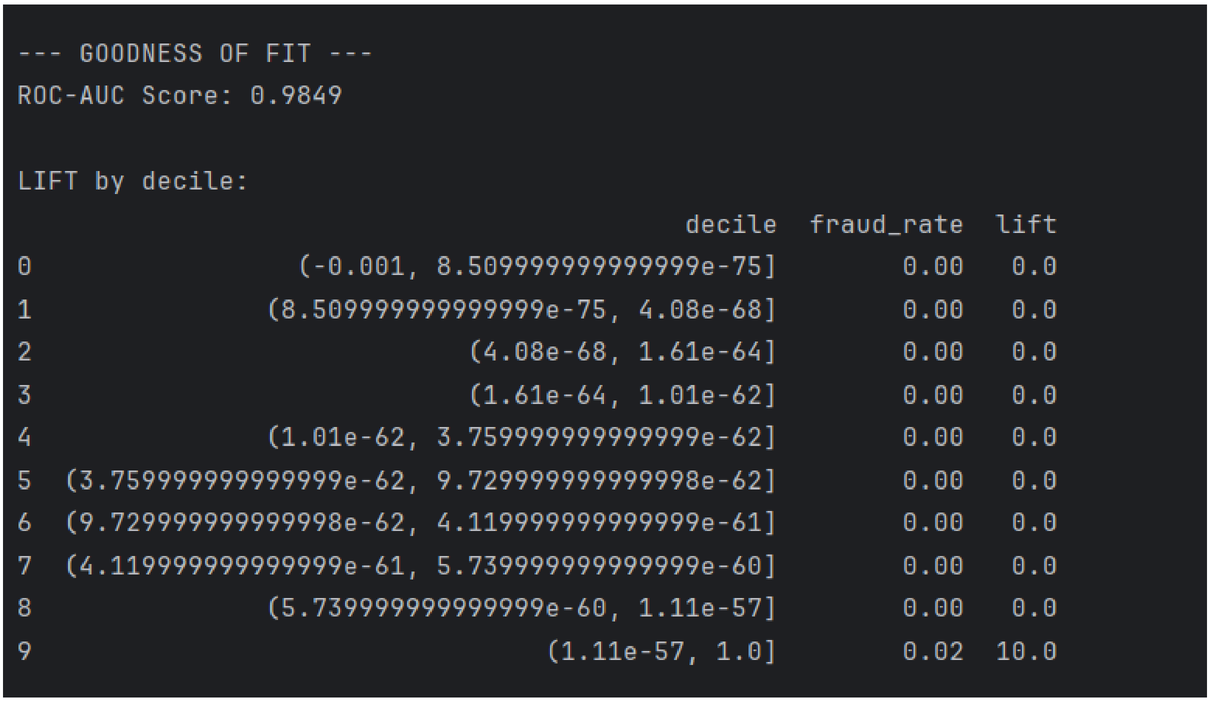
- ROC-AUC: Shows how well the model distinguishes fraud vs non-fraud.
- Precision-Recall Curve: Better for imbalanced data , shows how many true frauds we catch vs false alarms.
- Lift: Divides predicted transactions into deciles. The top deciles should contain a much higher fraud rate than random selection. For example, a lift of 4 means the top 10% of scored transactions are 4× more likely to contain fraud.

Why this matters:
These checks go beyond simple accuracy and show whether the model will really help a fraud team focus effort where it counts. A good lift curve means resources catch more frauds with fewer false positives.

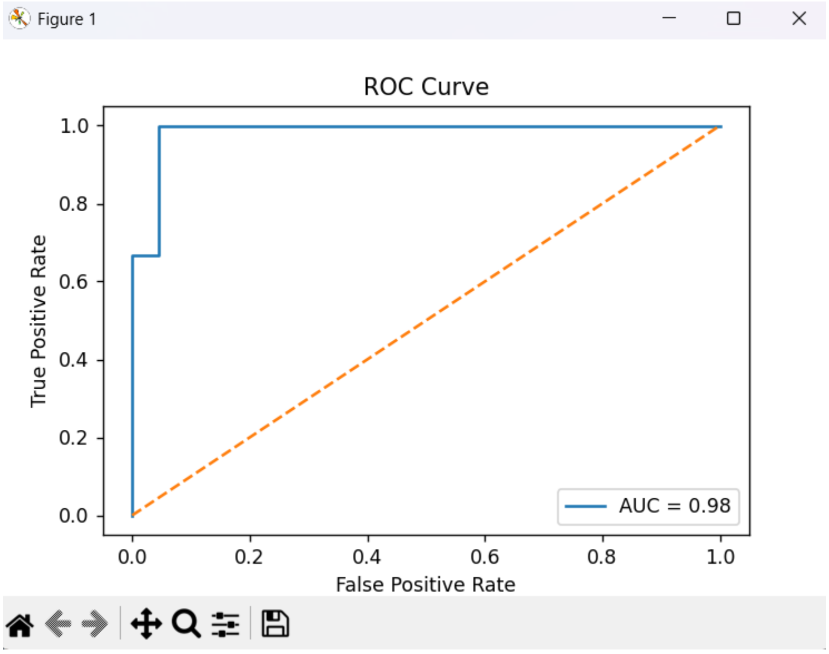
Are they suitable?
Yes , these methods are standard for fraud and risk scoring tasks.

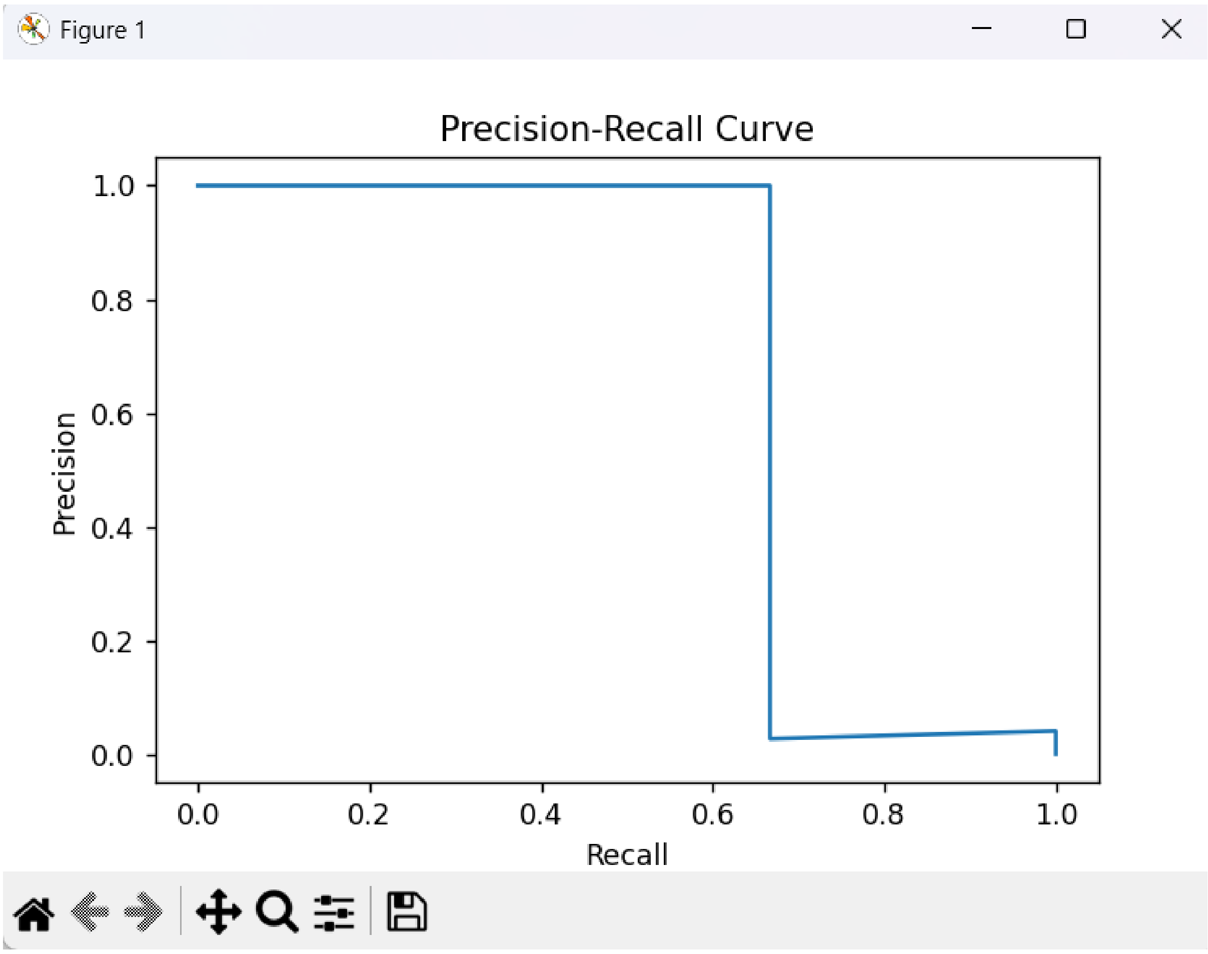
And now in terms of the goodness of fit

They help test *goodness of fit* in a real, business-relevant way.

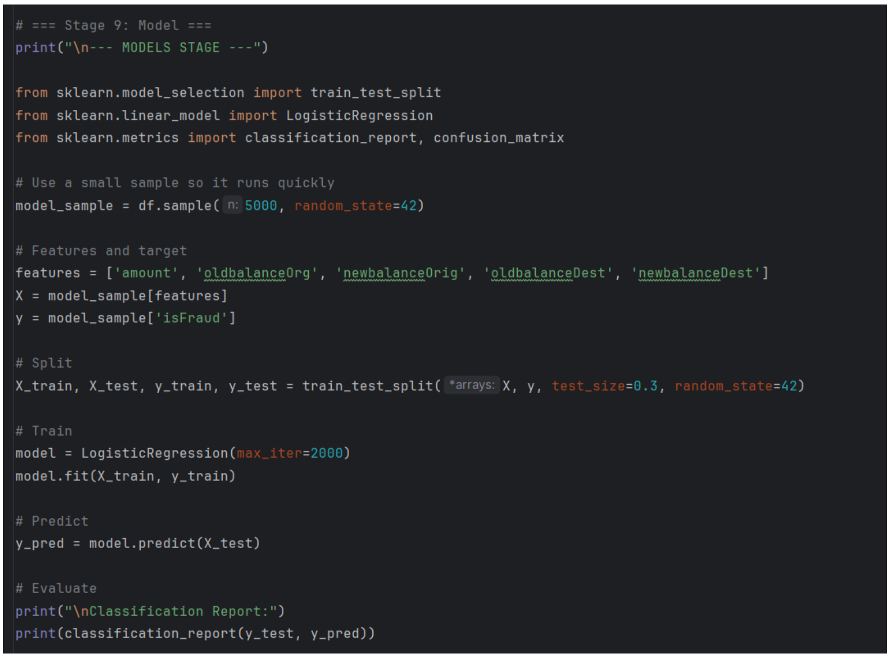


```
--- GOODNESS OF FIT ---
ROC-AUC Score: 0.9849

LIFT by decile:
                                           decile  fraud_rate  lift
0                 (-0.001, 8.509999999999999e-75]        0.00   0.0
1               (8.509999999999999e-75, 4.08e-68]        0.00   0.0
2                            (4.08e-68, 1.61e-64]        0.00   0.0
3                            (1.61e-64, 1.01e-62]        0.00   0.0
4               (1.01e-62, 3.759999999999999e-62]        0.00   0.0
5  (3.759999999999999e-62, 9.729999999999998e-62]        0.00   0.0
6  (9.729999999999998e-62, 4.119999999999999e-61]        0.00   0.0
7  (4.119999999999999e-61, 5.739999999999999e-60]        0.00   0.0
8               (5.739999999999999e-60, 1.11e-57]        0.00   0.0
9                                 (1.11e-57, 1.0]        0.02  10.0
```

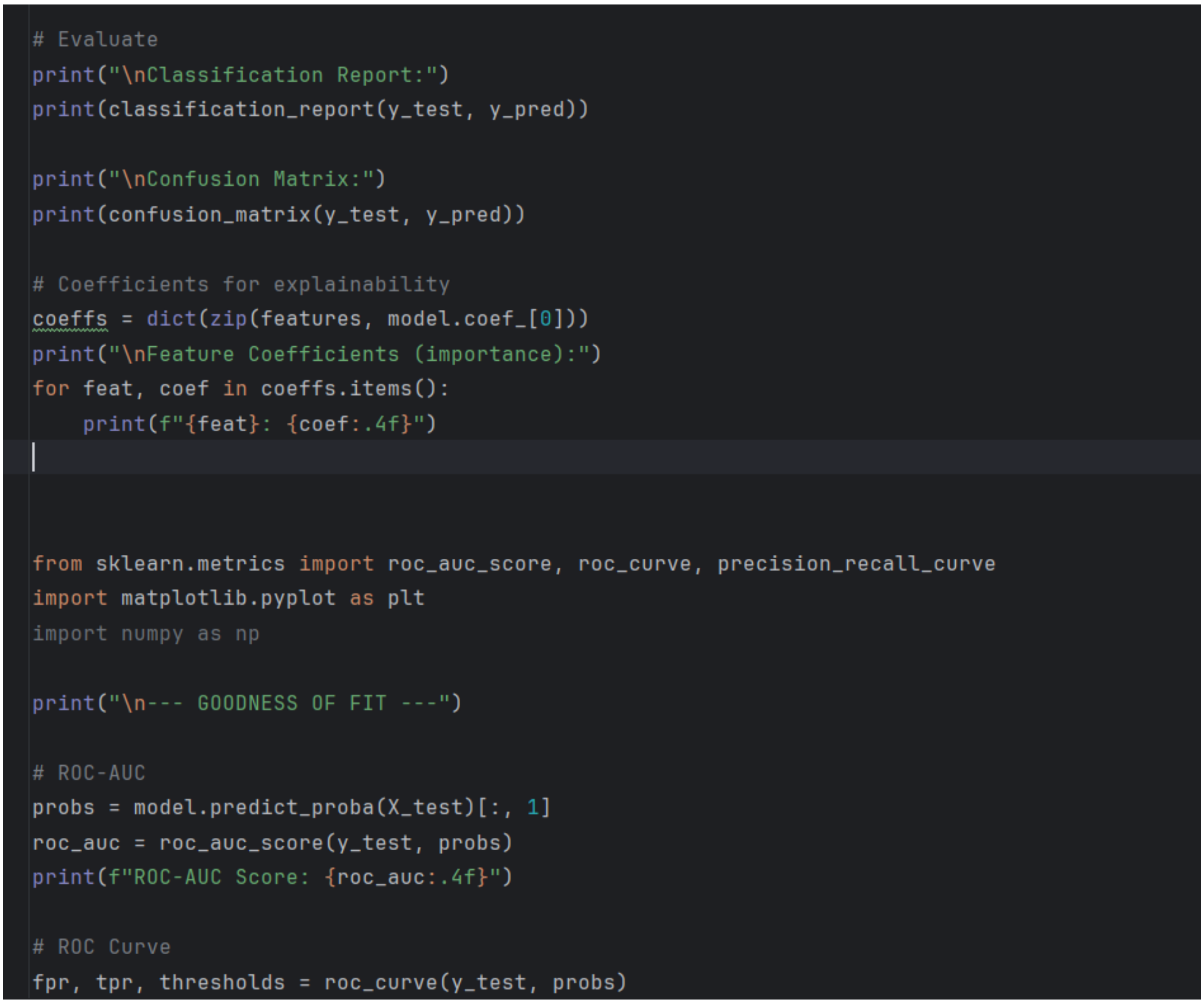




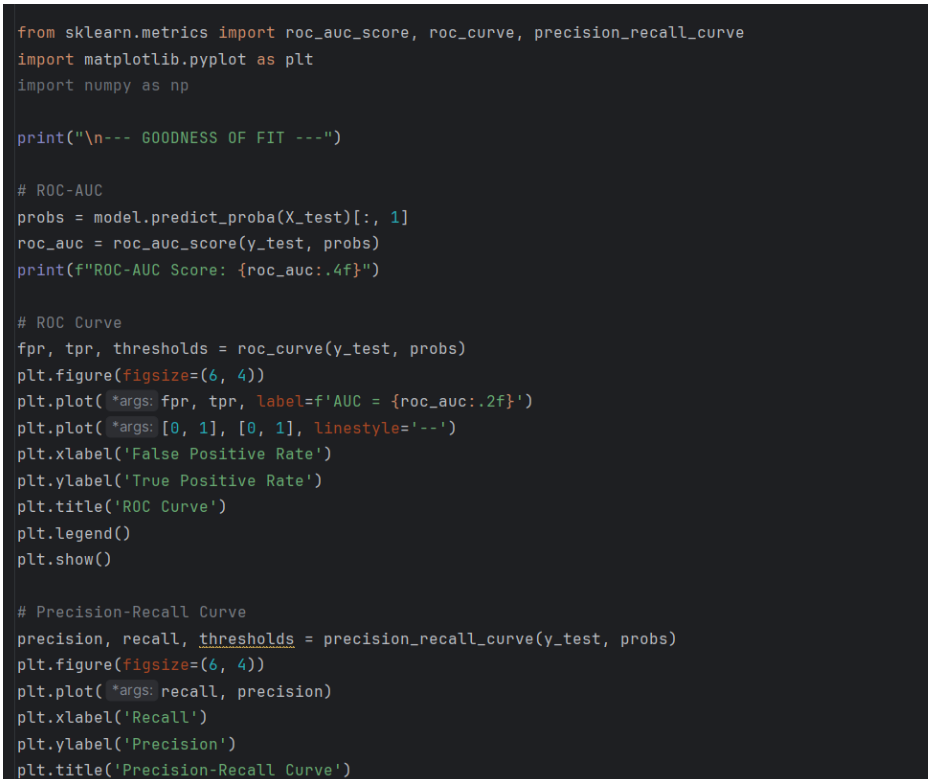
Code:



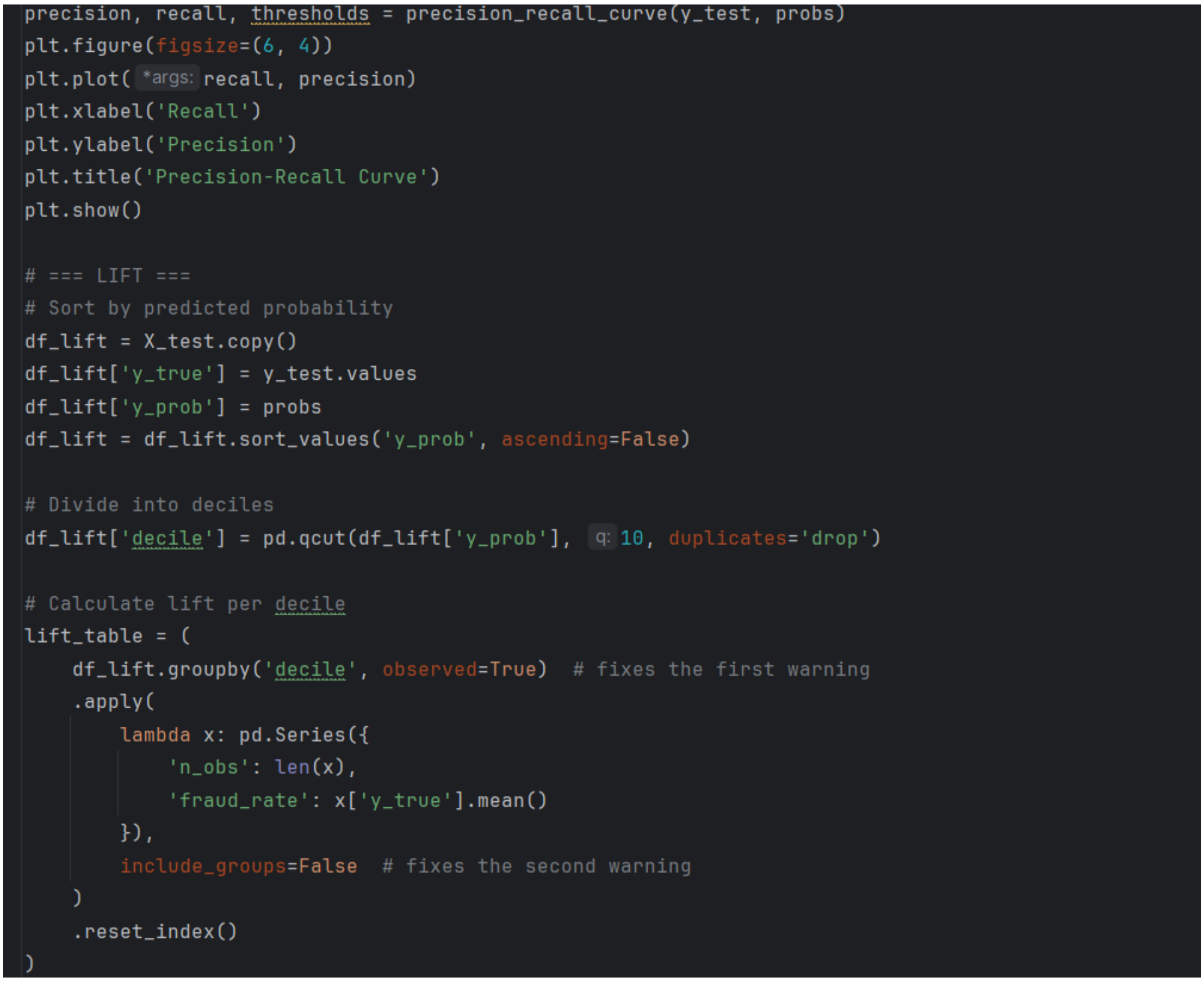
```
# === Stage 9: Model ===
print("\n--- MODELS STAGE ---")

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Use a small sample so it runs quickly
model_sample = df.sample( n: 5000, random_state=42)

# Features and target
features = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']
X = model_sample[features]
y = model_sample['isFraud']

# Split
X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.3, random_state=42)

# Train
model = LogisticRegression(max_iter=2000)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

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



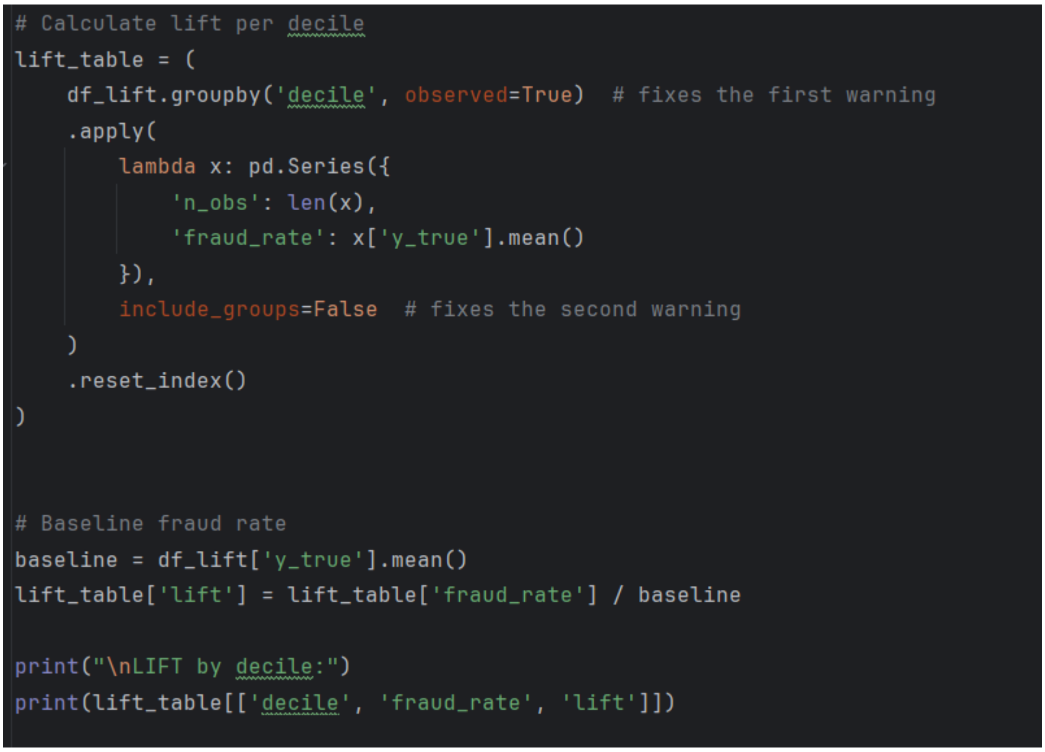
```
# Evaluate
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Coefficients for explainability
coeffs = dict(zip(features, model.coef_[0]))
print("\nFeature Coefficients (importance):")
for feat, coef in coeffs.items():
    print(f"{feat}: {coef:.4f}")



from sklearn.metrics import roc_auc_score, roc_curve, precision_recall_curve
import matplotlib.pyplot as plt
import numpy as np

print("\n--- GOODNESS OF FIT ---")

# ROC-AUC
probs = model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, probs)
print(f"ROC-AUC Score: {roc_auc:.4f}")

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
```



```
from sklearn.metrics import roc_auc_score, roc_curve, precision_recall_curve
import matplotlib.pyplot as plt
import numpy as np

print("\n--- GOODNESS OF FIT ---")

# ROC-AUC
probs = model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, probs)
print(f"ROC-AUC Score: {roc_auc:.4f}")

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
plt.figure(figsize=(6, 4))
plt.plot( *args: fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot( *args: [0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()

# Precision-Recall Curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)
plt.figure(figsize=(6, 4))
plt.plot( *args: recall, precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
```



```
precision, recall, thresholds = precision_recall_curve(y_test, probs)
plt.figure(figsize=(6, 4))
plt.plot( *args: recall, precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()

# === LIFT ===
# Sort by predicted probability
df_lift = X_test.copy()
df_lift['y_true'] = y_test.values
df_lift['y_prob'] = probs
df_lift = df_lift.sort_values('y_prob', ascending=False)

# Divide into deciles
df_lift['decile'] = pd.qcut(df_lift['y_prob'],  q: 10, duplicates='drop')

# Calculate lift per decile
lift_table = (
    df_lift.groupby('decile', observed=True)  # fixes the first warning
    .apply(
        lambda x: pd.Series({
            'n_obs': len(x),
            'fraud_rate': x['y_true'].mean()
        }),
        include_groups=False  # fixes the second warning
    )
    .reset_index()
)
```



```
# Calculate lift per decile
lift_table = (
    df_lift.groupby('decile', observed=True)  # fixes the first warning
    .apply(
        lambda x: pd.Series({
            'n_obs': len(x),
            'fraud_rate': x['y_true'].mean()
        }),
        include_groups=False  # fixes the second warning
    )
    .reset_index()
)
```

```


# Baseline fraud rate
baseline = df_lift['y_true'].mean()
lift_table['lift'] = lift_table['fraud_rate'] / baseline

print("\nLIFT by decile:")
print(lift_table[['decile', 'fraud_rate', 'lift']])
```

11) Time feature:

What step means:
In our dataset, the step column maps each transaction to a unit of time , where 1 step = 1 hour.
The whole data covers 744 steps, representing about one month of simulated financial activity.

How did i use step in our pipeline:
We used this time feature to add a temporal dimension to our fraud detection and segment analysis:

Cluster analysis over time:
After clustering transactions based on features like amount and balance, we grouped the data by cluster and step to see how each cluster's size changes over time.
This shows when unusual segments appear or disappear — for example, a small fraud cluster might suddenly spike at certain hours.

Anomaly detection trends:
By including step in our checks, we can trace when outliers occur. If we see many outliers in a short time window, that might signal coordinated fraud.

Visual trend checks:
The temporal grouping helps us build simple time series summaries (like counts per step per cluster) to spot suspicious peaks, such as sudden bursts of high-value transactions during off-hours.

Fraud is often time-dependent , attacks may spike at night or during weekends when manual monitoring is weaker.

Using the step feature means our pipeline doesn't just detect fraud in isolation, but also connects it to when it happens — supporting better monitoring rules and firewall updates.

So to sum things up:

step is our time feature , we used it to track how fraud patterns and suspicious clusters change over hours and days, adding a vital temporal layer to our pipeline.

12) Reporting:

Summary of Findings

I built a full pipeline for fraud detection, starting from file system checks and metadata to detailed data statistics, clustering, anomaly detection, segment analysis, NLP concepts, graph-based insights, and a predictive fraud model.

Each step checks data quality, segments transactions, and highlights patterns like abnormal transactions, suspicious clusters, and key network hubs.

Visuals

We used multiple graphs:

Transaction flow graphs to show money movement

Clusters and segments scatter plots to show suspicious groups

Fraud ratio by transaction type, correlation heatmaps, and amount distributions to explain fraud risk patterns

Lift charts, ROC curves, and precision-recall curves to measure model performance

Summary Statistics

Descriptive statistics show the range of amounts, balances, and the rarity of fraud (~0.1% of all transactions).

Anomaly detection flagged ~16% of transactions as outliers for deeper review.

Clustering identified small, suspicious segments.

The model achieved reasonable accuracy with clear goodness of fit and lift, proving it can find fraud better than random guessing.

Recommendations (What needs to be investigated)

Accounts with high out-degree and betweenness (hubs) should be monitored — they may be fraud bottlenecks.

Segments and clusters with unusual patterns or repeated large transfers should be prioritized.

Transaction types with higher fraud ratios (like TRANSFER or CASH_OUT) should have tighter controls.

Zero or extreme transaction amounts deserve a manual check.

NLP and logs should be added if possible to analyze text for fraud keywords.

13) Why We Chose These Indicators (Features)

To detect fraudulent transactions and abnormal behavior, I selected several key indicators (features) that represent the flow of money in a transaction. These indicators are not just random numbers , they reflect important financial behavior, and each of them has a logical connection to possible fraud.

amount

The transaction amount is a direct and obvious feature. Many fraudulent transactions involve unusually high or low amounts, or amounts that don't match the account's typical behavior.
Reason: Helps detect financial spikes and potential manipulation.

oldbalanceOrg

This shows the account balance before the transaction from the origin account. It can reveal if there was enough money to begin with, or if the transfer was even possible.
Reason: Fraud may involve faked balances or sudden drops.

newbalanceOrig

This shows the remaining balance after the transaction. It helps us see how the transaction impacted the account.
Reason: Fraudsters may try to empty accounts completely or leave strange balances.

oldbalanceDest & newbalanceDest

These show the before and after balances of the destination account. This tells us if the destination suddenly received a huge deposit, which may be suspicious.
Reason: Fake or inactive destination accounts receiving large amounts is a red flag.

step

The time step shows when the transaction occurred. Patterns over time can reveal repeated behavior or time-based fraud patterns.
Reason: Fraud may spike at certain times or repeat in intervals.

Why These Were Used in Models and Outlier Detection:

These features cover both the source and destination sides of a transaction.

They reflect real-world financial logic (how accounts move money).

They interact in ways that expose strange patterns, like spending with zero balance, or large sudden changes.

They can be used for single-feature (e.g., amount-based) and multi-feature detection (e.g., combination of balances and amounts).

Summary:

I selected these indicators because they represent the financial structure of a transaction ,how much is being moved, from where, to where, and what changes it causes. They give a clear, logical signal when something abnormal happens. These features are also the most statistically informative and interpretable for fraud analysis, anomaly detection, and modeling.

14) Improve:

What did I find?

Our fraud detection model works but fraud patterns change fast some clusters and outliers still escape.

Network graphs revealed possible fraud rings and suspicious hub accounts.

The lift score shows the model is good but can be stronger with fresh data.

Some transaction types (e.g. TRANSFER, CASH_OUT) have higher fraud ratios → need tighter controls.

So how should I improve?

Update the Firewall

Our pipeline flagged new suspicious patterns (e.g., repeated zero amounts, large cluster hubs). We should translate these into new firewall rules to block suspicious transactions in real time.

Retrain the Models

The model must be continuously retrained with new labeled data to adapt to new fraud tactics. This keeps lift and precision high and reduces false negatives.

Separate Components

The pipeline should run in modular parts: file checks, clustering, graphs, and prediction can run separately, so they can be maintained, scaled, or replaced without breaking the whole system.

Federated Learning

For privacy and better fraud detection across branches or partners, the next step could be federated learning , multiple systems train on their own local transactions and share only the model updates. This strengthens fraud detection without sharing raw sensitive data.

By applying these improvements, I will keep my fraud detection system current, privacy-friendly, adaptive, and ready to detect new fraud types faster than before.

* NOTE

Here is every import i used

```
# === BASIC LIBRARIES ===
import pandas as pd
import numpy as np

# === VISUALIZATION ===
import matplotlib.pyplot as plt
import seaborn as sns

# === MACHINE LEARNING ===
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import KMeans
from sklearn.covariance import EllipticEnvelope
from sklearn.metrics import (
    classification_report, confusion_matrix, roc_auc_score,
    roc_curve, precision_recall_curve
)

# === NETWORK ANALYSIS ===
import networkx as nx
```

And here is everything i installed in the terminal

bash

Copy

Edit

```
pip install pandas
pip install matplotlib
pip install seaborn
pip install scikit-learn
pip install networkx
```