Fraud Detection & Analysis for Financial Security Using Machine Learning

PROJECT INVOLVES:
Importing Packages
Loading Dtaset & Cleaning
Exploratary Data Analysis
Feature Selection & Engineering
Model Selection & Training
Predictive Model

Y.RAKESH
LINKEDIN:https://www.linkedin.com/in/rakeshyrc

Importing Packages

In [311…

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
import warnings, copy
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap
from scipy.stats import chi2_contingency
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
import lightgbm as lgb
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score, StratifiedKI
from sklearn.metrics import f1_score, precision_score, recall_score, roc_auc_score
```

loading data

In [320…

```
# Load data
data = pd.read_csv("/content/onlinefraudsmall.csv")
print(data.shape)
data.head(1)
```

```
(1048575, 10)
```

Out[320]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0


data cleaning
&
PREPARATION

analysis and cleaning

In [321…

```
data.describe().round(3)
data.info()
data.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 10 columns):
 #   Column          Non-Null Count    Dtype  
---  ------          --------------    -----  
 0   step            1048575 non-null  int64  
 1   type            1048575 non-null  object 
 2   amount          1048575 non-null  float64
 3   nameOrig        1048575 non-null  object 
 4   oldbalanceOrg   1048575 non-null  float64
 5   newbalanceOrig  1048575 non-null  float64
 6   nameDest        1048575 non-null  object 
 7   oldbalanceDest  1048575 non-null  float64
 8   newbalanceDest  1048575 non-null  float64
 9   isFraud         1048575 non-null  int64  
dtypes: float64(5), int64(2), object(3)
memory usage: 80.0+ MB
```

Out[321]:

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
dtype: int64
```

In [322…

```
# Drop rows with missing values
data.dropna(subset=['isFraud'], inplace=True)
# Convert "isFraud" columns from float to int
data['isFraud'] = data['isFraud'].astype(int)
```

In [323…

```
data.describe().round(3)
data.info()
data.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 10 columns):
 #   Column          Non-Null Count    Dtype  
---  ------          --------------    -----  
 0   step            1048575 non-null  int64  
 1   type            1048575 non-null  object 
 2   amount          1048575 non-null  float64
 3   nameOrig        1048575 non-null  object 
 4   oldbalanceOrg   1048575 non-null  float64
 5   newbalanceOrig  1048575 non-null  float64
 6   nameDest        1048575 non-null  object 
 7   oldbalanceDest  1048575 non-null  float64
 8   newbalanceDest  1048575 non-null  float64
 9   isFraud         1048575 non-null  int64  
dtypes: float64(5), int64(2), object(3)
memory usage: 80.0+ MB
```

Out[323]:

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
dtype: int64
```

In [324…

```
data.describe()
```

Out[324]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06
mean	2.696617e+01	1.586670e+05	8.740095e+05	8.938089e+05	9.781600e+05	1.114198e+06	1.089097e-03
std	1.562325e+01	2.649409e+05	2.971751e+06	3.008271e+06	2.296780e+06	2.416593e+06	3.298351e-02
min	1.000000e+00	1.000000e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.500000e+01	1.214907e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	2.000000e+01	7.634333e+04	1.600200e+04	0.000000e+00	1.263772e+05	2.182604e+05	0.000000e+00
75%	3.900000e+01	2.137619e+05	1.366420e+05	1.746000e+05	9.159235e+05	1.149808e+06	0.000000e+00
max	9.500000e+01	1.000000e+07	3.890000e+07	3.890000e+07	4.210000e+07	4.220000e+07	1.000000e+00

The average step is 23.98 hours. The average amount is 162,426.70. The average oldbalanceOrg is 884,346.10. The average newbalanceOrig is 905,079.70. The average oldbalanceDest is 987,699.90. The average newbalanceDest is 1,131,526.00. The percentage of fraudulent transactions is 0.054%.

- The average amount of a fraudulent transaction is much higher than the average amount of a non-fraudulent transaction. This is something that we will need to keep in mind when we build our machine learning model.

In [325…

```
# Check duplicate values
data=data
data.duplicated().sum()
```

Out[325]:

```
0
```

In [328…

```
data.head()
```

Out[328]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0

step: This column represents the time in hours since the start of the dataset. It is not necessary for fraud detection. nameDest: This column is the name of the recipient of the transaction. It is not necessary for fraud detection. oldbalanceDest: This column is the balance of the recipient's account before the transaction. It is not necessary for fraud detection.

Now data look clean and now can do the EDA to gain few insights from the data

EDA

In [406…

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

feature=['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']

for i in feature:
```


EXPLORATORY DATA
ANALYSIS

In [325...

```
# Check duplicate values
data=data
data.duplicated().sum()
```

Out[325]:

```
0
```

In [328...

```
data.head()
```

Out[328]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFrau
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	

step: This column represents the time in hours since the start of the dataset. It is not necessary for fraud detection. nameDest: This column is the name of the recipient of the transaction. It is not necessary for fraud detection. oldbalanceDest: This column is the balance of the recipient's account before the transaction. It is not necessary for fraud detection.

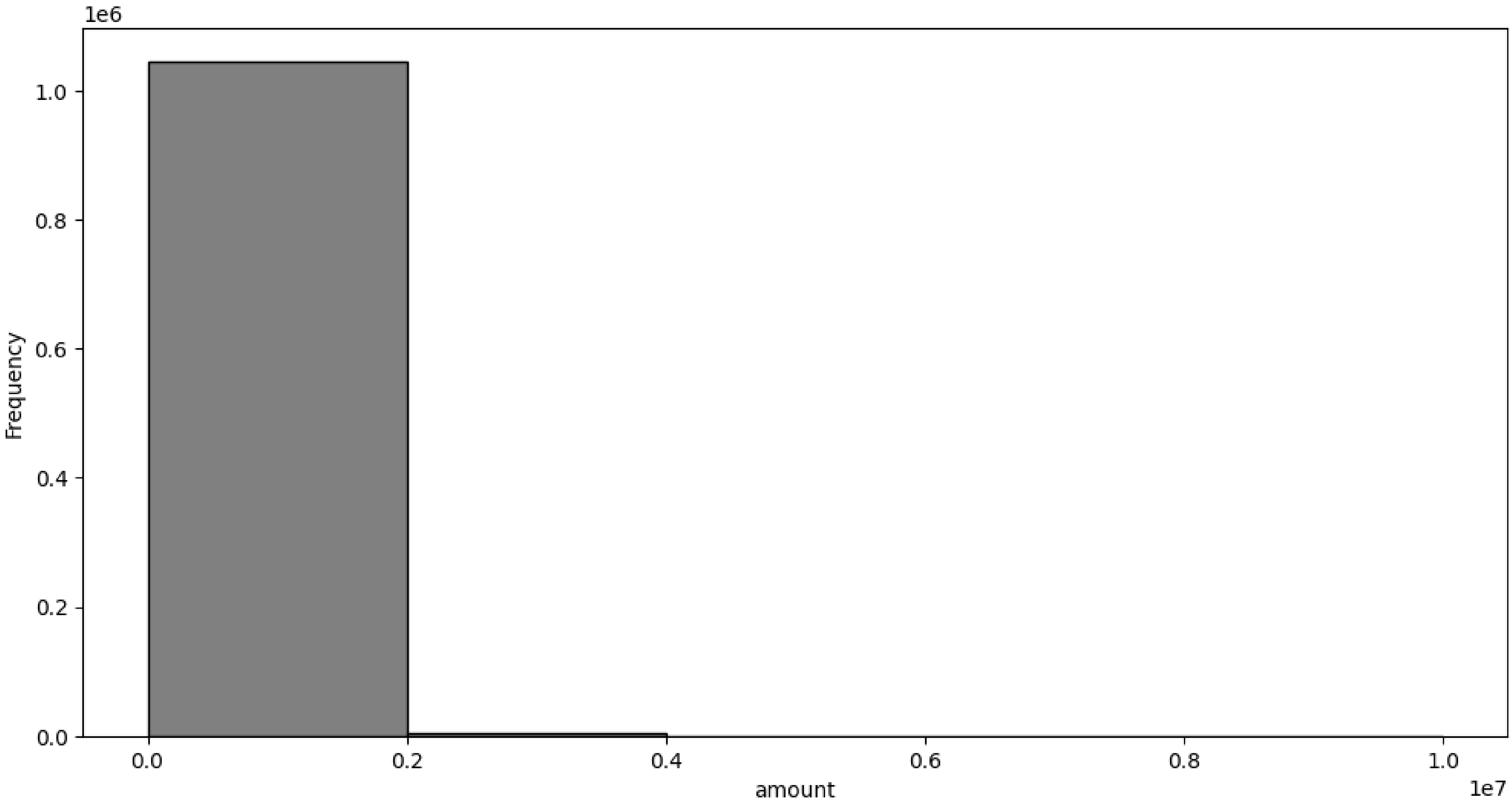
Now data look clean and now can do the EDA to gain few insights from the data

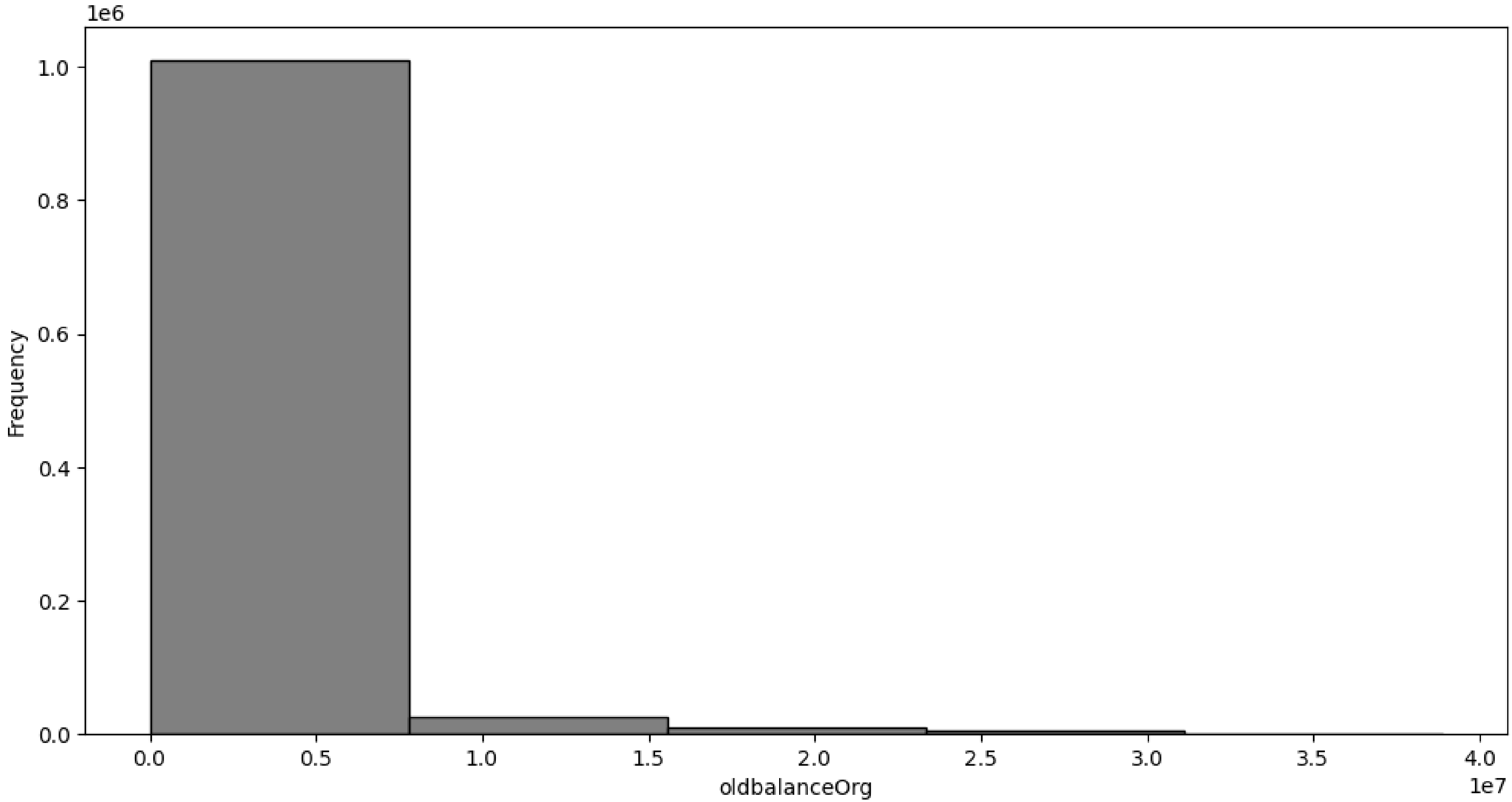
EDA

In [406...

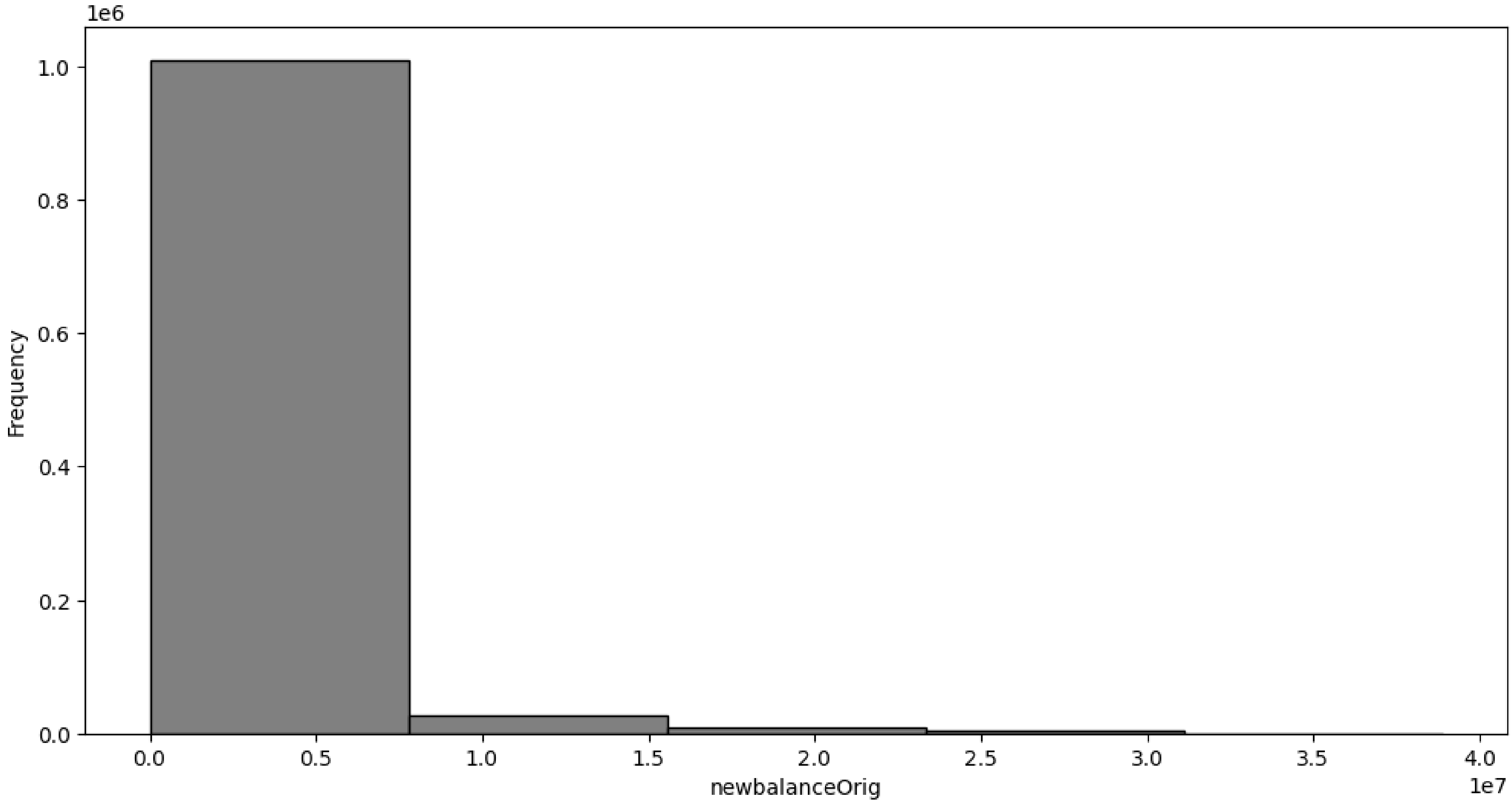
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

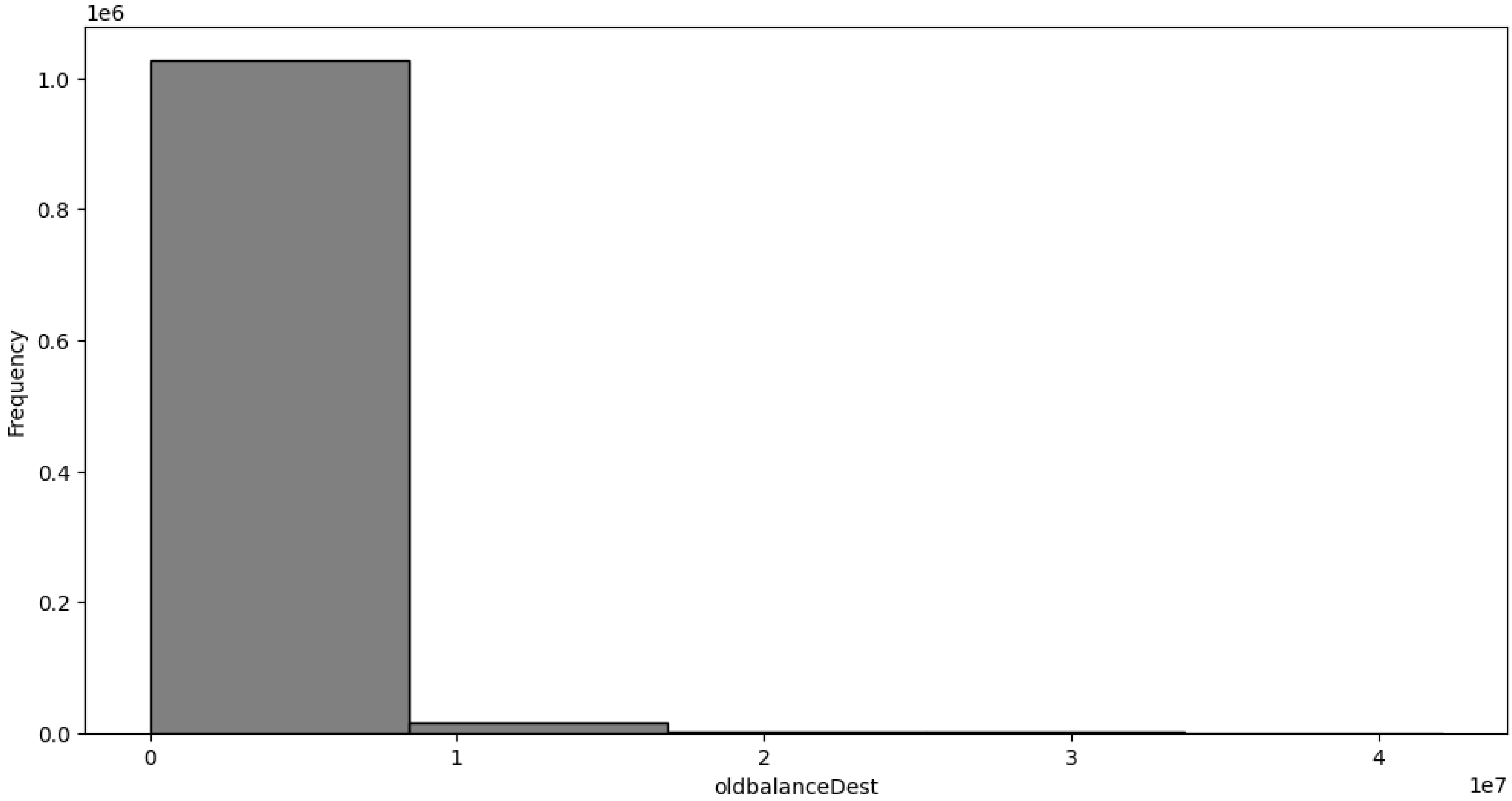
```
feature=['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']

for i in feature:
    plt.xlabel(i)
    data[i].plot(kind='hist', bins=5, figsize=(12,6), facecolor='grey',edgecolor='black')
    plt.show()
```



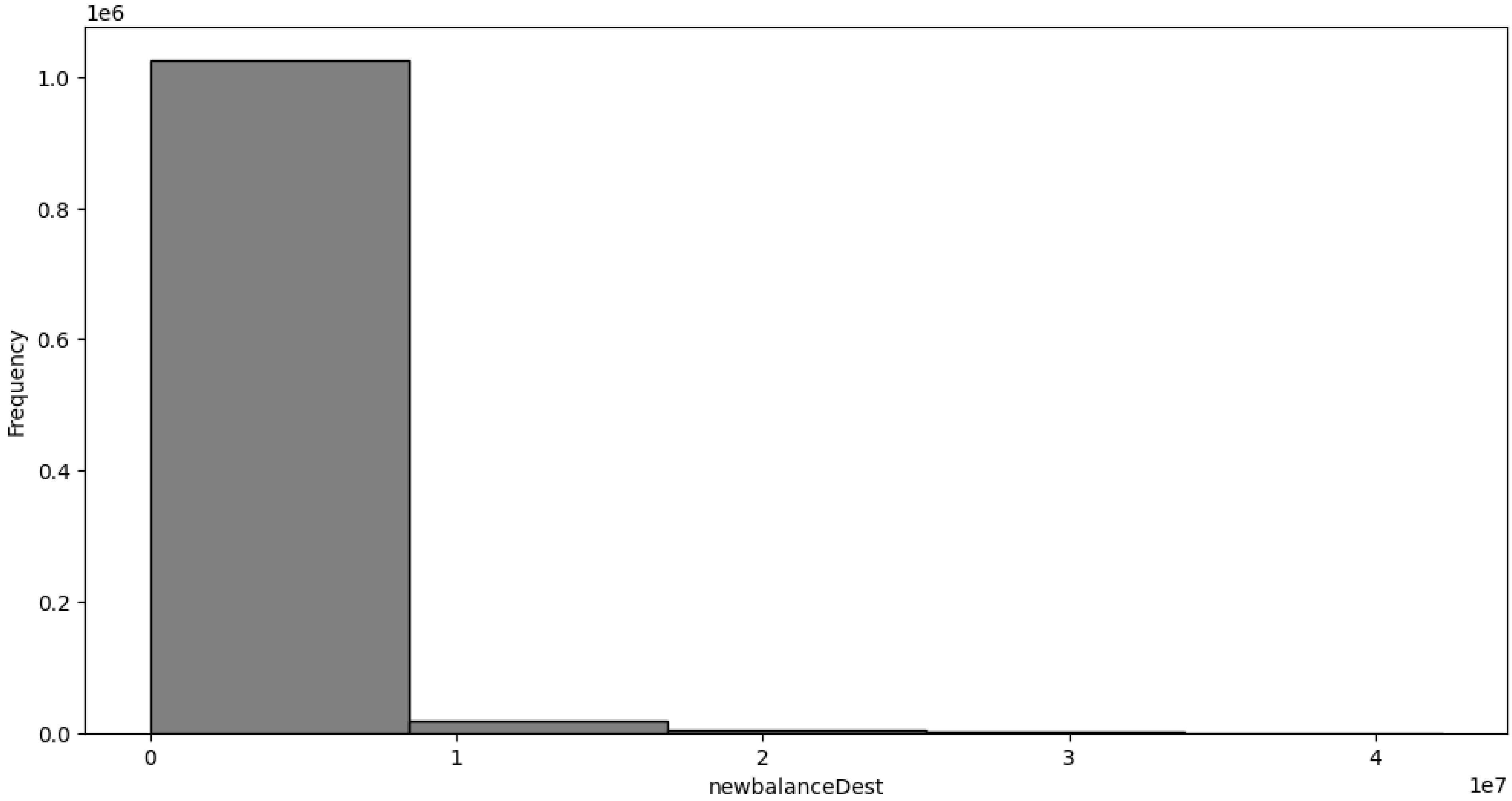


1e6
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1e7
oldbalanceOrg


1e6
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1e7
newbalanceOrig



1e6
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
0
1
2
3
4
oldbalanceDest
1e7



- By this plot we can see the distbunce in data due to outliers
- In our case or data it is better to go with capping & flooring than removing data.

In [407…

```
feature=['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']

for i in feature:
    print(i)
    print(data[i].quantile(0.10))
    print(data[i].quantile(0.90))
```

```
    print('\n')
```

```
amount
4220.57
373075.3779999999


oldbalanceOrg
0.0
1924613.1739999996


newbalanceOrig
0.0
2059503.9359999998


oldbalanceDest
0.0
2721593.4459999995


newbalanceDest
0.0
3102896.2
```

In [408…

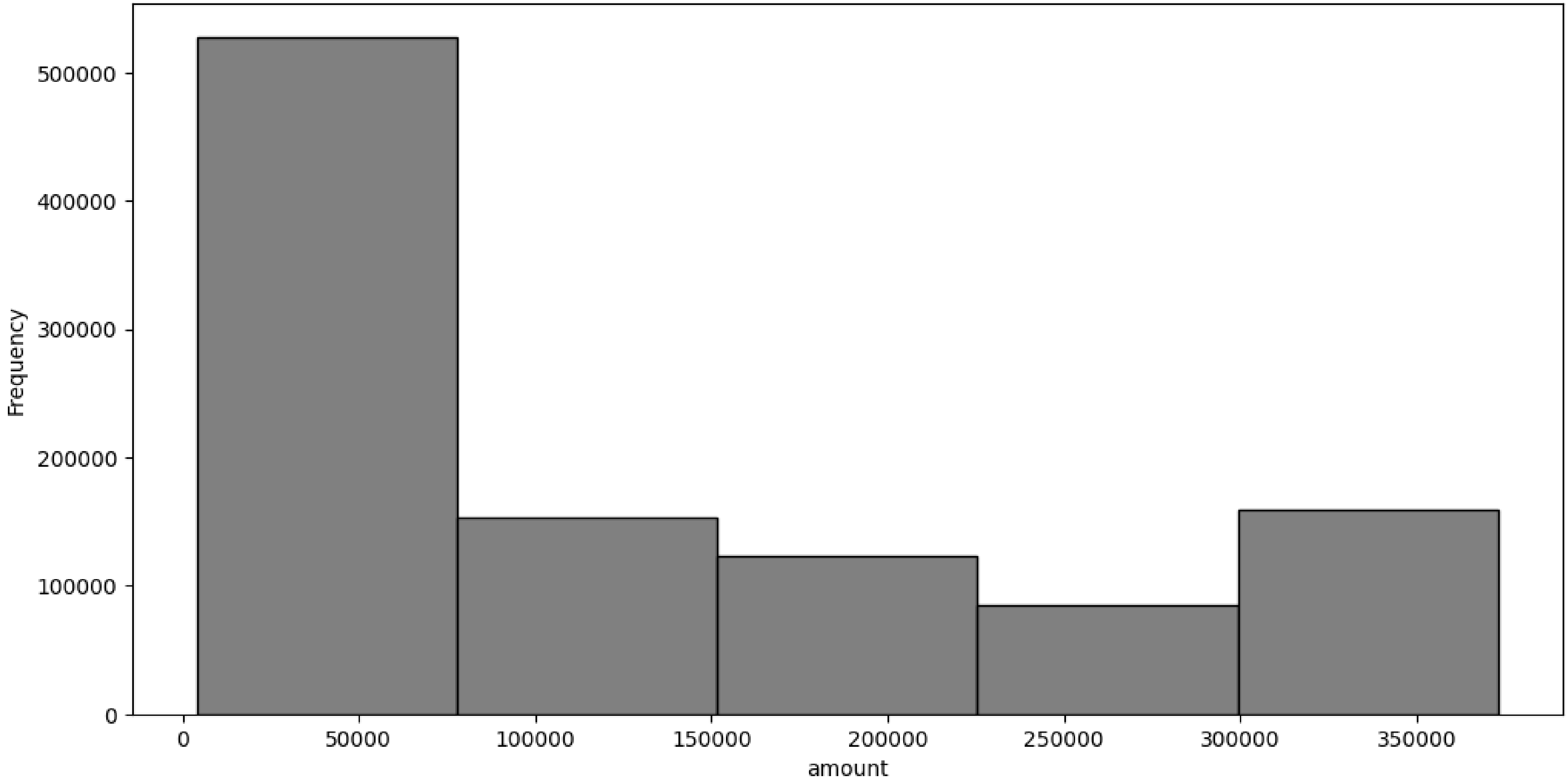
```
feature=['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']

for i in feature:
    lower = data[i].quantile(0.10)
    upper = data[i].quantile(0.90)
    data[i] = np.where(data[i] <lower, lower,data[i])
    data[i] = np.where(data[i] >upper, upper,data[i])
    print('Feature: ',i)
    print('Skewness value: ',data[i].skew())
    print('\n')
```

```
Feature:  amount
Skewness value:  0.7964930444208819


Feature:  oldbalanceOrg
Skewness value:  2.1881516694642875


Feature:  newbalanceOrig
Skewness value:  2.1770209559093807


Feature:  oldbalanceDest
Skewness value:  1.385454781137203


Feature:  newbalanceDest
Skewness value:  1.3046789943177446
```

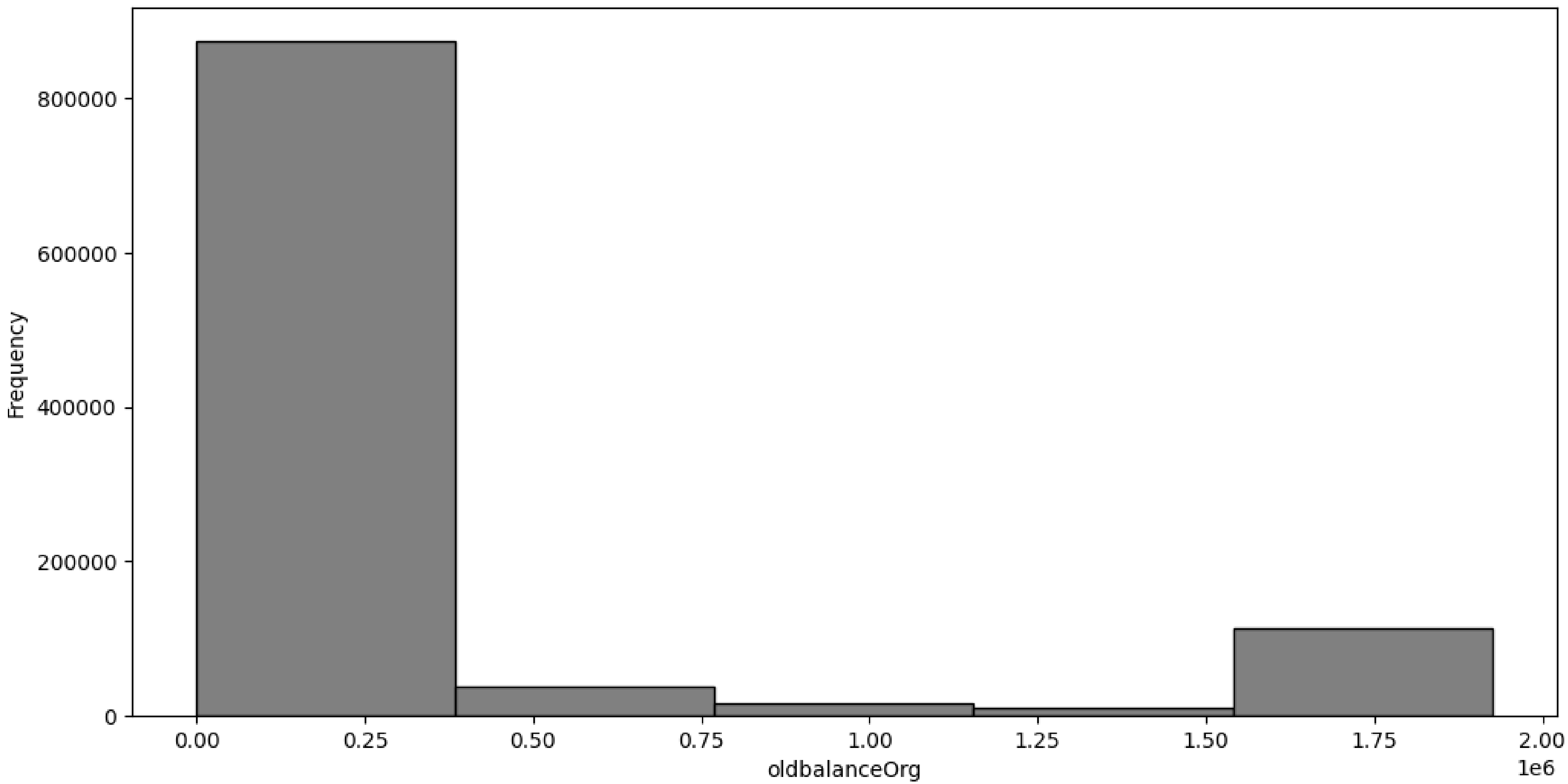
now lets look into the data by ploting after dealing with the outliers

In [409…

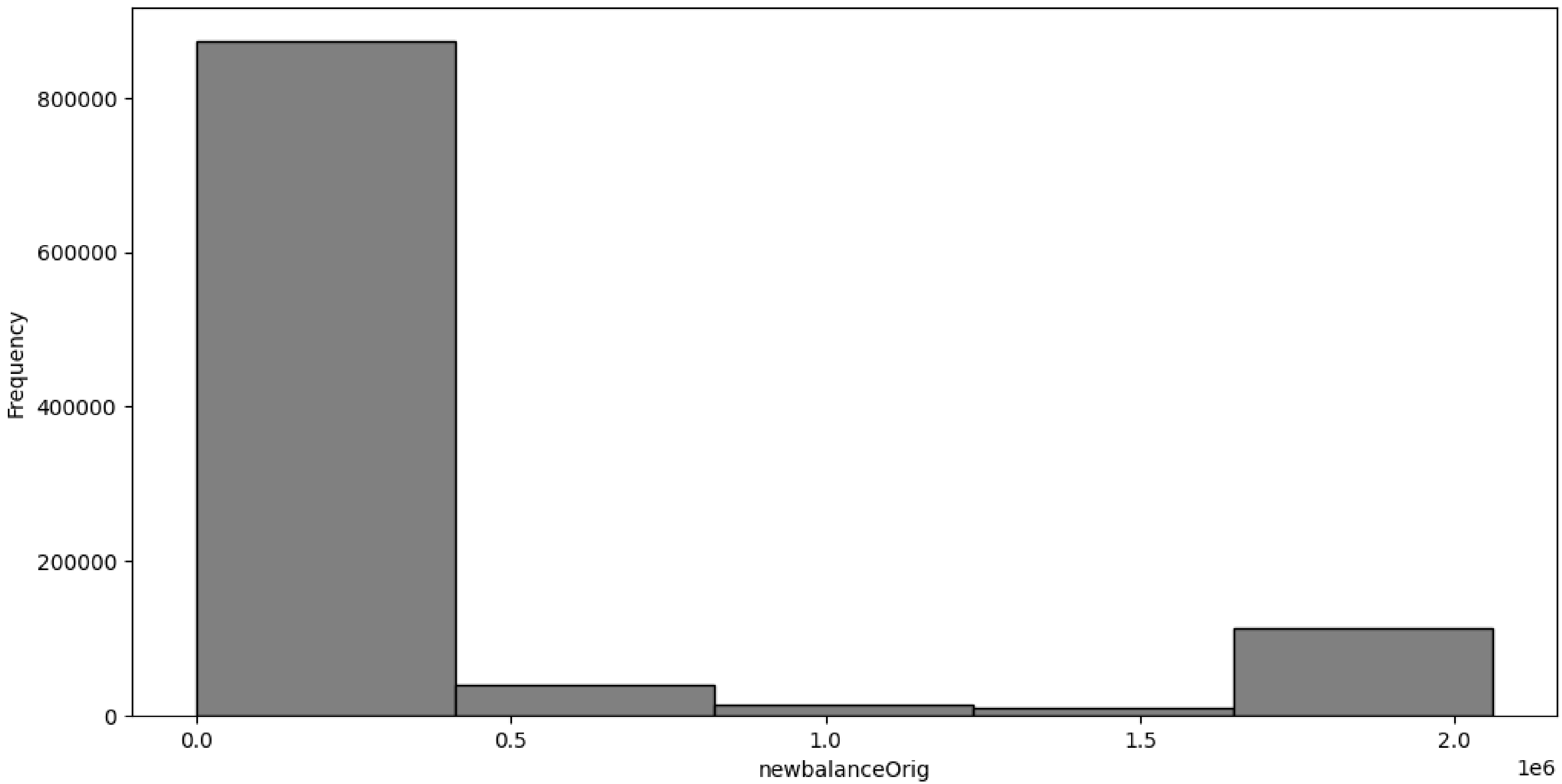
```
feature=['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']

for i in feature:
    plt.xlabel(i)
    data[i].plot(kind='hist', bins=5, figsize=(12,6), facecolor='grey',edgecolor='black')
    plt.show()
```



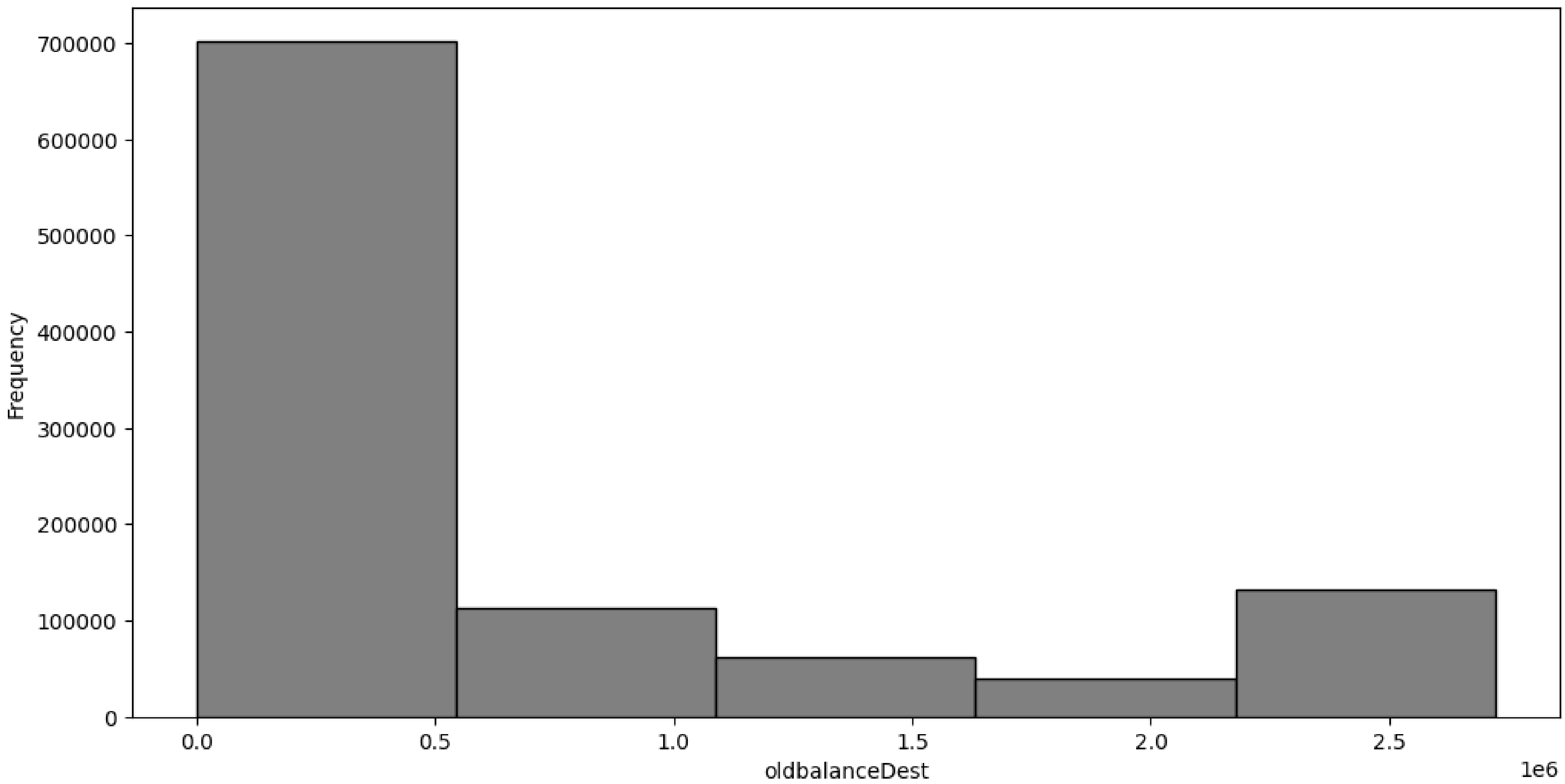
Frequency
500000
400000
300000
200000
100000
0
0
50000
100000
150000
200000
250000
300000
350000
amount

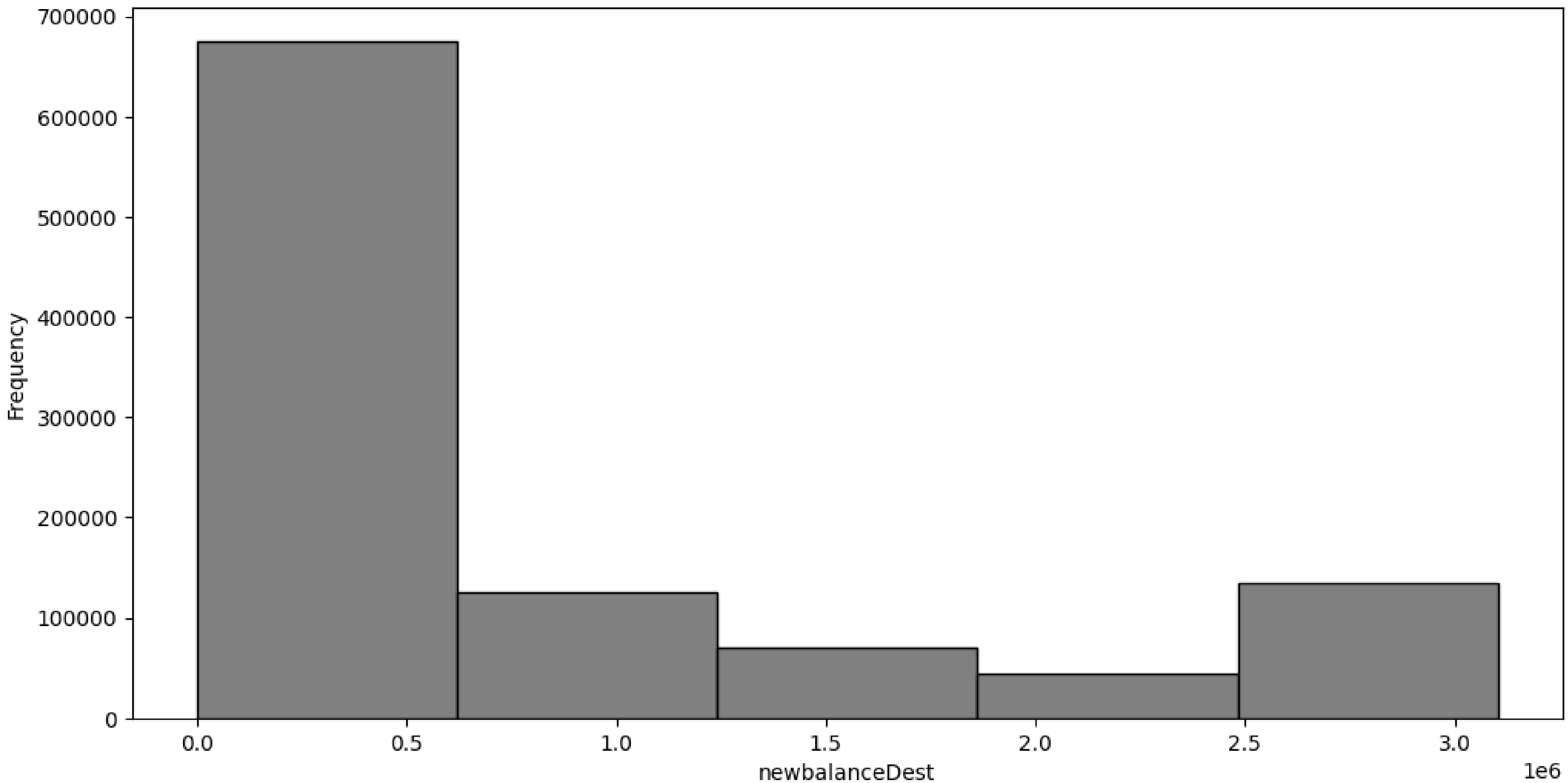


800000
600000
400000
200000
0
Frequency
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e6
oldbalanceOrg



800000
600000
400000
200000
0
Frequency
0.0
0.5
1.0
1.5
2.0
1e6
newbalanceOrig


700000
600000
500000
400000
300000
200000
100000
0
Frequency
0.0
0.5
1.0
1.5
2.0
2.5
1e6
oldbalanceDest



In [279…

```
data.type.unique()
```

Out[279]:

```
array(['PAYMENT', 'TRANSFER', 'CASH_OUT', 'DEBIT', 'CASH_IN'],
      dtype=object)
```

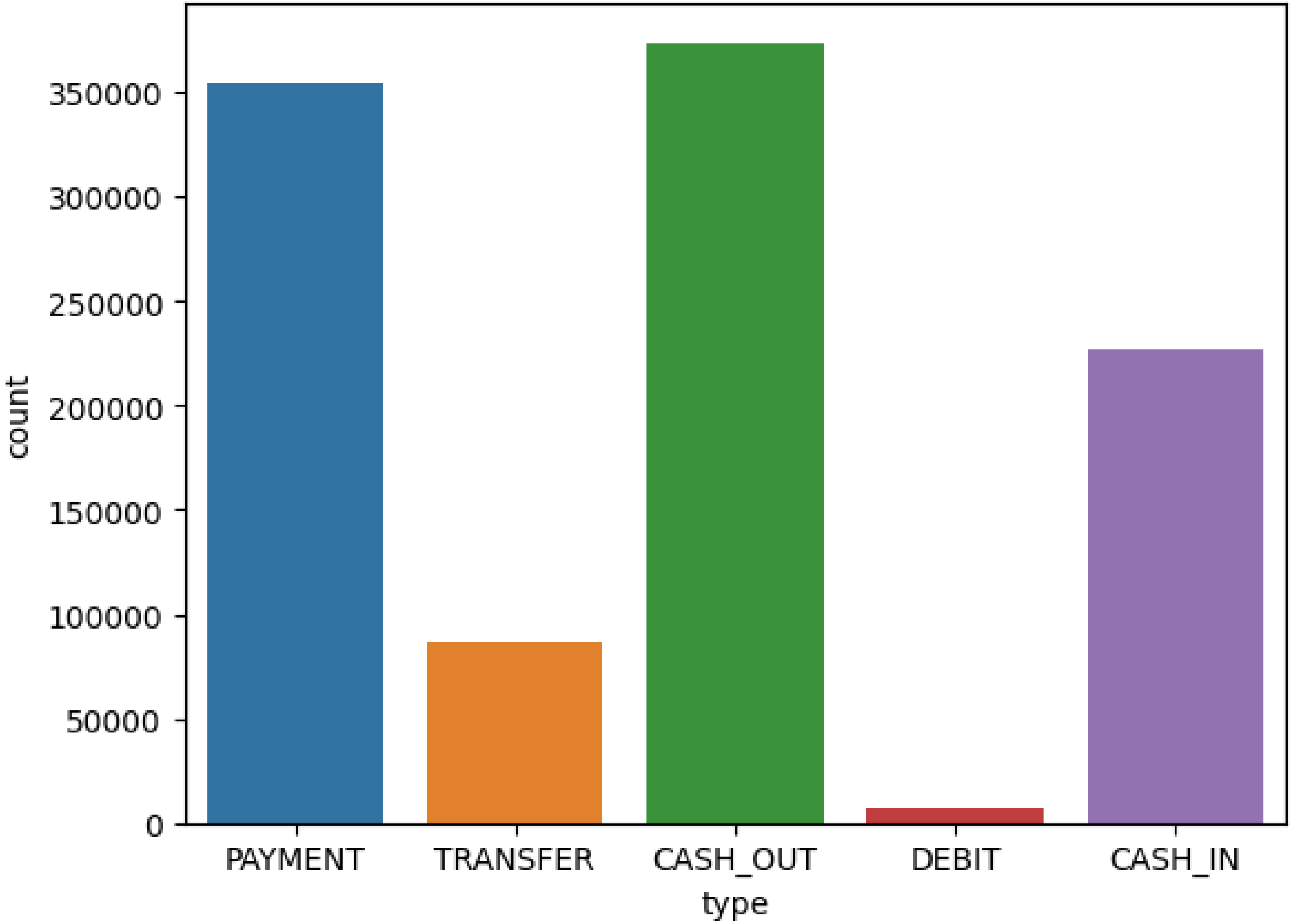
In [280…

```
payment = data.type.value_counts()
payment
```

Out[280]:
```
CASH_OUT    373641
PAYMENT     353873
CASH_IN     227130
TRANSFER     86753
DEBIT         7178
Name: type, dtype: int64
```

In [282…
```
sns.countplot(x='type', data=data)
```

Out[282]:
```
<Axes: xlabel='type', ylabel='count'>
```



- by the above visulization we can see the most transactions take place by CASH_OUT followed by PAYMENT
    - But this is not the usefull thing because we need to find relationship with type column with fraud

In [283…

```
pivot_=pd.crosstab(index=data.type,columns=data.isFraud)
pivot_
```

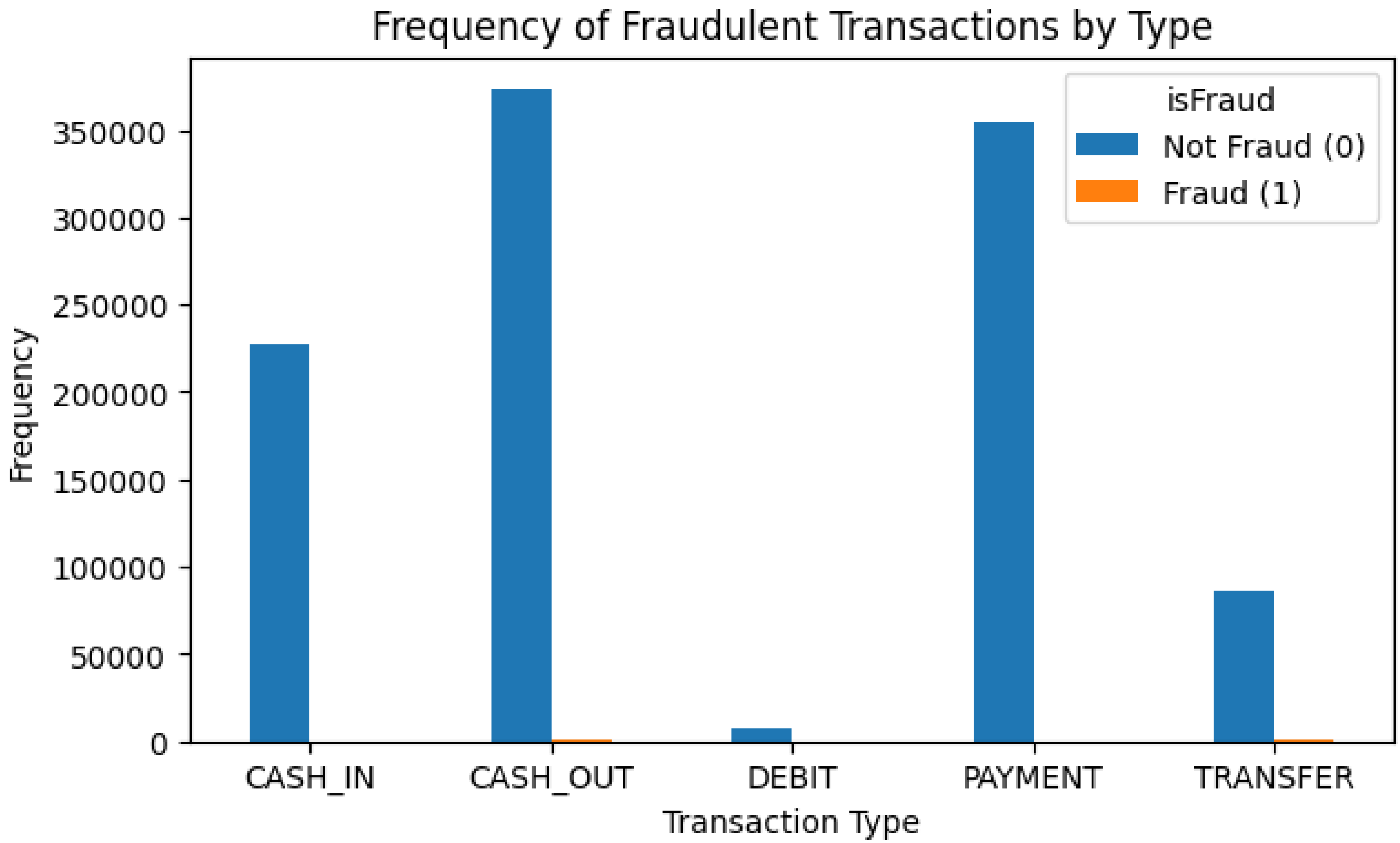
Out[283]:

isFraud	0	1
type		
CASH_IN	227130	0
CASH_OUT	373063	578
DEBIT	7178	0
PAYMENT	353873	0
TRANSFER	86189	564

In [286…

```
import matplotlib.pyplot as plt

plt.figure(figsize=(7, 4))
pivot_.plot.bar( figsize=(7, 4), rot=0)
plt.title('Frequency of Fraudulent Transactions by Type')
plt.xlabel('Transaction Type')
plt.ylabel('Frequency')
plt.legend(title='isFraud', labels=['Not Fraud (0)', 'Fraud (1)'])
plt.show()
```

```
<Figure size 700x400 with 0 Axes>
```

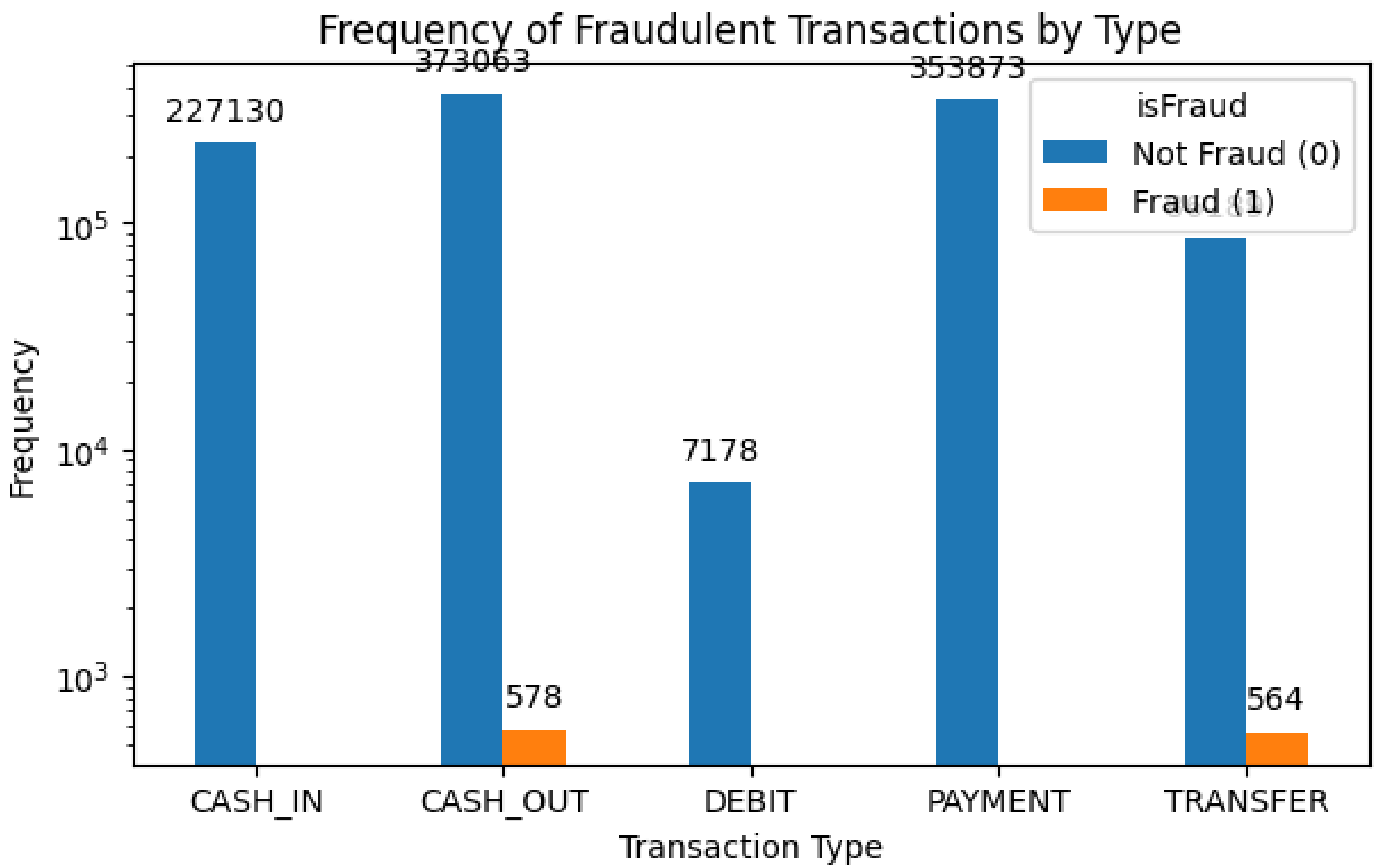


the numbers indicating fraud happened with fraud not happened is very low so apply logscale to visullize good

In [287…

```
import matplotlib.pyplot as plt

# Create the bar plot
plt.figure(figsize=(10, 6))
ax = pivot_.plot.bar(logy=True,figsize=(7, 4), rot=0, ax=plt.gca())

# Annotate each bar with its respective frequency value
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 10), textcoords='offset points')
```

```
# Add a title and labels to the plot
plt.title('Frequency of Fraudulent Transactions by Type')
plt.xlabel('Transaction Type')
plt.ylabel('Frequency')
plt.legend(title='isFraud', labels=['Not Fraud (0)', 'Fraud (1)'])
plt.show()
```



- Now after applying logscale and labling we can we can visullize it good
  - From this we can say it is unbalanced data
  - only in Cashout and Transfer we can see the fraud

In [288…

```
pivot_
```

Out[288]:

isFraud	0	1
type		
CASH_IN	227130	0
CASH_OUT	373063	578
DEBIT	7178	0
PAYMENT	353873	0
TRANSFER	86189	564

In [289…

```
cashout=        373063+578
cashout_fraud= 578/(cashout) * 100
cashout_fraud
```

Out[289]:

```
0.15469394418706728
```

In [290…

```
transfer= 86189+564
transfer_fraud = 564/(transfer) * 100
transfer_fraud
```

Out[290]:

```
0.6501216096273328
```

from the both this above observation

- 0.154% of frauds happend in total at cashout
- 0.650% of frauds happen in total at transfer

in this both cases the % of fraud occured is very less

In [291…

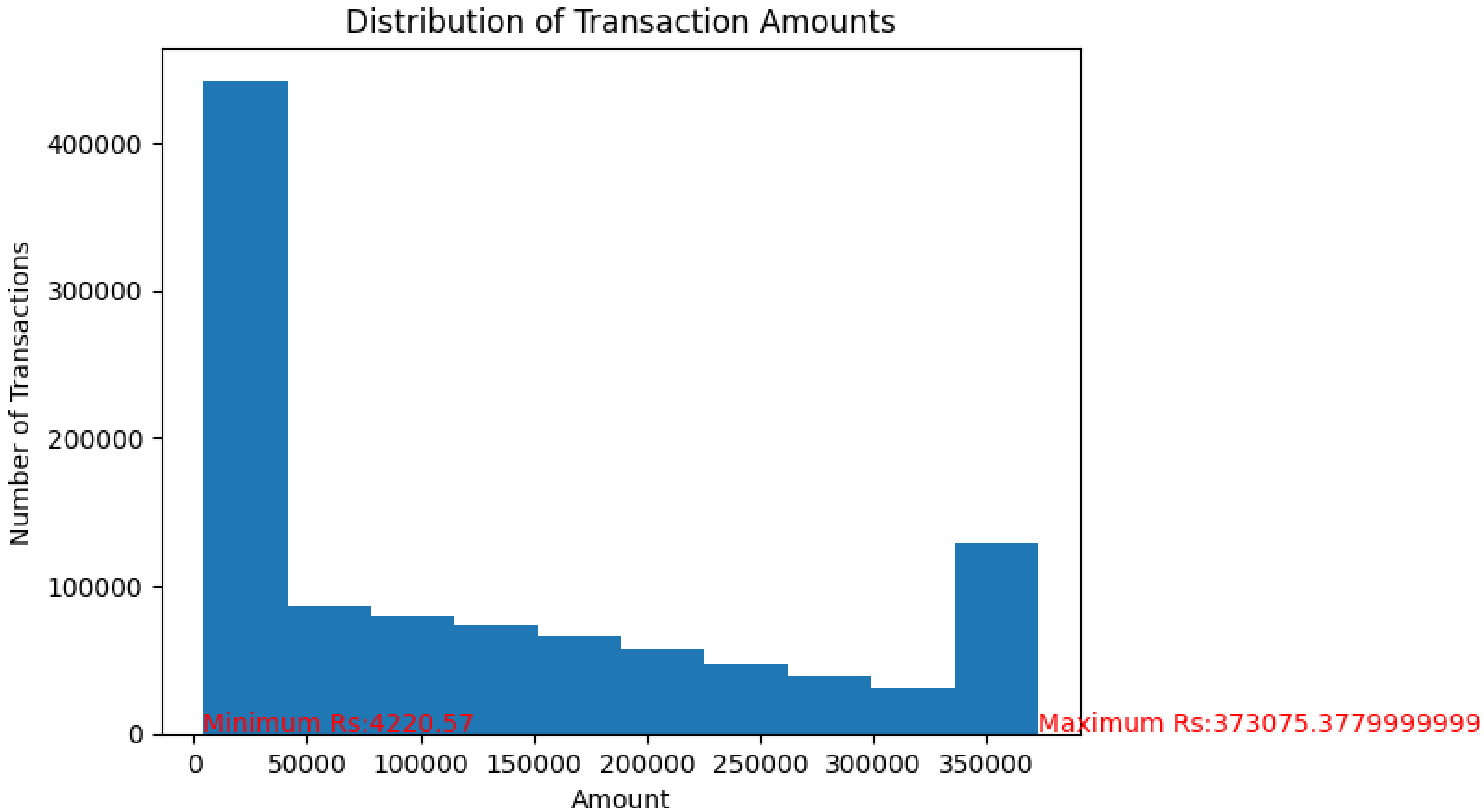
```
data.amount
```

Out[291]:
```
0            9839.64
1            4220.57
2            4220.57
3            4220.57
4           11668.14
              ...
1048570    132557.35
1048571      9917.36
1048572     14140.05
1048573     10020.05
1048574     11450.03
Name: amount, Length: 1048575, dtype: float64
```

In [292…
```
print('Minimum: ',data.amount.min())
print('Maximum: ',data.amount.max())
```

```
Minimum:  4220.57
Maximum:  373075.3779999999
```

In [293…
```
import matplotlib.pyplot as plt

plt.hist(data.amount)
plt.title('Distribution of Transaction Amounts')
plt.xlabel('Amount')
plt.ylabel('Number of Transactions')
plt.annotate('Minimum Rs:' + str(data.amount.min()), (data.amount.min(), 1000), color='red')
plt.annotate('Maximum Rs:' + str(data.amount.max()), (data.amount.max(), 1000), color='red')
plt.show()
```



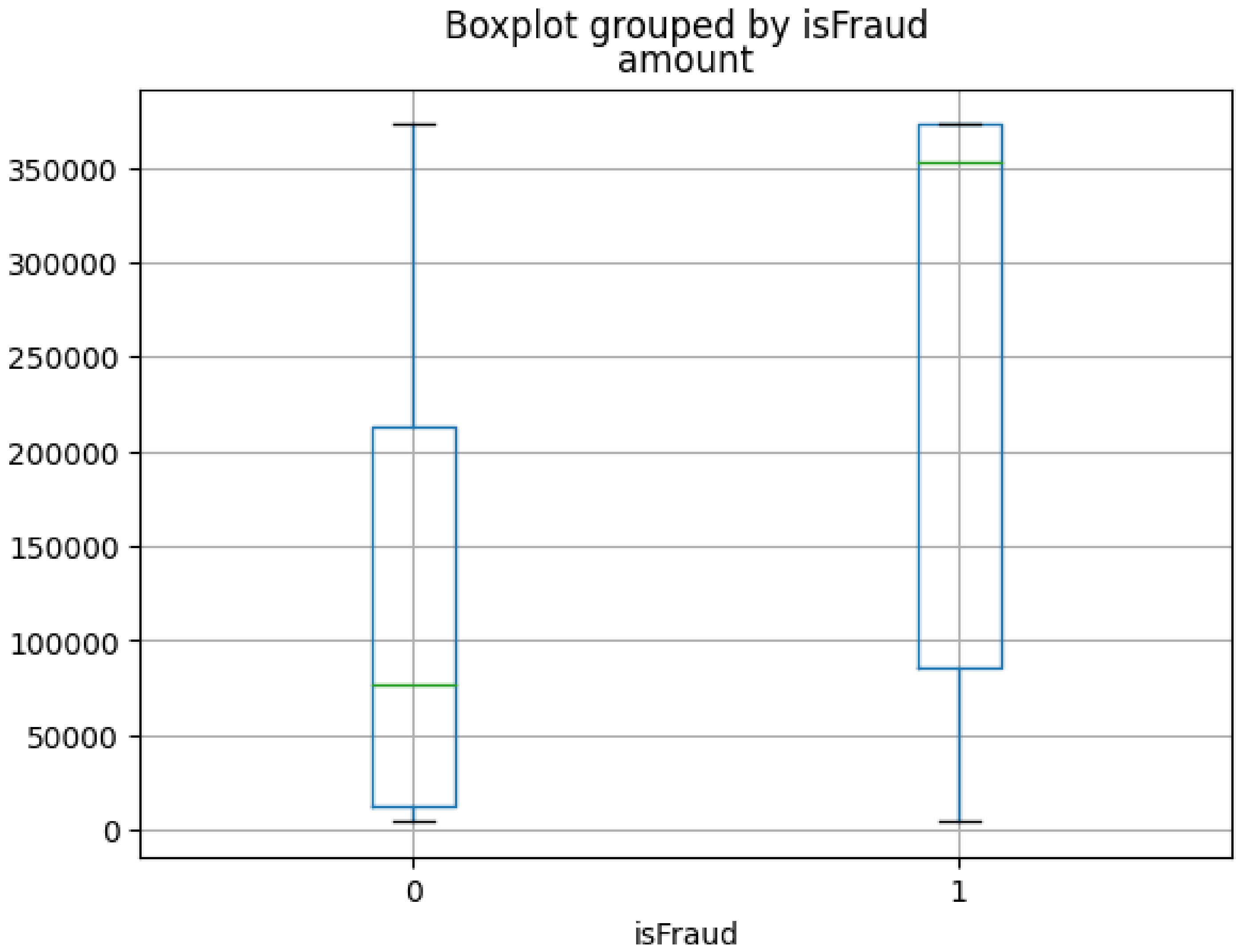
Lowest amount transaction starts from 4220.57 and highest amount transaction goes upto 3.7 lakh

In [294…

```
data.boxplot(column='amount', by='isFraud')
```

Out[294]:

```
<Axes: title={'center': 'amount'}, xlabel='isFraud'>
```



Fraud amount transaction range is between 75k-3.7 lakh

In [295…

```
total_transactions = data.shape[0]

fraud_transaction = data[data.isFraud==1].shape[0]

fraud_percent= fraud_transaction/total_transactions * 100
```

In [296…

```
print('Total transactions: ',total_transactions)
print('Total fraud transactions happened: ',fraud_transaction)
print("Total fraud transaction percent: ",round(fraud_percent,2))
```

```
Total transactions:  1048575
Total fraud transactions happened:  1142
Total fraud transaction percent:  0.11
```

once again is shows that the dataset is very much imbalanced

In [297…

```
fraud_amount= data[data.isFraud==1]
fraud_amount=fraud_amount.sort_values(by=['amount'],ascending=False)
fraud_amount.head()
```

Out[297]:

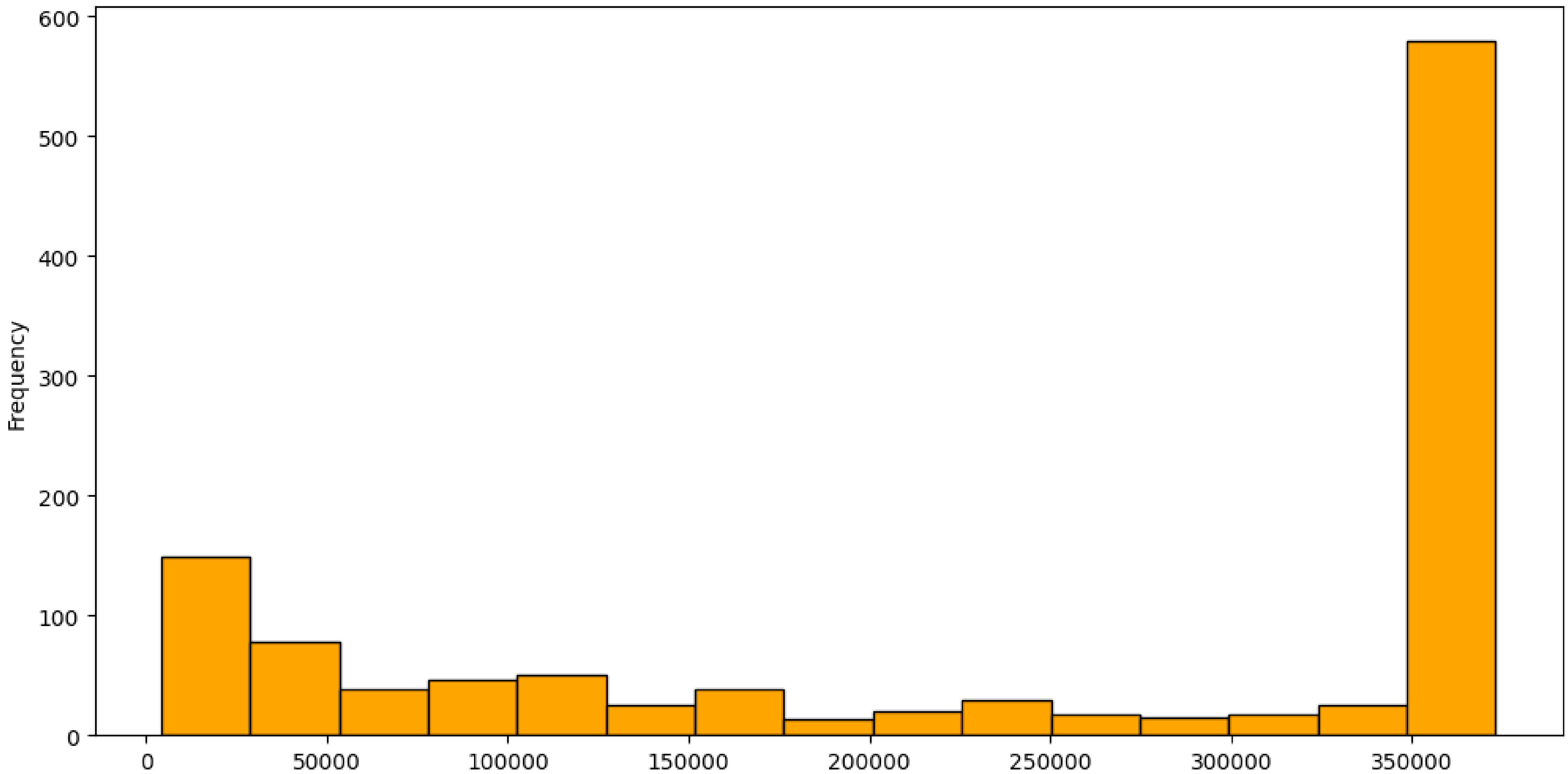
	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDes
1025194	48	CASH_OUT	373075.378	C274359236	1215297.01	0.0	C1653022223	2497294.92	3102896.2(
992140	45	TRANSFER	373075.378	C1582972194	1069508.42	0.0	C284364603	0.00	0.0(
955157	44	TRANSFER	373075.378	C369936121	1649818.97	0.0	C1347315975	0.00	0.0(
955158	44	CASH_OUT	373075.378	C2052172437	1649818.97	0.0	C1401780750	560704.68	2210523.6
956900	44	TRANSFER	373075.378	C374179954	387952.42	0.0	C1213274351	0.00	0.0(

In [298…

```
fraud_amount.amount.plot(kind='hist', bins=15, figsize=(12,6), facecolor='orange',edgecolor='black')
```

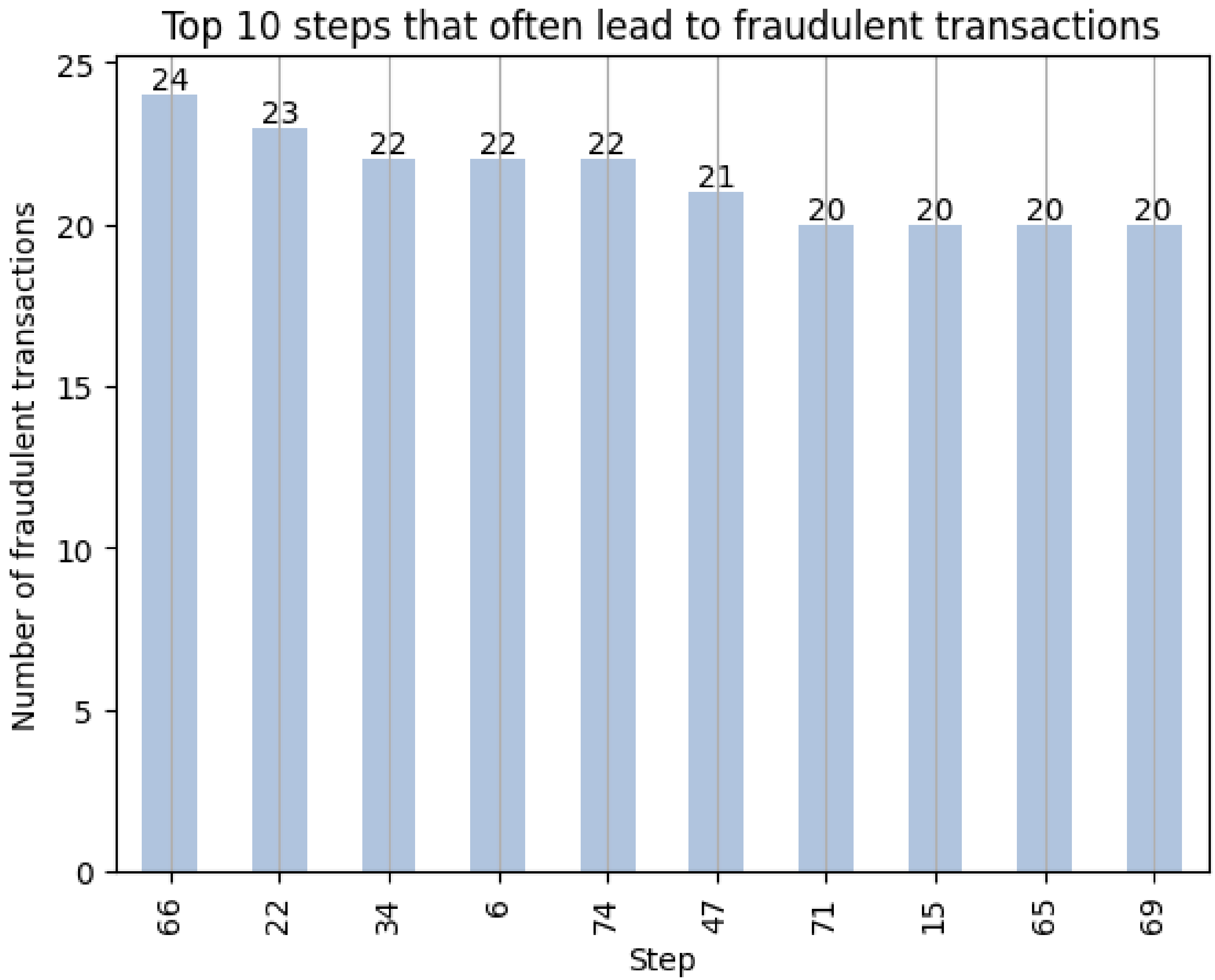
Out[298]:

```
<Axes: ylabel='Frequency'>
```



In [299…

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

df1 = data[data['isFraud'] == 1]
df2 = df1['step'].value_counts().head(10)
ax = df2.plot(kind='bar', color='lightsteelblue')
for container in ax.containers:
    ax.bar_label(container)
plt.title('Top 10 steps that often lead to fraudulent transactions')
plt.ylabel('Number of fraudulent transactions')
plt.xlabel('Step')
plt.grid(axis='x')
```



Step 66 has the highest number of fraudulent transactions, 24 cases. This indicates that Step 66 is the step that will most likely lead to fraudulent transactions.

In [300…

```
fraudster= data.nameDest.value_counts()
fraudster
```

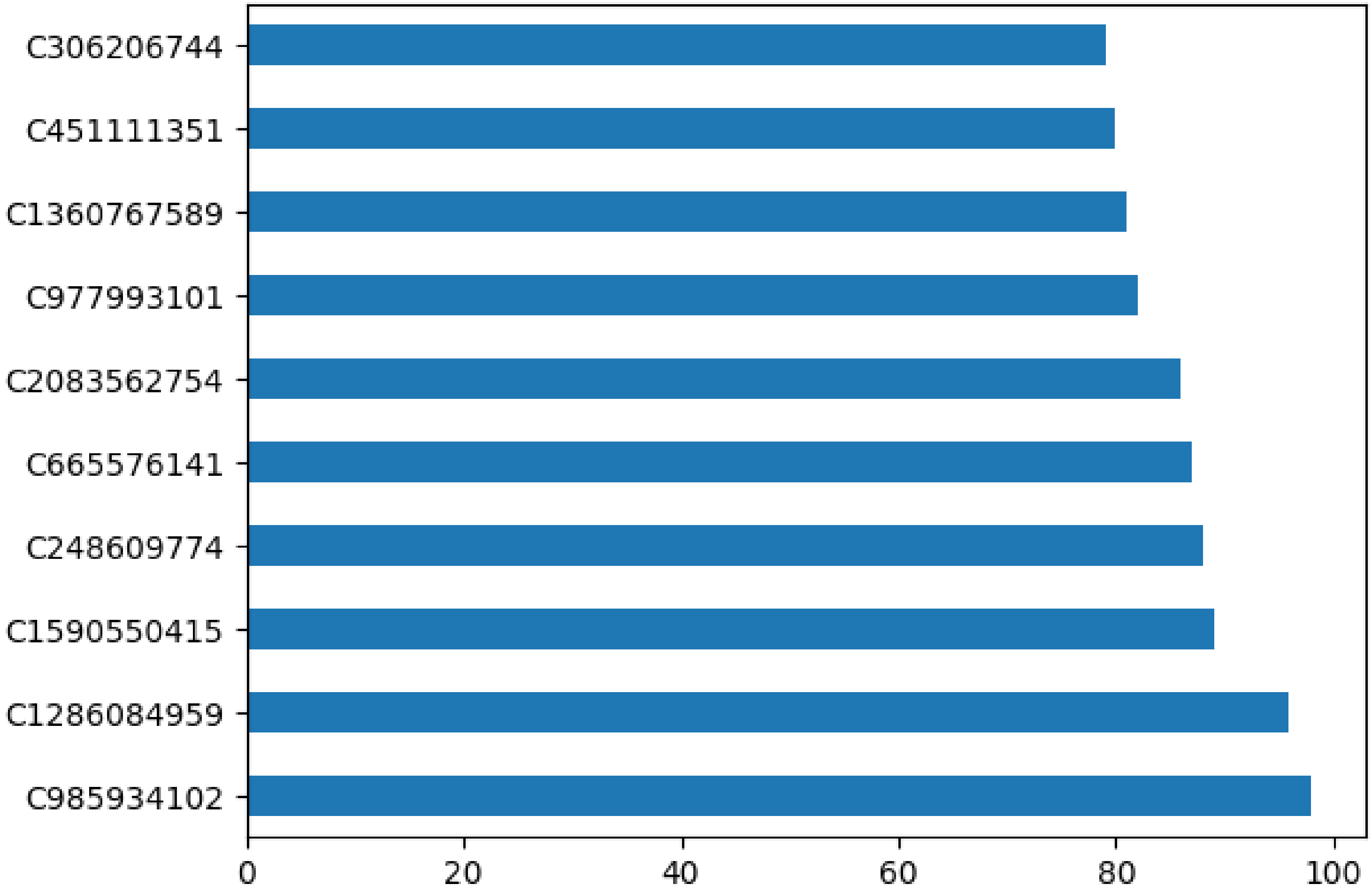
Out[300]:
```
C985934102     98
C1286084959    96
C1590550415    89
C248609774     88
C665576141     87
               ..
M382871047      1
M322765556      1
M1118794441     1
M1127250627     1
M677577406      1
Name: nameDest, Length: 449635, dtype: int64
```

In [301…
```
fraudster[:10]
```

Out[301]:
```
C985934102     98
C1286084959    96
C1590550415    89
C248609774     88
C665576141     87
C2083562754    86
C977993101     82
C1360767589    81
C451111351     80
C306206744     79
Name: nameDest, dtype: int64
```

In [302…
```
fraudster[:10].plot(kind='barh')
```

Out[302]:
```
<Axes: >
```



the above nameDest are the top 10 fraud people with most fraud transfers

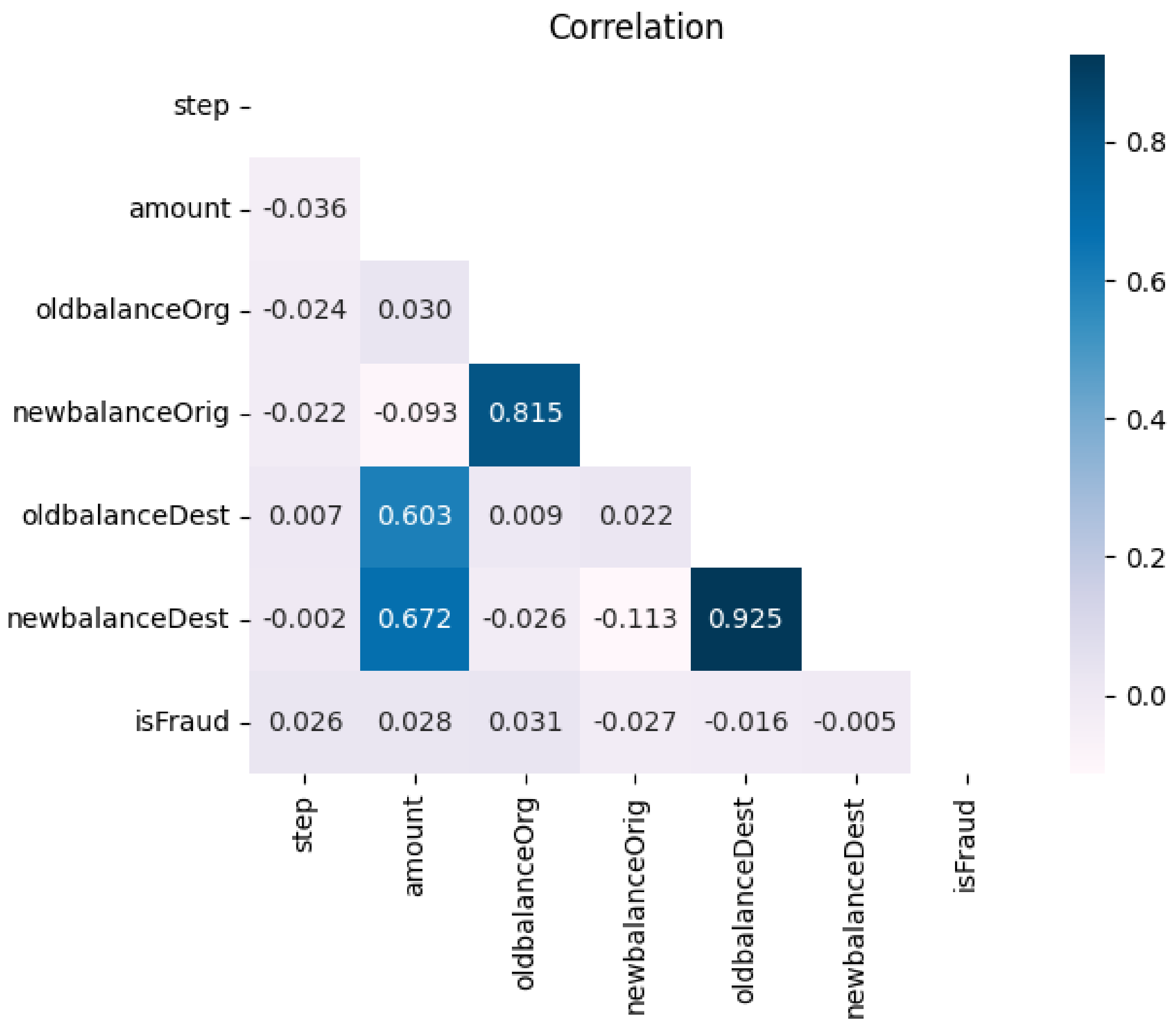
Correlation

In [330…

```
corr_viz=data.corr('spearman')
sns.heatmap(corr_viz, cbar=True, annot=True, mask = np.triu(np.ones_like(corr_viz, dtype = bool)), fmt='.3f', c
plt.title('Correlation')
```

Out[330]:

```
Text(0.5, 1.0, 'Correlation')
```



oldbalanceOrg and newbalanceOrig has strong positive relationship.

oldbalanceDest and newbalanceDest has strong positive relationship.

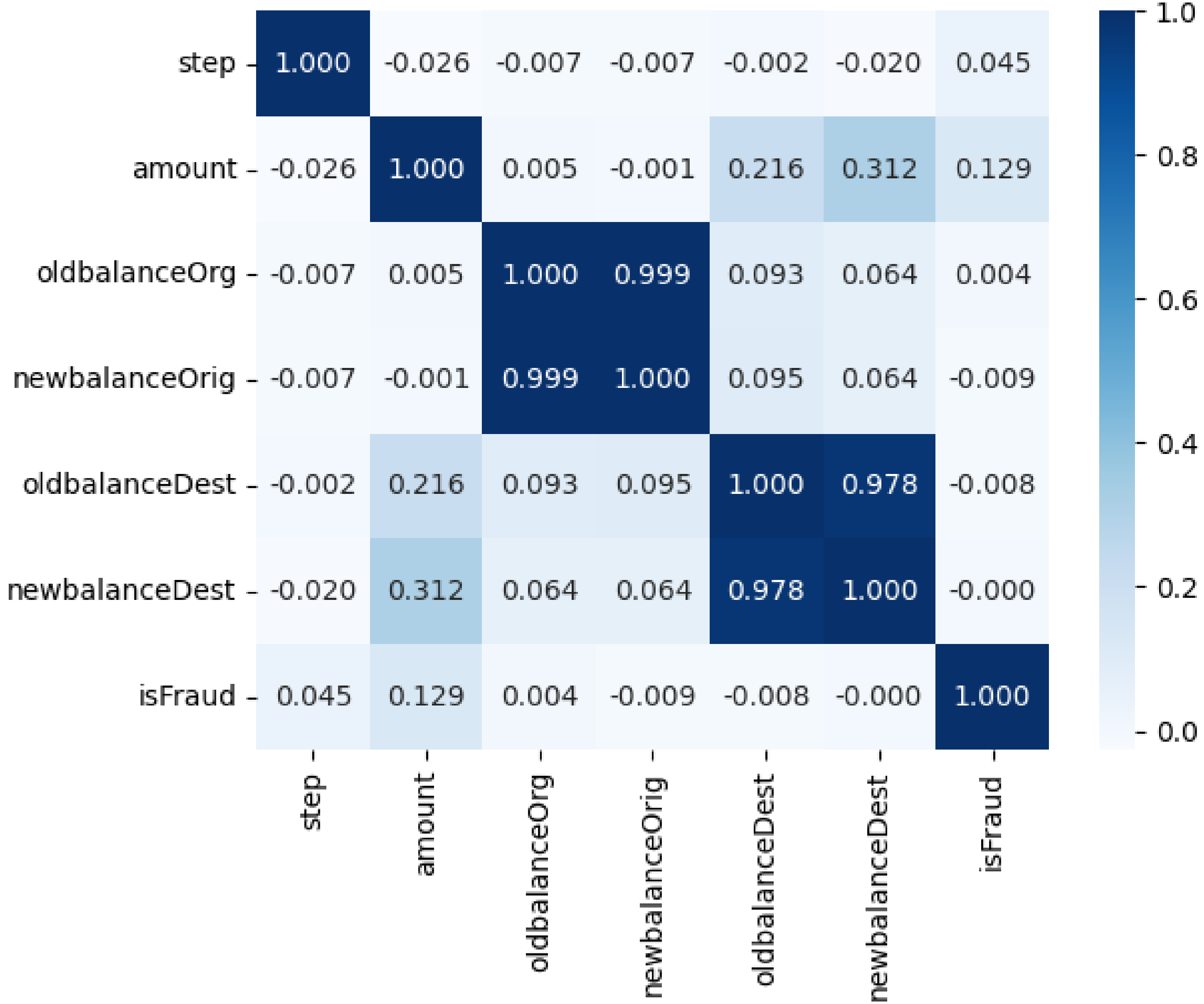
oldbalanceOrg and amount has weak positive relationship. newbalanceOrig and amount has moderate positive relationship.

In [331…

```
# calculate correlation matrix
corr_viz_=data.corr()# plot the heatmap
sns.heatmap(corr_viz_, xticklabels=corr_viz_.columns, yticklabels=corr_viz_.columns, annot=True, cmap='Blues' ,
```

Out[331]:

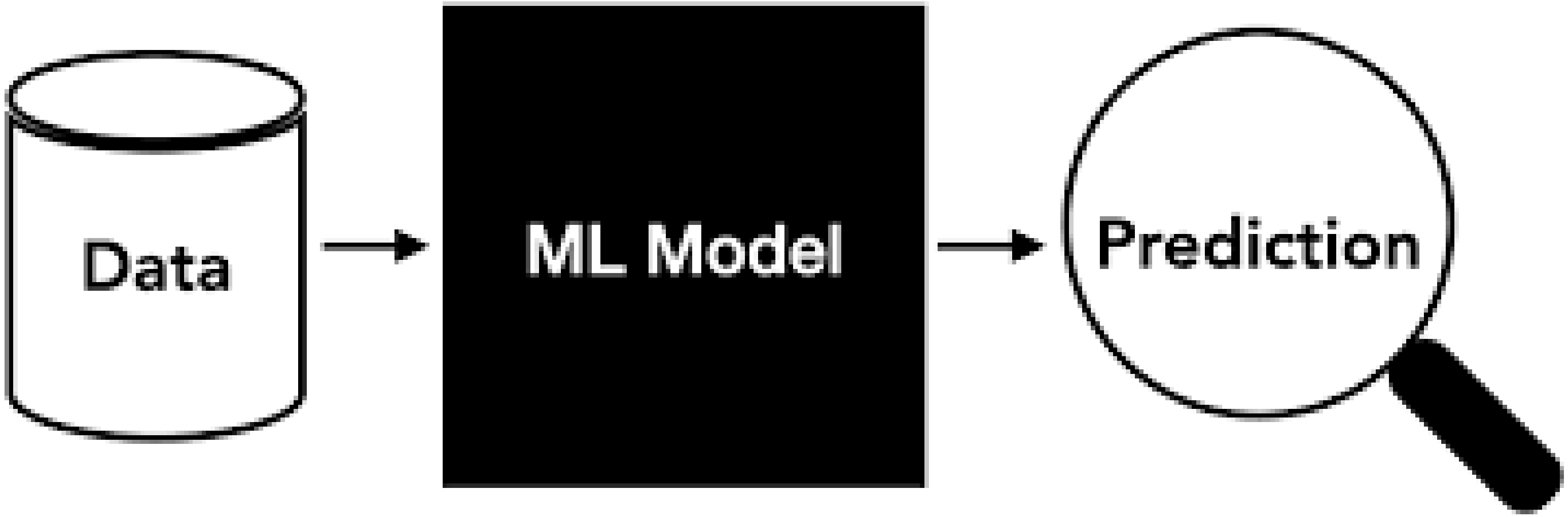
```
<Axes: >
```

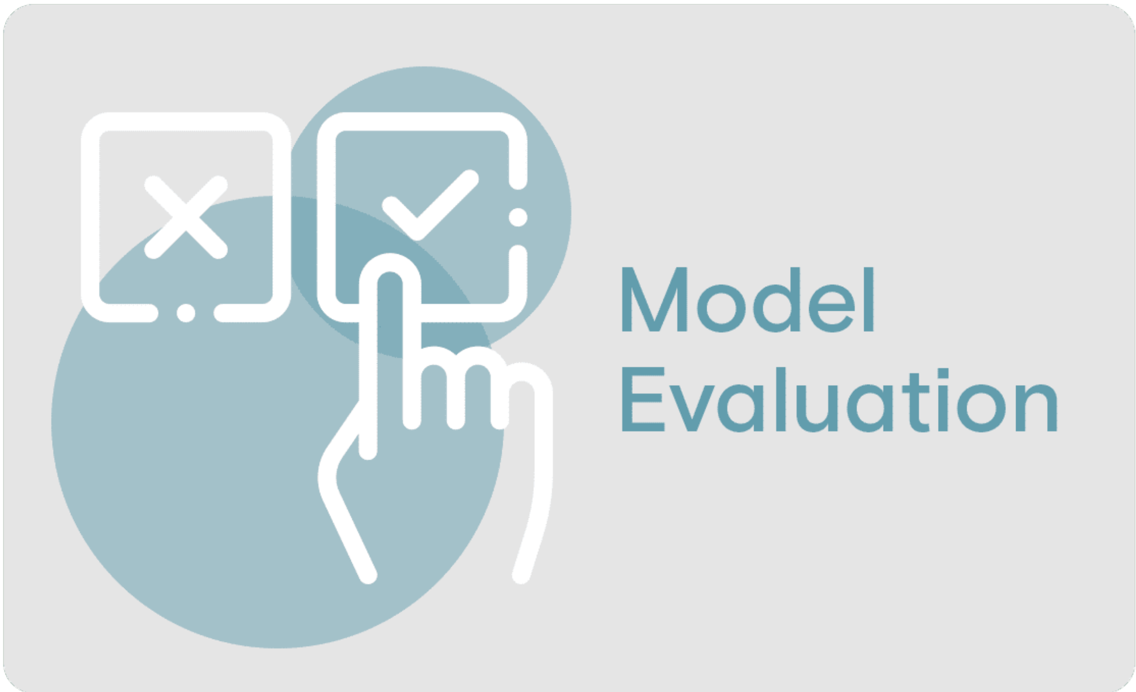


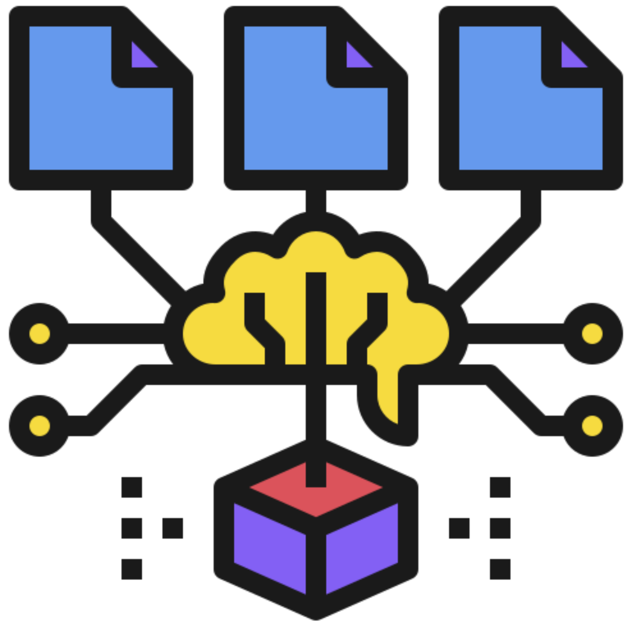
There is a high correlation between newbalanceOrig and oldbalanceOrg.

Also, between newbalanceDest and oldbalanceDest.

Apart from that, we have a relatively high correlation between amount and newbalanceDest and amount with oldbalanceDest


Data
ML Model
Prediction


Model
Evaluation



ML-1 undersampling

- LogisticRegression
- DecisionTreeClassifier

In [360…

```
data.head(1)
```

Out[360]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	0	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0

In [358…

```
data['type'] = data['type'].map({'PAYMENT':0, 'CASH_IN':1, 'DEBIT':2, 'CASH_OUT':3, 'TRANSFER':4})
```

In [361…

```
data.tail()
```

Out[361]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
1048570	95	3	132557.35	C1179511630	479803.00	347245.65	C435674507	484329.37	616886.72	0
1048571	95	0	9917.36	C1956161225	90545.00	80627.64	M668364942	0.00	0.00	0
1048572	95	0	14140.05	C2037964975	20545.00	6404.95	M1355182933	0.00	0.00	0
1048573	95	0	10020.05	C1633237354	90605.00	80584.95	M1964992463	0.00	0.00	0
1048574	95	0	11450.03	C1264356443	80584.95	69134.92	M677577406	0.00	0.00	0

In [362…

```
data['isFraud'].value_counts()
```

Out[362]:

```
0    1047433
1       1142
Name: isFraud, dtype: int64
```

In [363…

```
legit_txns = data[data.isFraud == 0]
fraud_txns = data[data.isFraud == 1]
print(legit_txns.shape)
print(fraud_txns.shape)
```

```
(1047433, 10)
(1142, 10)
```

In [364…

```
legit_txns.amount.describe()
```

Out[364]:

```
count    1.047433e+06
mean     1.575397e+05
std      2.541883e+05
min      1.000000e-01
25%      1.213487e+04
50%      7.621497e+04
75%      2.134928e+05
max      6.419835e+06
Name: amount, dtype: float64
```

In [365…

```
fraud_txns.amount.describe()
```

Out[365]:

```
count    1.142000e+03
mean     1.192629e+06
std      2.030599e+06
min      1.190000e+02
25%      8.607017e+04
50%      3.531794e+05
75%      1.248759e+06
max      1.000000e+07
Name: amount, dtype: float64
```

In [366…

```
data.groupby('isFraud').mean()
```

Out[366]:

isFraud	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	26.942944	1.628201	1.575397e+05	8.736338e+05	894746.395080	978732.769117	1.114237e+06
1	48.272329	3.493870	1.192629e+06	1.218636e+06	33944.321208	452866.124527	1.077940e+06

In [368…

```
# Samples 1142 transactions out of the legit transactions
legit_sample = legit_txns.sample(n=1142)
# Concatenates all the 8213 the fraud_txns and the 8213 samples of the legit txns
undersampled_dataset = pd.concat([legit_sample, fraud_txns], axis=0)
```

In [369…

```
undersampled_dataset.head()
```

Out[369]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
1028063	48	0	25820.01	C117878797	0.0	0.00	M2105717108	0.00	0.00	0
244895	14	1	143321.16	C1051005422	693366.8	836687.96	C232844939	1051074.14	871010.12	0
965729	44	3	306263.71	C1932186313	1472.0	0.00	C258311078	1525878.22	2104187.63	0
300208	15	2	9759.29	C1467666922	6109.0	0.00	C497088507	292340.44	302099.73	0
575790	25	0	1995.48	C465603619	0.0	0.00	M370670310	0.00	0.00	0

In [370…

```
undersampled_dataset['isFraud'].value_counts()
```

Out[370]:

```
0    1142
1    1142
Name: isFraud, dtype: int64
```

In [371…

```
undersampled_dataset.groupby('isFraud').mean()
```

Out[371]:

isFraud	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	27.028897	1.689142	1.622242e+05	7.838778e+05	801553.637373	884234.647268	1.026784e+06
1	48.272329	3.493870	1.192629e+06	1.218636e+06	33944.321208	452866.124527	1.077940e+06

In [372…

```
data.groupby('isFraud').mean()
```

Out[372]:

isFraud	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	26.942944	1.628201	1.575397e+05	8.736338e+05	894746.395080	978732.769117	1.114237e+06
1	48.272329	3.493870	1.192629e+06	1.218636e+06	33944.321208	452866.124527	1.077940e+06

In [373…

```
X = undersampled_dataset.drop(columns=['isFraud','nameDest','nameOrig'], axis=1)
# Remove the class column from the undersampled dataset
X.head()
```

Out[373]:

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
1028063	48	0	25820.01	0.0	0.00	0.00	0.00
244895	14	1	143321.16	693366.8	836687.96	1051074.14	871010.12
965729	44	3	306263.71	1472.0	0.00	1525878.22	2104187.63
300208	15	2	9759.29	6109.0	0.00	292340.44	302099.73
575790	25	0	1995.48	0.0	0.00	0.00	0.00

In [374…

```
Y = undersampled_dataset['isFraud']
Y
```

Out[374]:

```
1028063    0
244895     0
965729     0
300208     0
575790     0
          ..
1047888    1
1048221    1
1048222    1
1048323    1
1048324    1
Name: isFraud, Length: 2284, dtype: int64
```

In [375…

```
X_train_undersampled, X_test_undersampled, Y_train_undersampled, Y_test_undersampled = train_test_split(X, Y, test_size=0.2, str
```

In [376…

```
print(X.shape, X_train_undersampled.shape, X_test_undersampled.shape)
```

```
(2284, 7) (1827, 7) (457, 7)
```

In [387…

```
scaler = StandardScaler()
X_train_scaled_undersampled = scaler.fit_transform(X_train_undersampled)
X_test_scaled_undersampled = scaler.transform(X_test_undersampled)
```

In [388…

```python
LogisticRegressionModel = LogisticRegression()
SVM_SVC_Model_CLF = SVC(kernel='linear')
NaiveBayes_Model_CLF = GaussianNB()
KNN_Model_CLF = KNeighborsClassifier(n_neighbors=5)
DecisionTree_Model = DecisionTreeClassifier()
params = {'max_depth': [2, 4, 6, 8, 10],
          'min_samples_split': [2, 4, 6, 8, 10],
          'min_samples_leaf': [1, 2, 3, 4, 5]}
DecisionTree_GridSearch_CLF = GridSearchCV(DecisionTree_Model, params, cv=5)
```

In [389…

```python
LogisticRegressionModel.fit(X_train_scaled_undersampled, Y_train_undersampled)
```

Out[389]:

```
▾ LogisticRegression

LogisticRegression()
```

In [390…

```python
# accuracy on training data
X_train_prediction = LogisticRegressionModel.predict(X_train_scaled_undersampled)
LR_undersampling_training_data_accuracy = accuracy_score(X_train_prediction, Y_train_undersampled)
print('Accuracy on Training data : ', LR_undersampling_training_data_accuracy)
LR_undersampling_training_data_classification_report = classification_report(X_train_prediction, Y_train_undersampled)
print('\nClassification Report on Training data : \n', LR_undersampling_training_data_classification_report)
```

```
Accuracy on Training data :  0.8801313628899836

Classification Report on Training data :
               precision    recall  f1-score   support

           0       0.89      0.88      0.88       924
           1       0.87      0.88      0.88       903

    accuracy                           0.88      1827
   macro avg       0.88      0.88      0.88      1827
weighted avg       0.88      0.88      0.88      1827
```

In [391…

```python
# accuracy on test data
X_test_prediction = LogisticRegressionModel.predict(X_test_scaled_undersampled)
LR_undersampling_test_data_accuracy = accuracy_score(X_test_prediction, Y_test_undersampled)
print('Accuracy score on Test Data : ', LR_undersampling_test_data_accuracy)
LR_undersampling_testing_data_classification_report = classification_report(X_test_prediction, Y_test_undersampled)
print('\nClassification Report on Training data : \n', LR_undersampling_testing_data_classification_report)
```

```
Accuracy score on Test Data :  0.9059080962800875

Classification Report on Training data :
               precision    recall  f1-score   support

           0       0.93      0.89      0.91       240
           1       0.88      0.93      0.90       217

    accuracy                           0.91       457
   macro avg       0.91      0.91      0.91       457
weighted avg       0.91      0.91      0.91       457
```

In [392…

```python
DecisionTree_Model.fit(X_train_scaled_undersampled, Y_train_undersampled)
```

Out[392]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [393…

```python
# accuracy on training data
X_train_prediction = DecisionTree_Model.predict(X_train_scaled_undersampled)
DT_undersampling_training_data_accuracy = accuracy_score(X_train_prediction, Y_train_undersampled)
print('Accuracy on Training data : ', DT_undersampling_training_data_accuracy)
DT_undersampling_training_data_classification_report = classification_report(X_train_prediction, Y_train_undersampled)
print('\nClassification Report on Training data : \n', DT_undersampling_training_data_classification_report)
```

```
Accuracy on Training data :  1.0

Classification Report on Training data :
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       913
           1       1.00      1.00      1.00       914

    accuracy                           1.00      1827
   macro avg       1.00      1.00      1.00      1827
weighted avg       1.00      1.00      1.00      1827
```

In [394…

```python
# accuracy on test data
X_test_prediction = DecisionTree_Model.predict(X_test_scaled_undersampled)
DT_undersampling_test_data_accuracy = accuracy_score(X_test_prediction, Y_test_undersampled)
```

```
print('Accuracy score on Test Data : ', DT_undersampling_test_data_accuracy)
DT_undersampling_testing_data_classification_report = classification_report(X_test_prediction, Y_test_undersampled)
print('\nClassification Report on Training data : \n', DT_undersampling_testing_data_classification_report)
```

```
Accuracy score on Test Data :  0.975929978118162

Classification Report on Training data :
               precision    recall  f1-score   support

           0       0.97      0.98      0.98       226
           1       0.98      0.97      0.98       231

    accuracy                           0.98       457
   macro avg       0.98      0.98      0.98       457
weighted avg       0.98      0.98      0.98       457
```

ML2

- Random Forest Classifier
- Gradient Boosting

In [386…

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Initialize Random Forest Classifier
rf_classifier_undersampled = RandomForestClassifier(random_state=42, class_weight='balanced')

# Train the Random Forest model on the undersampled data
rf_classifier_undersampled.fit(X_train_undersampled, Y_train_undersampled)

# Predict on the test set
rf_predictions_undersampled = rf_classifier_undersampled.predict(X_test_undersampled)

# Evaluate Random Forest Classifier
print("Random Forest Classifier Results on Undersampled Data:")
print("Accuracy:", accuracy_score(Y_test_undersampled, rf_predictions_undersampled))
print("Precision:", precision_score(Y_test_undersampled, rf_predictions_undersampled))
print("Recall:", recall_score(Y_test_undersampled, rf_predictions_undersampled))
print("F1 Score:", f1_score(Y_test_undersampled, rf_predictions_undersampled))
print("ROC AUC Score:", roc_auc_score(Y_test_undersampled, rf_classifier_undersampled.predict_proba(X_test_undersampled)[:,1]))
```

```
# Initialize Gradient Boosting Classifier
gb_classifier_undersampled = GradientBoostingClassifier(random_state=42)

# Train the Gradient Boosting model on the undersampled data
gb_classifier_undersampled.fit(X_train_undersampled, Y_train_undersampled)

# Predict on the test set
gb_predictions_undersampled = gb_classifier_undersampled.predict(X_test_undersampled)

# Evaluate Gradient Boosting Classifier
print("\nGradient Boosting Classifier Results on Undersampled Data:")
print("Accuracy:", accuracy_score(Y_test_undersampled, gb_predictions_undersampled))
print("Precision:", precision_score(Y_test_undersampled, gb_predictions_undersampled))
print("Recall:", recall_score(Y_test_undersampled, gb_predictions_undersampled))
print("F1 Score:", f1_score(Y_test_undersampled, gb_predictions_undersampled))
print("ROC AUC Score:", roc_auc_score(Y_test_undersampled, gb_classifier_undersampled.predict_proba(X_test_undersampled)[:,1]))
```

```
Random Forest Classifier Results on Undersampled Data:
Accuracy: 0.975929978118162
Precision: 0.9696969696969697
Recall: 0.9824561403508771
F1 Score: 0.9760348583877996
ROC AUC Score: 0.9950969125871447

Gradient Boosting Classifier Results on Undersampled Data:
Accuracy: 0.975929978118162
Precision: 0.9656652360515021
Recall: 0.9868421052631579
F1 Score: 0.9761388286334057
ROC AUC Score: 0.9968398069409331
```

PREDICTION

- Random Forest Classifier

In [400…

```
# Define features and target
X = undersampled_dataset.drop(columns=['isFraud','nameDest','nameOrig'], axis=1)
Y = undersampled_dataset['isFraud']

# Split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

```python
# Train Random Forest Classifier
rf_classifier_undersampled = RandomForestClassifier(random_state=42, class_weight='balanced')
rf_classifier_undersampled.fit(X_train, Y_train)

# User input for new data
user_input = {
    'step': int(input("Enter step: ")),
    'type': int(input("Enter transaction type (0: PAYMENT, 1: TRANSFER, 2: CASH_OUT): ")),
    'amount': float(input("Enter transaction amount: ")),
    'oldbalanceOrg': float(input("Enter old balance of origin account: ")),
    'newbalanceOrig': float(input("Enter new balance of origin account: ")),
    'oldbalanceDest': float(input("Enter old balance of destination account: ")),
    'newbalanceDest': float(input("Enter new balance of destination account: "))
}

# Create a DataFrame from user input
user_new_data = pd.DataFrame(user_input, index=[0])

# Make prediction using the trained Random Forest model
user_new_data_predictions = rf_classifier_undersampled.predict(user_new_data)

# Print the prediction
if user_new_data_predictions[0] == 1:
    print("Prediction for the new data: Fraudulent")
else:
    print("Prediction for the new data: Not Fraudulent")
```

```
Enter step: 1
Enter transaction type (0: PAYMENT, 1: TRANSFER, 2: CASH_OUT): 0
Enter transaction amount:  9839.64
Enter old balance of origin account:  170136.0
Enter new balance of origin account: 160296.36
Enter old balance of destination account: 0.0
Enter new balance of destination account: 0.0
Prediction for the new data: Not Fraudulent
```