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

from sklearn.linear_model import LinearRegression
```

Import the dataset

In [2]:
```
data = 'FinanceFraudDetectionModified.csv'
ori_df = pd.read_csv(data)
```

Question 1. Is there any missing dada? If yes, how many are they per column?

In [3]:
```
ori_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 11 columns):
 #   Column          Non-Null Count    Dtype  
---  ------          --------------    -----  
 0   step            1048575 non-null  int64  
 1   type            1048575 non-null  object 
 2   amount          1048570 non-null  float64
 3   nameOrig        1048575 non-null  object 
 4   oldbalanceOrg   1048572 non-null  float64
 5   newbalanceOrig  1048575 non-null  float64
 6   nameDest        1048572 non-null  object 
 7   oldbalanceDest  1048575 non-null  float64
 8   newbalanceDest  1048575 non-null  float64
 9   isFraud         1048575 non-null  int64  
 10  isFlaggedFraud  1048575 non-null  int64  
dtypes: float64(5), int64(3), object(3)
memory usage: 88.0+ MB
```

Answer 1. There is no missing data because the Non-Null Count column shows that

In [4]:
```
ori_df.head()
```

Out[4]:

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

Drop the last column using iloc

In [5]:
```
df = ori_df.iloc[:, :-1]
```

In [6]:
```
df.head(15)
```

Out[6]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldba
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	
3	1	CASH_OUT	NaN	C840083671	181.00	0.00	C38997010	
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	
5	1	PAYMENT	7817.71	C90045638	53860.00	46042.29	M573487274	
6	1	PAYMENT	7107.77	C154988899	183195.00	176087.23	M408069119	
7	1	PAYMENT	7861.64	C1912850431	176087.23	168225.59	M633326333	
8	1	PAYMENT	NaN	C1265012928	2671.00	0.00	M1176932104	
9	1	DEBIT	5337.77	C712410124	NaN	36382.23	C195600860	
10	1	DEBIT	9644.94	C1900366749	4465.00	0.00	NaN	
11	1	PAYMENT	3099.97	C249177573	20771.00	17671.03	M2096539129	
12	1	PAYMENT	2560.74	C1648232591	5070.00	2509.26	M972865270	
13	1	PAYMENT	NaN	C1716932897	10127.00	0.00	M801569151	
14	1	PAYMENT	4098.78	C1026483832	503264.00	499165.22	M1635378213	

In [7]:
```
print(f"The shape of the data we are working with is {df.shape}")
```

```
The shape of the data we are working with is (1048575, 10)
```

In [8]:
```
df.info()
```

```
<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          1048570 non-null  float64
 3   nameOrig        1048575 non-null  object 
 4   oldbalanceOrg   1048572 non-null  float64
 5   newbalanceOrig  1048575 non-null  float64
 6   nameDest        1048572 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
```

Checking missing values

In [9]:
```
df.isnull().values.any()
```

Out[9]: True

True: There is missing data

In [10]:
```
df.isnull()
```

Out[10]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceD
0	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	F
2	False	False	False	False	False	False	False	F
3	False	False	True	False	False	False	False	F
4	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	
1048570	False	False	False	False	False	False	False	F
1048571	False	False	False	False	False	False	False	F
1048572	False	False	False	False	False	False	False	F
1048573	False	False	False	False	False	False	False	F
1048574	False	False	False	False	False	False	False	F

1048575 rows × 10 columns

True: missing value, False: Real values

Print out any rows that have missing values

df[condition to filter on]

In [11]:
```
#df.isnull() == True
```

In [12]:
```
#df[df.isnull()==True].head(20)
```

In [13]:
```
df.shape[0]
```

Out[13]: 1048575

In [14]:
```
df.shape[1]
```

Out[14]: 10

In [15]:
```
# Print out the rows that have missing values
for col in df.columns:
    temp_df = df[df[col].isna()]
    if temp_df.shape[0] != 0:
        display(temp_df)
        print(f"Processing {col}")
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbal
3	1	CASH_OUT	NaN	C840083671	181.0	0.00	C38997010	
8	1	PAYMENT	NaN	C1265012928	2671.0	0.00	M1176932104	
13	1	PAYMENT	NaN	C1716932897	10127.0	0.00	M801569151	
18	1	PAYMENT	NaN	C2033524545	15123.0	14451.36	M473053293	
23	1	PAYMENT	NaN	C504336483	67852.0	63975.59	M1404932042	

```
Processing amount
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldba
9	1	DEBIT	5337.77	C712410124	NaN	36382.23	C195600860	
19	1	TRANSFER	215310.30	C1670993182	NaN	0.00	C1100439041	
29	1	PAYMENT	9920.52	C764826684	NaN	0.00	M1940055334	

```
Processing oldbalanceOrg
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalance
10	1	DEBIT	9644.94	C1900366749	4465.0	0.00	NaN	108
20	1	PAYMENT	1373.43	C20804602	13854.0	12480.57	NaN	
30	1	PAYMENT	3448.92	C2103763750	0.0	0.00	NaN	

Imputation

In [16]:

```
df.fillna(0)
```

Out[16]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065
3	1	CASH_OUT	0.00	C840083671	181.00	0.00	C38997010
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703
...	...	...	...	...	...	...	...
1048570	95	CASH_OUT	132557.35	C1179511630	479803.00	347245.65	C435674507
1048571	95	PAYMENT	9917.36	C1956161225	90545.00	80627.64	M668364942
1048572	95	PAYMENT	14140.05	C2037964975	20545.00	6404.95	M1355182933
1048573	95	PAYMENT	10020.05	C1633237354	90605.00	80584.95	M1964992463
1048574	95	PAYMENT	11450.03	C1264356443	80584.95	69134.92	M677577406

1048575 rows × 10 columns

In [17]:

```
df.columns
```

Out[17]:

```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud'],
      dtype='object')
```

In [18]:

```
# for col in df.columns:
#     print(col)
#     temp_df = df[df[col].isna()] #
#     #display(temp_df)

#     if temp_df.shape[0] != 0: # The rows that have missing values
#         #display(temp_df)
#         #print(f"Processing {col}")
#         temp_df[col] = temp_df[col].fillna(df[col].mean())
#         display(temp_df)
```

In [19]:
```
for col in df.select_dtypes(include=np.number).columns:
    df[col] = df[col].fillna(df[col].mean())
```

In [20]:
```
df
```

Out[20]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	name
0	1	PAYMENT	9839.640000	C1231006815	170136.00	160296.36	M197978
1	1	PAYMENT	1864.280000	C1666544295	21249.00	19384.72	M204428
2	1	TRANSFER	181.000000	C1305486145	181.00	0.00	C55326
3	1	CASH_OUT	158667.712672	C840083671	181.00	0.00	C3899
4	1	PAYMENT	11668.140000	C2048537720	41554.00	29885.86	M123070
...	...	...	...	...	...	...	
1048570	95	CASH_OUT	132557.350000	C1179511630	479803.00	347245.65	C43567
1048571	95	PAYMENT	9917.360000	C1956161225	90545.00	80627.64	M66836
1048572	95	PAYMENT	14140.050000	C2037964975	20545.00	6404.95	M135518
1048573	95	PAYMENT	10020.050000	C1633237354	90605.00	80584.95	M196499
1048574	95	PAYMENT	11450.030000	C1264356443	80584.95	69134.92	M67757

1048575 rows × 10 columns

In [21]:
```
col_names = df.columns
print('The names of columns are:', col_names)
```

```
The names of columns are: Index(['step', 'type', 'amount', 'nameOrig', 'oldbalance
Org', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud'],
      dtype='object')
```

In [22]:
```
cols = df.columns.tolist()

print('There are {} categorical variables\n'.format(len(cols)))
print('All variables are :', cols)
```

```
There are 10 categorical variables

All variables are : ['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newba
lanceOrig', 'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud']
```

In [23]:
```
categorical = [var for var in df.columns if df[var].dtype == 'O']     #'O' represen

print('There are {} categorical variables\n'.format(len(categorical)))

print('The categorical variables are :', categorical)
```

```
There are 3 categorical variables

The categorical variables are : ['type', 'nameOrig', 'nameDest']
```

In [24]:
```
df[categorical].head()
```

Out[24]:

	type	nameOrig	nameDest
0	PAYMENT	C1231006815	M1979787155
1	PAYMENT	C1666544295	M2044282225
2	TRANSFER	C1305486145	C553264065
3	CASH_OUT	C840083671	C38997010
4	PAYMENT	C2048537720	M1230701703

In [25]:
```
# Check missing values in categorical variables

df[categorical].isnull().sum()
```

Out[25]:
```
type        0
nameOrig    0
nameDest    3
dtype: int64
```

There are no missing values in the given dataset from the above exercution

View the contribution of each column and See what they mean

In [26]:
```
for var in categorical:
    print(df[var].value_counts()/np.single(len(df)))
```

```
CASH_OUT    0.356332
PAYMENT     0.337480
CASH_IN     0.216608
TRANSFER    0.082734
DEBIT       0.006845
Name: type, dtype: float64
C1214450722    1.907350e-06
C309111136     1.907350e-06
C1268675361    1.907350e-06
C720460198     1.907350e-06
C1109092856    1.907350e-06
                   ...
C560131732     9.536752e-07
C455251560     9.536752e-07
C650578540     9.536752e-07
C1883668225    9.536752e-07
C1264356443    9.536752e-07
Name: nameOrig, Length: 1048317, dtype: float64
C985934102     9.346017e-05
C1286084959    9.155282e-05
C1590550415    8.487710e-05
C248609774     8.392342e-05
C665576141     8.296974e-05
                   ...
M2036888797    9.536752e-07
M1545238325    9.536752e-07
M382871047     9.536752e-07
M322765556     9.536752e-07
M677577406     9.536752e-07
Name: nameDest, Length: 449633, dtype: float64
```

In [27]:

```python
Numerical = [var for var in df.columns if df[var].dtype != 'O']  # 'O' represents o

print('There are {} numerical variables\n'.format(len(Numerical)))

print('The numerical variables are :', Numerical)
```

```
There are 7 numerical variables

The numerical variables are : ['step', 'amount', 'oldbalanceOrg', 'newbalanceOri
g', 'oldbalanceDest', 'newbalanceDest', 'isFraud']
```

In [28]:

```python
# Check missing values in categorical variables

df[Numerical].isnull().sum()
```

Out[28]:

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

In [29]:

```python
for var in categorical:
```

```
    print(df[var].value_counts()/np.single(len(df)))
```

```
CASH_OUT    0.356332
PAYMENT     0.337480
CASH_IN     0.216608
TRANSFER    0.082734
DEBIT       0.006845
Name: type, dtype: float64
C1214450722    1.907350e-06
C309111136     1.907350e-06
C1268675361    1.907350e-06
C720460198     1.907350e-06
C1109092856    1.907350e-06
                   ...
C560131732     9.536752e-07
C455251560     9.536752e-07
C650578540     9.536752e-07
C1883668225    9.536752e-07
C1264356443    9.536752e-07
Name: nameOrig, Length: 1048317, dtype: float64
C985934102     9.346017e-05
C1286084959    9.155282e-05
C1590550415    8.487710e-05
C248609774     8.392342e-05
C665576141     8.296974e-05
                   ...
M2036888797    9.536752e-07
M1545238325    9.536752e-07
M382871047     9.536752e-07
M322765556     9.536752e-07
M677577406     9.536752e-07
Name: nameDest, Length: 449633, dtype: float64
```

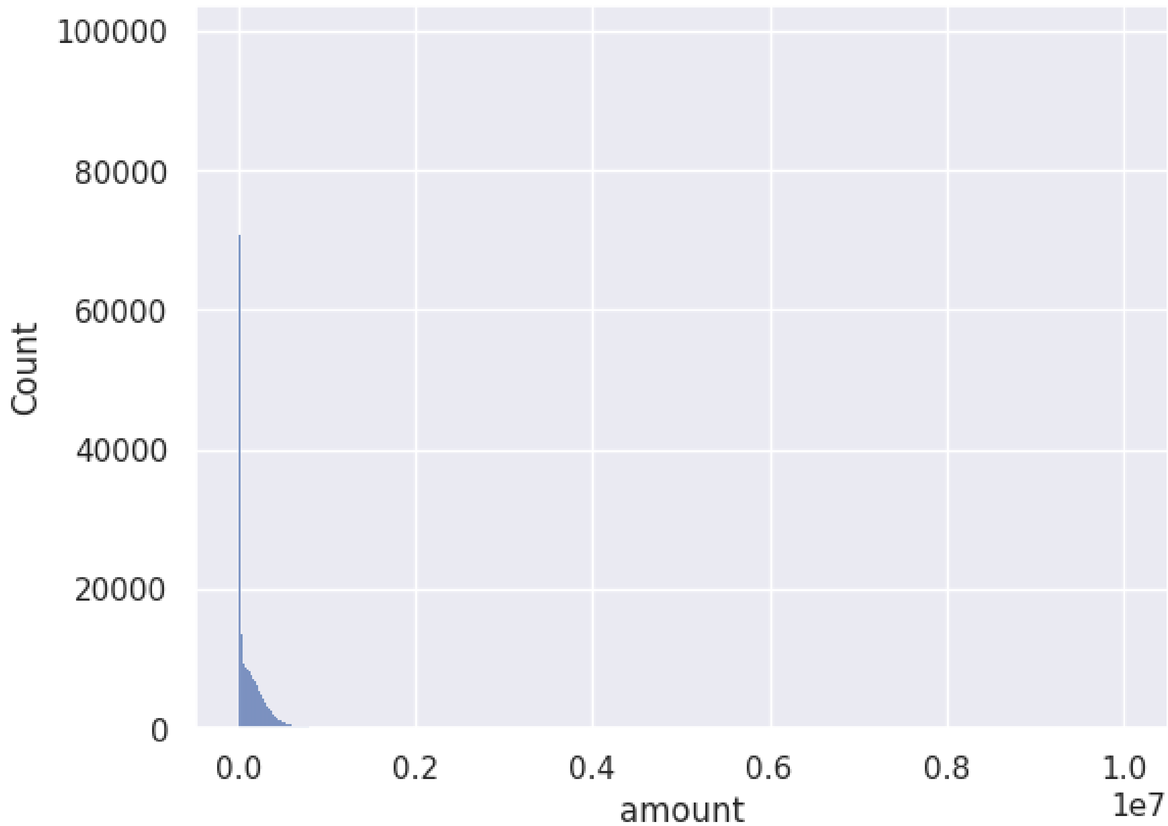
Plot the histograms to check distributions to find out if they are normal or skewed.If the variable follows normal distribution, then I will do Extreme Value Analysis. Otherwise if they are skewed, I will find IQR (Interquantile range).

In [30]:
```
import matplotlib as plt
import seaborn as sns
```

In [31]:
```
sns.set_theme()
```

In [32]:
```
sns.histplot(df["amount"])
```

Out[32]:
```
<AxesSubplot: xlabel='amount', ylabel='Count'>
```



From the histograms, the distribution is left skew because there may have some data points on right but the count is very small, and they are not shown in the histograms.

In [33]:
```
# sns.displot(df["amount"])
```

In [34]:
```
print(round(df[Numerical].describe()),2)
```

```
             step      amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
count   1048575.0   1048575.0      1048575.0       1048575.0       1048575.0   
mean         27.0    158668.0       874012.0        893809.0        978160.0   
std          16.0    264941.0      2971750.0       3008271.0       2296780.0   
min           1.0         0.0            0.0             0.0             0.0   
25%          15.0     12149.0            0.0             0.0             0.0   
50%          20.0     76346.0        16002.0             0.0        126377.0   
75%          39.0    213762.0       136647.0        174600.0        915923.0   
max          95.0  10000000.0     38900000.0      38900000.0      42100000.0   

       newbalanceDest    isFraud  
count       1048575.0  1048575.0  
mean        1114198.0        0.0  
std         2416593.0        0.0  
min               0.0        0.0  
25%               0.0        0.0  
50%          218260.0        0.0  
75%         1149808.0        0.0  
max        42200000.0        1.0   2
```

In [35]:
```
rslt_df_amount = df[df['amount'] > 2000000]
```

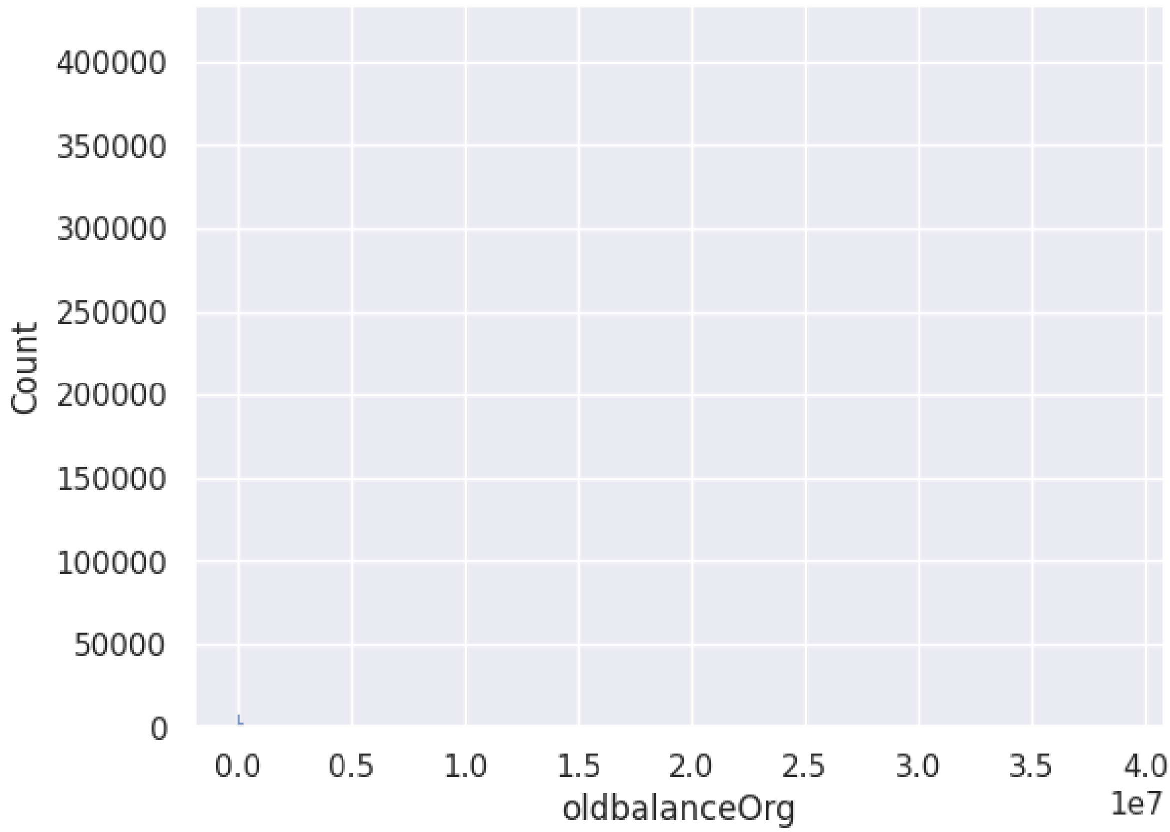
```
print('\nResult dataframe :\n', rslt_df_amount)
```

```
Result dataframe :
          step      type      amount     nameOrig  oldbalanceOrg  \
359         1  TRANSFER  2421578.09   C106297322           0.00
375         1  TRANSFER  2545478.01  C1057507014           0.00
376         1  TRANSFER  2061082.82  C2007599722           0.00
1153        1  TRANSFER  3776389.09   C197491520           0.00
1202        1  TRANSFER  2258388.15    C12139181           0.00
...       ...       ...         ...          ...            ...
1046593    95  TRANSFER  3605241.70   C971075452           0.00
1047484    95  TRANSFER  3148886.01   C857538059       18166.13
1047555    95  TRANSFER  2236090.83  C1291317126           0.00
1048027    95  TRANSFER  3572499.78  C1076671504           0.00
1048082    95  TRANSFER  2905341.96  C1244477950       41666.26

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud
359                 0.0  C1590550415      8515645.77     19200000.00        0
375                 0.0  C1590550415     12400000.00     19200000.00        0
376                 0.0  C1590550415     14900000.00     19200000.00        0
1153                0.0  C1883840933     10100000.00     16900000.00        0
1202                0.0  C1789550256      2784129.27      4619798.56        0
...                 ...          ...             ...             ...      ...
1046593             0.0   C273747506      4938367.64      8543609.34        0
1047484             0.0  C1022342908      3427656.32      6576542.33        0
1047555             0.0   C594568402      5612081.62      7848172.45        0
1048027             0.0  C1411451832      3835064.84      7407564.62        0
1048082             0.0  C1839656095      7592851.22     10500000.00        0

[3226 rows x 10 columns]
```

There are over 3000 data points with the "amount" fewer than 2 million that are not shown in the histogram.

In [36]:
```
sns.histplot(df['oldbalanceOrg'])
```

Out[36]: <AxesSubplot: xlabel='oldbalanceOrg', ylabel='Count'>



From the histogram, the distribution is extremely skew due to there are data points on the right with a small count and they are not displayed on the plot.

Selecting rows based on condition

In [37]:
```
rslt_df_old1 = df[df['oldbalanceOrg'] > 10000000]
print('\nResult dataframe :\n', rslt_df_old1)
```

```
Result dataframe :
            step     type      amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
1332           1  CASH_IN   143405.80  C2108708444     10100000.0      10200000.0
2956           2  CASH_IN   168681.06   C728236551     10300000.0      10500000.0
2957           2  CASH_IN   160370.33  C1887991591     10500000.0      10700000.0
3666           2  CASH_IN    10443.68  C2122225197     10200000.0      10300000.0
3667           2  CASH_IN    61344.23   C388143137     10300000.0      10300000.0
...          ...      ...         ...          ...            ...             ...
1044845       94  CASH_IN   218400.09  C1289764241     10700000.0      10900000.0
1044846       94  CASH_IN    32430.55  C1258100769     10900000.0      10900000.0
1044847       94  CASH_IN    92263.87  C1993376698     10900000.0      11000000.0
1044848       94  CASH_IN   288621.09  C1076570932     11000000.0      11300000.0
1044849       94  CASH_IN    56224.07   C225085973     11300000.0      11400000.0

              nameDest  oldbalanceDest  newbalanceDest  isFraud
1332        C667346055       195636.81      9291619.62        0
2956       C1401316767       229832.29        61151.23        0
2957         C11003494     10300000.00     10200000.00        0
3666       C1498438472        68235.87        57792.19        0
3667        C932583850      4410975.57      3512606.05        0
...                ...             ...             ...      ...
1044845     C840534489      1208196.76       989796.67        0
1044846    C1174072748      5071246.70      5038816.15        0
1044847    C2054046813       586755.72       494491.85        0
1044848     C506325881       902466.99       613845.90        0
1044849     C498120368       198792.29       142568.22        0

[24962 rows x 10 columns]
```

In [38]:
```
rslt_df_old2 = df[(df['oldbalanceOrg'] > 1000000) & (df['oldbalanceOrg'] <= 1000000
print('\nResult dataframe :\n', rslt_df_old2)
```

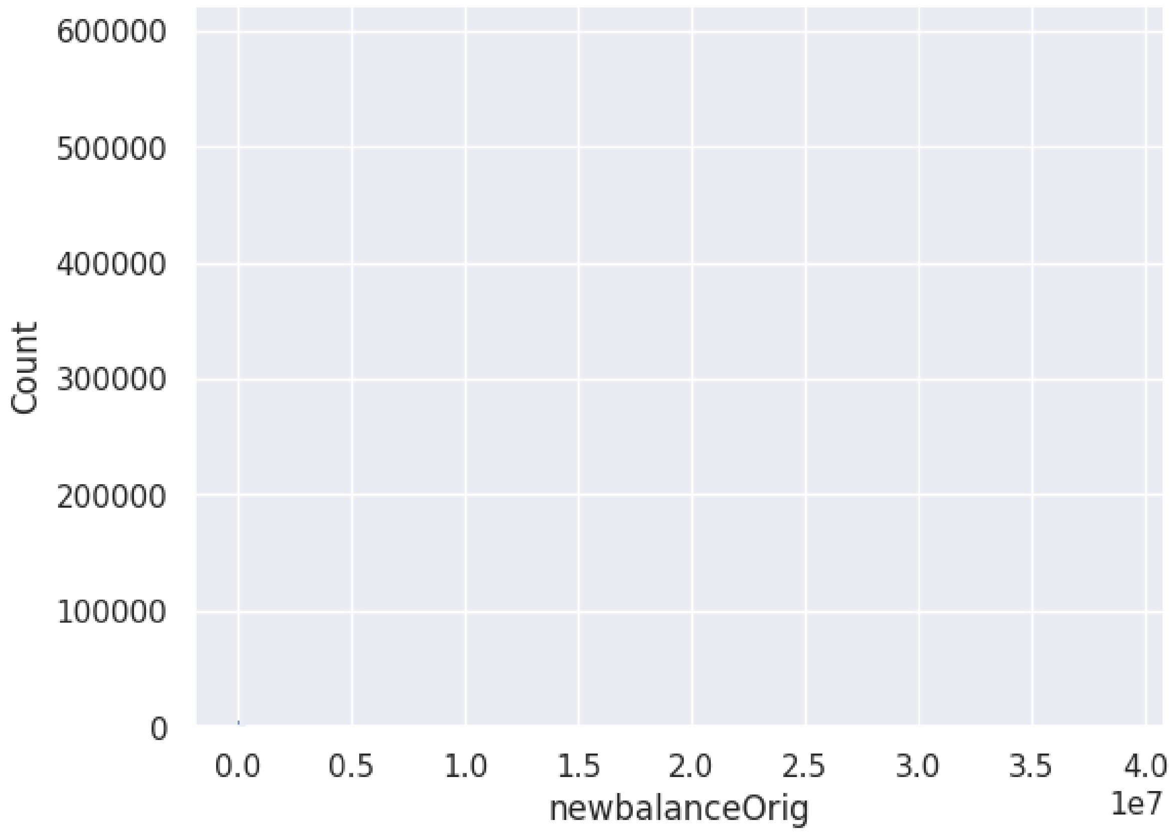
```
Result dataframe :
         step     type     amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
241         1  PAYMENT    4635.18  C1110698130     6313782.05      6309146.87
242         1  PAYMENT    1267.97  C1053632127     6309146.87      6307878.90
243         1  PAYMENT    6911.99    C89509666     6307878.90      6300966.92
244         1  PAYMENT    1795.67  C1016856028     6300966.92      6299171.25
245         1  PAYMENT    3199.06   C832292933     6299171.25      6295972.18
...       ...      ...        ...          ...            ...             ...
1048250    95  CASH_IN   48893.79  C1698409889     1772630.46      1821524.26
1048251    95  CASH_IN  210899.89   C569878803     1821524.26      2032424.15
1048252    95  CASH_IN  164119.42  C1912056627     2032424.15      2196543.58
1048253    95  CASH_IN   64558.24  C2056251453     2196543.58      2261101.82
1048254    95  CASH_IN  116591.73  C1113658472     2261101.82      2377693.55

            nameDest  oldbalanceDest  newbalanceDest  isFraud
241       M125644421            0.00            0.00        0
242      M1493158871            0.00            0.00        0
243      M1806880779            0.00            0.00        0
244       M446445803            0.00            0.00        0
245      M1280603381            0.00            0.00        0
...              ...             ...             ...      ...
1048250  C1597670440      4221410.32      4172516.53        0
1048251  C2020820364       261447.39        50547.50        0
1048252  C1763524045      3312216.93      3148097.51        0
1048253  C1602483192       195044.39       130486.15        0
1048254  C1024668435       629081.93       512490.20        0

[100897 rows x 10 columns]
```

In [39]:
```
print(Numerical)
```

```
['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanc
eDest', 'isFraud']
```

In [40]:
```
sns.histplot(df['newbalanceOrig'])
```

Out[40]:
```
<AxesSubplot: xlabel='newbalanceOrig', ylabel='Count'>
```



In [41]:
```
rslt_df_new1 = df[df['newbalanceOrig'] > 10000000]
print('\nResult dataframe :\n', rslt_df_new1)
```

```
Result dataframe :
         step     type     amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
1331        1  CASH_IN  183816.31   C205615258     9919025.73      10100000.0
1332        1  CASH_IN  143405.80  C2108708444    10100000.00      10200000.0
2955        2  CASH_IN  313452.48  C1842102517    10000000.00      10300000.0
2956        2  CASH_IN  168681.06   C728236551    10300000.00      10500000.0
2957        2  CASH_IN  160370.33  C1887991591    10500000.00      10700000.0
...       ...      ...        ...          ...            ...             ...
1044846    94  CASH_IN   32430.55  C1258100769    10900000.00      10900000.0
1044847    94  CASH_IN   92263.87  C1993376698    10900000.00      11000000.0
1044848    94  CASH_IN  288621.09  C1076570932    11000000.00      11300000.0
1044849    94  CASH_IN   56224.07   C225085973    11300000.00      11400000.0
1045429    94  CASH_IN  185599.69   C745726873     9919426.52      10100000.0

            nameDest  oldbalanceDest  newbalanceDest  isFraud
1331     C1023714065      2391652.26      1412484.09        0
1332      C667346055       195636.81      9291619.62        0
2955      C998351292       537694.37      1097218.45        0
2956     C1401316767       229832.29        61151.23        0
2957       C11003494     10300000.00     10200000.00        0
...              ...             ...             ...      ...
1044846  C1174072748      5071246.70      5038816.15        0
1044847  C2054046813       586755.72       494491.85        0
1044848   C506325881       902466.99       613845.90        0
1044849   C498120368       198792.29       142568.22        0
1045429  C1370748616       282298.75        96699.06        0

[25665 rows x 10 columns]
```

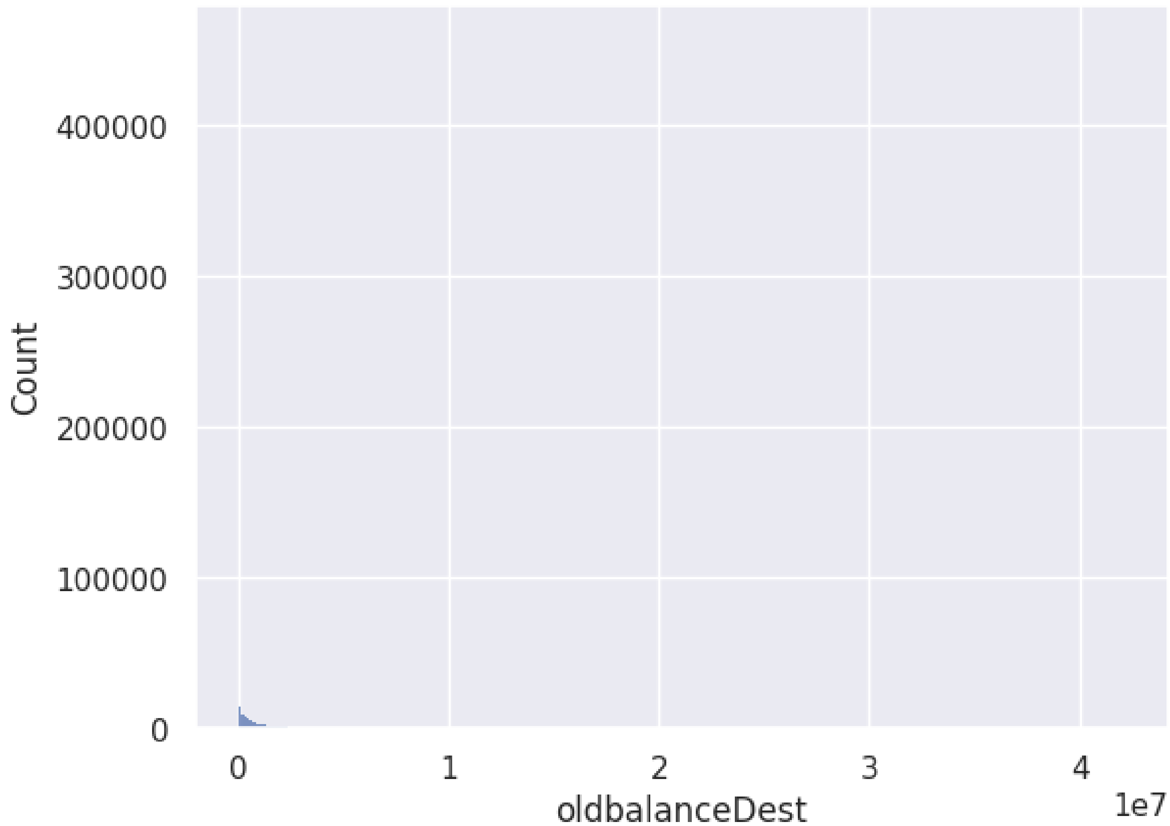
From the histogram, the distribution is extremely skew due to there are data points on the right with a small count and they are not displayed on the plot as seen below.

In [42]:
```python
rslt_df_new2 = df[(df['newbalanceOrig'] > 1000000) & (df['newbalanceOrig'] <= 10000
print('\nResult dataframe :\n', rslt_df_new2)
```

```
Result dataframe :
         step     type     amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
241         1  PAYMENT    4635.18  C1110698130     6313782.05      6309146.87
242         1  PAYMENT    1267.97  C1053632127     6309146.87      6307878.90
243         1  PAYMENT    6911.99   C89509666      6307878.90      6300966.92
244         1  PAYMENT    1795.67  C1016856028     6300966.92      6299171.25
245         1  PAYMENT    3199.06   C832292933     6299171.25      6295972.18
...       ...      ...        ...          ...            ...             ...
1048250    95  CASH_IN   48893.79  C1698409889     1772630.46      1821524.26
1048251    95  CASH_IN  210899.89   C569878803     1821524.26      2032424.15
1048252    95  CASH_IN  164119.42  C1912056627     2032424.15      2196543.58
1048253    95  CASH_IN   64558.24  C2056251453     2196543.58      2261101.82
1048254    95  CASH_IN  116591.73  C1113658472     2261101.82      2377693.55

            nameDest  oldbalanceDest  newbalanceDest  isFraud
241       M125644421            0.00            0.00        0
242      M1493158871            0.00            0.00        0
243      M1806880779            0.00            0.00        0
244       M446445803            0.00            0.00        0
245      M1280603381            0.00            0.00        0
...              ...             ...             ...      ...
1048250  C1597670440      4221410.32      4172516.53        0
1048251  C2020820364       261447.39        50547.50        0
1048252  C1763524045      3312216.93      3148097.51        0
1048253  C1602483192       195044.39       130486.15        0
1048254  C1024668435       629081.93       512490.20        0

[102619 rows x 10 columns]
```

In [43]:
```
sns.histplot(df['oldbalanceDest'])
```

Out[43]: <AxesSubplot: xlabel='oldbalanceDest', ylabel='Count'>



In [44]:
```
rslt_df_oldb1 = df[df['oldbalanceDest'] > 10000000]
print('\nResult dataframe :\n', rslt_df_oldb1)
```

```
Result dataframe :
         step      type      amount     nameOrig  oldbalanceOrg  \
362         1  TRANSFER  1457213.54   C396918327           0.00
375         1  TRANSFER  2545478.01  C1057507014           0.00
376         1  TRANSFER  2061082.82  C2007599722           0.00
432         1   CASH_IN   349505.89   C173791568     7330235.59
463         1   CASH_IN   222711.47  C2123533871     2419068.96
...       ...       ...         ...          ...            ...
1047904    95  CASH_OUT   257225.25  C1001899290           0.00
1047974    95  TRANSFER   166846.64  C2140727709       72723.00
1048011    95  TRANSFER  1339844.75  C1604159675           0.00
1048311    95  CASH_OUT   127581.61  C1567904281       19465.38
1048561    95     DEBIT     7880.88   C233708423       31489.00

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud
362                0.00  C1590550415      10900000.0      19200000.0        0
375                0.00  C1590550415      12400000.0      19200000.0        0
376                0.00  C1590550415      14900000.0      19200000.0        0
432          7679741.48  C1590550415      17000000.0      19200000.0        0
463          2641780.43  C1590550415      16700000.0      19200000.0        0
...                 ...          ...             ...             ...      ...
1047904            0.00   C703592419      24800000.0      25100000.0        0
1047974            0.00  C1753106347      13400000.0      13600000.0        0
1048011            0.00   C891089759      31500000.0      32800000.0        0
1048311            0.00   C623154526      12200000.0      12400000.0        0
1048561        23608.12   C794801857      18700000.0      18700000.0        0

[13936 rows x 10 columns]
```

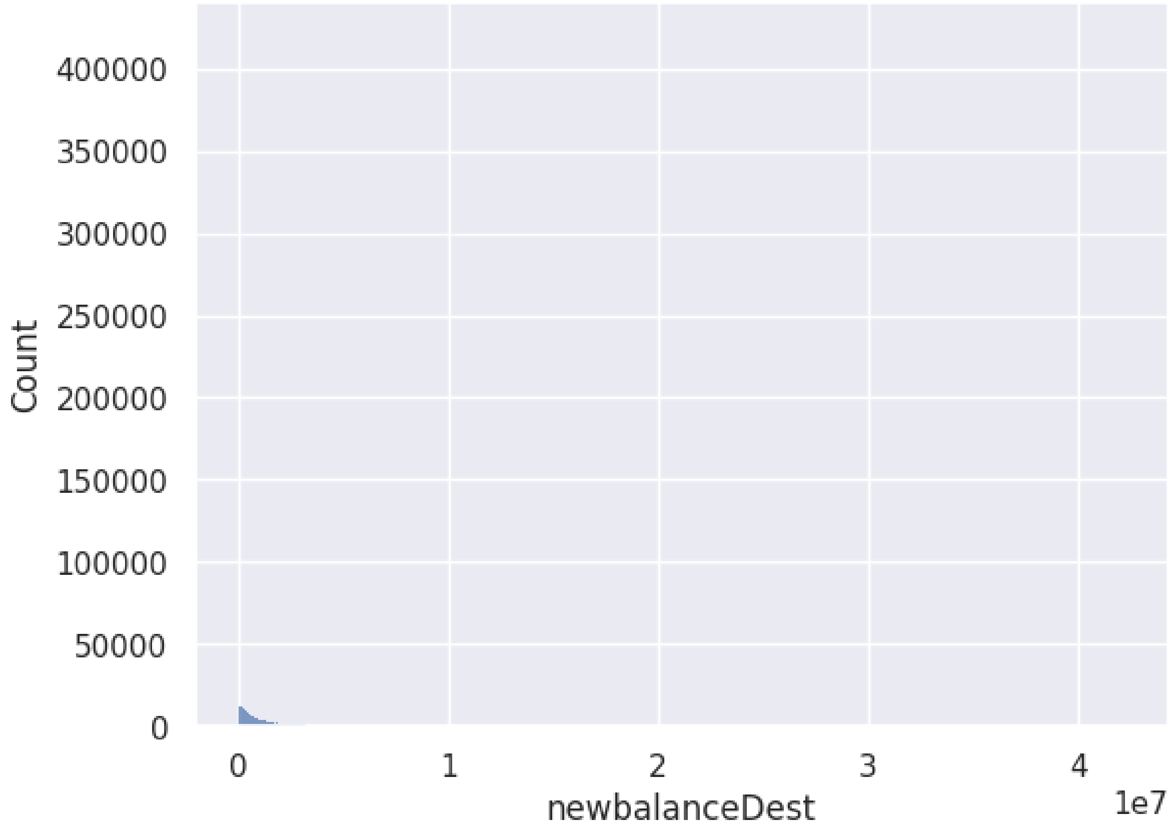
In [45]:
```
rslt_df_oldb2 = df[(df['oldbalanceDest'] > 1000000) & (df['oldbalanceDest'] <= 1000
print('\nResult dataframe :\n', rslt_df_oldb2)
```

```
Result dataframe :
         step      type      amount     nameOrig  oldbalanceOrg  \
88          1  TRANSFER   761507.39   C412788346           0.00   
89          1  TRANSFER  1429051.47  C1520267010           0.00   
94          1  TRANSFER  1724887.05  C1495608502           0.00   
96          1  TRANSFER   581294.26   C843299092           0.00   
113         1  CASH_OUT    85423.63   C460741164           0.00   
...       ...       ...         ...          ...            ...   
1048514    95  TRANSFER   136218.95   C969666269       43866.00   
1048519    95  CASH_OUT   271378.96  C1617333943        6268.62   
1048549    95     DEBIT     9983.95   C130161561         997.00   
1048553    95  CASH_OUT   103391.93  C2021893664      107623.00   
1048567    95  CASH_OUT   279674.05   C990252469       18002.85   

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  
88                 0.00  C1590550415      1280036.23     19200000.00        0  
89                 0.00  C1590550415      2041543.62     19200000.00        0  
94                 0.00  C1590550415      3470595.10     19200000.00        0  
96                 0.00  C1590550415      5195482.15     19200000.00        0  
113                0.00  C1590550415      5776776.41     19200000.00        0  
...                 ...          ...             ...             ...      ...  
1048514            0.00  C1199486666      1756330.45      1892549.39        0  
1048519            0.00  C1654714274      3538250.81      3809629.77        0  
1048549            0.00   C740635949      2341925.24      2351909.19        0  
1048553         4231.07   C178178755      5752648.68      5856040.61        0  
1048567            0.00   C574439165      1847488.28      2127162.32        0  

[233950 rows x 10 columns]
```

From the histogram, the distribution is extremely skew due to there are data points on the right with a small count and they are not displayed on the plot as seen above.

In [46]:
```python
sns.histplot(df['newbalanceDest'])
```

Out[46]:
```
<AxesSubplot: xlabel='newbalanceDest', ylabel='Count'>
```



From the histogram, the distribution is extremely skew due to there are data points on the right with a small count and they are not displayed on the plot as seen below.

In [47]:
```
rslt_df_newb1 = df[df['newbalanceDest'] > 10000000]
print('\nResult dataframe :\n', rslt_df_newb1)
```

```
Result dataframe :
         step      type      amount     nameOrig  oldbalanceOrg  \
84          1  TRANSFER   379856.23  C1449772539           0.00
88          1  TRANSFER   761507.39   C412788346           0.00
89          1  TRANSFER  1429051.47  C1520267010           0.00
94          1  TRANSFER  1724887.05  C1495608502           0.00
96          1  TRANSFER   581294.26   C843299092           0.00
...       ...       ...         ...          ...            ...
1047974    95  TRANSFER   166846.64  C2140727709       72723.00
1048011    95  TRANSFER  1339844.75  C1604159675           0.00
1048082    95  TRANSFER  2905341.96  C1244477950       41666.26
1048311    95  CASH_OUT   127581.61  C1567904281       19465.38
1048561    95     DEBIT     7880.88   C233708423       31489.00

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud
84                 0.00  C1590550415       900180.00      19200000.0        0
88                 0.00  C1590550415      1280036.23      19200000.0        0
89                 0.00  C1590550415      2041543.62      19200000.0        0
94                 0.00  C1590550415      3470595.10      19200000.0        0
96                 0.00  C1590550415      5195482.15      19200000.0        0
...                 ...          ...             ...             ...      ...
1047974            0.00  C1753106347     13400000.00      13600000.0        0
1048011            0.00   C891089759     31500000.00      32800000.0        0
1048082            0.00  C1839656095      7592851.22      10500000.0        0
1048311            0.00   C623154526     12200000.00      12400000.0        0
1048561        23608.12   C794801857     18700000.00      18700000.0        0

[15636 rows x 10 columns]
```

In [48]:
```
rslt_df_newb2 = df[(df['newbalanceDest'] > 1000000) & (df['newbalanceDest'] <= 1000
print('\nResult dataframe :\n', rslt_df_newb2)
```

```
Result dataframe :
         step      type      amount     nameOrig  oldbalanceOrg  \
24          1  TRANSFER   311685.89  C1984094095       10835.00
48          1  CASH_OUT     5346.89   C512549200           0.00
83          1  TRANSFER   125872.53  C1443967876           0.00
85          1  TRANSFER  1505626.01   C926859124           0.00
90          1  TRANSFER   358831.92   C908084672           0.00
...       ...       ...         ...          ...            ...
1048519    95  CASH_OUT   271378.96  C1617333943        6268.62
1048520    95  CASH_OUT    61654.72   C187514699       16057.00
1048549    95     DEBIT     9983.95   C130161561         997.00
1048553    95  CASH_OUT   103391.93  C2021893664      107623.00
1048567    95  CASH_OUT   279674.05   C990252469       18002.85

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud
24                 0.00   C932583850         6267.00      2719172.89        0
48                 0.00   C248609774       652637.00      6453430.91        0
83                 0.00   C392292416       348512.00      3420103.09        0
85                 0.00   C665576141        29031.00      5515763.34        0
90                 0.00   C392292416       474384.53      3420103.09        0
...                 ...          ...             ...             ...      ...
1048519            0.00  C1654714274      3538250.81      3809629.77        0
1048520            0.00  C1195667457       849667.70      1082415.65        0
1048549            0.00   C740635949      2341925.24      2351909.19        0
1048553         4231.07   C178178755      5752648.68      5856040.61        0
1048567            0.00   C574439165      1847488.28      2127162.32        0

[271694 rows x 10 columns]
```

In [49]:
```python
rslt_df_newb1 = df[df['newbalanceDest'] > 1000000]
print('\nResult dataframe :\n', rslt_df_newb1)
```

```
Result dataframe :
         step      type      amount     nameOrig  oldbalanceOrg  \
24          1  TRANSFER   311685.89  C1984094095       10835.00   
48          1  CASH_OUT     5346.89   C512549200           0.00   
83          1  TRANSFER   125872.53  C1443967876           0.00   
84          1  TRANSFER   379856.23  C1449772539           0.00   
85          1  TRANSFER  1505626.01   C926859124           0.00   
...       ...       ...         ...          ...            ...   
1048520    95  CASH_OUT    61654.72   C187514699       16057.00   
1048549    95     DEBIT     9983.95   C130161561         997.00   
1048553    95  CASH_OUT   103391.93  C2021893664      107623.00   
1048561    95     DEBIT     7880.88   C233708423       31489.00   
1048567    95  CASH_OUT   279674.05   C990252469       18002.85   

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  
24                 0.00   C932583850         6267.00      2719172.89        0  
48                 0.00   C248609774       652637.00      6453430.91        0  
83                 0.00   C392292416       348512.00      3420103.09        0  
84                 0.00  C1590550415       900180.00     19200000.00        0  
85                 0.00   C665576141        29031.00      5515763.34        0  
...                 ...          ...             ...             ...      ...  
1048520            0.00  C1195667457       849667.70      1082415.65        0  
1048549            0.00   C740635949      2341925.24      2351909.19        0  
1048553         4231.07   C178178755      5752648.68      5856040.61        0  
1048561        23608.12   C794801857     18700000.00     18700000.00        0  
1048567            0.00   C574439165      1847488.28      2127162.32        0  

[287330 rows x 10 columns]
```

There are over 280,000 "newbalanceDest" data points greater than one million that are not displayed on the histogram.

Correlation Analysis

In [50]:
```
df.head(20)
```

Out[50]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest
0	1	PAYMENT	9839.640000	C1231006815	170136.000000	160296.36	M1979787155
1	1	PAYMENT	1864.280000	C1666544295	21249.000000	19384.72	M2044282225
2	1	TRANSFER	181.000000	C1305486145	181.000000	0.00	C553264065
3	1	CASH_OUT	158667.712672	C840083671	181.000000	0.00	C38997010
4	1	PAYMENT	11668.140000	C2048537720	41554.000000	29885.86	M1230701703
5	1	PAYMENT	7817.710000	C90045638	53860.000000	46042.29	M573487274
6	1	PAYMENT	7107.770000	C154988899	183195.000000	176087.23	M408069119
7	1	PAYMENT	7861.640000	C1912850431	176087.230000	168225.59	M633326333
8	1	PAYMENT	158667.712672	C1265012928	2671.000000	0.00	M1176932104
9	1	DEBIT	5337.770000	C712410124	874012.004459	36382.23	C195600860
10	1	DEBIT	9644.940000	C1900366749	4465.000000	0.00	NaN
11	1	PAYMENT	3099.970000	C249177573	20771.000000	17671.03	M2096539129
12	1	PAYMENT	2560.740000	C1648232591	5070.000000	2509.26	M972865270
13	1	PAYMENT	158667.712672	C1716932897	10127.000000	0.00	M801569151
14	1	PAYMENT	4098.780000	C1026483832	503264.000000	499165.22	M1635378213
15	1	CASH_OUT	229133.940000	C905080434	15325.000000	0.00	C476402209
16	1	PAYMENT	1563.820000	C761750706	450.000000	0.00	M1731217984
17	1	PAYMENT	1157.860000	C1237762639	21156.000000	19998.14	M1877062907
18	1	PAYMENT	158667.712672	C2033524545	15123.000000	14451.36	M473053293
19	1	TRANSFER	215310.300000	C1670993182	874012.004459	0.00	C1100439041

In [51]:

```
new_df = df.dropna()

new_df.head(20)
```

Out[51]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest
0	1	PAYMENT	9839.640000	C1231006815	170136.000000	160296.36	M1979787155
1	1	PAYMENT	1864.280000	C1666544295	21249.000000	19384.72	M2044282225
2	1	TRANSFER	181.000000	C1305486145	181.000000	0.00	C553264065
3	1	CASH_OUT	158667.712672	C840083671	181.000000	0.00	C38997010
4	1	PAYMENT	11668.140000	C2048537720	41554.000000	29885.86	M1230701703
5	1	PAYMENT	7817.710000	C90045638	53860.000000	46042.29	M573487274
6	1	PAYMENT	7107.770000	C154988899	183195.000000	176087.23	M408069119
7	1	PAYMENT	7861.640000	C1912850431	176087.230000	168225.59	M633326333
8	1	PAYMENT	158667.712672	C1265012928	2671.000000	0.00	M1176932104
9	1	DEBIT	5337.770000	C712410124	874012.004459	36382.23	C195600860
11	1	PAYMENT	3099.970000	C249177573	20771.000000	17671.03	M2096539129
12	1	PAYMENT	2560.740000	C1648232591	5070.000000	2509.26	M972865270
13	1	PAYMENT	158667.712672	C1716932897	10127.000000	0.00	M801569151
14	1	PAYMENT	4098.780000	C1026483832	503264.000000	499165.22	M1635378213
15	1	CASH_OUT	229133.940000	C905080434	15325.000000	0.00	C476402209
16	1	PAYMENT	1563.820000	C761750706	450.000000	0.00	M1731217984
17	1	PAYMENT	1157.860000	C1237762639	21156.000000	19998.14	M1877062907
18	1	PAYMENT	158667.712672	C2033524545	15123.000000	14451.36	M473053293
19	1	TRANSFER	215310.300000	C1670993182	874012.004459	0.00	C1100439041
21	1	DEBIT	9302.790000	C1566511282	11299.000000	1996.21	C1973538135

In [52]:

```
sub_df = new_df.iloc[:1000]

sub_df.shape
```

Out[52]: (1000, 10)

In [53]:

```
sub_df['amount'].dtypes
```

Out[53]: dtype('float64')

In [54]:

```
corr = sub_df['amount'].corr(sub_df['oldbalanceOrg'])
```

In [55]:

```
corr
```

Out[55]: 0.05746595805193277

In [56]:

```
sub_df.corr()
```

```
/tmp/ipykernel_341/1540764471.py:1: FutureWarning: The default value of numeric_on
ly in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  sub_df.corr()
```

Out[56]:

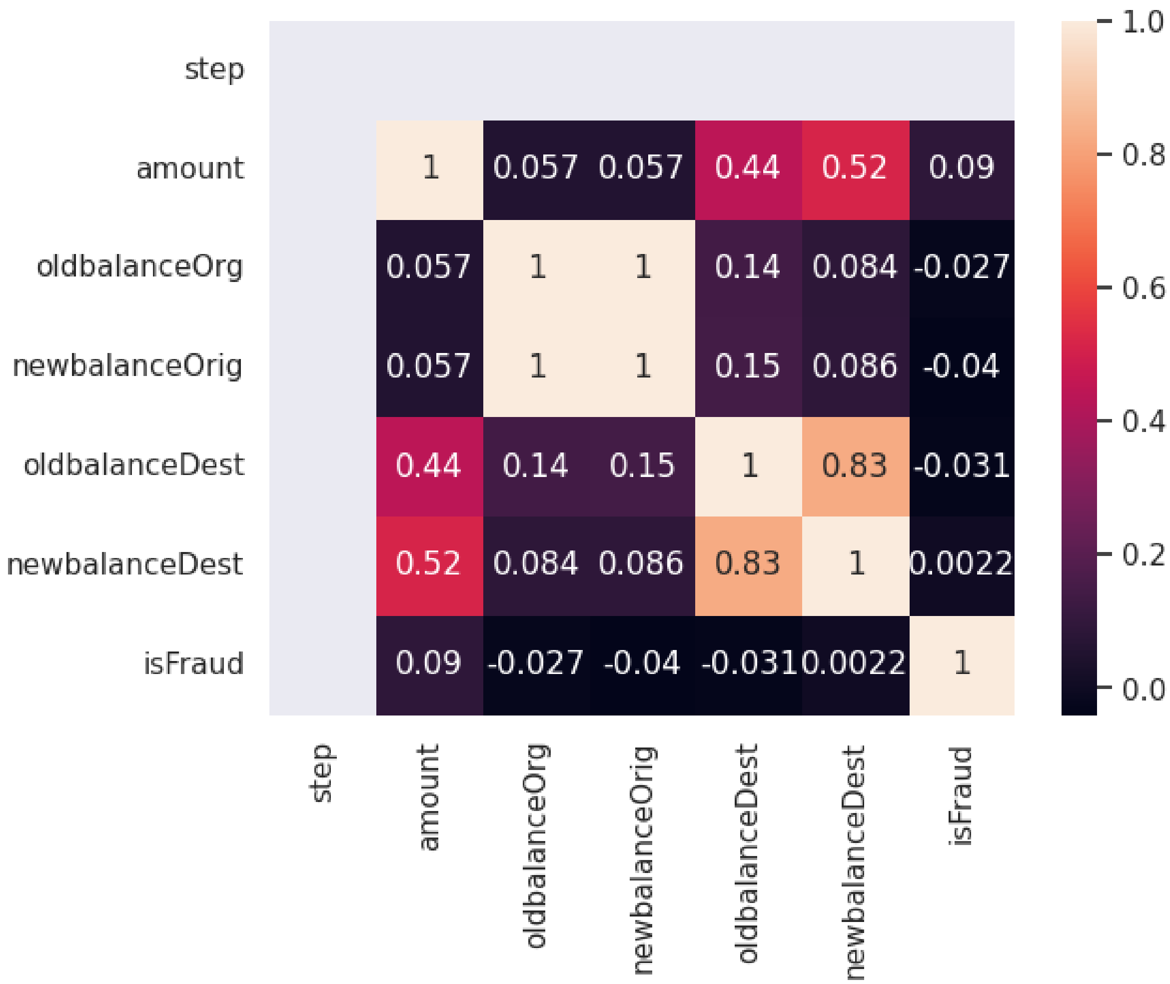
	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
step	NaN	NaN	NaN	NaN	NaN	NaN
amount	NaN	1.000000	0.057466	0.057246	0.440153	0.515642
oldbalanceOrg	NaN	0.057466	1.000000	0.998889	0.139919	0.084473
newbalanceOrig	NaN	0.057246	0.998889	1.000000	0.145127	0.086371
oldbalanceDest	NaN	0.440153	0.139919	0.145127	1.000000	0.826707
newbalanceDest	NaN	0.515642	0.084473	0.086371	0.826707	1.000000
isFraud	NaN	0.089688	-0.026537	-0.039657	-0.030532	0.002209

In [57]:

```
sns.heatmap(sub_df.corr(), annot = True)
```

```
/tmp/ipykernel_341/3445893564.py:1: FutureWarning: The default value of numeric_on
ly in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  sns.heatmap(sub_df.corr(), annot = True)
```

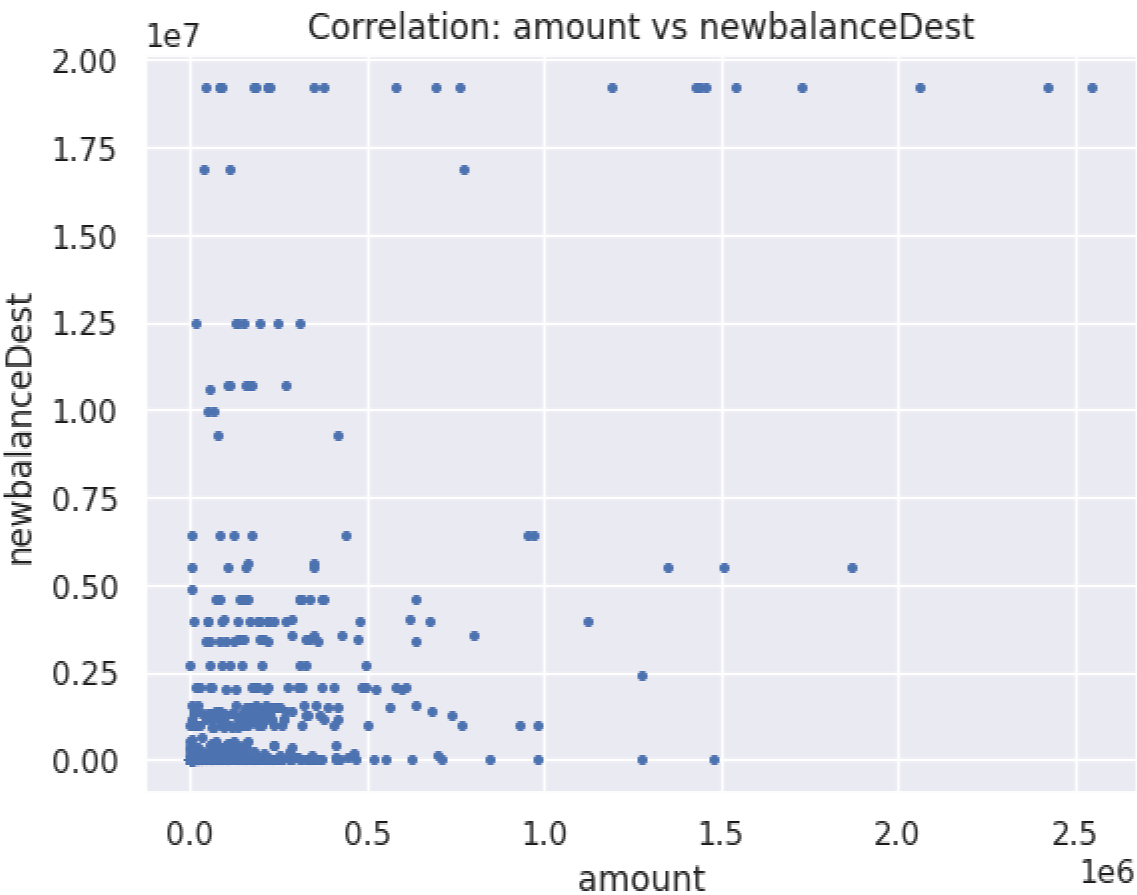
Out[57]: <AxesSubplot: >



Correlation Interpretation

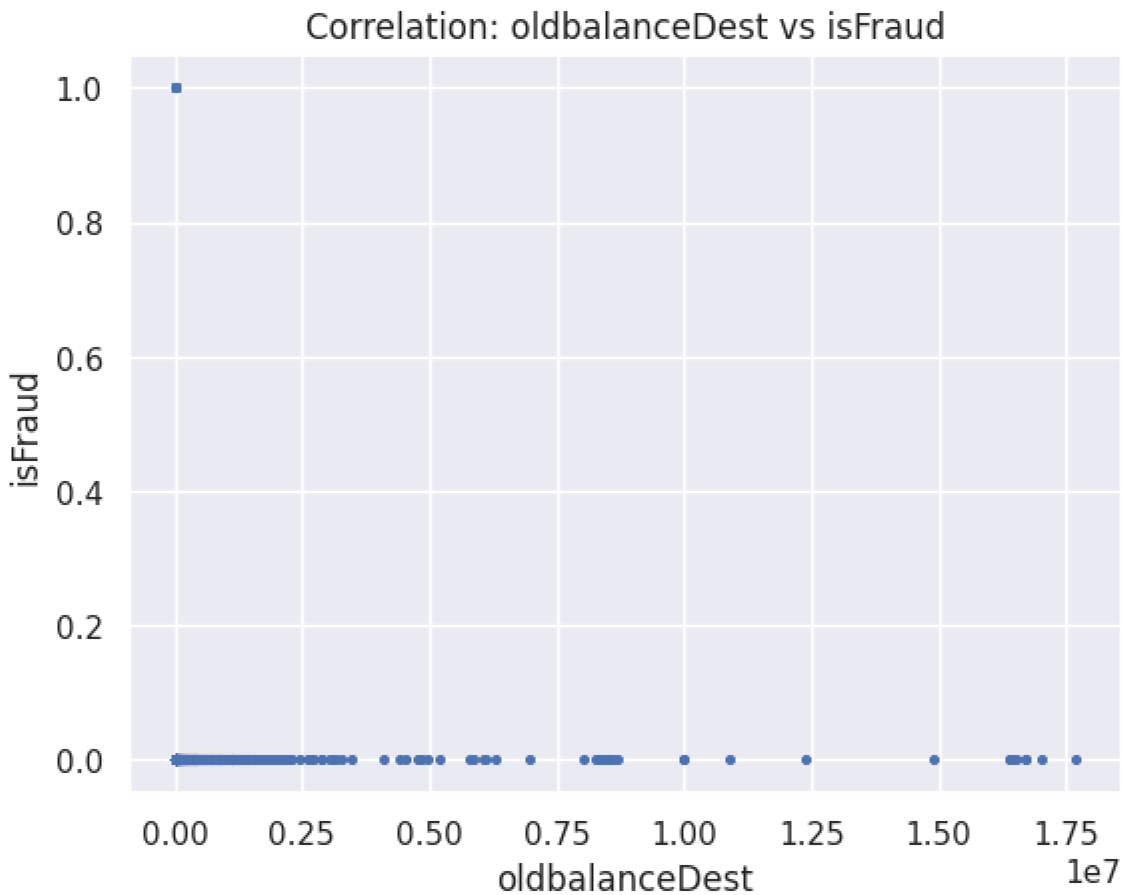
The Heatmap represents the correlation between all numerical varialbes together. The number which is closer to 1 means these two varialbes are strongly positive correlated. The correlation value which is closer to 0 tells us that the two variables are not correlated.

In [58]:
```
import matplotlib.pyplot as plt

plt.scatter(sub_df['amount'], sub_df['newbalanceDest'], s = 7)
plt.title("Correlation: amount vs newbalanceDest")
plt.xlabel("amount")
plt.ylabel("newbalanceDest")
plt.show()
```



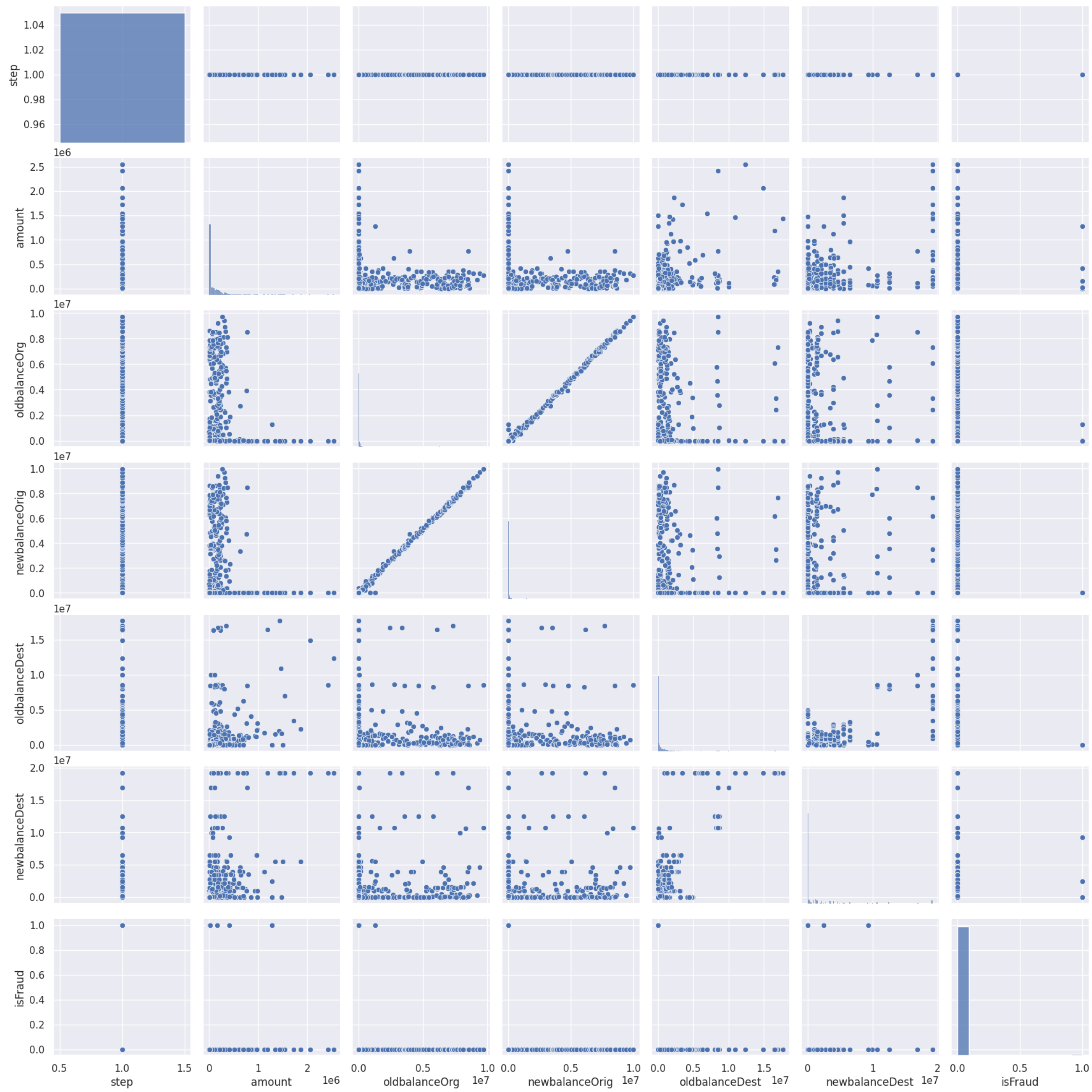
From plot, When the amount of transction is not too large, under 1M, the two varialbes are strongly correlated. That means when the newbalacneDest value is too large, then there are very few transactions or deposit to that account

In [59]:
```
plt.scatter(sub_df['oldbalanceDest'], sub_df['isFraud'], s = 7)
plt.title("Correlation: oldbalanceDest vs isFraud")
plt.xlabel("oldbalanceDest")
plt.ylabel("isFraud")
plt.show()
```



In [60]:
```
sns.pairplot(sub_df)
```

Out[60]: <seaborn.axisgrid.PairGrid at 0x7fb999d0eca0>



Looking at this plot, we see that there is only one point for the variable isFraud because this variable just has two unique values, 0 or 1.

The point is scattered away from the plot of the oldbalanceDest and this is corroborated by the fact that the correlation coefficient for this pair is close to 0.

Also, the single point at the upper left corner tells us that when the value of oldbalanceDest increases, further going to right, the value of isFraud is further to the left, which is supported by the fact that the correlation coefficient for this pair is negative (-0.031)

In [61]:
```
sub_df1 = new_df.iloc[:20000]
```

```
sub_df1.shape
```

Out[61]:
```
(20000, 10)
```

In [62]:
```
corr1 = sub_df1['amount'].corr(sub_df1['oldbalanceOrg'])

display(corr)
```

```
0.05746595805193277
```

In [ ]:

In [63]:
```
sub_df1.corr()
```

```
/tmp/ipykernel_341/1118181706.py:1: FutureWarning: The default value of numeric_on
ly in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  sub_df1.corr()
```

Out[63]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalance
step	1.000000	0.067545	-0.038007	-0.038375	-0.012620	0.03
amount	0.067545	1.000000	0.054308	0.035146	0.340439	0.44
oldbalanceOrg	-0.038007	0.054308	1.000000	0.997794	0.166233	0.12
newbalanceOrig	-0.038375	0.035146	0.997794	1.000000	0.171693	0.12
oldbalanceDest	-0.012620	0.340439	0.166233	0.171693	1.000000	0.90
newbalanceDest	0.030528	0.449894	0.120893	0.122769	0.907202	1.00
isFraud	-0.039192	0.101625	-0.005229	-0.022207	-0.015409	-0.00

In [64]:
```
sub_df.corr()
```

```
/tmp/ipykernel_341/1540764471.py:1: FutureWarning: The default value of numeric_on
ly in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  sub_df.corr()
```

Out[64]:

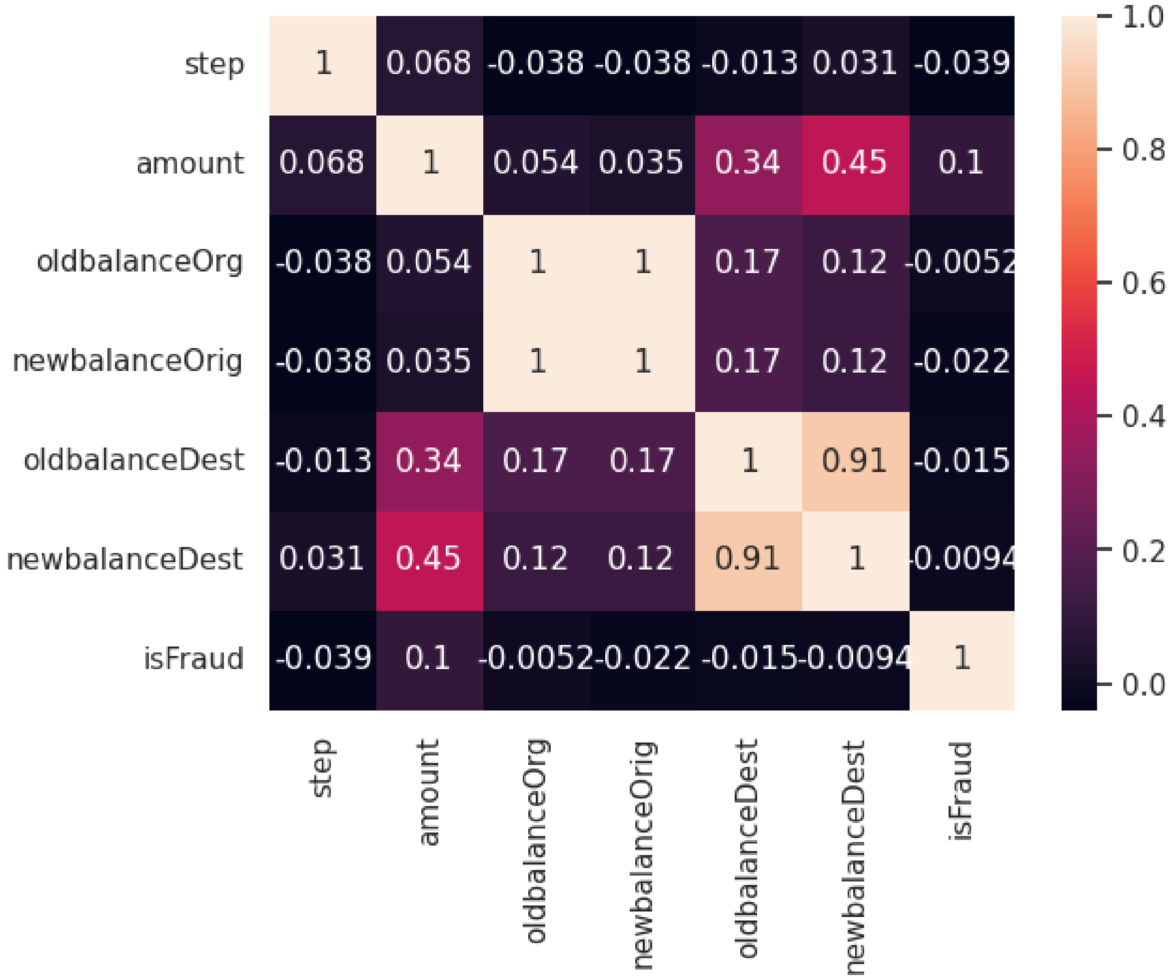
	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
step	NaN	NaN	NaN	NaN	NaN	NaN
amount	NaN	1.000000	0.057466	0.057246	0.440153	0.515642
oldbalanceOrg	NaN	0.057466	1.000000	0.998889	0.139919	0.084473
newbalanceOrig	NaN	0.057246	0.998889	1.000000	0.145127	0.086371
oldbalanceDest	NaN	0.440153	0.139919	0.145127	1.000000	0.826707
newbalanceDest	NaN	0.515642	0.084473	0.086371	0.826707	1.000000
isFraud	NaN	0.089688	-0.026537	-0.039657	-0.030532	0.002209

In [65]:

```
h1 = sns.heatmap(sub_df1.corr(), annot = True)

h1
```

```
/tmp/ipykernel_341/433119671.py:1: FutureWarning: The default value of numeric_onl
y in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  h1 = sns.heatmap(sub_df1.corr(), annot = True)
```

Out[65]: <AxesSubplot: >

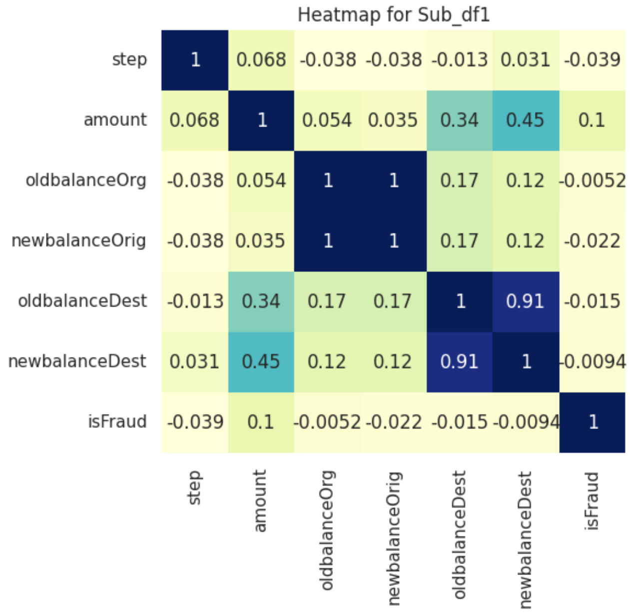


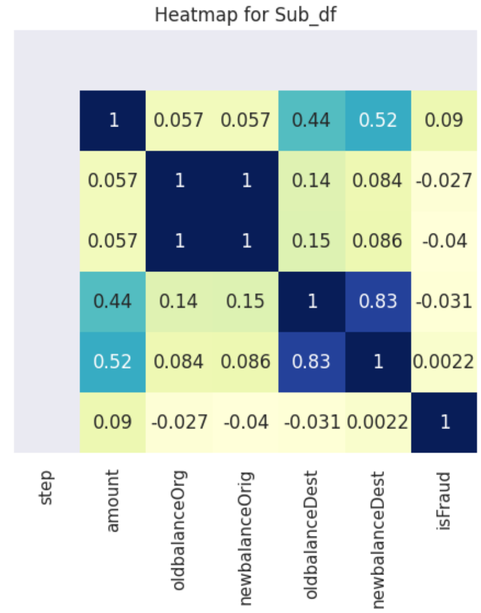
In [66]:

```
# sns.heatmap(sub_df1.corr(), annot = True)

# sns.heatmap(sub_df.corr(), annot = True)

f,(ax1,ax2) = plt.subplots(1,2,sharey=True, figsize=(12,5))
g1 = sns.heatmap(sub_df1.corr(),cmap="YlGnBu", annot = True, cbar=False,ax=ax1)
g1.set_title('Heatmap for Sub_df1')
g1.set_ylabel('')
g1.set_xlabel('')
g2 = sns.heatmap(sub_df.corr(),cmap="YlGnBu",annot = True, cbar=False,ax=ax2)
g2.set_title('Heatmap for Sub_df')
g2.set_ylabel('')
g2.set_xlabel('')
```

```
/tmp/ipykernel_341/220771305.py:6: FutureWarning: The default value of numeric_onl
y in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  g1 = sns.heatmap(sub_df1.corr(),cmap="YlGnBu", annot = True, cbar=False,ax=ax1)
/tmp/ipykernel_341/220771305.py:10: FutureWarning: The default value of numeric_on
ly in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  g2 = sns.heatmap(sub_df.corr(),cmap="YlGnBu",annot = True, cbar=False,ax=ax2)
```

Out[66]: `Text(0.5, 21.249999999999993, '')`



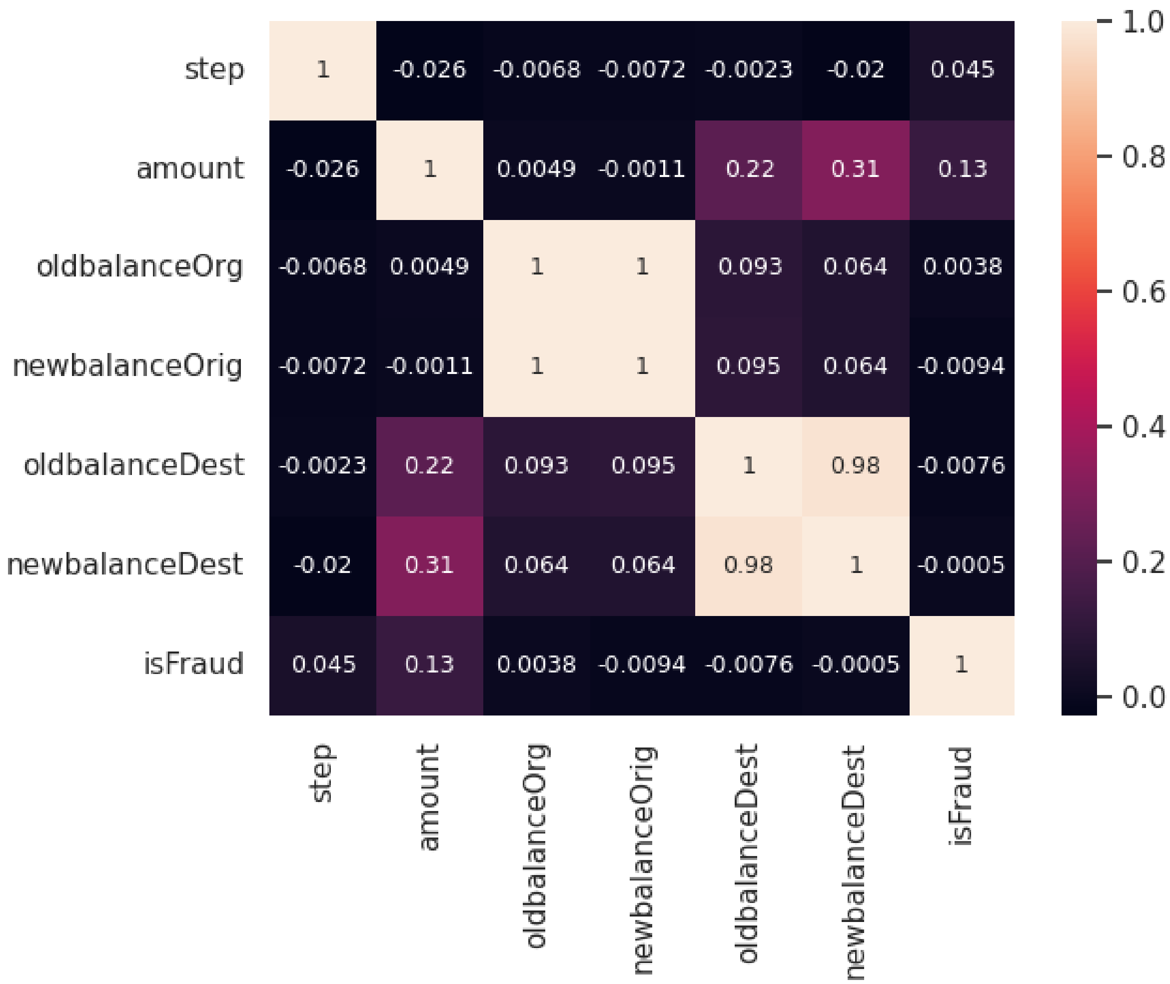


In [67]:

```
sns.heatmap(df.corr(), annot = True, annot_kws = {'fontsize': 9})
```

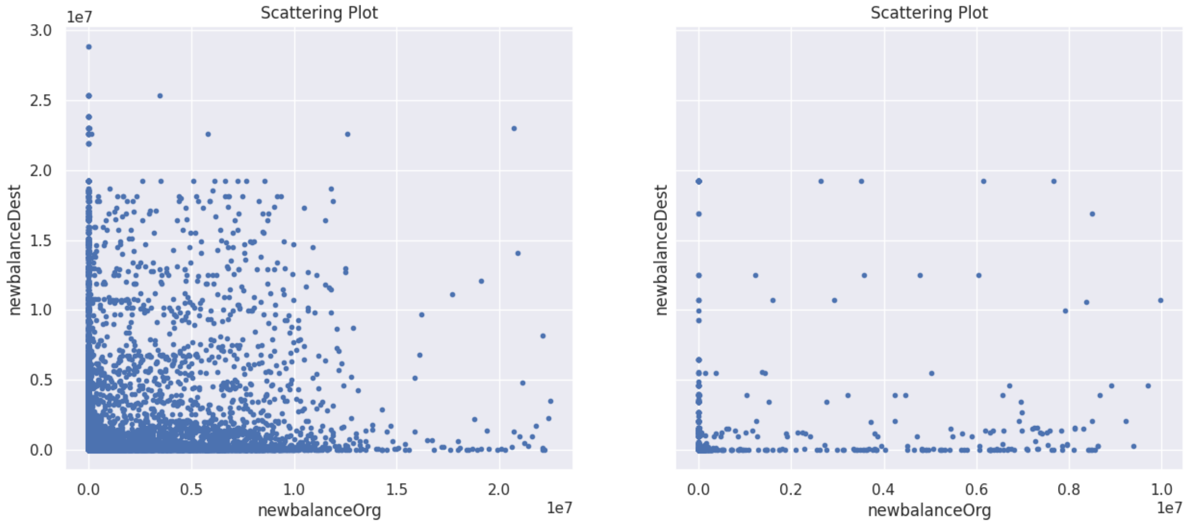
```
/tmp/ipykernel_341/1941133742.py:1: FutureWarning: The default value of numeric_on
ly in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  sns.heatmap(df.corr(), annot = True, annot_kws = {'fontsize': 9})
```

Out[67]: <AxesSubplot: >



In [68]:
```
# Create two subplots and unpack the output array immediately
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(15,6))
ax1.scatter(sub_df1['newbalanceOrig'], sub_df1['newbalanceDest'], s =9)
ax1.set_title("Scattering Plot")
ax1.set_xlabel('newbalanceOrg')
ax1.set_ylabel('newbalanceDest')

ax2.scatter(sub_df['newbalanceOrig'], sub_df['newbalanceDest'], s =9)
ax2.set_title("Scattering Plot")
ax2.set_xlabel('newbalanceOrg')
ax2.set_ylabel('newbalanceDest')
```

Out[68]: `Text(0, 0.5, 'newbalanceDest')`



Comparison the correlation from the two data sets

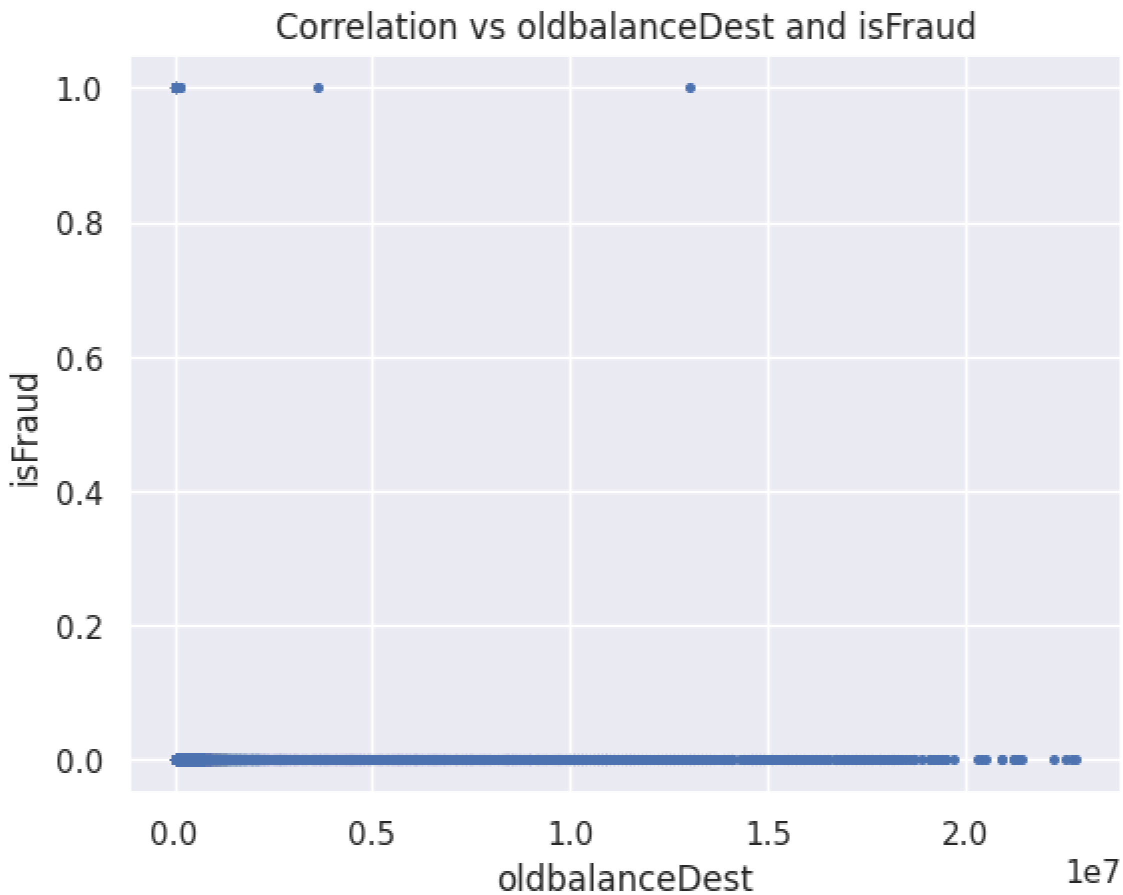
Sub_df1 has values for step while the sub_df doesn't have because when haveing more records the correlation coefficients have changed.

From the heatmap plot, the values under the diagoanl in sub_df1 a lit larger than that of in the sub_df because when there are more records, the number of transactions increase. This means there is more money the destination.

On the contrary, the values above the diagoanl in sub_df1 a lit smaller than that of in the sub_df because there is less money in Origin account.

In [69]:
```
import matplotlib.pyplot as plt

plt.scatter(sub_df1['amount'], sub_df1['newbalanceDest'], s = 7)
plt.title("Correlation vs amount and newbalanceDest")
plt.xlabel("amount")
plt.ylabel("newbalanceDest")
plt.show()
```

In [70]:
```
plt.scatter(sub_df1['oldbalanceDest'], sub_df1['isFraud'], s = 7)
plt.title("Correlation vs oldbalanceDest and isFraud")
plt.xlabel("oldbalanceDest")
plt.ylabel("isFraud")
plt.show()
```



EDA Summary

1. Missing Values: There are missing values. Specifically, there are 5 missing values in column name at rows 3, 8, 13, 18, and 23. There are 3 missing values in column oldbalanceOrig at rows 9, 19, and 29. There are also 3 missing values in nameDest at rows 10, 20 , and 30.

2. By looking at histogram plots and the nature of the dataset, we decided to use mean for the imputation for the missing values.

3. Hostogram plots: Almost features have left-skew histogram. This means there may have some data points on right but the count is very small, and they are not shown in the histograms.

4. Correlation Analysis. We have conducted both scatter plots and Heatmap matrices for all pairs. We saw that there are some paris that are highly positive correlated such as oldbalanceDest and newbalanceDest features. There seeems no negative correlation between the independent variables

and bwtween the independent variables and the target variables because the negative correlation coefficients are very close to zero.

Again, this is corbororated by the reality that the oldbalceOrig and the isFraud is unlikely correlated.

In [71]:
```
df.head()
```

Out[71]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	c
0	1	PAYMENT	9839.640000	C1231006815	170136.0	160296.36	M1979787155	
1	1	PAYMENT	1864.280000	C1666544295	21249.0	19384.72	M2044282225	
2	1	TRANSFER	181.000000	C1305486145	181.0	0.00	C553264065	
3	1	CASH_OUT	158667.712672	C840083671	181.0	0.00	C38997010	
4	1	PAYMENT	11668.140000	C2048537720	41554.0	29885.86	M1230701703	

In [72]:
```
model_df = df.drop(['step', 'nameOrig', 'nameDest'], axis = 1)

model_df.head()
```

Out[72]:

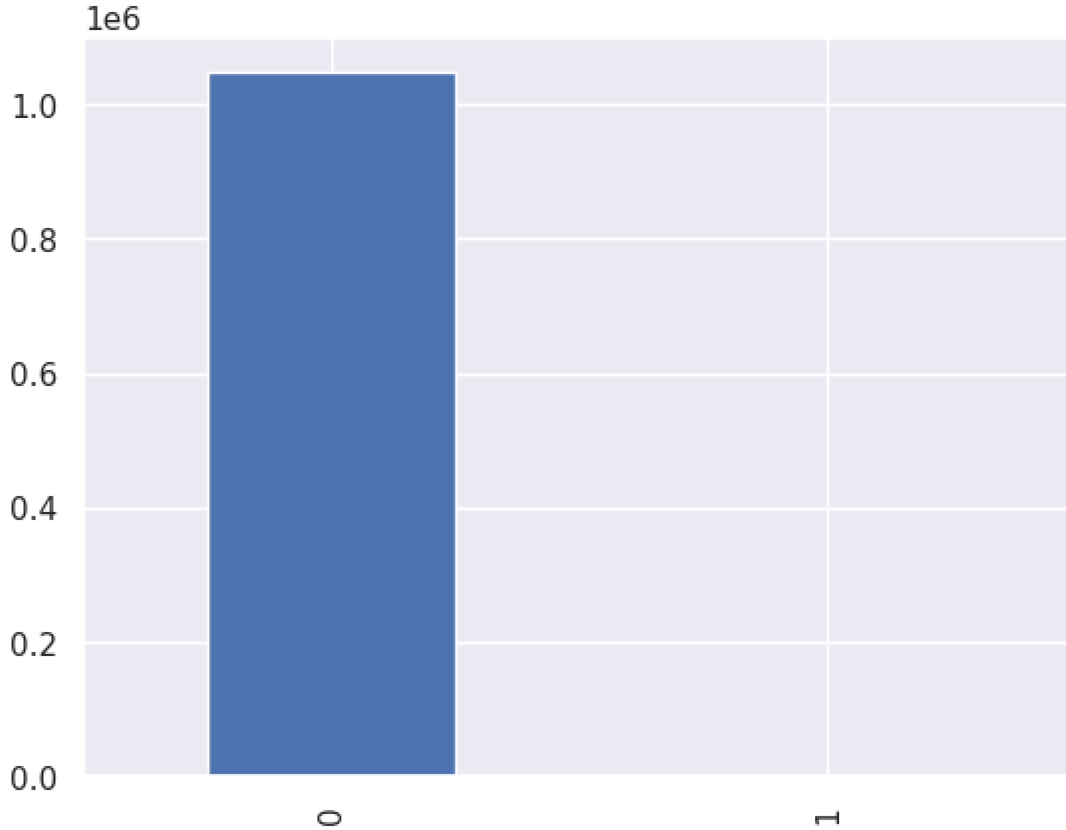
	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isl
0	PAYMENT	9839.640000	170136.0	160296.36	0.0	0.0	
1	PAYMENT	1864.280000	21249.0	19384.72	0.0	0.0	
2	TRANSFER	181.000000	181.0	0.00	0.0	0.0	
3	CASH_OUT	158667.712672	181.0	0.00	21182.0	0.0	
4	PAYMENT	11668.140000	41554.0	29885.86	0.0	0.0	

In [73]:
```
x = model_df['isFraud'].value_counts()[0]

y = model_df['isFraud'].value_counts()[1]

print(x)
print(y)
```

```
1047433
1142
```

In [74]:
```
model_df['isFraud'].value_counts().plot(kind='bar')
```

Out[74]: <AxesSubplot: >



There are several ways for encoding. 5 basic encoding methods:

One-Hot ncoding

Binary Encoding

Ordianal Encoding

Quantile Encoding

Counting Encoding

Replace column 'type' with a numerical column using onehot-encoder

In [75]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [76]:
```
model_df['type'].unique()
```

Out[76]:
```
array(['PAYMENT', 'TRANSFER', 'CASH_OUT', 'DEBIT', 'CASH_IN'],
      dtype=object)
```

In [77]:
```
# Perform one-hot encoding using pandas
one_hot_encoded = pd.get_dummies(model_df['type'])

# Concatenate the one-hot encoded columns with the original DataFrame
df_encoded = pd.concat([model_df, one_hot_encoded], axis=1)

df_encoded.head()
```

Out[77]:

	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isI
0	PAYMENT	9839.640000	170136.0	160296.36	0.0	0.0	
1	PAYMENT	1864.280000	21249.0	19384.72	0.0	0.0	
2	TRANSFER	181.000000	181.0	0.00	0.0	0.0	
3	CASH_OUT	158667.712672	181.0	0.00	21182.0	0.0	
4	PAYMENT	11668.140000	41554.0	29885.86	0.0	0.0	

In [78]:
```
df_final = df_encoded.drop('type', axis = 1)

df_final.head()
```

Out[78]:

	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	CASH
0	9839.640000	170136.0	160296.36	0.0	0.0	0	
1	1864.280000	21249.0	19384.72	0.0	0.0	0	
2	181.000000	181.0	0.00	0.0	0.0	1	
3	158667.712672	181.0	0.00	21182.0	0.0	1	
4	11668.140000	41554.0	29885.86	0.0	0.0	0	

Train-Test-Split

Why don't we use the entire dataset? We need training dataset and test dataset because we want estimate the performance of the model when there are extra data points that were not seen before. In other word, the trained model with the entire dataset might not be very reliable when there are new dat points added.

In [79]:
```
from sklearn.model_selection import train_test_split
```

In [80]:
```
y = df_final['isFraud']
y.head()
```

Out[80]:
```
0    0
1    0
2    1
3    1
4    0
Name: isFraud, dtype: int64
```

In [81]:
```
X = df_final.drop('isFraud', axis =1)
X.head()
```

Out[81]:

	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	CASH_IN	CA
0	9839.640000	170136.0	160296.36	0.0	0.0	0	
1	1864.280000	21249.0	19384.72	0.0	0.0	0	
2	181.000000	181.0	0.00	0.0	0.0	0	
3	158667.712672	181.0	0.00	21182.0	0.0	0	
4	11668.140000	41554.0	29885.86	0.0	0.0	0	

X and y are split into two subsets called X_train and X_test (y_train and y_test). The size of the test dataset sepends how large the dataset we are working on. In this problem we pick 33% for the test dataset.

In [82]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_st
```

In [83]:
```
X_train.head()
```

Out[83]:

	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	CASH_IN	C
664918	352265.95	223595.69	0.00	1830686.45	2182952.39	0	
741964	6839.81	0.00	0.00	0.00	0.00	0	
314050	189558.87	1148217.48	1337776.35	608722.11	419163.25	1	
631759	11593.26	0.00	0.00	0.00	0.00	0	
555223	109714.54	2254.00	0.00	85409.59	195124.14	0	

Model Training

Train the logistic regression

Evaluate/calculate model performance with classification metrics

Train the decision tree AND Evaluate/calculate model performance with classification metrics

Train the random forest AND Evaluate/calculate model performance with classification metrics

Train the XGBoost AND Evaluate/calculate model performance with classification metrics

Train a logistic regression model on the training set

In [84]:
```
from sklearn.linear_model import LogisticRegression
```

Instantiate the model

In [85]:
```
logreg = LogisticRegression(solver='liblinear', random_state=0, max_iter=250)
```

Fit the model

fit(X,y): Establish a relation between dependent variable and independent varialbes given a dataset.

logreg.fit(X,y): Establish a relation between dependent variable and independent varialbes to predict the log of the odds given a dataset.

fitting = learning = training = developing the model.

In [86]:
```
logreg.fit(X_train, y_train)

## predict(X_test): To predict the labels of X_test, given the independent testing
## logreg.precict (X_test): To predict the labels of X_test in logreg model, given
```

```
y_pred_test = logreg.predict(X_test)
y_pred_test
```

Out[86]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

Interpretation

X_train: The training dataset containing independent variables that are used to train the model.

y_train: The training dataset containing dependent variables that are used to train the model.

X_test: The testing dataset containing independent variables that are used to test the model.

y_test: The testing dataset containing dependent variables that are used to test the model.

The model predicts it is Not Fraud

In [89]:
```
# probability of getting output as 0 - Not Fraud

logreg.predict_proba(X_test)
```

Out[89]:
```
array([[1.00000000e+000, 3.59376188e-201],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 2.73798382e-028],
       ...,
       [1.00000000e+000, 1.15585677e-105],
       [1.00000000e+000, 9.37429067e-157],
       [1.00000000e+000, 1.28596795e-062]])
```

In [90]:
```
# Predict the logarithm of probability estimates

logreg.predict_log_proba(X_test)
```

```
/home/studio-lab-user/.conda/envs/default/lib/python3.9/site-packages/sklearn/line
ar_model/_logistic.py:1402: RuntimeWarning: divide by zero encountered in log
  return np.log(self.predict_proba(X))
```

Out[90]:
```
array([[   0.        , -461.54040416],
       [   0.        ,         -inf],
       [   0.        ,  -63.46516079],
       ...,
       [   0.        , -241.62659291],
       [   0.        , -359.26788869],
       [   0.        , -142.50876406]])
```

In [91]:
```
from sklearn.metrics import classification_report
```

In [126…

```
target_names = ['notFraud-0', 'isFraud-1']
# y_true = y_test
# y_pred = y_test_pred
print(classification_report(y_test, y_pred_test, target_names=target_names))
```

```
              precision    recall  f1-score   support

  notFraud-0       1.00      1.00      1.00    345655
   isFraud-1       0.61      0.81      0.70       375

    accuracy                           1.00    346030
   macro avg       0.81      0.90      0.85    346030
weighted avg       1.00      1.00      1.00    346030
```

In [125…

```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    345655
           1       0.61      0.81      0.70       375

    accuracy                           1.00    346030
   macro avg       0.81      0.90      0.85    346030
weighted avg       1.00      1.00      1.00    346030
```

Confusion Matrix

In [94]:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Negatives(TP) = ', cm[0,0])

print('\nTrue Positives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[345465    190]
 [    73    302]]

True Negatives(TP) =  345465

True Positives(TN) =  302

False Positives(FP) =  190

False Negatives(FN) =  73
```

In [95]:

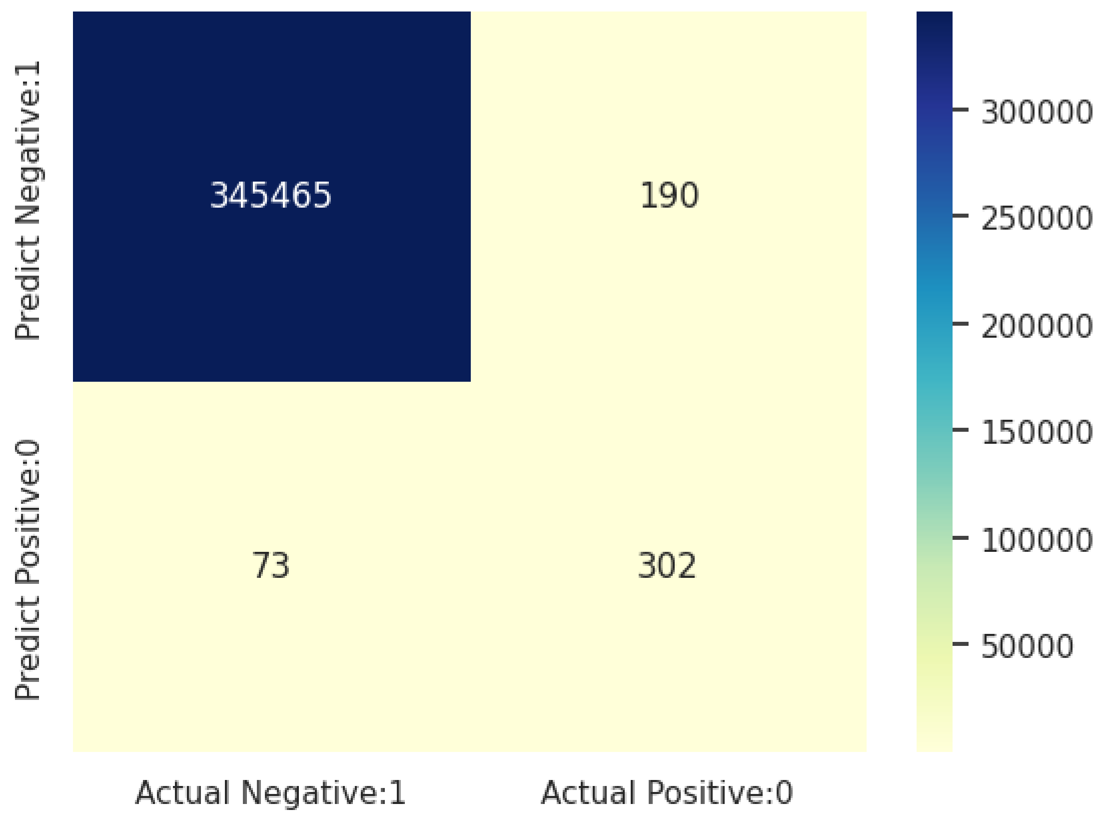
```python
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test).ravel()
print(tn, fp, fn, tp)
```

```
345465 190 73 302
```

In [96]:

```python
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Negative:1', 'Actual Positive:0'
                                 index=['Predict Negative:1', 'Predict Positive:0']

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[96]: <AxesSubplot: >



Classification accuracy

Accuracy = Correct prediction / Total prediction

In [163…
```
Accuracy = (cm[0,0] + cm[1,1])/(cm[0,0] + cm[1,1] + cm[0,1] + cm[1,0])

print(Accuracy)
```

```
0.9992399502933271
```

Precision = No of true PP that are true and it's true in fact / Total No of PP) = TP/(TP+FP)

Look at the Model (The bottom for the model)

In [164…
```
Precision = (cm[1,1])/(cm[1,1] + cm[0,1])

print(Precision)
```

```
0.6138211382113821
```

Recall = True Positive Rate = Sensitivity = TP/(TP+FN)

Look at the Dataset (The bottom for the dataset)

In [165…
```
Recall = (cm[1,1])/(cm[1,1] + cm[1,0])

print(Recall)
```

```
0.8053333333333333
```

F1_Score = 2(Precision.Recall)/(Precision + Recall)

In [166…
```
F1_Score = 2*(Precision*Recall)/(Precision+Recall)
print(F1_Score)
```

```
0.6966551326412918
```

Over-Sampling

The data set is not balanced. We try to make it balanced.

Undersampling method: Take a random sample of the original dataset in which the nmuber of majority calss is about the number of minority class.

Disadvantage: Reduce the data points. Then modelis biased.

In [101…
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in ./.conda/envs/default/lib/python3.9/sit
e-packages (0.0)
Requirement already satisfied: imbalanced-learn in ./.conda/envs/default/lib/pytho
n3.9/site-packages (from imblearn) (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in ./.conda/envs/default/lib/python3.
9/site-packages (from imbalanced-learn->imblearn) (1.24.1)
Requirement already satisfied: scipy>=1.3.2 in ./.conda/envs/default/lib/python3.
9/site-packages (from imbalanced-learn->imblearn) (1.10.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./.conda/envs/default/lib/p
ython3.9/site-packages (from imbalanced-learn->imblearn) (3.1.0)
Requirement already satisfied: joblib>=1.1.1 in ./.conda/envs/default/lib/python3.
9/site-packages (from imbalanced-learn->imblearn) (1.2.0)
Requirement already satisfied: scikit-learn>=1.0.2 in ./.conda/envs/default/lib/py
thon3.9/site-packages (from imbalanced-learn->imblearn) (1.2.1)
```

In [112…
```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=0)

X_resampled1, y_resampled1 = ros.fit_resample(X_train, y_train)
```

In [113…
```
print(X_resampled1.shape)

print(y_resampled1.shape)
```

```
(1403556, 10)
(1403556,)
```

In [114…
```
from collections import Counter
print(sorted(Counter(y_resampled1).items()))
```

```
[(0, 701778), (1, 701778)]
```

Under-Sampling

The data set is not balanced. We try to make it balanced.

Undersampling method: Take a random sample of the original dataset in which the nmuber of majority calss is about the number of minority class.

Disadvantage: Reduce the data points. Then modelis biased.

In [101…
```
from imblearn.under_sampling import ClusterCentroids
```

In [103…
```
# cc = ClusterCentroids(random_state=0)
# X_resampled2, y_resampled2 = cc.fit_resample(X_train, y_train)
```

```
# print(sorted(Counter(y_resampled2).items()))
```

SMOTE-Synthetic Minority Oversampling Technique

This method creates fake data points based on the original data points for which the new data points are colinear with existing data points.

Disadvantage: Because the new data points are created depending on the existing data points, it violates the independent assumption.

In [107…
```
from imblearn.over_sampling import SMOTE
from collections import Counter
```

In [108…
```
X_resampled3, y_resampled3 = SMOTE().fit_resample(X_train, y_train)
```

In [111…
```
print(sorted(Counter(y_resampled3).items()))
```

```
[(0, 701778), (1, 701778)]
```

Train the model with resampled dataset

In [120…
```
logreg.fit(X_resampled1, y_resampled1)

y_pred_test1 = logreg.predict(X_test)
y_pred_test1
```

Out[120]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [129…
```
target_names = ['notFraud', 'isFraud']

print(classification_report(y_test, y_pred_test1, target_names=target_names))
```

```
              precision    recall  f1-score   support

    notFraud       1.00      0.92      0.96    345655
     isFraud       0.01      0.96      0.03       375

    accuracy                           0.92    346030
   macro avg       0.51      0.94      0.49    346030
weighted avg       1.00      0.92      0.96    346030
```

In [128…
```
print(classification_report(y_test, y_pred_test1))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96    345655
           1       0.01      0.96      0.03       375

    accuracy                           0.92    346030
   macro avg       0.51      0.94      0.49    346030
weighted avg       1.00      0.92      0.96    346030
```

In [130…

```
logreg.fit(X_resampled3, y_resampled3)

y_pred_test3 = logreg.predict(X_test)
y_pred_test3
```

Out[130]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [131…

```
print(classification_report(y_test, y_pred_test3))
```

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.96    345655
           1       0.01      0.96      0.03       375

    accuracy                           0.93    346030
   macro avg       0.51      0.94      0.50    346030
weighted avg       1.00      0.93      0.96    346030
```

Confision Matrix with Resampled Dataset 1

In [132…

```
cm1 = confusion_matrix(y_test, y_pred_test1)
```

In [135…

```
print('Confusion Matrix\n\n', cm1)
```

```
Confusion Matrix

 [[318275  27380]
 [    15    360]]
```

In [137…

```
print('\n True Negative', cm1[0,0])

print('\n True Positive', cm1[1,1])

print('\n False Negative', cm1[1,0])

print('\n False Positive', cm1[0,1])
```

```
True Negative 318275

True Positive 360

False Negative 15

False Positive 27380
```

Calcualating Metrics 1

In [162…

```
Accuracy_OVER = (cm1[0,0]+ cm1[1,1])/( cm1[0,0]+ cm1[1,1]+ cm1[1,0]+ cm1[0,1])
print('\n Accuracy_OVER: ', Accuracy_OVER)

Precision_OVER = (cm1[1,1])/(cm1[1,1]+cm1[0,1])
print('\n Precision_OVER: ', Precision_OVER)

Recall_OVER = cm1[1,1]/(cm1[1,1]+cm1[1,0])
print('\n Recall_OVER: ', Recall_OVER)

F1_score_OVER = 2*(Precision_OVER*Recall_OVER)/(Precision_OVER+Recall_OVER)
print('\n F1_Score_OVER: ', F1_score_OVER)
```

```
Accuracy_OVER:  0.9208305638239459

Precision_OVER:  0.012977649603460706

Recall_OVER:  0.96

F1_Score_OVER:  0.02560910545971901
```

Confusion Matrix with Resampled Dataset 3

In [145…

```
cm3 = confusion_matrix(y_test, y_pred_test3)
```

In [156…

```
print('Confusion Matrix \n \n', cm3)

print('\n True Negative: ', cm3[0,0])

print('\n True Positive: ', cm3[1,1])

print('\n False Negative: ', cm3[1,0])

print('\n False Positive: ', cm3[0,1])
```

```
Confusion Matrix

 [[321048  24607]
 [    16    359]]

 True Negative:  321048

 True Positive:  359

 False Negative:  16

 False Positive:  24607
```

Calculating Metrics

In [160…

```
Accuracy_SMOTE = (cm3[0,0]+cm3[1,1])/(cm3[0,0]+cm3[1,1]+cm3[1,0]+cm3[0,1])
print('\n Accuracy_SMOTE: ', Accuracy_SMOTE)

Precision_SMOTE = (cm3[1,1])/(cm3[1,1]+cm3[0,1])
print('\n Precision_SMOTE: ', Precision_SMOTE)

Recall_SMOTE = (cm3[1,1])/(cm3[1,1]+cm3[1,0])
print('\n Recall_SMOTE: ', Recall_SMOTE)

F1_score_SMOTE = 2*(Precision_SMOTE*Recall_SMOTE)/(Precision_SMOTE+Recall_SMOTE)
print('\n F1_score_SMOTE: ', F1_score_SMOTE)
```

```
 Accuracy_SMOTE:  0.9288414299338208

 Precision_SMOTE:  0.014379556196427l4

 Recall_SMOTE:  0.9573333333333334

 F1_score_SMOTE:  0.02833353064204254
```

In [ ]: