Project Development
Phase-I
Code and Outputs

Date	6 November 2023
Team ID	PNT2023TMID592341
Project Name	Project – Online Payments Fraud Detection Using Machine Learning
Maximum Marks	4 Marks

Online Payments Frauds Detection:

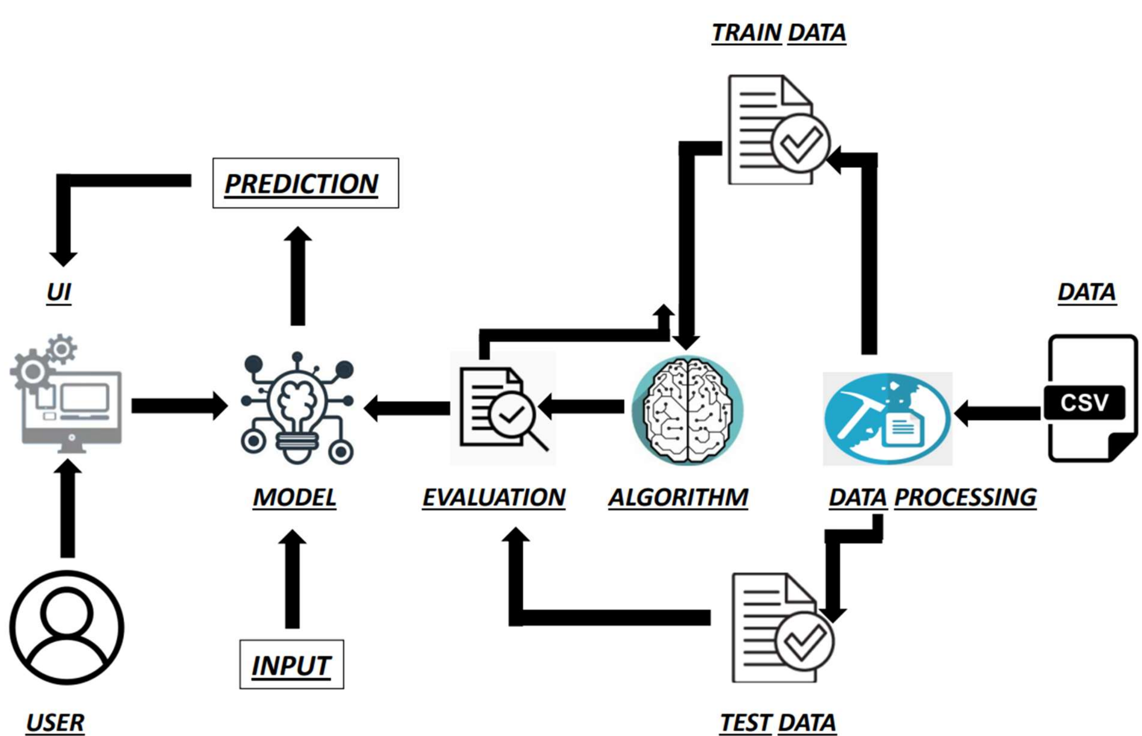
In response to the surge in online credit/debit card transactions, our project, "Online Payments Fraud Detection Using Machine Learning," pioneers a cutting-edge solution.

Leveraging classification algorithms like Decision Trees, Random Forest, SVM, Extra Tree, and XGBoost, we aim to detect and prevent fraud effectively.

The system ensures real-time monitoring, issues alerts for potential fraud, and features continuous learning for adaptation. With a user-friendly interface, administrators can efficiently review flagged transactions.

This initiative contributes to a secure online payment ecosystem through an adaptive, accurate, and efficient fraud detection system. Integration involves flask and IBM deployment.

Technical Architecture:



Data Collection, Preprocessing and Model building code:

fraud-detection

November 18, 2023

1 Data Collection

We are using a dataset from kaggle for this project

https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset

2 Visualising and analysing data

2.0.1 Importing libaries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import SVC
import xgboost as xgb
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
import pickle
```

2.0.2 Read dataset

[2]:
```
df = pd.read_csv("fraud detection dataset.csv")
```

[3]:
```
df
```

[3]:
```
        step      type     amount     nameOrig  oldbalanceOrg  \
0          1   PAYMENT    9839.64  C1231006815      170136.00
1          1   PAYMENT    1864.28  C1666544295       21249.00
2          1  TRANSFER     181.00  C1305486145         181.00
```

```
3            1  CASH_OUT      181.00   C840083671         181.00
4            1   PAYMENT    11668.14  C2048537720       41554.00
…         …        …         …           …             …
6362615    743  CASH_OUT   339682.13   C786484425      339682.13
6362616    743  TRANSFER  6311409.28  C1529008245     6311409.28
6362617    743  CASH_OUT  6311409.28  C1162922333     6311409.28
6362618    743  TRANSFER   850002.52  C1685995037      850002.52
6362619    743  CASH_OUT   850002.52  C1280323807      850002.52

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
…                     …          …             …             …       …
6362615            0.00   C776919290            0.00       339682.13        1
6362616            0.00  C1881841831            0.00            0.00        1
6362617            0.00  C1365125890        68488.84      6379898.11        1
6362618            0.00  C2080388513            0.00            0.00        1
6362619            0.00   C873221189      6510099.11      7360101.63        1

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
4                     0
…                     …
6362615               0
6362616               0
6362617               0
6362618               0
6362619               0

[6362620 rows x 11 columns]
```

[4]:
```
df.drop(["isFlaggedFraud"], axis=1, inplace=True)
```

Here the "isFlagged" is removed as it has only 1 value i.e, 0 and is unecessary

[5]:
```
df.head()
```

[5]:
```
   step      type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0     1   PAYMENT   9839.64  C1231006815       170136.0       160296.36
1     1   PAYMENT   1864.28  C1666544295        21249.0        19384.72
2     1  TRANSFER    181.00  C1305486145          181.0            0.00
```

```
3     1  CASH_OUT    181.00   C840083671          181.0            0.00
4     1   PAYMENT  11668.14  C2048537720        41554.0        29885.86

      nameDest  oldbalanceDest  newbalanceDest  isFraud
0  M1979787155             0.0             0.0        0
1  M2044282225             0.0             0.0        0
2   C553264065             0.0             0.0        1
3    C38997010         21182.0             0.0        1
4  M1230701703             0.0             0.0        0
```

[6]:
```
df.tail()
```

[6]:
```
         step      type      amount     nameOrig  oldbalanceOrg  \
6362615   743  CASH_OUT   339682.13   C786484425      339682.13
6362616   743  TRANSFER  6311409.28  C1529008245     6311409.28
6362617   743  CASH_OUT  6311409.28  C1162922333     6311409.28
6362618   743  TRANSFER   850002.52  C1685995037      850002.52
6362619   743  CASH_OUT   850002.52  C1280323807      850002.52

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud
6362615             0.0   C776919290            0.00       339682.13        1
6362616             0.0  C1881841831            0.00            0.00        1
6362617             0.0  C1365125890        68488.84      6379898.11        1
6362618             0.0  C2080388513            0.00            0.00        1
6362619             0.0   C873221189      6510099.11      7360101.63        1
```

[7]:
```
df.columns
```

[7]:
```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud'],
      dtype='object')
```

We will modify the "matplotlib" plots to "ggplot" style as used in R programming. Additionally, we will also suppress any warning messages generated by matplotlib or any other libraries.

[8]:
```
plt.style.use("ggplot")
warnings.filterwarnings("ignore")
```

[9]:
```
df["isFraud"].value_counts()
```

[9]:
```
isFraud
0    6354407
1       8213
Name: count, dtype: int64
```

0 means legal transaction and 1 means fraudulent transaction

It is clearly seen that the number of legal transactions is way more than the number of fraudulent transactions which makes it an imbalanced dataset

Hence we need to balance it

[10]:
```
legit = df[df["isFraud"]==0]
```

[11]:
```
fraud = df[df["isFraud"]==1]
```

We shall make the number of legal transactions equal to the number of fraudulent transactions to make it even

[12]:
```
legit = legit.sample(n=8213)
```

[13]:
```
legit.shape, fraud.shape
```

[13]:
```
((8213, 10), (8213, 10))
```

Now they are equal

[14]:
```
new_df = pd.concat([legit, fraud], axis=0)
```

[15]:
```
new_df.head()
```

[15]:
```
         step      type     amount     nameOrig  oldbalanceOrg  \
2472014   204   PAYMENT    2711.51  C1894501645      155609.95
1329496   137   PAYMENT   50062.45  C1730518885           0.00
3569677   260  TRANSFER  142407.65  C2110794520           0.00
1698217   159   PAYMENT   11379.43   C195346257       25424.00
5190368   369   CASH_IN  220293.91   C481241988      293672.58

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud
2472014       152898.44  M1335909317            0.00            0.00        0
1329496            0.00   M341053193            0.00            0.00        0
3569677            0.00  C2102807958       830469.20      1181738.48        0
1698217        14044.57   M371911161            0.00            0.00        0
5190368       513966.49  C1138529772      3017311.84      2797017.93        0
```

[16]:
```
new_df.tail()
```

[16]:
```
         step      type      amount     nameOrig  oldbalanceOrg  \
6362615   743  CASH_OUT   339682.13   C786484425      339682.13
6362616   743  TRANSFER  6311409.28  C1529008245     6311409.28
6362617   743  CASH_OUT  6311409.28  C1162922333     6311409.28
6362618   743  TRANSFER   850002.52  C1685995037      850002.52
6362619   743  CASH_OUT   850002.52  C1280323807      850002.52

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud
6362615             0.0   C776919290            0.00       339682.13        1
6362616             0.0  C1881841831            0.00            0.00        1
```

```
6362617             0.0  C1365125890       68488.84      6379898.11        1
6362618             0.0  C2080388513           0.00            0.00        1
6362619             0.0   C873221189     6510099.11      7360101.63        1
```

[17]:
```
new_df["isFraud"].value_counts()
```

[17]:
```
isFraud
0    8213
1    8213
Name: count, dtype: int64
```

[18]:
```
new_df.to_csv('balanced_dataset.csv', index=False, encoding='utf-8')
```

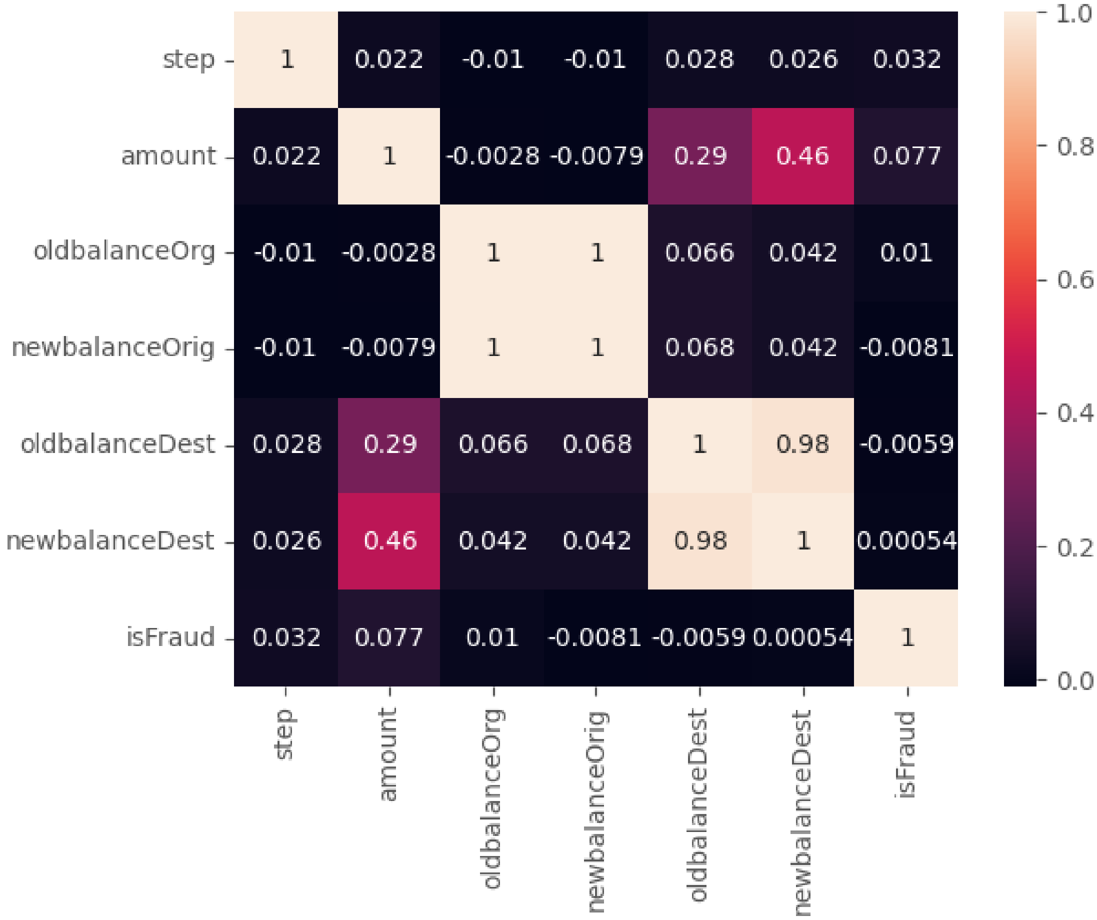
[19]:
```
corr = df.corr(numeric_only=True)
```

[20]:
```
corr
```

[20]:
```
                    step    amount  oldbalanceOrg  newbalanceOrig  \
step            1.000000  0.022373      -0.010058       -0.010299
amount          0.022373  1.000000      -0.002762       -0.007861
oldbalanceOrg  -0.010058 -0.002762       1.000000        0.998803
newbalanceOrig -0.010299 -0.007861       0.998803        1.000000
oldbalanceDest  0.027665  0.294137       0.066243        0.067812
newbalanceDest  0.025888  0.459304       0.042029        0.041837
isFraud         0.031578  0.076688       0.010154       -0.008148

                oldbalanceDest  newbalanceDest   isFraud
step                  0.027665        0.025888  0.031578
amount                0.294137        0.459304  0.076688
oldbalanceOrg         0.066243        0.042029  0.010154
newbalanceOrig        0.067812        0.041837 -0.008148
oldbalanceDest        1.000000        0.976569 -0.005885
newbalanceDest        0.976569        1.000000  0.000535
isFraud              -0.005885        0.000535  1.000000
```

[21]:
```
sns.heatmap(corr, annot=True)
```

[21]:
```
<Axes: >
```

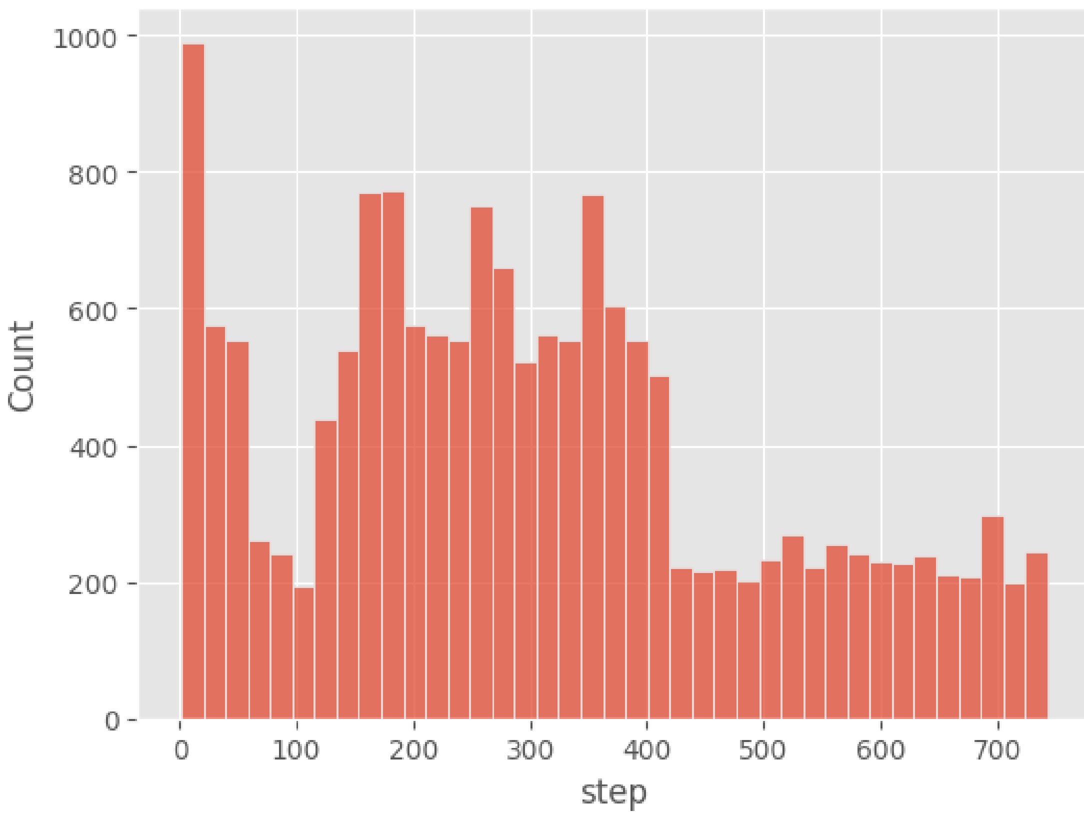


2.0.3 Univariate analysis

The process of understanding data with a single feature is called univariate analysis

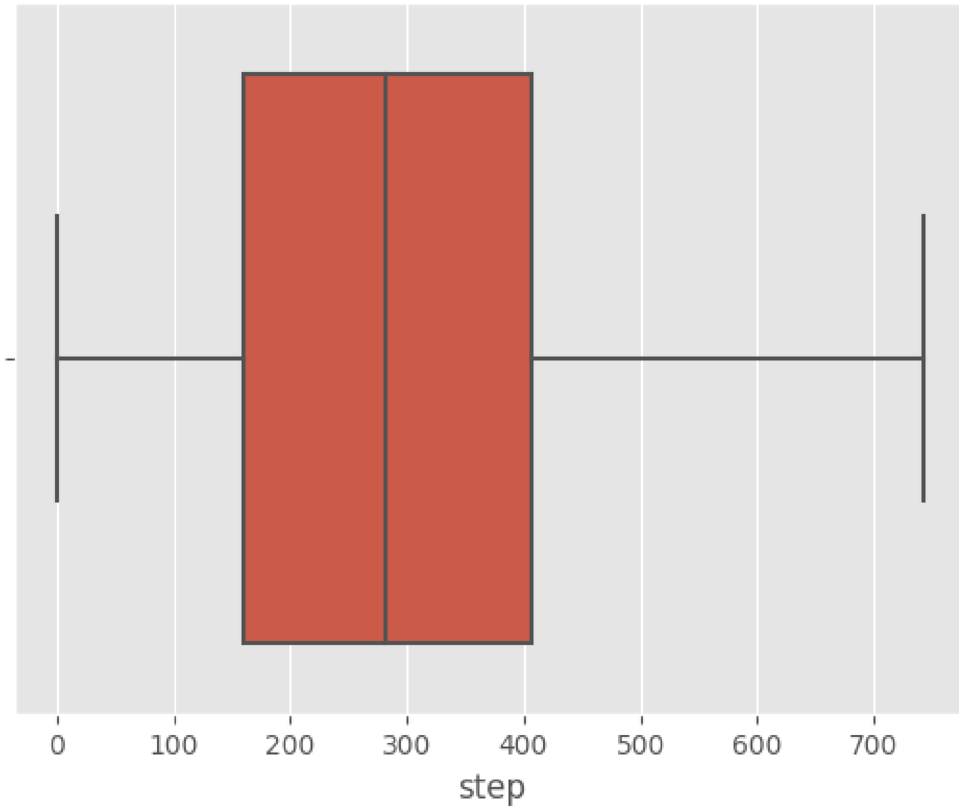
[22]:
```
sns.histplot(data=new_df, x="step")
```

[22]: <Axes: xlabel='step', ylabel='Count'>



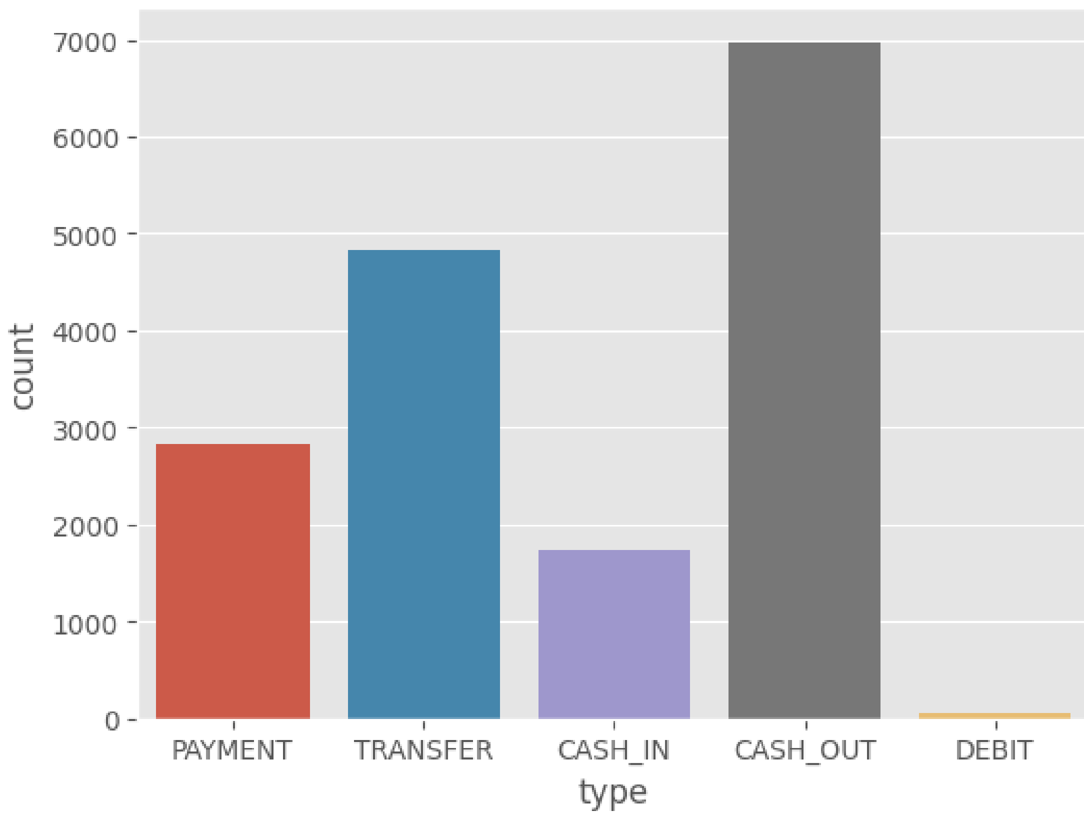
[23]:
```
sns.boxplot(data=new_df, x="step")
```

[23]: <Axes: xlabel='step'>



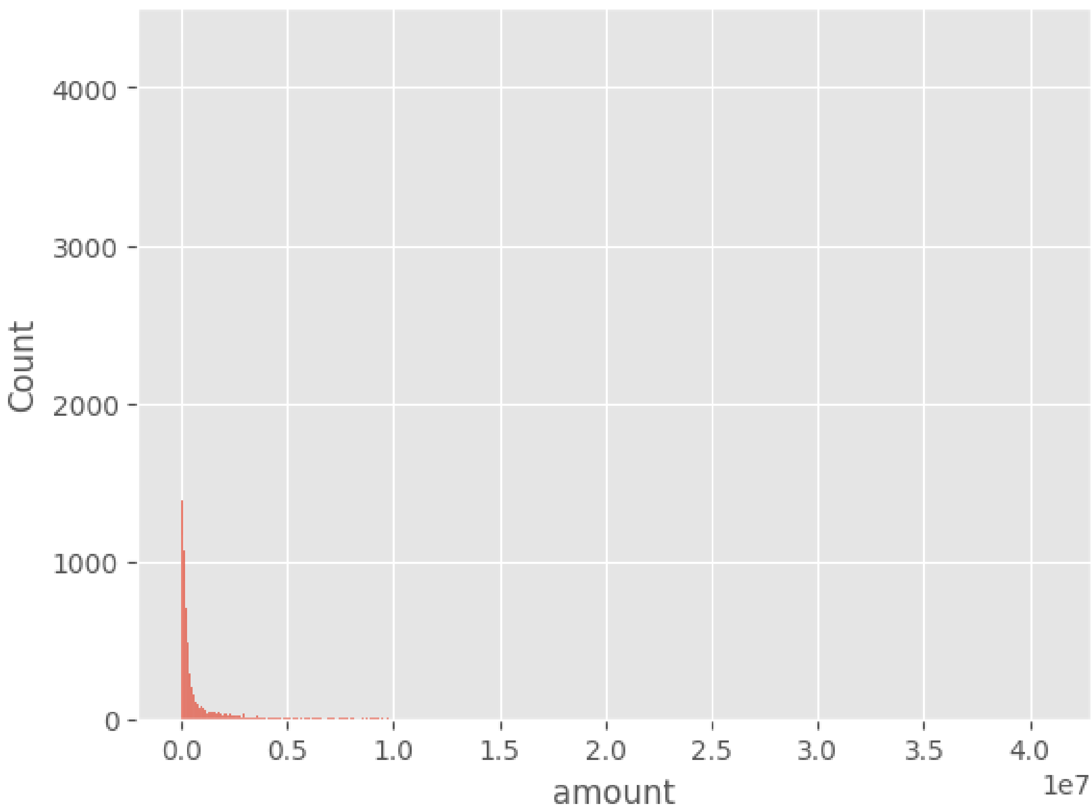
[24]:
```
sns.countplot(data=new_df, x="type")
```

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



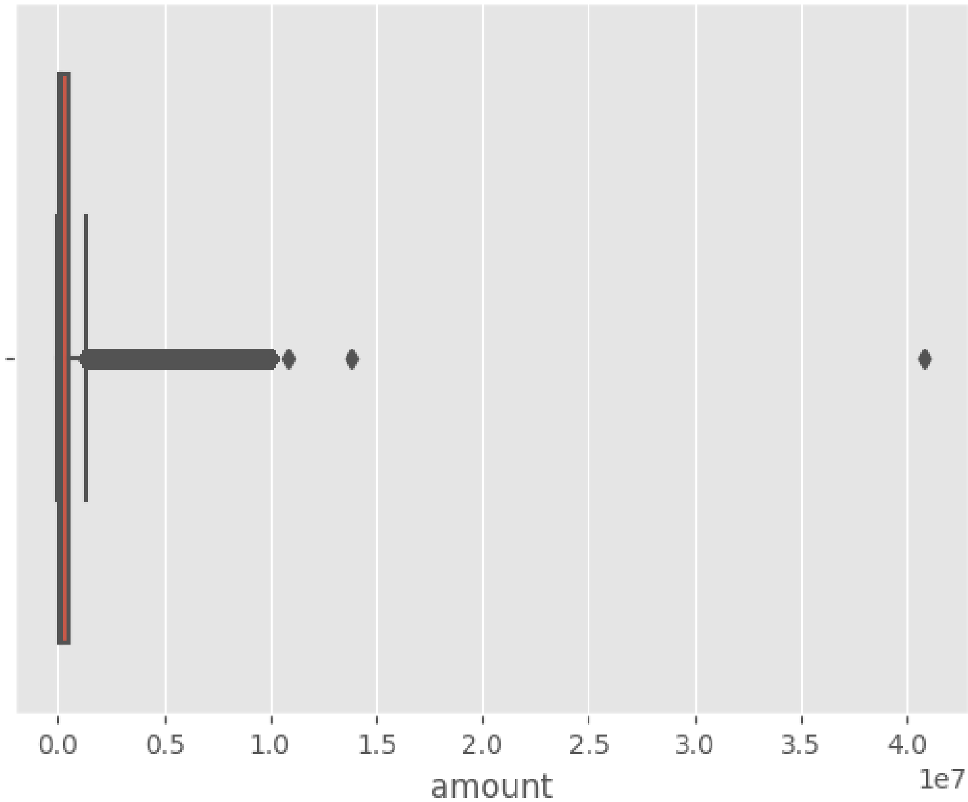
[25]:
```
sns.histplot(data=new_df, x="amount")
```

[25]: <Axes: xlabel='amount', ylabel='Count'>



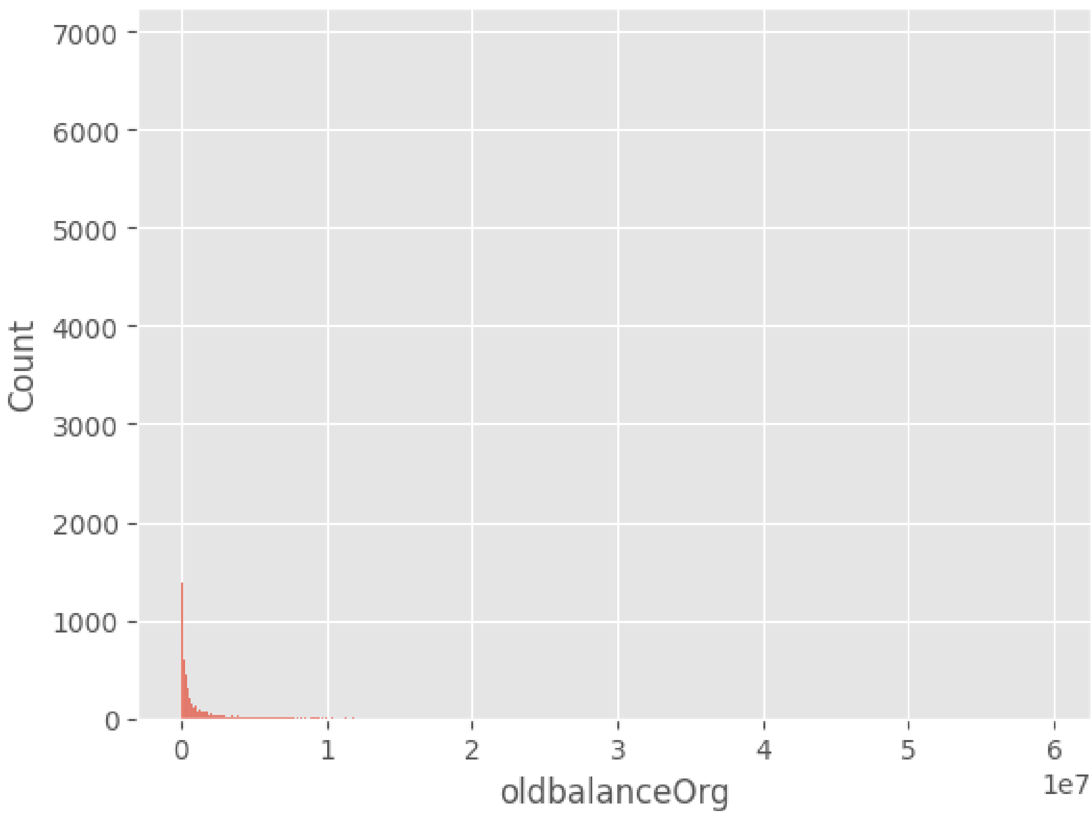
[26]:
```
sns.boxplot(data=new_df, x="amount")
```

[26]: <Axes: xlabel='amount'>



[27]:
```
sns.histplot(data=new_df, x="oldbalanceOrg")
```

[27]: <Axes: xlabel='oldbalanceOrg', ylabel='Count'>

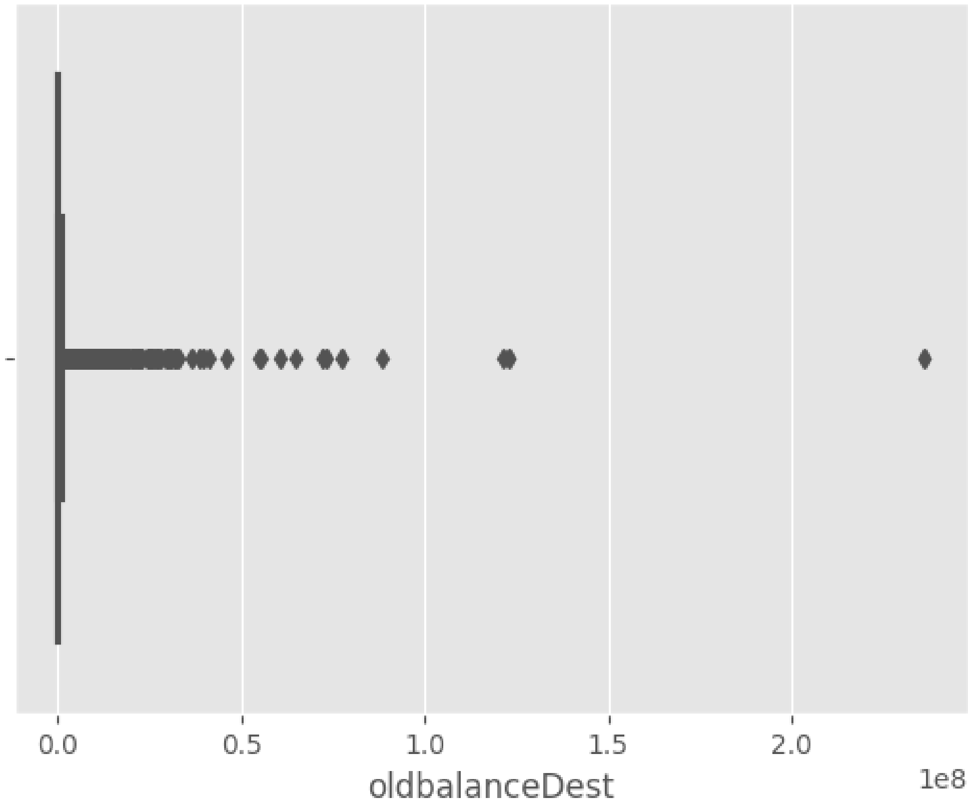


[28]:
```
df["nameDest"].value_counts()
```

[28]:
```
nameDest
C1286084959    113
C985934102     109
C665576141     105
C2083562754    102
C248609774     101
                ...
M1470027725      1
M1330329251      1
M1784358659      1
M2081431099      1
C2080388513      1
Name: count, Length: 2722362, dtype: int64
```

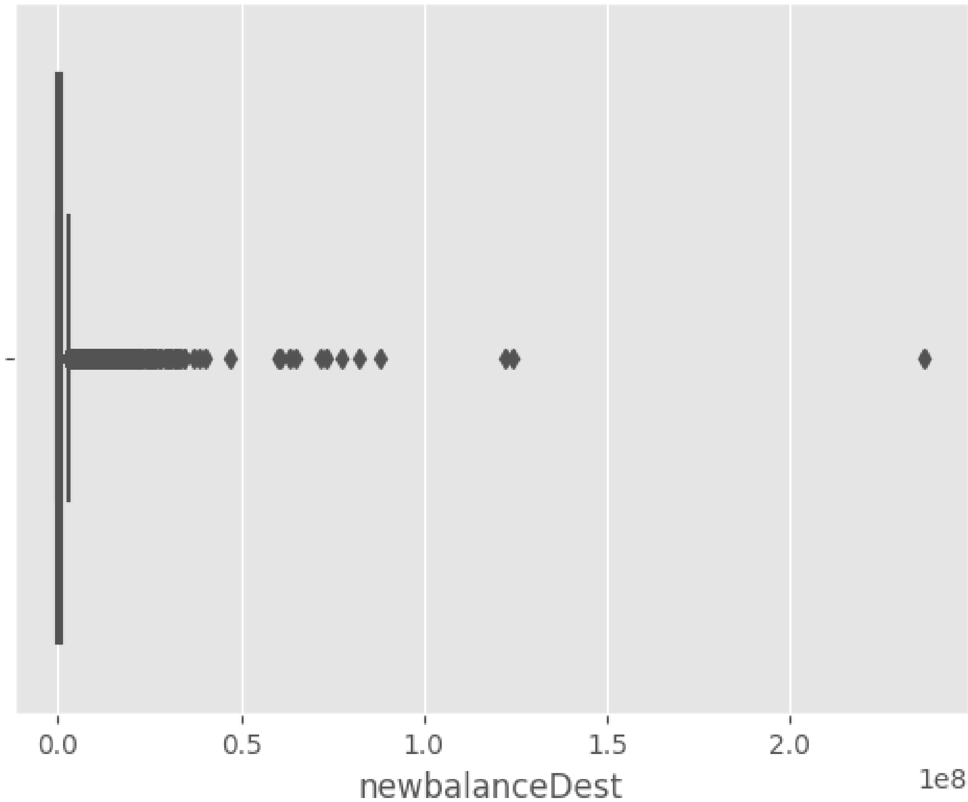
[29]:
```
sns.boxplot(data=new_df, x="oldbalanceDest")
```

[29]:
```
<Axes: xlabel='oldbalanceDest'>
```



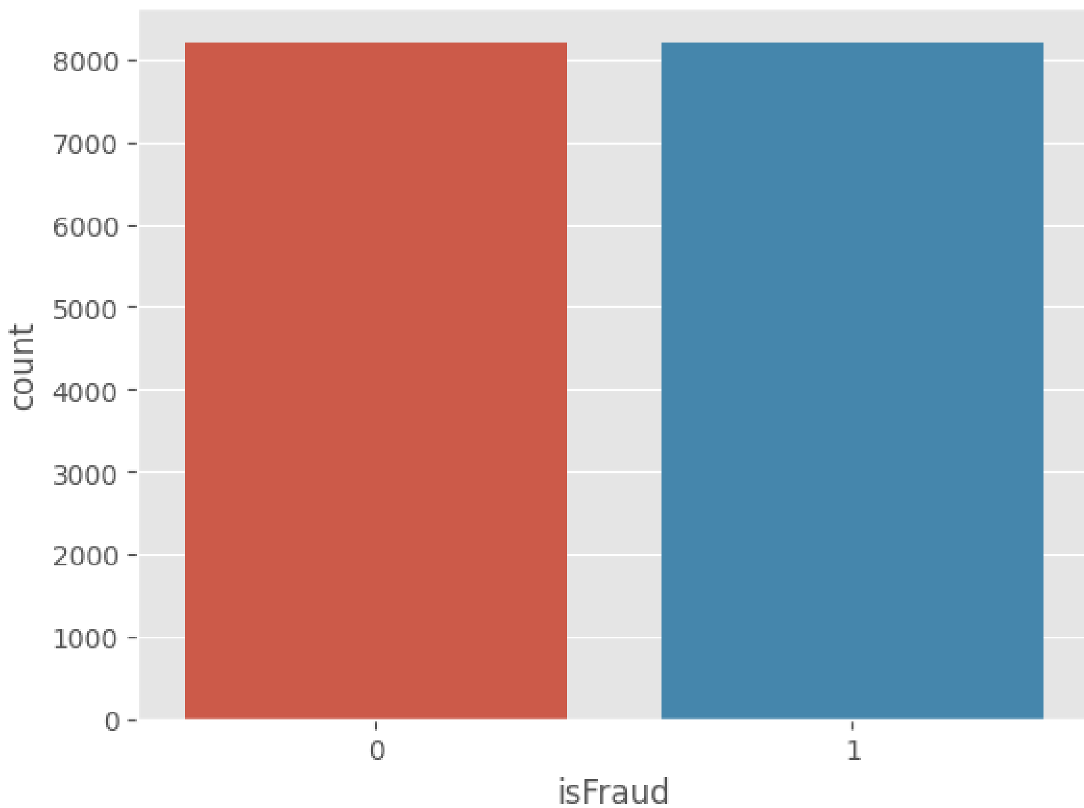
[30]:
```
sns.boxplot(data=new_df, x="newbalanceDest")
```

[30]: `<Axes: xlabel='newbalanceDest'>`



[31]:
```
sns.countplot(data=new_df, x="isFraud")
```

[31]: <Axes: xlabel='isFraud', ylabel='count'>

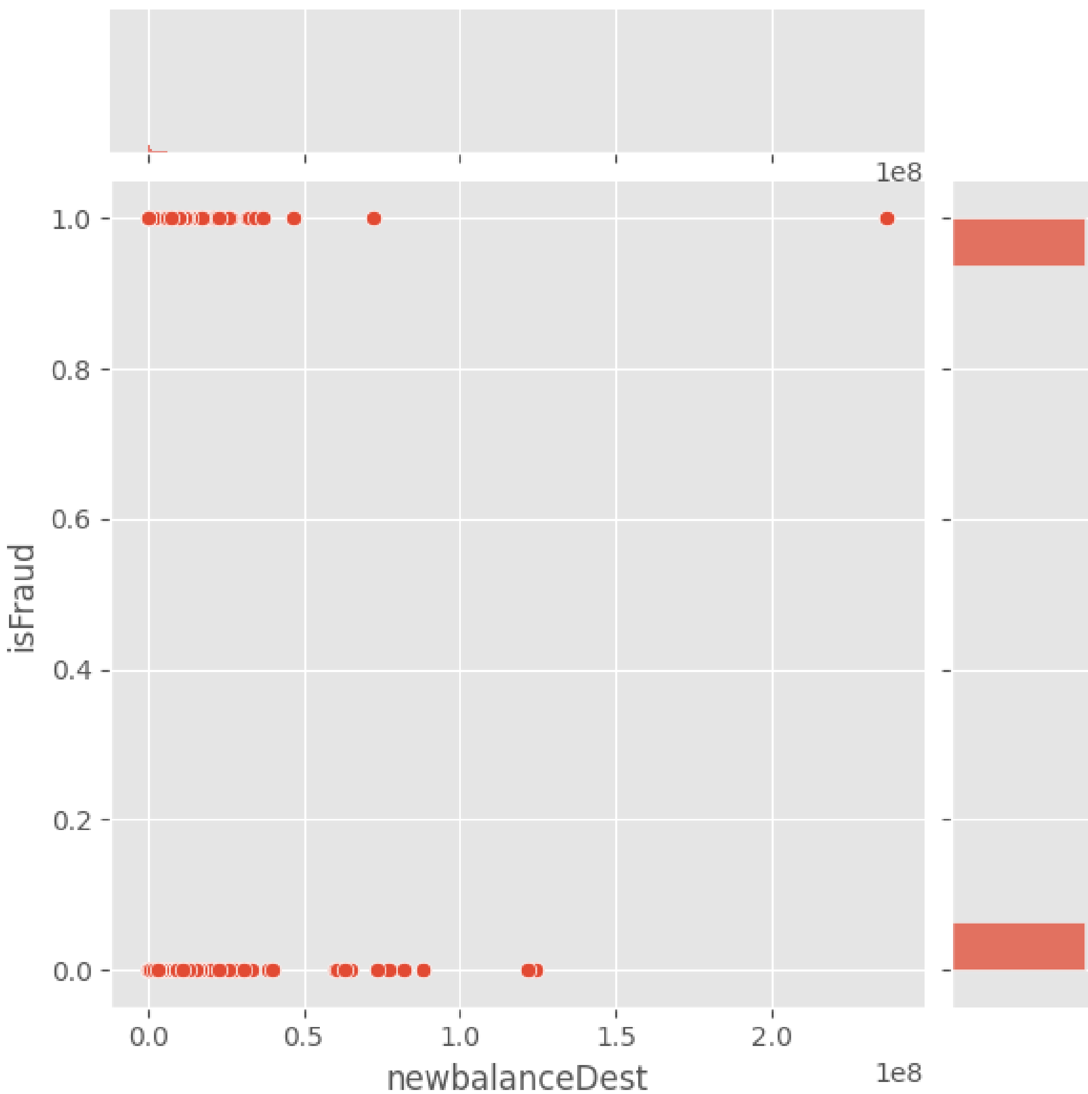


2.0.4 Bivariate analysis

The process of finding the relation between two features is called bivariate analysis

[32]:
```
sns.jointplot(data=new_df, x="newbalanceDest", y="isFraud")
```

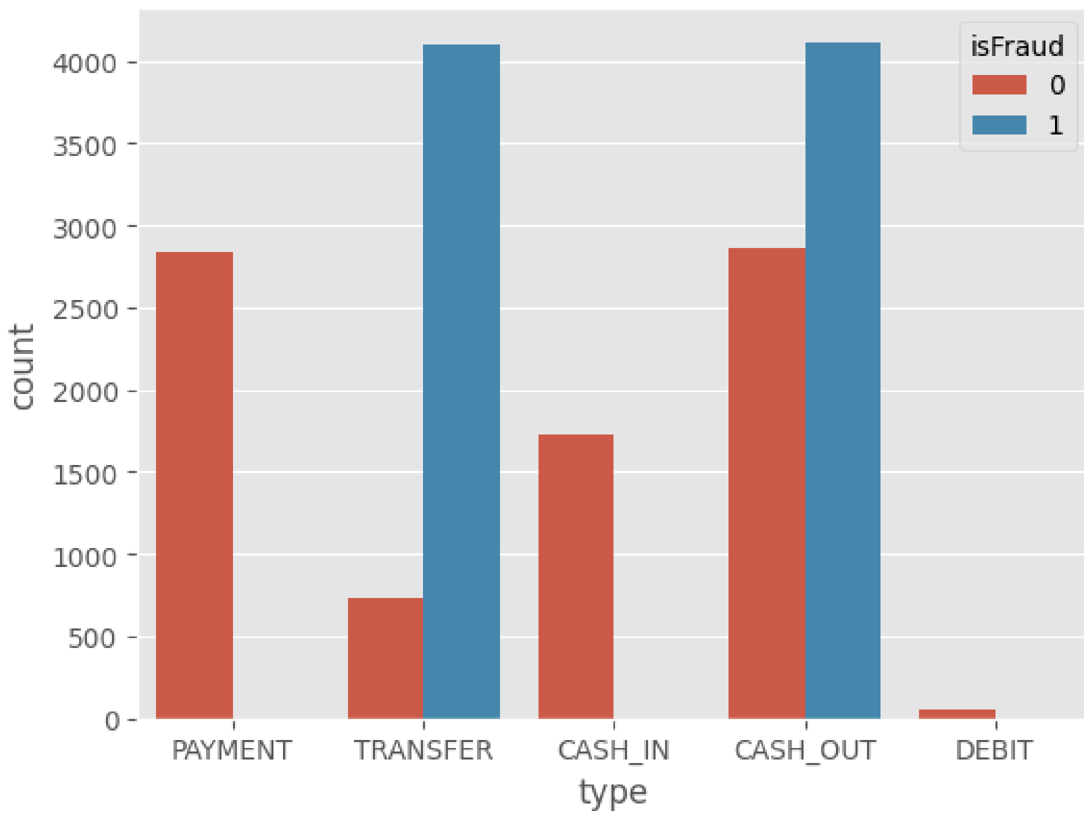
[32]: <seaborn.axisgrid.JointGrid at 0x2086ebe3c70>



0 means legal transaction and 1 means fraudulent transaction

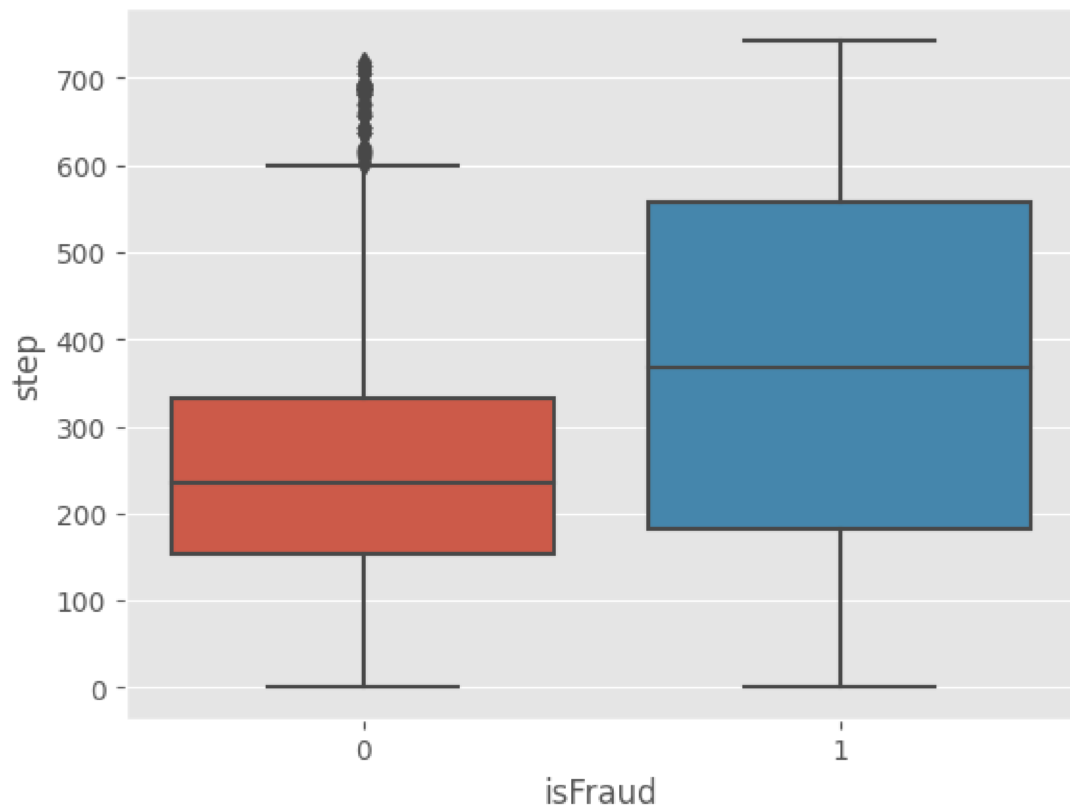
[33]:
```
sns.countplot(data=new_df, x="type", hue="isFraud")
```

[33]: <Axes: xlabel='type', ylabel='count'>



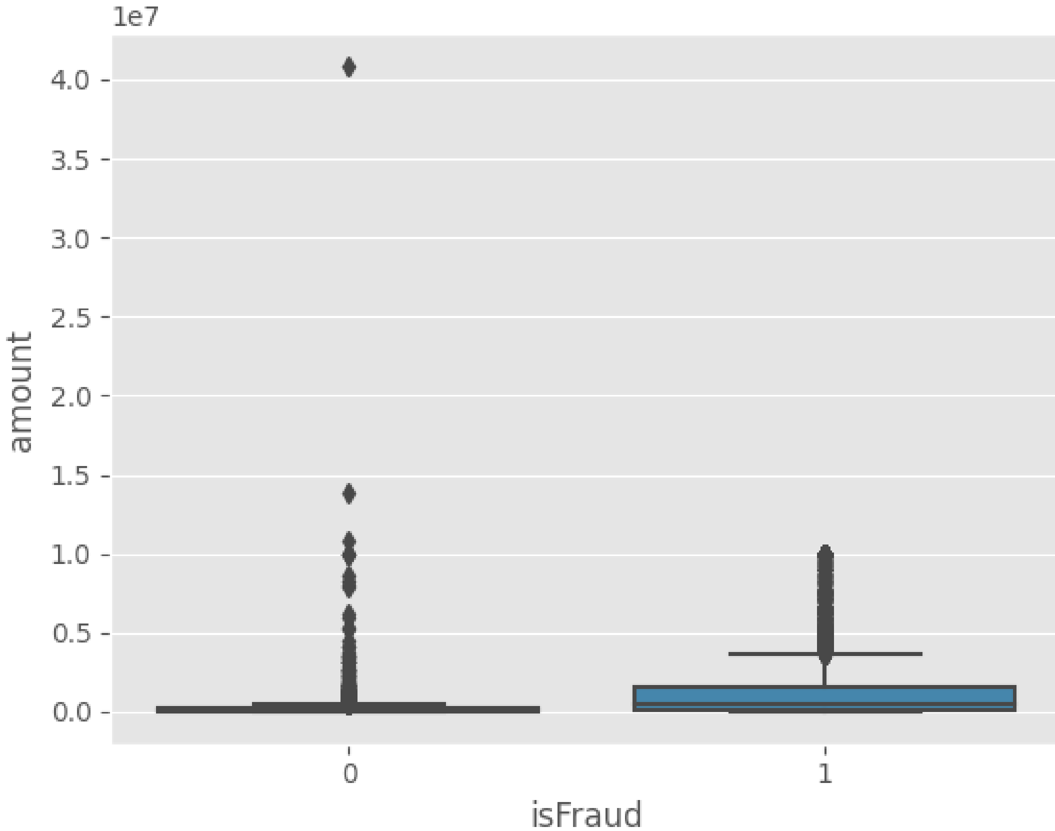
[34]:
```
sns.boxplot(data=new_df, x="isFraud", y="step")
```

[34]: <Axes: xlabel='isFraud', ylabel='step'>



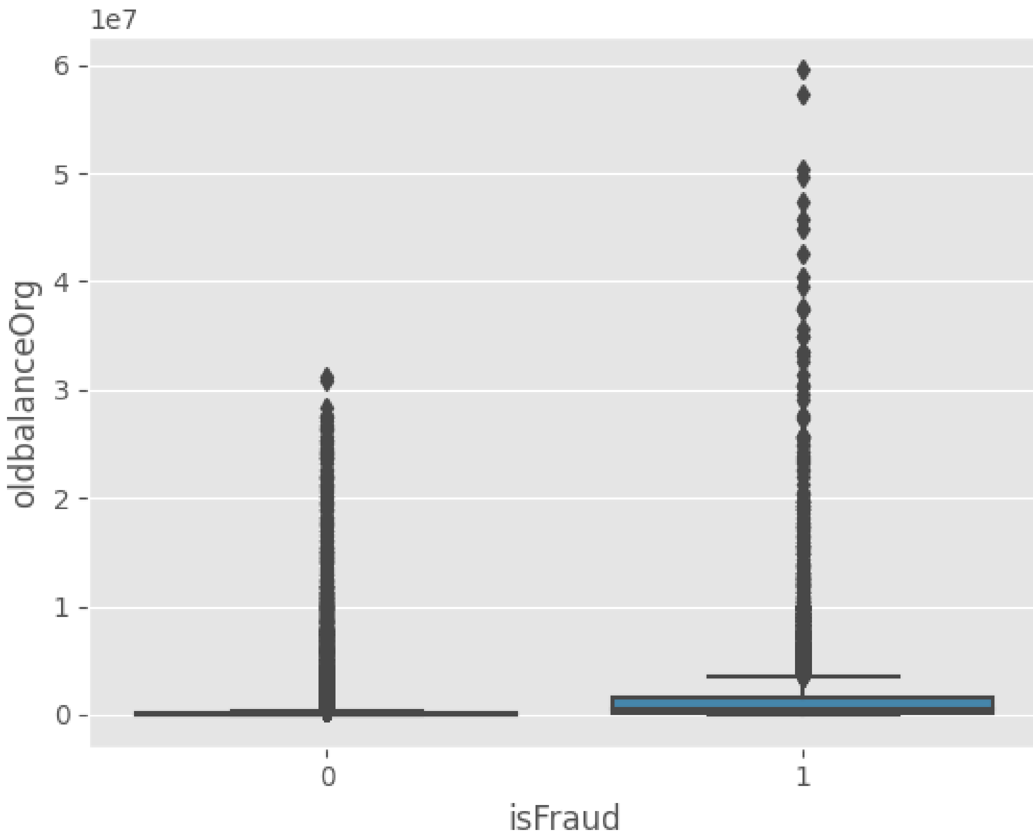
[35]:
```
sns.boxplot(data=new_df, x="isFraud", y="amount")
```

[35]: <Axes: xlabel='isFraud', ylabel='amount'>



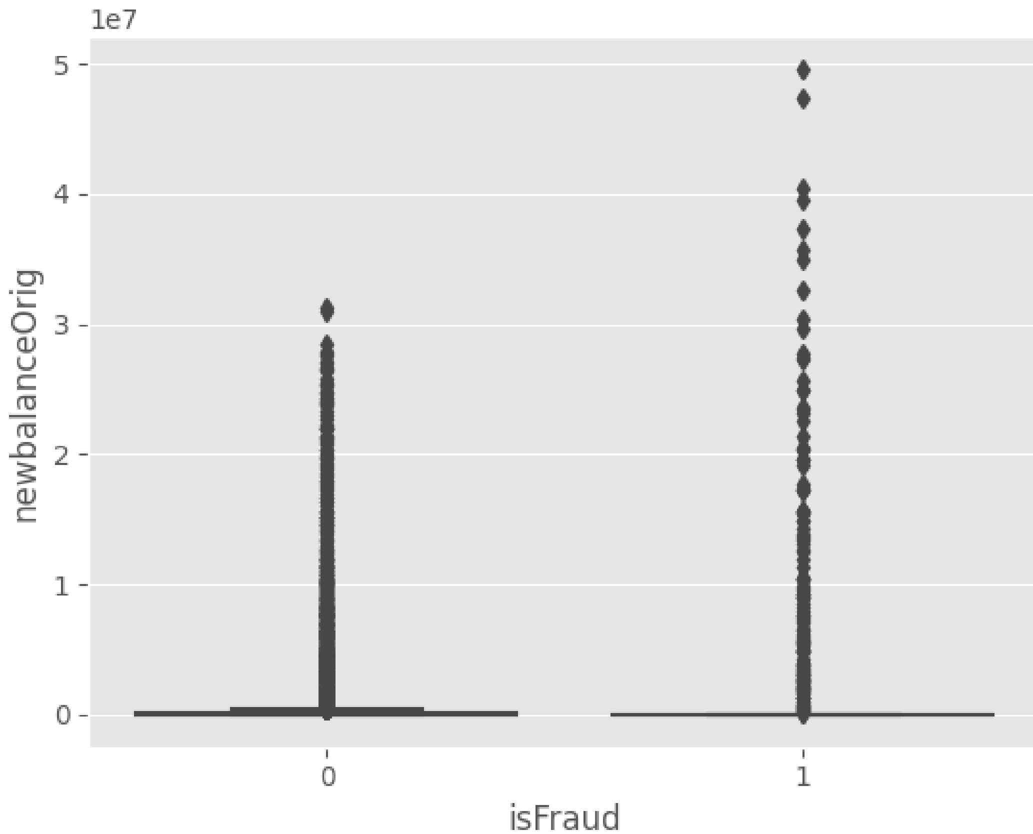
[36]:
```
sns.boxplot(data=new_df, x="isFraud", y="oldbalanceOrg")
```

[36]: <Axes: xlabel='isFraud', ylabel='oldbalanceOrg'>



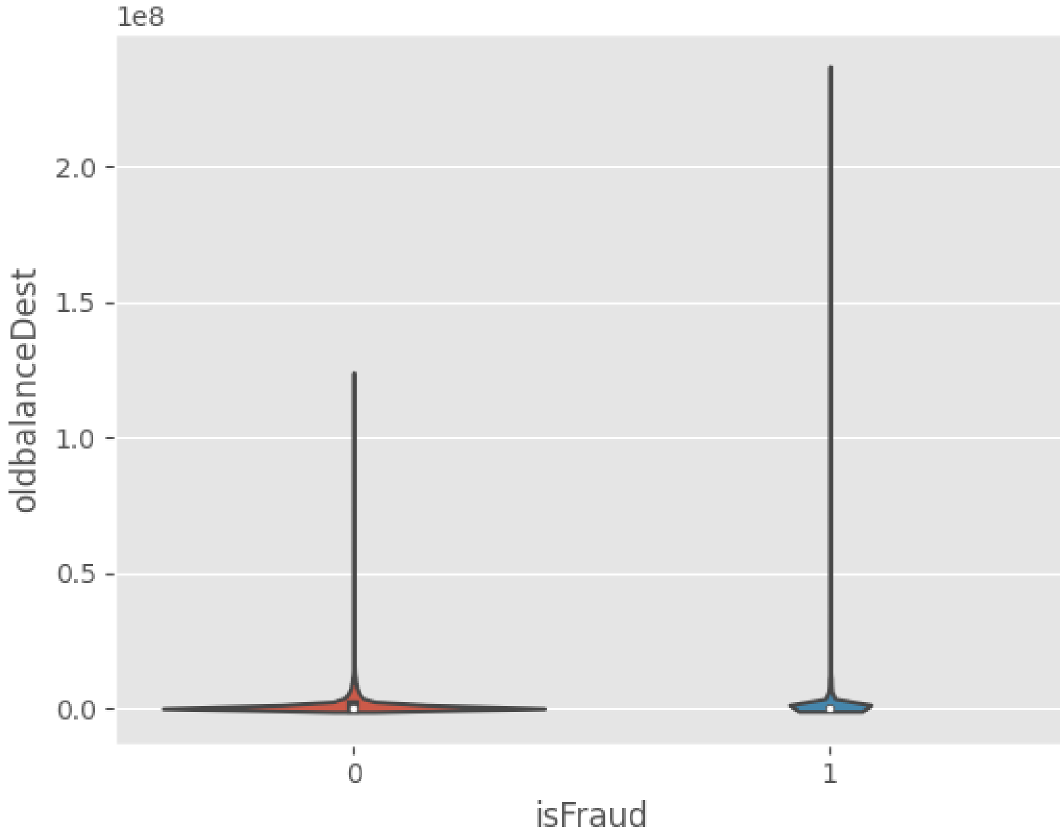
[37]:
```
sns.boxplot(data=new_df, x="isFraud", y="newbalanceOrig")
```

[37]: <Axes: xlabel='isFraud', ylabel='newbalanceOrig'>



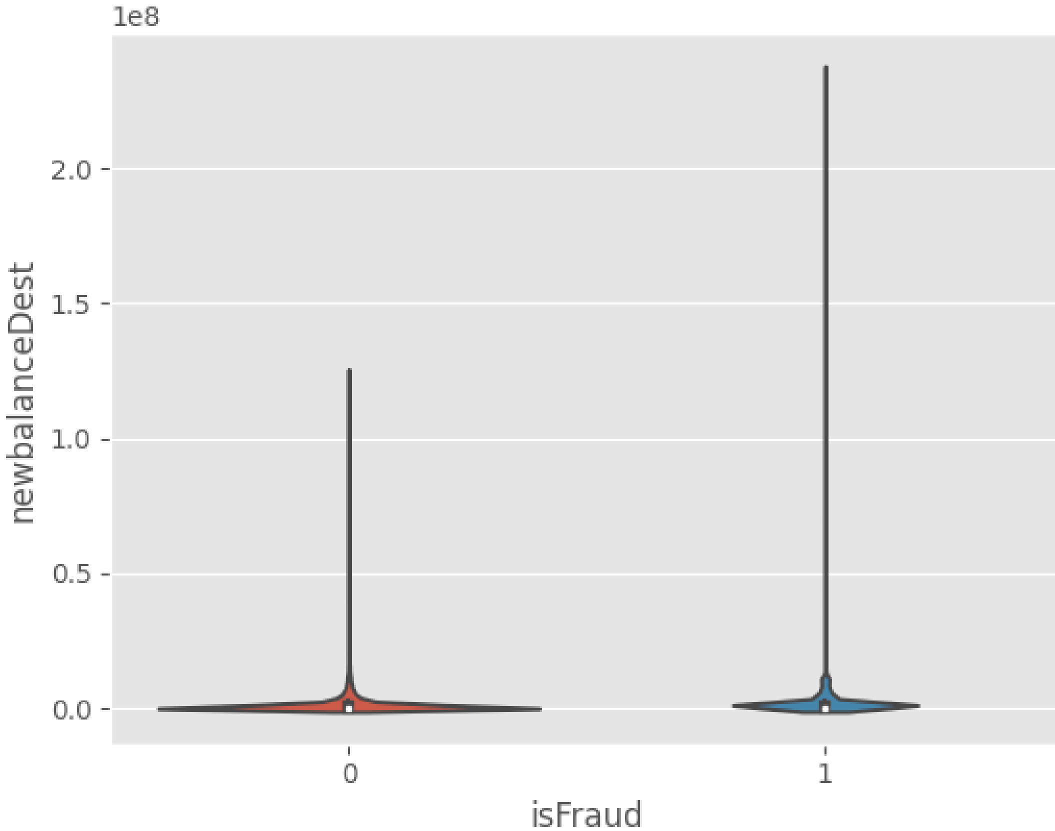
[38]:
```
sns.violinplot(data=new_df, x="isFraud", y="oldbalanceDest")
```

[38]: <Axes: xlabel='isFraud', ylabel='oldbalanceDest'>



[39]:
```
sns.violinplot(data=new_df, x="isFraud", y="newbalanceDest")
```

[39]: <Axes: xlabel='isFraud', ylabel='newbalanceDest'>



2.0.5 Descriptive analysis

The process of studying the basic features of data with the statistical process is called descriptive analysis

[40]:
```
new_df.describe(include="all")
```

[40]:

	step	type	amount	nameOrig	oldbalanceOrg	\
count	16426.000000	16426	1.642600e+04	16426	1.642600e+04	
unique	NaN	5	NaN	16426	NaN	
top	NaN	CASH_OUT	NaN	C1894501645	NaN	
freq	NaN	6977	NaN	1	NaN	
mean	303.528004	NaN	8.236633e+05	NaN	1.215890e+06	
std	194.265068	NaN	1.874245e+06	NaN	3.216881e+06	
min	1.000000	NaN	0.000000e+00	NaN	0.000000e+00	
25%	160.000000	NaN	3.600852e+04	NaN	1.037175e+04	
50%	282.000000	NaN	1.724345e+05	NaN	1.181601e+05	
75%	407.000000	NaN	5.425444e+05	NaN	7.800662e+05	
max	743.000000	NaN	4.083851e+07	NaN	5.958504e+07	

```
        newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  \
count     1.642600e+04        16426    1.642600e+04    1.642600e+04
unique             NaN        16243             NaN             NaN
top                NaN  C2069255486             NaN             NaN
freq               NaN            3             NaN             NaN
mean      4.981792e+05          NaN    8.541002e+05    1.283866e+06
std       2.449074e+06          NaN    3.585214e+06    3.963415e+06
min       0.000000e+00          NaN    0.000000e+00    0.000000e+00
25%       0.000000e+00          NaN    0.000000e+00    0.000000e+00
50%       0.000000e+00          NaN    0.000000e+00    1.145759e+05
75%       0.000000e+00          NaN    5.062906e+05    1.086951e+06
max       4.958504e+07          NaN    2.362305e+08    2.367265e+08

             isFraud
count   16426.000000
unique           NaN
top              NaN
freq             NaN
mean        0.500000
std         0.500015
min         0.000000
25%         0.000000
50%         0.500000
75%         1.000000
max         1.000000
```

3 Data preprocessing

[41]:
```python
new_df.shape
```

[41]:
```
(16426, 10)
```

[42]:
```python
new_df.drop(["nameOrig", "nameDest"], axis=1, inplace=True)
df.columns
```

[42]:
```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud'],
      dtype='object')
```

[43]:
```python
new_df.head()
```

[43]:
```
         step      type     amount  oldbalanceOrg  newbalanceOrig  \
2472014   204   PAYMENT    2711.51      155609.95       152898.44
1329496   137   PAYMENT   50062.45           0.00            0.00
3569677   260  TRANSFER  142407.65           0.00            0.00
1698217   159   PAYMENT   11379.43       25424.00        14044.57
```

```
5190368   369   CASH_IN  220293.91      293672.58       513966.49

         oldbalanceDest  newbalanceDest  isFraud
2472014            0.00            0.00        0
1329496            0.00            0.00        0
3569677       830469.20      1181738.48        0
1698217            0.00            0.00        0
5190368      3017311.84      2797017.93        0
```

[44]:
```
new_df.tail()
```

[44]:
```
         step      type      amount  oldbalanceOrg  newbalanceOrig  \
6362615   743  CASH_OUT   339682.13      339682.13             0.0
6362616   743  TRANSFER  6311409.28     6311409.28             0.0
6362617   743  CASH_OUT  6311409.28     6311409.28             0.0
6362618   743  TRANSFER   850002.52      850002.52             0.0
6362619   743  CASH_OUT   850002.52      850002.52             0.0

         oldbalanceDest  newbalanceDest  isFraud
6362615            0.00       339682.13        1
6362616            0.00            0.00        1
6362617        68488.84      6379898.11        1
6362618            0.00            0.00        1
6362619      6510099.11      7360101.63        1
```

3.0.1 Checking null values

[45]:
```
new_df.isnull().any()
```

[45]:
```
step              False
type              False
amount            False
oldbalanceOrg     False
newbalanceOrig    False
oldbalanceDest    False
newbalanceDest    False
isFraud           False
dtype: bool
```

[46]:
```
new_df.isnull().sum()
```

[46]:
```
step              0
type              0
amount            0
oldbalanceOrg     0
newbalanceOrig    0
oldbalanceDest    0
```

```
newbalanceDest    0
isFraud           0
dtype: int64
```

Clearly there are no null values

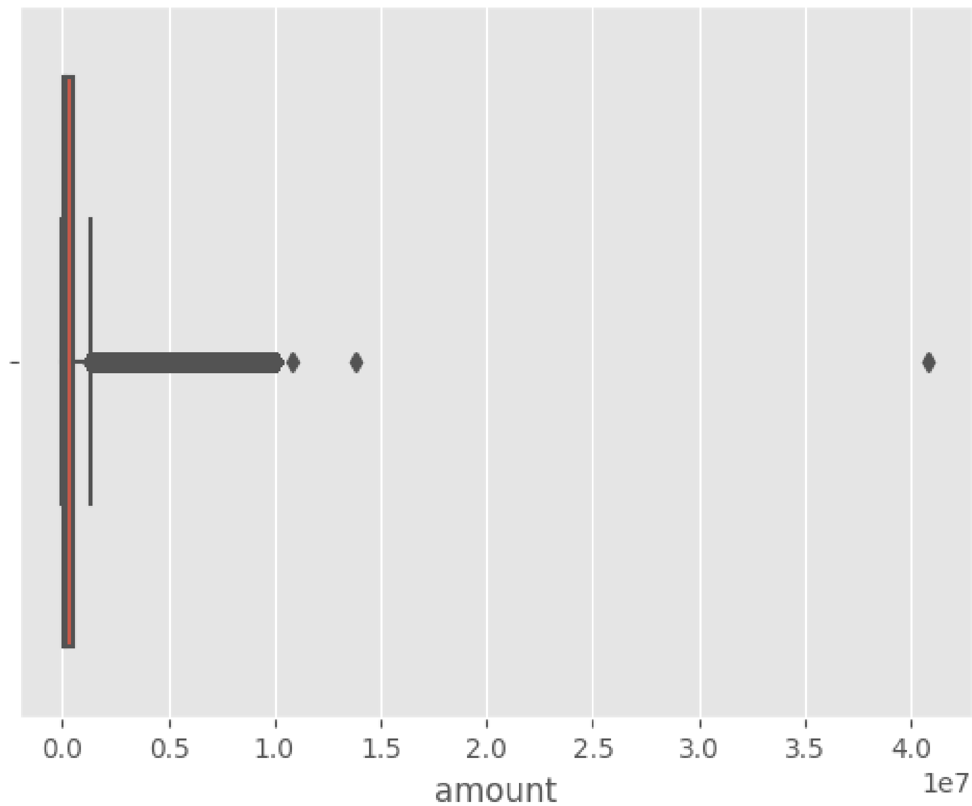
[47]:
```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16426 entries, 2472014 to 6362619
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   step            16426 non-null  int64
 1   type            16426 non-null  object
 2   amount          16426 non-null  float64
 3   oldbalanceOrg   16426 non-null  float64
 4   newbalanceOrig  16426 non-null  float64
 5   oldbalanceDest  16426 non-null  float64
 6   newbalanceDest  16426 non-null  float64
 7   isFraud         16426 non-null  int64
dtypes: float64(5), int64(2), object(1)
memory usage: 1.1+ MB
```

3.0.2 Handling outliers

[48]:
```
sns.boxplot(x=new_df["amount"])
```

[48]:
```
<Axes: xlabel='amount'>
```

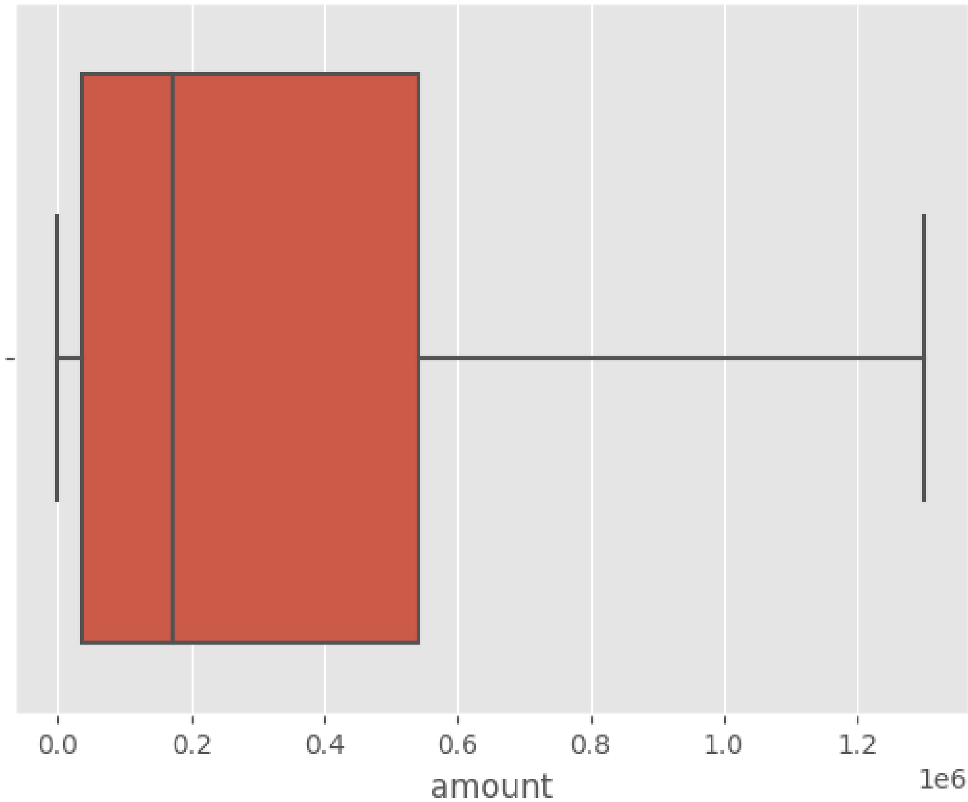


There are outliers in the “amount” column which need to be removed

[49]:
```
q1 = new_df['amount'].quantile(0.25)
q3 = new_df['amount'].quantile(0.75)

iqr = q3 - q1

upper_limit = q3 + 1.5 * iqr
lower_limit = q1 - 1.5 * iqr

mean = new_df['amount'].mean()

# Replace outliers with the mean value
new_df['amount'] = np.where(new_df['amount'] > upper_limit, mean,␣
 ↪new_df['amount'])
new_df['amount'] = np.where(new_df['amount'] < lower_limit, mean,␣
 ↪new_df['amount'])
```

[50]:
```
sns.boxplot(x=new_df["amount"])
```

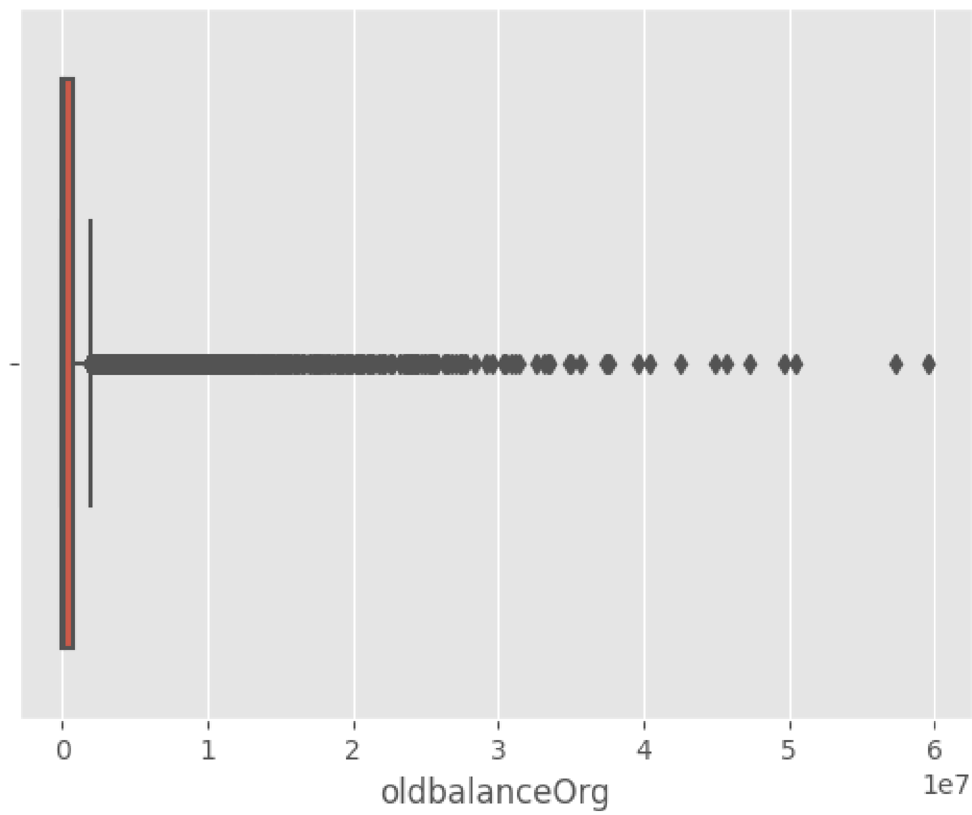
[50]:
```
<Axes: xlabel='amount'>
```



Now outliers are removed in “amount” columns

[51]:
```
sns.boxplot(x=new_df["oldbalanceOrg"])
```

[51]: <Axes: xlabel='oldbalanceOrg'>

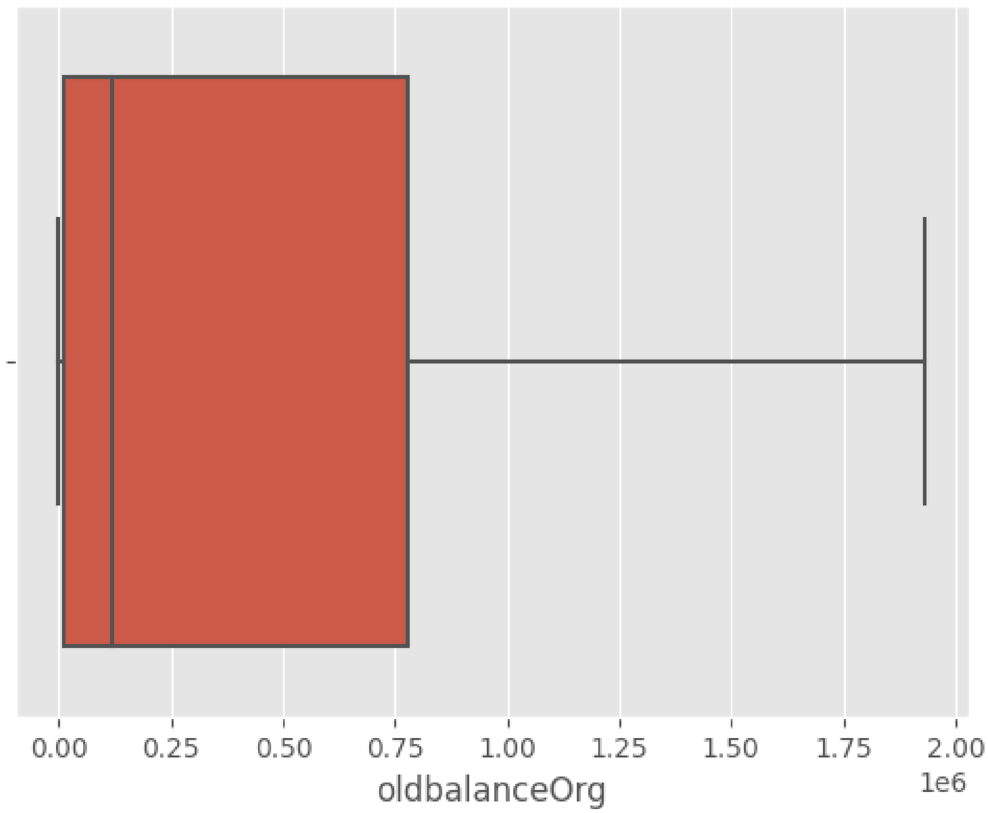


The column “oldbalanceOrg” has outliers which need to be treated

[52]:
```
q1 = new_df['oldbalanceOrg'].quantile(0.25)
q3 = new_df['oldbalanceOrg'].quantile(0.75)

iqr = q3 - q1

upper_limit = q3 + 1.5 * iqr
lower_limit = q1 - 1.5 * iqr

mean = new_df['oldbalanceOrg'].mean()

# Replace outliers with the mean value
new_df['oldbalanceOrg'] = np.where(new_df['oldbalanceOrg'] > upper_limit, mean,␣
↪new_df['oldbalanceOrg'])
new_df['oldbalanceOrg'] = np.where(new_df['oldbalanceOrg'] < lower_limit, mean,␣
↪new_df['oldbalanceOrg'])
```

[53]:
```
sns.boxplot(x=new_df["oldbalanceOrg"])
```

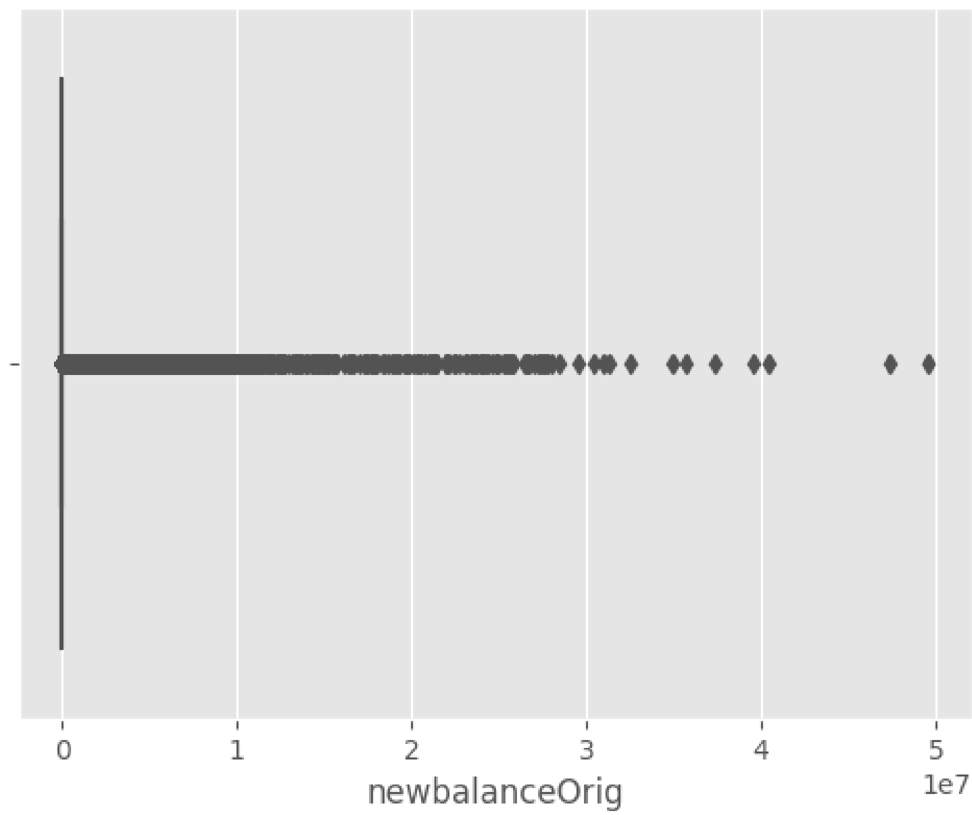
[53]:
```
<Axes: xlabel='oldbalanceOrg'>
```



Now “oldbalanceOrg” is free of outliers

[54]:
```
sns.boxplot(x=new_df["newbalanceOrig"])
```

[54]: `<Axes: xlabel='newbalanceOrig'>`

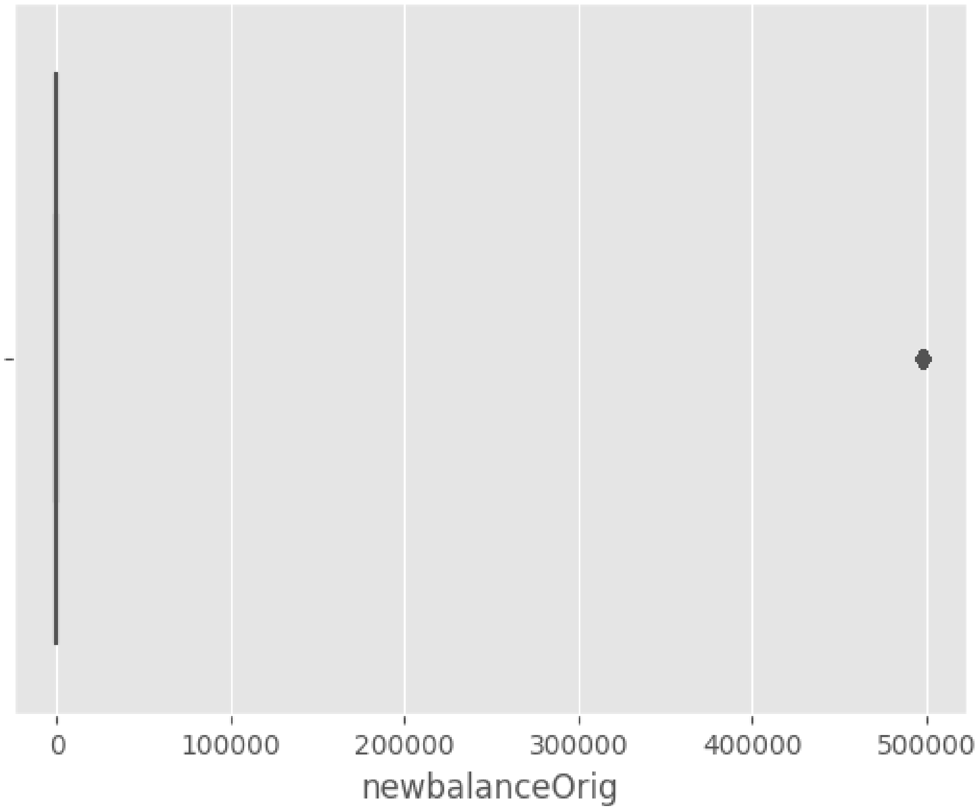


The column "newbalanceOrig" has outliers which need to be treated

[55]:
```
q1 = new_df['newbalanceOrig'].quantile(0.25)
q3 = new_df['newbalanceOrig'].quantile(0.75)

iqr = q3 - q1

upper_limit = q3 + 1.5 * iqr
lower_limit = q1 - 1.5 * iqr

mean = new_df['newbalanceOrig'].mean()

# Replace outliers with the mean value
new_df['newbalanceOrig'] = np.where(new_df['newbalanceOrig'] > upper_limit,␣
 ↪mean, new_df['newbalanceOrig'])
new_df['newbalanceOrig'] = np.where(new_df['newbalanceOrig'] < lower_limit,␣
 ↪mean, new_df['newbalanceOrig'])
```

[56]:
```
sns.boxplot(x=new_df["newbalanceOrig"])
```

[56]: <Axes: xlabel='newbalanceOrig'>



Now "newbalanceOrig" is free of outliers

3.0.3 Handling categorial or object data using label encoding

[57]:
```
from sklearn.preprocessing import LabelEncoder
```

[58]:
```
le = LabelEncoder()

new_df["type"] = le.fit_transform(new_df["type"])
```

[60]:
```
new_df["type"].value_counts()
```

[60]:
```
type
1    6977
4    4830
3    2834
0    1731
2      54
Name: count, dtype: int64
```

[63]:
```
le.classes_
```

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

[64]:
```
map = dict(zip(le.classes_, range(len(le.classes_))))
```

[65]:
```
map
```

[65]: {'CASH_IN': 0, 'CASH_OUT': 1, 'DEBIT': 2, 'PAYMENT': 3, 'TRANSFER': 4}

3.0.4 Dividing dataset into dependent and independent variable

[60]:
```
x = new_df.drop("isFraud", axis=1)
y = new_df["isFraud"]
```

[61]:
```
x.head()
```

[61]:
```
         step  type     amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
156272     12     0  193273.15   7.988900e+04   522204.069453       559880.30
6194917   573     0  139530.26   1.238643e+06   522204.069453      1028962.30
1199699   133     3   37315.10   1.139180e+05   522204.069453            0.00
186011     13     3   12722.06   4.601860e+03        0.000000            0.00
1157583   131     1   49391.70   0.000000e+00        0.000000      1872470.67

         newbalanceDest
156272        366607.14
6194917       889432.04
1199699            0.00
186011             0.00
1157583      1921862.37
```

[62]:
```
y.head()
```

[62]:
```
156272     0
6194917    0
1199699    0
186011     0
1157583    0
Name: isFraud, dtype: int64
```

[63]:
```
type(x)
```

[63]: pandas.core.frame.DataFrame

[64]:
```
type(y)
```

[64]: pandas.core.series.Series

3.0.5 Splitting data into training and testing set

[65]:
```
from sklearn.model_selection import train_test_split
```

[66]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
↪test_size=0.2)
```

[67]:
```
print(x_train.shape)
print(x_test.shape)
print(y_test.shape)
print(y_train.shape)
```

```
(13140, 7)
(3286, 7)
(3286,)
(13140,)
```

4 Model building

Now that our data is clean, we can train it on different models and pick the best performing model

4.1 1. Random forest classifier

A Random Forest is an ensemble of decision trees. It builds multiple trees from random subsets of data and features, combines their predictions, and reduces overfitting to create a robust and accurate classification model.

4.1.1 Import model building libraries

[68]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

4.1.2 Initialising the model

[69]:
```
rfc = RandomForestClassifier()
rfc
```

[69]: RandomForestClassifier()

4.1.3 Training and testing the model

[70]:
```
rfc.fit(x_train, y_train)
```

[70]: RandomForestClassifier()

[71]:
```
# testing accuracy
```

```
y_test_predict1 = rfc.predict(x_test)
test_accuracy = accuracy_score(y_test, y_test_predict1)
test_accuracy
```

[71]: 0.9914790018259282

[72]:
```
# training accuracy

y_train_predict1 = rfc.predict(x_train)
train_accuracy = accuracy_score(y_train, y_train_predict1)
train_accuracy
```

[72]: 1.0

4.1.4 Evaluating performance of the model

[73]:
```
pd.crosstab(y_test, y_test_predict1)
```

[73]:
```
col_0       0     1
isFraud
0        1609    21
1           7  1649
```

[ ]:

[74]:
```
print(classification_report(y_test, y_test_predict1))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1630
           1       0.99      1.00      0.99      1656

    accuracy                           0.99      3286
   macro avg       0.99      0.99      0.99      3286
weighted avg       0.99      0.99      0.99      3286
```

[ ]:

[ ]:

4.2 2. Decision trees

A Decision Tree is a tree-like model that makes decisions by recursively splitting the data based on features, aiming to create homogeneous groups. It's a simple yet interpretable way to perform classification and regression tasks.

4.2.1 Import model building libraries

[75]:
```
from sklearn.tree import DecisionTreeClassifier
```

4.2.2 Initialising the model

[76]:
```
dtc = DecisionTreeClassifier()
dtc
```

[76]:
```
DecisionTreeClassifier()
```

4.2.3 Training and testing the model

[77]:
```
dtc.fit(x_train, y_train)
```

[77]:
```
DecisionTreeClassifier()
```

[78]:
```
# testing accuracy

y_test_predict2 = dtc.predict(x_test)
test_accuracy = accuracy_score(y_test, y_test_predict2)
test_accuracy
```

[78]:
```
0.9902617163724894
```

[79]:
```
# training accuracy

y_train_predict2 = dtc.predict(x_train)
train_accuracy = accuracy_score(y_train, y_train_predict2)
train_accuracy
```

[79]:
```
1.0
```

4.2.4 Evaluating the performance of the model

[80]:
```
pd.crosstab(y_test, y_test_predict2)
```

[80]:

col_0	0	1
isFraud		
0	1609	21
1	11	1645

[ ]:
```

```

[81]:
```
print(classification_report(y_test, y_test_predict2))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1630
           1       0.99      0.99      0.99      1656

    accuracy                           0.99      3286
   macro avg       0.99      0.99      0.99      3286
weighted avg       0.99      0.99      0.99      3286
```

4.3 3. ExtraTrees classifier

An ExtraTrees Classifier (Extremely Randomized Trees Classifier) is an ensemble machine learning model that builds multiple decision trees with a key difference from Random Forest. It adds randomness by selecting random subsets of features and choosing the best split points, which makes it computationally efficient. It combines the predictions from these trees to make accurate classifications and reduce overfitting.

4.3.1 Import model building libraries

[82]:
```
from sklearn.ensemble import ExtraTreesClassifier
```

4.3.2 Initialising the model

[83]:
```
etc = ExtraTreesClassifier()
etc
```

[83]:
```
ExtraTreesClassifier()
```

4.3.3 Training and testing the model

[84]:
```
etc.fit(x_train,y_train)
```

[84]:
```
ExtraTreesClassifier()
```

[85]:
```
# testing accuracy

y_test_predict3 = etc.predict(x_test)
test_accuracy = accuracy_score(y_test, y_test_predict3)
test_accuracy
```

[85]:
```
0.9899573950091296
```

[86]:
```
# training accuracy

y_train_predict3 = etc.predict(x_train)
train_accuracy = accuracy_score(y_train, y_train_predict3)
```

```
train_accuracy
```

[86]: 1.0

4.3.4 Evaluating the performance of the model

[87]:
```
pd.crosstab(y_test,y_test_predict3)
```

[87]:
```
col_0       0     1
isFraud
0        1609    21
1          12  1644
```

[ ]:
```

```

[88]:
```
print(classification_report (y_test,y_test_predict3))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1630
           1       0.99      0.99      0.99      1656

    accuracy                           0.99      3286
   macro avg       0.99      0.99      0.99      3286
weighted avg       0.99      0.99      0.99      3286
```

4.4 4. Support vector machine classifier

A Support Vector Machine (SVM) Classifier is a powerful machine learning model used for both classification and regression tasks. It works by finding the best hyperplane that separates different classes in the data, aiming to maximize the margin between the classes. SVMs can handle linear and nonlinear data, making them effective for a wide range of applications, from image recognition to text classification.

4.4.1 Import model building libraries

[89]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

4.4.2 Initialising the model

[90]:
```
svc = SVC()
svc
```

[90]: SVC()

4.4.3 Training and testing the model

[91]:
```
svc.fit(x_train,y_train)
```

[91]:
```
SVC()
```

[92]:
```
# testing accuracy

y_test_predict4 = svc.predict(x_test)
test_accuracy = accuracy_score(y_test, y_test_predict4)
test_accuracy
```

[92]:
```
0.8703590992087644
```

[93]:
```
# training accuracy

y_train_predict4 = svc.predict(x_train)
train_accuracy = accuracy_score(y_train, y_train_predict4)
train_accuracy
```

[93]:
```
0.8622526636225266
```

4.4.4 Evaluating the performance of the model

[94]:
```
pd.crosstab(y_test,y_test_predict4)
```

[94]:

col_0	0	1
isFraud		
0	1621	9
1	417	1239

[ ]:
```

```

[95]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, y_test_predict4))
```

```
              precision    recall  f1-score   support

           0       0.80      0.99      0.88      1630
           1       0.99      0.75      0.85      1656

    accuracy                           0.87      3286
   macro avg       0.89      0.87      0.87      3286
weighted avg       0.89      0.87      0.87      3286
```

4.5 5. xgboost classifier

XGBoost (Extreme Gradient Boosting) Classifier is a popular and powerful machine learning algorithm known for its accuracy and speed. It belongs to the gradient boosting family and combines the predictions of multiple weak learners (typically decision trees) in an iterative manner. XGBoost is designed to minimize prediction errors and can handle complex datasets, making it a top choice for various classification tasks.

4.5.1 Import model building libraries

[96]:
```
import xgboost as xgb
from sklearn.metrics import accuracy_score
```

4.5.2 Initialising the model

[97]:
```
xgb1 = xgb.XGBClassifier()
xgb1
```

[97]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, …)
```

4.5.3 Training and testing the model

[98]:
```
xgb1.fit(x_train,y_train)
```

[98]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, …)
```

[99]:
```python
# testing accuracy

y_test_predict5 = xgb1.predict(x_test)
test_accuracy = accuracy_score(y_test, y_test_predict5)
test_accuracy
```

[99]: 0.9945222154595252

[100]:
```python
# training accuracy

y_train_predict5 = svc.predict(x_train)
train_accuracy = accuracy_score(y_train, y_train_predict5)
train_accuracy
```

[100]: 0.8622526636225266

4.5.4 Evaluating the performance of the model

[101]:
```python
pd.crosstab(y_test, y_test_predict5)
```

[101]:
```
col_0       0     1
isFraud
0        1617    13
1           5  1651
```

[ ]:
```python

```

[102]:
```python
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report (y_test, y_test_predict5))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1630
           1       0.99      1.00      0.99      1656

    accuracy                           0.99      3286
   macro avg       0.99      0.99      0.99      3286
weighted avg       0.99      0.99      0.99      3286
```

4.6 Comparing models

[103]:
```python
def compareModel():
    print("Train accuracy for RFC: ",␣
↪accuracy_score(y_train_predict1,y_train)*100)
    print("Test accuracy for RFC: ", accuracy_score(y_test_predict1,y_test)*100)
```

```
    print("\n")
    print("Train accuracy for DTC: ", 
 ↪accuracy_score(y_train_predict2,y_train)*100)
    print("Test accuracy for DTC: ", accuracy_score(y_test_predict2,y_test)*100)
    print("\n")
    print("Train accuracy for ETC: ", 
 ↪accuracy_score(y_train_predict3,y_train)*100)
    print("Test accuracy for ETC: ", accuracy_score 
 ↪(y_test_predict3,y_test)*100)
    print("\n")
    print("Train accuracy for SVC: ", 
 ↪accuracy_score(y_train_predict4,y_train)*100)
    print("Test accuracy for SVC: ", accuracy_score(y_test_predict4,y_test)*100)
    print("\n")
    print("Train accuracy for XGB: ", 
 ↪accuracy_score(y_train_predict5,y_train)*100)
    print("Test accuracy for XGB: ", accuracy_score(y_test_predict5,y_test)*100)
```

[104]:
```
compareModel()
```

```
Train accuracy for RFC:  100.0
Test accuracy for RFC:  99.14790018259282


Train accuracy for DTC:  100.0
Test accuracy for DTC:  99.02617163724894


Train accuracy for ETC:  100.0
Test accuracy for ETC:  98.99573950091296


Train accuracy for SVC:  86.22526636225267
Test accuracy for SVC:  87.03590992087643


Train accuracy for XGB:  86.22526636225267
Test accuracy for XGB:  99.45222154595253
```

On comparing the training and testing of 5 different models trained by 5 different algorithms, we have found that XGBoost Classifier is the best model as it has the highest testing accuracy and is not overfitting.

4.7 Save the model

[105]:
```
import pickle
pickle.dump(xgb1, open('payments.pkl', 'wb'))
```

[66]:
```
new_df
```

[66]:
```
         step  type         amount  oldbalanceOrg  newbalanceOrig  \
2472014   204     3    2711.510000   1.556100e+05   498179.214218
1329496   137     3   50062.450000   0.000000e+00        0.000000
3569677   260     4  142407.650000   0.000000e+00        0.000000
1698217   159     3   11379.430000   2.542400e+04   498179.214218
5190368   369     0  220293.910000   2.936726e+05   498179.214218
…         …     …            …            …             …
6362615   743     1  339682.130000   3.396821e+05        0.000000
6362616   743     4  823663.257501   1.215890e+06        0.000000
6362617   743     1  823663.257501   1.215890e+06        0.000000
6362618   743     4  850002.520000   8.500025e+05        0.000000
6362619   743     1  850002.520000   8.500025e+05        0.000000

         oldbalanceDest  newbalanceDest  isFraud
2472014            0.00            0.00        0
1329496            0.00            0.00        0
3569677       830469.20      1181738.48        0
1698217            0.00            0.00        0
5190368      3017311.84      2797017.93        0
…                  …             …       …
6362615            0.00       339682.13        1
6362616            0.00            0.00        1
6362617        68488.84      6379898.11        1
6362618            0.00            0.00        1
6362619      6510099.11      7360101.63        1

[16426 rows x 8 columns]
```

[ ]:
```

```

Flask Application Code:

Directory structure –

```
opfd
  > .ipynb_checkpoints
  v static
      background.jpg
      logo.png
      style.css
      Techno.ttf
  v templates
      home.html
      predict.html
      submit.html
    app.py
    balanced_dataset.csv
    fraud detection data...
    fraud detection.ipynb
    payments.pkl
```

home.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Share+Tech+Mono&display=swap">

    <link rel="stylesheet" href="https://www.dafont.com/techno-2.font">

    <link rel="icon" href="static/logo.png" type="image/png">

    <title>Online Payments Fraud Detection</title>
</head>
<body style="background-image: url('/static/background.jpg');">
    <h1 class="google-font">Online Payments Fraud Detection</h1>
    <div class="content google-font">
        The Online Payments Fraud Detection project addresses the growing
concerns
            related to the surge in online credit/debit card transactions, which
has
            concurrently led to an increase in fraudulent activities.
```

```
            Recognizing the need for effective fraud detection, our solution
employs machine learning
            classification algorithms, including Decision Tree, Random Forest,
SVM,
            Extra Tree Classifier, and XGBoost Classifier. This diverse set of
algorithms
            aims to enhance accuracy and overcome specific drawbacks associated
with
            traditional approaches. By leveraging these algorithms, we predict
and further
            process potential frauds, particularly focusing on detecting changes
in
            transaction behavior.

            The proposed method becomes crucial in handling the
            substantial volume of data inherent in credit/debit card
transactions. After
            training and testing the data with various algorithms, the model
exhibiting
            the highest performance is selected and saved in a pkl format. To
ensure
            practical usability, we integrate this predictive model with a Flask
web
            application, providing users with a seamless and intuitive interface
for
            real-time fraud detection, thereby fortifying the security of online
payment
            systems.
    </div>
    <div class="buttons google-font">
        <a href="{{ url_for('home') }}">Home</a>
        <a href="{{ url_for('predict') }}">Predict</a>
    </div>
</body>
</html>
```

predict.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Share+Tech+Mono&display=swap">

    <link rel="stylesheet" href="https://www.dafont.com/techno-2.font">

    <link rel="icon" href="static/logo.png" type="image/png">

    <title>Predict</title>
```

```
</head>
<body style="background-image: url('/static/background.jpg');">
    <h1 class="google-font">Online Payments Fraud Detection</h1>
    <div class="content google-font">
        <form action="{{ url_for('predict') }}" method="post">
            <label for="step">Step:</label><br>
            <input type="text" name="step" required><br><br>

            <label for="type">Type:</label><br>
            <input type="text" name="type" required><br><br>

            <label for="amount">Amount:</label><br>
            <input type="text" name="amount" required><br><br>

            <label for="oldbalanceOrg">Old Balance Org:</label><br>
            <input type="text" name="oldbalanceOrg" required><br><br>

            <label for="newbalanceOrig">New Balance Orig:</label><br>
            <input type="text" name="newbalanceOrig" required><br><br>

            <label for="oldbalanceDest">Old Balance Dest:</label><br>
            <input type="text" name="oldbalanceDest" required><br><br>

            <label for="newbalanceDest">New Balance Dest:</label><br>
            <input type="text" name="newbalanceDest" required><br><br>

            <button class="google-font" type="submit">Submit</button>
        </form>
    </div>
    <div class="buttons google-font">
        <a href="{{ url_for('home') }}">Home</a>
        <a href="{{ url_for('predict') }}">Predict</a>
    </div>
</body>
</html>
```

submit.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Share+Tech+Mono&display=swap">

    <link rel="stylesheet" href="https://www.dafont.com/techno-2.font">

    <link rel="icon" href="static/logo.png" type="image/png">

    <title>Result</title>
```

```html
</head>
<body style="background-image: url('/static/background.jpg');">
    <h1 class="google-font">Online Payments Fraud Detection</h1>
    <div class="buttons google-font">
        <a href="{{ url_for('home') }}">Home</a>
        <a href="{{ url_for('predict') }}">Predict</a>
    </div>
    <div class="content google-font">
        <h2>Result:</h2>
        <p>This trasaction is {{ result }}</p>
    </div>
</body>
</html>
```

style.css

```css
h1{
    color: white;
    font-size: 50px;
    font-family: 'Times New Roman', Times, serif;
}

body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-size: cover;
    background-repeat: no-repeat;
}

.google-font {
    font-family: 'Share Tech Mono', sans-serif;
    color: white;
    padding: 20px;
}

.buttons {
    position: absolute;
    top: 20px;
    right: 20px;
}

.buttons a {
    color: white;
    text-decoration: none;
    padding: 10px;
    margin: 0 10px;
    border: 1px solid white;
    border-radius: 5px;
}

.content {
    padding: 20px;
```

```
    color: white;
    max-width: 600px;
}

button {
    color: white;
    background: none;
    text-decoration: none;
    border: 1px solid white;
    border-radius: 5px;
}
```

app.py

```
from flask import Flask, render_template, request
import pandas as pd
import joblib
import os

app = Flask(__name__)

current_dir = os.getcwd()

model_path = os.path.join(current_dir, 'payments.pkl')

if os.path.exists(model_path):
    model = joblib.load(model_path)
else:
    print(f"Model file '{model_path}' not found.")

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        features = [
            float(request.form['step']),
            float(request.form['type']),
            float(request.form['amount']),
            float(request.form['oldbalanceOrg']),
            float(request.form['newbalanceOrig']),
            float(request.form['oldbalanceDest']),
            float(request.form['newbalanceDest'])
        ]

        df = pd.DataFrame([features], columns=['step', 'type', 'amount',
'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest'])

        prediction = model.predict(df)[0]

        result = "Fraud" if prediction == 1 else "Not Fraud"
```

```python
        return render_template('submit.html', result=result)

    return render_template('predict.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Output pages:

home.html

predict.html

submit.html

For input A,

For input B,


Online Payments Fraud Detection
127.0.0.1:5000/predict
Refer the below indices for the respective type of transactions and use them in the input field.
'CASH_IN': 0, 'CASH_OUT': 1, 'DEBIT': 2, 'PAYMENT': 3, 'TRANSFER': 4
Step:
743
Type:
1
Amount:
339682.130000
Old Balance Org:
339682.1
New Balance Orig:
0.000000
Old Balance Dest:
0
New Balance Dest:
339682.13
Submit
ENG IN
01:38
19-11-2023


Online Payments Fraud Detection
127.0.0.1:5000/predict
Home
Predict
Online Payments Fraud Detection
Result:
This trasaction is Fraud
ENG IN
01:38
19-11-2023