In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter("ignore")
```

In [2]:

```
df = pd.read_csv('C:\\Users\\Jayan\\Downloads\\Fraud.csv')
df.head()
```

Out[2]:

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

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype  
---  ------          -----  
 0   step            int64  
 1   type            object 
 2   amount          float64
 3   nameOrig        object 
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object 
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64  
 10  isFlaggedFraud  int64  
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

In [4]:

```
 continous=["amount","oldbalanceOrg","newbalanceOrig","oldbalanceDest","newbalanceDest"]
```

In [5]:

```
df.isnull().sum()
```

Out[5]:

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

In [6]:

```
df.shape
```

Out[6]:

```
(6362620, 11)
```

In [7]:

```
df.corr()
```

Out[7]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlagg
step	1.000000	0.022373	-0.010058	-0.010299	0.027665	0.025888	0.031578	
amount	0.022373	1.000000	-0.002762	-0.007861	0.294137	0.459304	0.076688	
oldbalanceOrg	-0.010058	-0.002762	1.000000	0.998803	0.066243	0.042029	0.010154	
newbalanceOrig	-0.010299	-0.007861	0.998803	1.000000	0.067812	0.041837	-0.008148	
oldbalanceDest	0.027665	0.294137	0.066243	0.067812	1.000000	0.976569	-0.005885	
newbalanceDest	0.025888	0.459304	0.042029	0.041837	0.976569	1.000000	0.000535	
isFraud	0.031578	0.076688	0.010154	-0.008148	-0.005885	0.000535	1.000000	
isFlaggedFraud	0.003277	0.012295	0.003835	0.003776	-0.000513	-0.000529	0.044109	

In [8]:

```
# df.drop_duplicates()
```

In [9]:

```
df['type'].unique()
```

Out[9]:

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

In [10]:

```
df['isFlaggedFraud'].unique()
```

Out[10]:

```
array([0, 1], dtype=int64)
```

In [11]:

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

In [12]:

```
df
```

Out[12]:

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	170136.00	160296.36	0.00	0.00	0
1	1	PAYMENT	1864.28	21249.00	19384.72	0.00	0.00	0
2	1	TRANSFER	181.00	181.00	0.00	0.00	0.00	1
3	1	CASH_OUT	181.00	181.00	0.00	21182.00	0.00	1
4	1	PAYMENT	11668.14	41554.00	29885.86	0.00	0.00	0
...	...	...	...	...	...	...	...	...
6362615	743	CASH_OUT	339682.13	339682.13	0.00	0.00	339682.13	1
6362616	743	TRANSFER	6311409.28	6311409.28	0.00	0.00	0.00	1
6362617	743	CASH_OUT	6311409.28	6311409.28	0.00	68488.84	6379898.11	1
6362618	743	TRANSFER	850002.52	850002.52	0.00	0.00	0.00	1
6362619	743	CASH_OUT	850002.52	850002.52	0.00	6510099.11	7360101.63	1

6362620 rows × 8 columns

In [13]:

```
df['type'].count()
```

Out[13]:

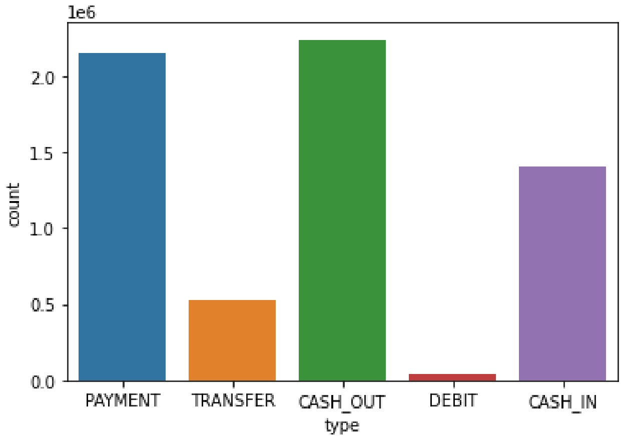
```
6362620
```

In [14]:

```
sns.countplot(df.type)
```

Out[14]:

```
<AxesSubplot:xlabel='type', ylabel='count'>
```



In [15]:

```
df["type"].value_counts()
```

Out[15]:

```
CASH_OUT    2237500
PAYMENT     2151495
CASH_IN     1399284
TRANSFER     532909
DEBIT         41432
Name: type, dtype: int64
```

In [16]:

```
plt.show()
```

In [17]:

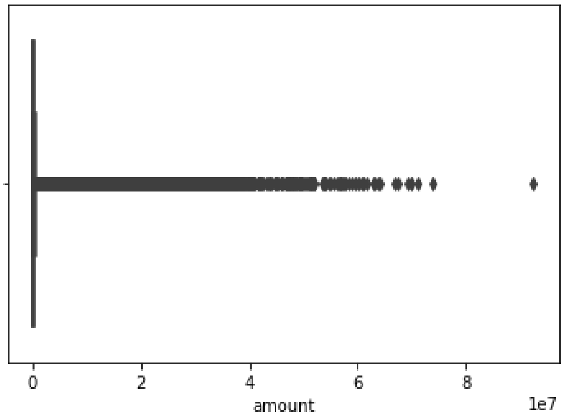
```
df[continous].describe()
```

Out[17]:

	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06
mean	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+06
std	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+05
75%	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+06
max	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+08

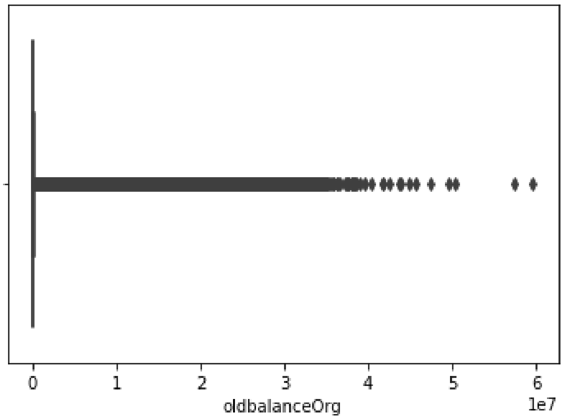
In [18]:

```
sns.boxplot(x=df["amount"])   #outliar in the column
plt.show()
```



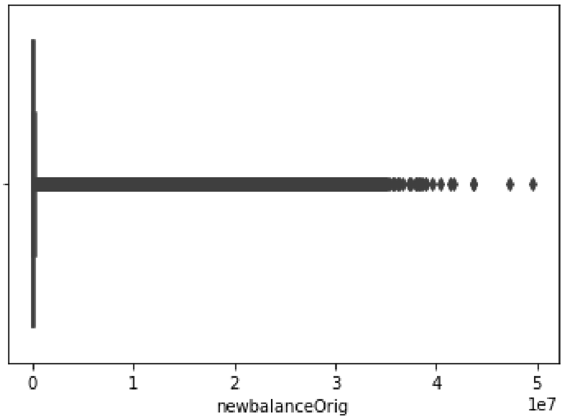
In [19]:

```
sns.boxplot(x=df["oldbalanceOrg"])   #outliar in the column
plt.show()
```



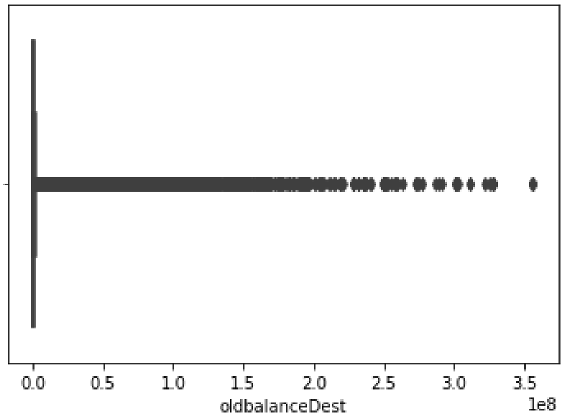
In [20]:

```
sns.boxplot(x=df["newbalanceOrig"])   #outliar in the column
plt.show()
```



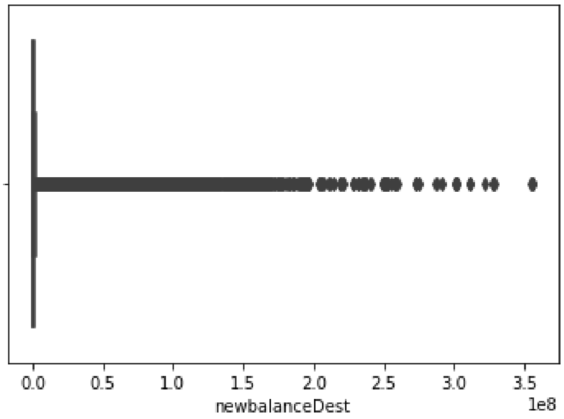
In [21]:

```
sns.boxplot(x=df["oldbalanceDest"])   #outliar in the column
plt.show()
```



In [22]:

```
sns.boxplot(x=df["newbalanceDest"])    #outliar in the column
plt.show()
```

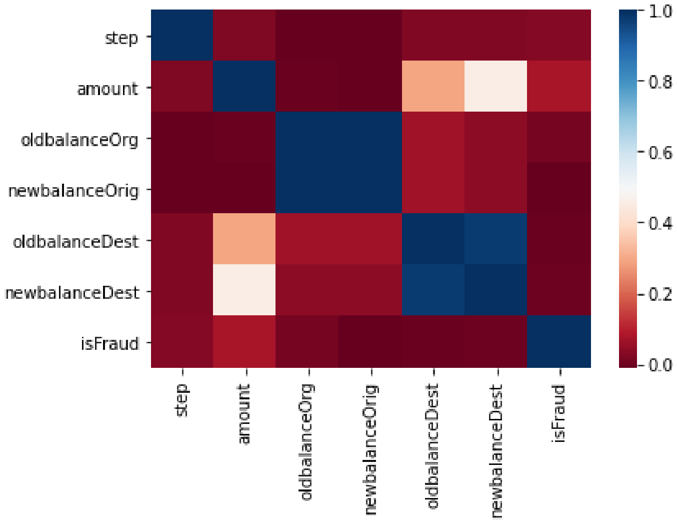


In [23]:

```
 continous=df[["amount","oldbalanceOrg","newbalanceOrig","oldbalanceDest","newbalanceDest"]]
```

In [24]:

```
sns.heatmap(df.corr(),cmap='RdBu',);
```



In [25]:

```
# DATA WRANGLING
df = pd.get_dummies(df,drop_first=True)
```

In [26]:

```
x = df.drop("isFraud",axis=1)
y = df["isFraud"]
```

In [27]:

```
# Train & Test Split
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=10)
```

In [28]:

```
# Scaling Data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

LOGISTIC REGRESSION BASE MODEL

In [54]:

```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression()
log_model.fit(x_train,y_train)
```

Out[54]:

```
LogisticRegression()
```

In [55]:

```
# Prediction
ypred_train= log_model.predict(x_train)
ypred_test =log_model.predict (x_test)
```

In [56]:

```
# Evalution
from sklearn.metrics import accuracy_score
print("train accuracy",accuracy_score(y_train,ypred_train))
print("test accuracy",accuracy_score(y_test,ypred_test))
```

```
train accuracy 0.9992170790379704
test accuracy 0.9992309247867492
```

In [57]:

```
# Cross Validation
from sklearn.model_selection import cross_val_score  # cv score
scores = cross_val_score(log_model,x,y,cv=5)
print(scores)
scores.mean()
```

```
[0.9745828  0.99929589 0.99920473 0.99906328 0.9990177 ]
```

Out[57]:

```
0.9942328789083742
```

In [30]:

```
# svm and knn can't use beacuse of large data set
```

In [31]:

```
# 2 # decision tree
```

In [32]:

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
dt_model.fit(x_train,y_train)
```

Out[32]:

```
DecisionTreeClassifier()
```

In [33]:

```
# prediction
ypred_train= dt_model.predict(x_train)
ypred_test = dt_model.predict (x_test)
```

In [34]:

```
# accuracy
# Evalution
from sklearn.metrics import accuracy_score
print("train accuracy",accuracy_score(y_train,ypred_train))
print("test accuracy",accuracy_score(y_test,ypred_test))
```

```
train accuracy 1.0
test accuracy 0.9996956180525214
```

In [35]:

```
# Cross Validation
from sklearn.model_selection import cross_val_score  # cv score
scores = cross_val_score(dt_model,x,y,cv=5)
print(scores)
scores.mean()
```

```
[0.99919451 0.99941376 0.99738551 0.99972103 0.02203102]
```

Out[35]:

```
0.8035491668526488
```

In [36]:

```
dt_model.feature_importances_
```

Out[36]:

```
array([0.0613073 , 0.15634119, 0.33847038, 0.03780783, 0.09541906,
       0.29392073, 0.01485646, 0.        , 0.        , 0.00187706])
```

In [ ]:

```
# can't do HYPER PARAMETER TUNNING because it take to much time IN LAPTOP
```

In [ ]:

```
# RANDOM FOREST CAN'T USE BEACUSE IT TAKE TO MUCH TIME .
```

ADA BOOST

In [42]:

```
from sklearn.ensemble import AdaBoostClassifier
ab_model = AdaBoostClassifier()
ab_model.fit(x_train,y_train)
```

Out[42]:

```
AdaBoostClassifier()
```

In [43]:

```
ypred_train= ab_model.predict(x_train)
ypred_test = ab_model.predict(x_test)
```

In [44]:

```
 #accuracy
 #Evalution
from sklearn.metrics import accuracy_score
print("train accuracy",accuracy_score(y_train,ypred_train))
print("test accuracy",accuracy_score(y_test,ypred_test))
```

```
train accuracy 0.9994056805889039
test accuracy 0.9994043334349686
```

In [45]:

```
# Cross Validation
from sklearn.model_selection import cross_val_score  # cv score
scores = cross_val_score(ab_model,x,y,cv=5)
print(scores)
scores.mean()
```

```
[0.99871594 0.99901377 0.99879845 0.99859492 0.20339734]
```

Out[45]:

```
0.8397040841665854
```

In [ ]:

```
# CANT CONSIDER THIS MODEL BEACUSE CV LOW CANCEL THIS MODEL AND CANT DO HPT BEACUSE IT TAKE TIME IN LAPTOP.
```

In [ ]:

```
# GRADENT BOOST CANT USE IT TAKE TOO MUCH TIME IN LAPTOP .SO I REMOVE THIS GRADIENT BOOST CODE
```

XGBOOST

In [48]:

```
pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.7.6-py3-none-win_amd64.whl (70.9 MB)
Requirement already satisfied: scipy in c:\users\jayan\anaconda3\lib\site-packages (from xgboost)
(1.6.2)
Requirement already satisfied: numpy in c:\users\jayan\anaconda3\lib\site-packages (from xgboost)
(1.20.1)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.6
Note: you may need to restart the kernel to use updated packages.
```

In [49]:

```
from xgboost import XGBClassifier
xgb_model = XGBClassifier()
xgb_model.fit(x_train,y_train)
```

Out[49]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=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,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [50]:

```
ypred_train= xgb_model.predict(x_train)
ypred_test = xgb_model.predict(x_test)
```

In [51]:

```
# accuracy
# Evalution
from sklearn.metrics import accuracy_score
print("train accuracy",accuracy_score(y_train,ypred_train))
print("test accuracy",accuracy_score(y_test,ypred_test))
```

```
train accuracy 0.9998839202359135
test accuracy 0.9998009205851258
```

In [52]:

```
# Cross Validation
from sklearn.model_selection import cross_val_score  # cv score
scores= cross_val_score(estimator = xgb_model, X=x , y=y , cv=5) # X ha aaplya la capital ch pahije.chota x ha
print("Cross Validation Score:",scores.mean())
```

```
Cross Validation Score: 0.9894356412924236
```

In [ ]:

```
# XGB BOOST MODEL OUR FINAL MODEL BECAUSE XGB MODEL HAS HIGH TEST ACCURACY  0.9998009205851258 THAN OTHER MODELS
```

SAVE THE MODEL

In [59]:

```
from joblib import dump
```

In [60]:

```
dump(xgb_model,"fraudulent_transaction.joblib")
```

Out[60]:

```
['fraudulent_transaction.joblib']
```

In [ ]:

In [ ]: