In [1]:
```
# import libriers
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt # visulation of data
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

1. Numpy Stands for numerical python and used for working with arrays.
2. Pandas is used for analyze and manipulate the data.
3. Matplotlib is also a python library which is used for Data Visualization.
4. Seaborn provides a high-level interface for drawing attractive and informative statistical graphics and it is closely integrated with pandas data structures in python.

In [2]:
```
# importing Data
df=pd.read_csv('C:\\Users\\honey\\Downloads\\Fraud.csv',encoding='unicode_escape')
```

In [3]:
```
df.isnull().any()
```

Out[3]:
```
step              False
type              False
amount            False
nameOrig          False
oldbalanceOrg     False
newbalanceOrig    False
nameDest          False
oldbalanceDest    False
newbalanceDest    False
isFraud           False
isFlaggedFraud    False
dtype: bool
```

In this data we do not have any null values.

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

Out[4]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbal
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 [5]:
```
df.rename(columns={'newbalanceOrig':'newbalanceOrg'},inplace=True)
df.drop(labels=['nameOrig','nameDest'],axis=1,inplace=True)
```

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

Out[6]:

	step	type	amount	oldbalanceOrg	newbalanceOrg	oldbalanceDest	newbalanceDest	isFraud	isFlaggedI
0	1	PAYMENT	9839.64	170136.0	160296.36	0.0	0.0	0	
1	1	PAYMENT	1864.28	21249.0	19384.72	0.0	0.0	0	
2	1	TRANSFER	181.00	181.0	0.00	0.0	0.0	1	
3	1	CASH_OUT	181.00	181.0	0.00	21182.0	0.0	1	
4	1	PAYMENT	11668.14	41554.0	29885.86	0.0	0.0	0	

In [8]:
```
df.shape
```

Out[8]:
```
(6362620, 9)
```

The provided data has the financial transaction data as well as the target variable isFraud

In [9]:
```
print('Minimum value of Amount, Old/New Balance of Origin/Destination:')
df[[ 'amount','oldbalanceOrg', 'newbalanceOrg', 'oldbalanceDest', 'newbalanceDest']].min()
```

```
Minimum value of Amount, Old/New Balance of Origin/Destination:
```

Out[9]:
```
amount            0.0
oldbalanceOrg     0.0
newbalanceOrg     0.0
oldbalanceDest    0.0
newbalanceDest    0.0
dtype: float64
```

In [10]:
```
print('Maximum value of Amount, Old/New Balance of Origin/Destination:')
df[[ 'amount','oldbalanceOrg', 'newbalanceOrg', 'oldbalanceDest', 'newbalanceDest']].max()
```

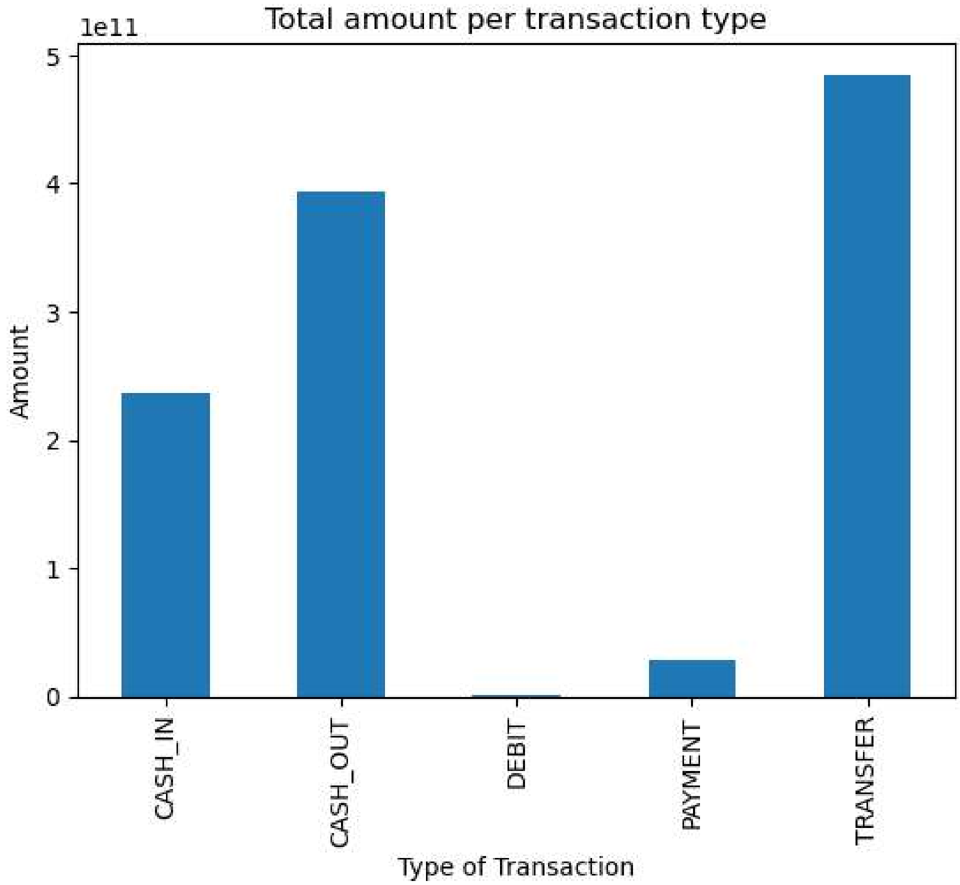
```
Maximum value of Amount, Old/New Balance of Origin/Destination:
```

Out[10]:
```
amount            9.244552e+07
oldbalanceOrg     5.958504e+07
newbalanceOrg     4.958504e+07
oldbalanceDest    3.560159e+08
newbalanceDest    3.561793e+08
dtype: float64
```

Data Analysis

Since there is no missing and garbage value, there is no need for data cleaning, but we still need to perform data analysis as data contaion huge variation of the value in different columns. Normalization will also imporve the overall accuracy of the machine learning model.

In [11]:
```
var = df.groupby('type').amount.sum()
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
var.plot(kind='bar')
ax1.set_title("Total amount per transaction type")
ax1.set_xlabel('Type of Transaction')
ax1.set_ylabel('Amount');
```

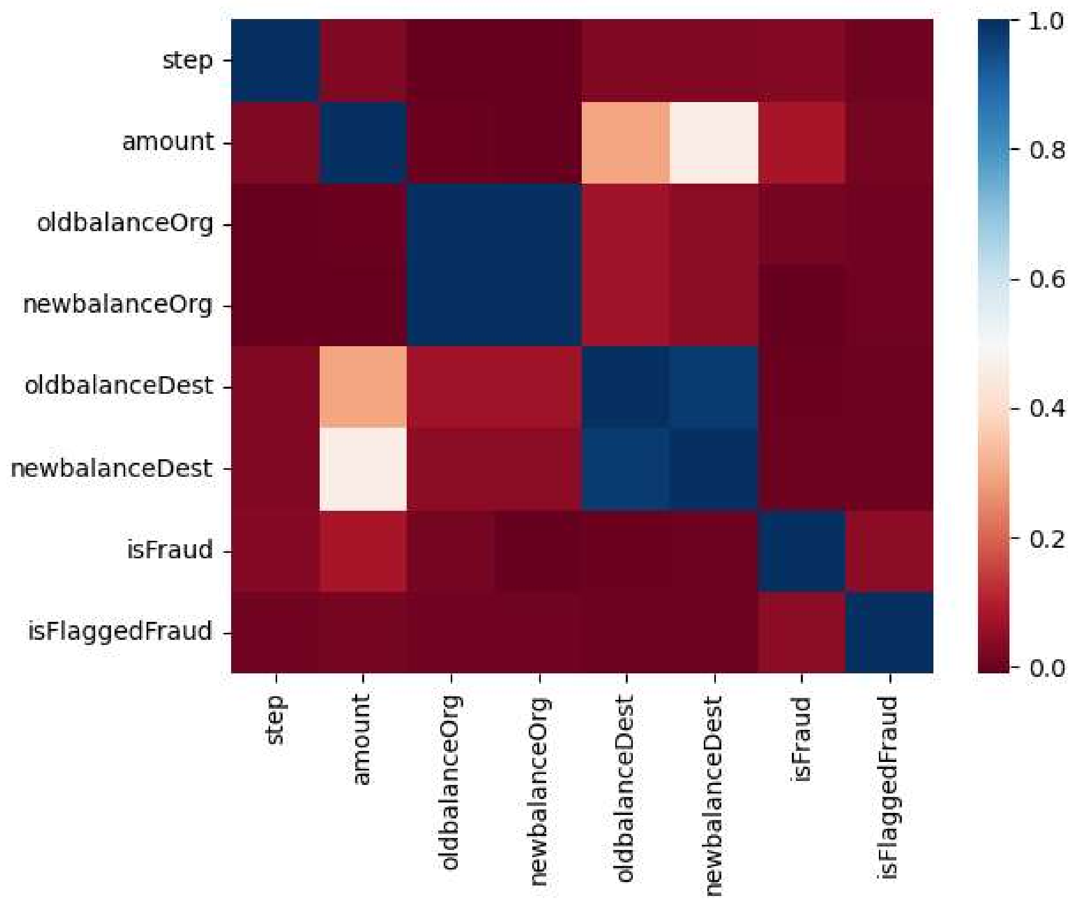


In [12]:
```
df.loc[df.isFraud == 1].type.unique()
```

Out[12]:
```
array(['TRANSFER', 'CASH_OUT'], dtype=object)
```

The graph above shows that TRANSFER and CASH_OUT are two most used mode of transaction and we can see that TRANSFER and CASH_OUT are also the only way in which fraud happen.

In [13]:
```
sns.heatmap(df.corr(),cmap='RdBu');
```



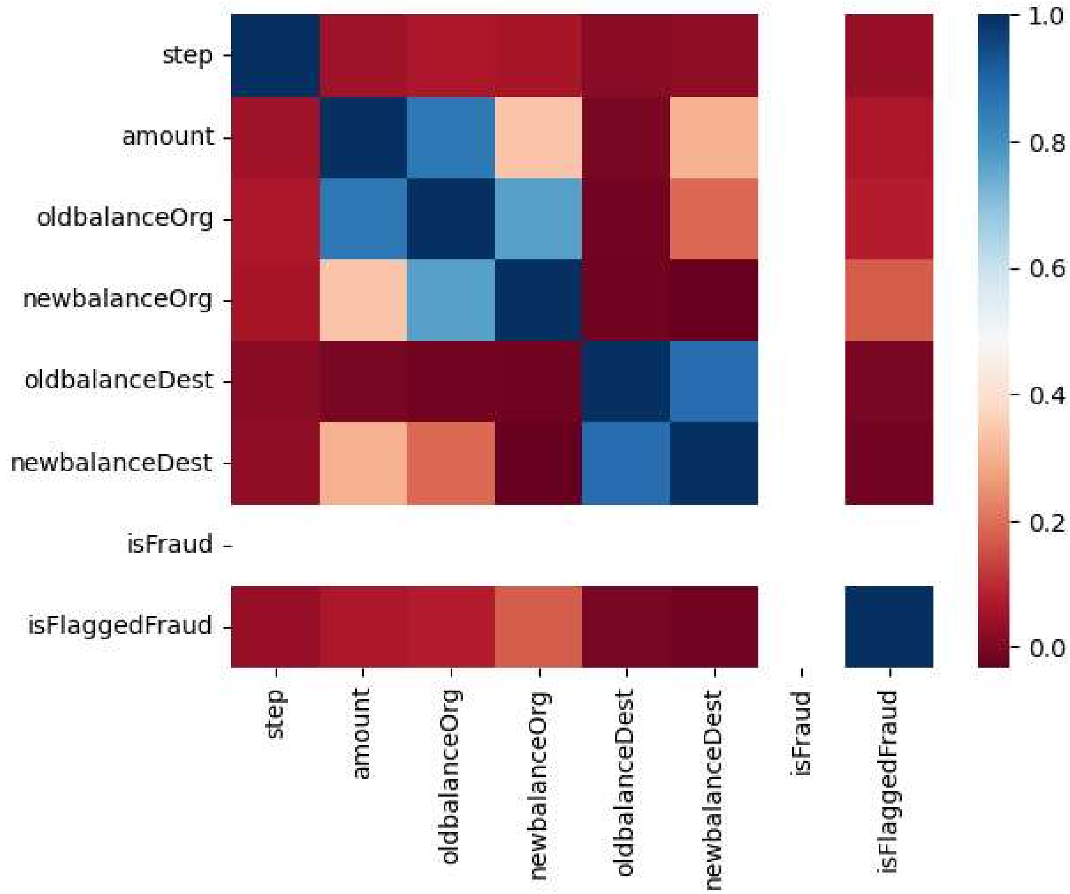
Things we can conclude from this heatmap:

OldbalanceOrg and NewbalanceOrg are highly correlated. OldbalanceDest and NewbalanceDest are highly correlated. Amount is correlated with isFraud(Target Variable). There is not much relation between the features, so we need to understand where the relationship between them depends on the type of transaction and amount. To do so, we need to see the heat map of fraud and nonfraud transactions differently.

In [14]:
```
fraud = df.loc[df.isFraud == 1]
nonfraud = df.loc[df.isFraud == 0]
```

In [15]:
```
fraudcount = fraud.isFraud.count()
nonfraudcount = nonfraud.isFraud.count()
```

In [16]:
```
sns.heatmap(fraud.corr(),cmap='RdBu',);
```



There are 2 flags which stand out to me and it's interesting to look onto: isFraud and isFlaggedFraud column. From the hypothesis, isFraud is the indicator which indicates the actual fraud transactions whereas isFlaggedFraud is what the system prevents the transaction due to some thresholds being triggered. From the above heatmap we can see that there is some relation between other columns and isFlaggedFraud thus there must be relation between isFraud.

In [17]:
```
print('The total number of fraud transaction is {}.'.format(df.isFraud.sum()))
print('The total number of fraud transaction which is marked as fraud {}.'.format(df.isFlagge
print('Ratio of fraud transaction vs non-fraud transaction is 1:{}.'.format(int(nonfraudcount/
```

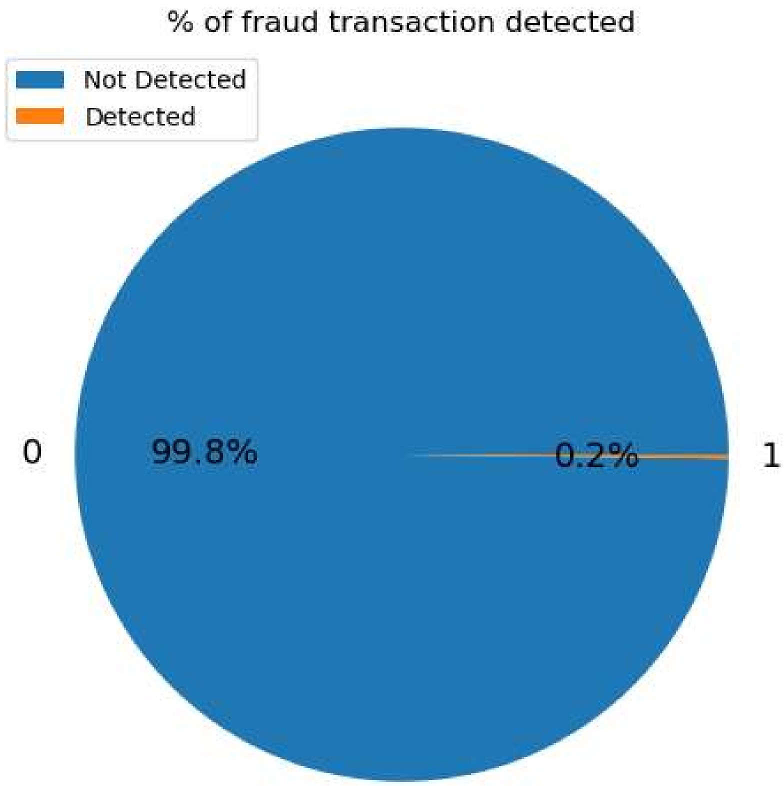
```
The total number of fraud transaction is 8213.
The total number of fraud transaction which is marked as fraud 16.
Ratio of fraud transaction vs non-fraud transaction is 1:773.
```

In [18]:
```
print('Thus in every 773 transaction there is 1 fraud transaction happening.')
print('Amount lost due to these fraud transaction is ${}.'.format(int(fraud.amount.sum())))
```

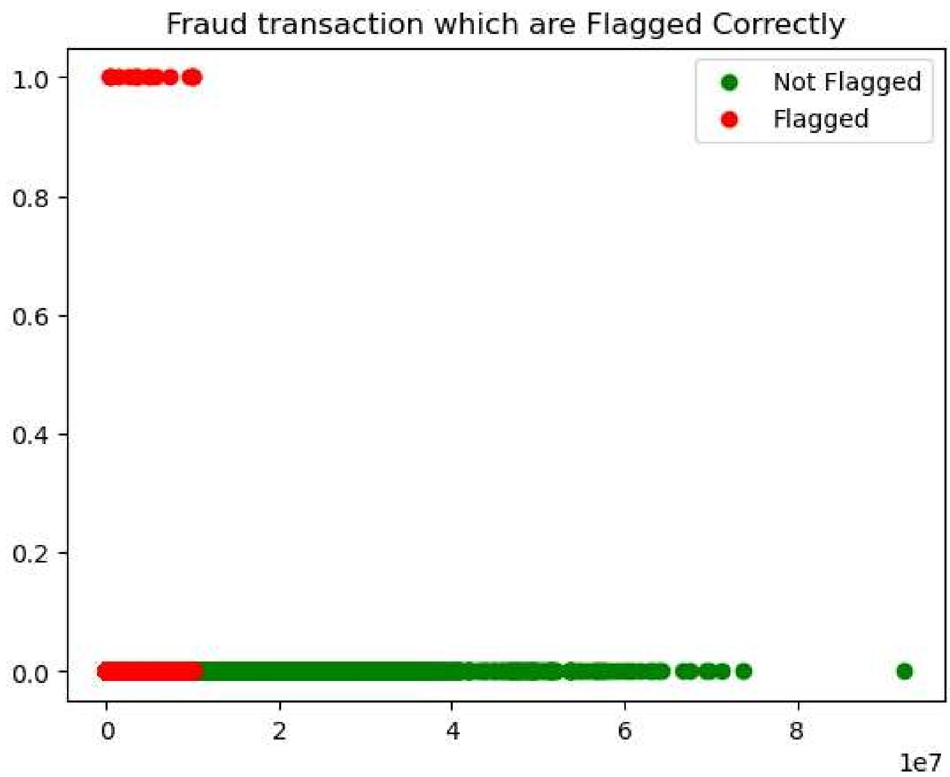
```
Thus in every 773 transaction there is 1 fraud transaction happening.
Amount lost due to these fraud transaction is $12056415427.
```

In [19]:
```
piedata = fraud.groupby(['isFlaggedFraud']).sum()
```

In [20]:
```
f, axes = plt.subplots(1,1, figsize=(6,6))
axes.set_title("% of fraud transaction detected")
piedata.plot(kind='pie',y='isFraud',ax=axes, fontsize=14,shadow=False,autopct='%1.1f%%');
axes.set_ylabel('');
plt.legend(loc='upper left',labels=['Not Detected','Detected'])
plt.show()
```



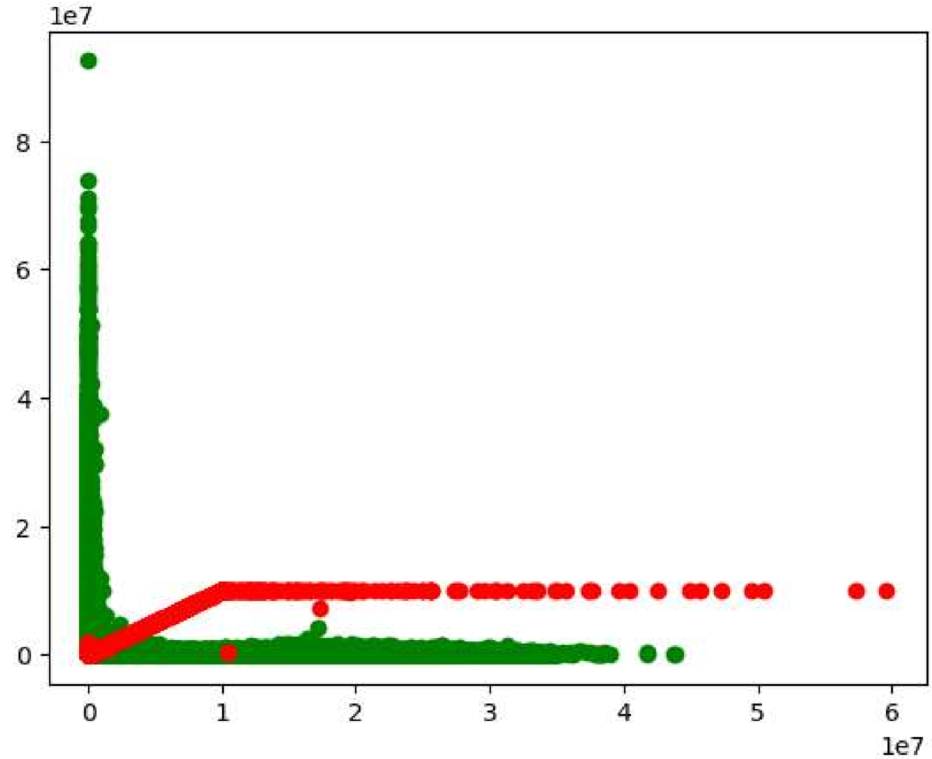
In [21]:
```
fig = plt.figure()
axes = fig.add_subplot(1,1,1)
axes.set_title("Fraud transaction which are Flagged Correctly")
axes.scatter(nonfraud['amount'],nonfraud['isFlaggedFraud'],c='g')
axes.scatter(fraud['amount'],fraud['isFlaggedFraud'],c='r')
plt.legend(loc='upper right',labels=['Not Flagged','Flagged'])
plt.show()
```



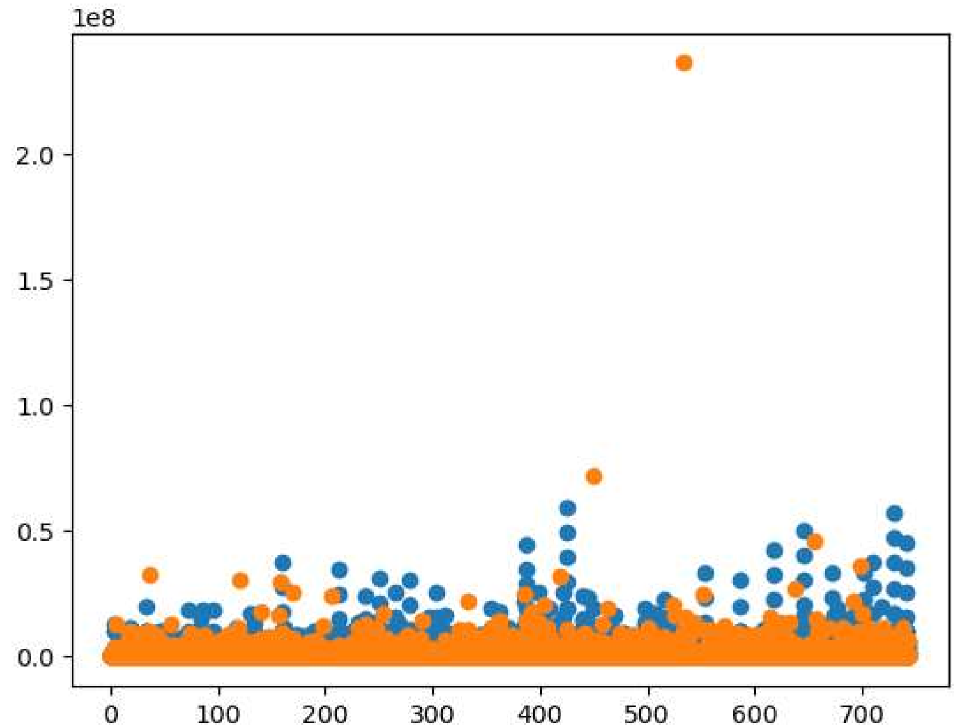
The plot above clearly shows the need for a system which can be fast and reliable to mark the transaction which is fraud. Since, the current system is letting fraud transaction able to pass through a system which is not labeling them as a fraud. Some data exploration can be helpful to check for the relation between features.

Data Exploration

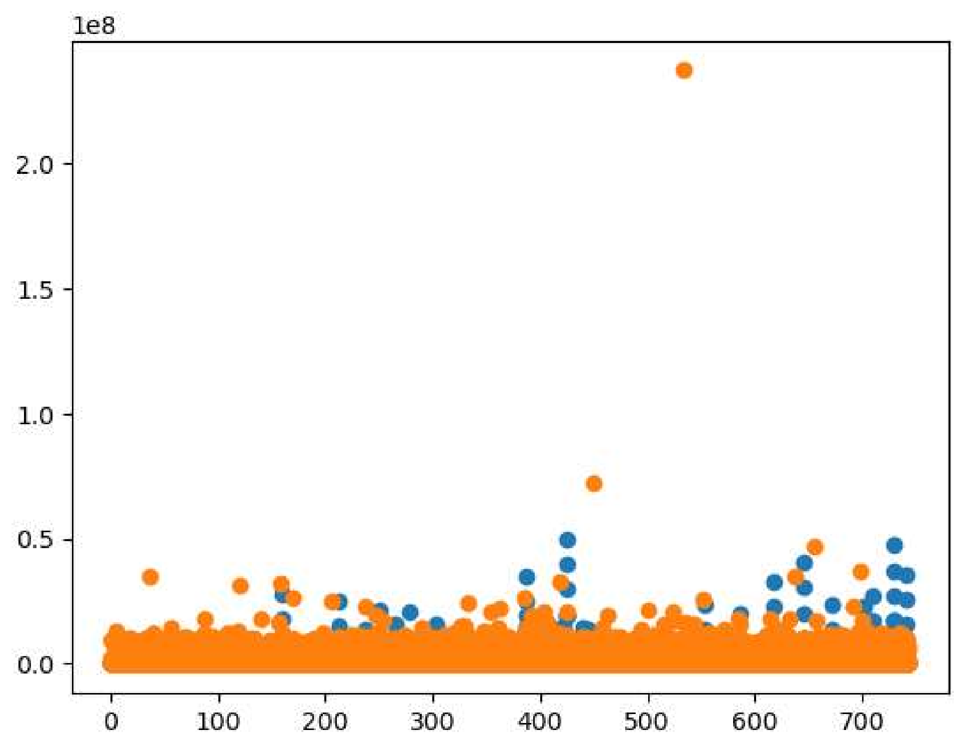
In [22]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(nonfraud['oldbalanceOrg'],nonfraud['amount'],c='g')
ax.scatter(fraud['oldbalanceOrg'],fraud['amount'],c='r')
plt.show()
```



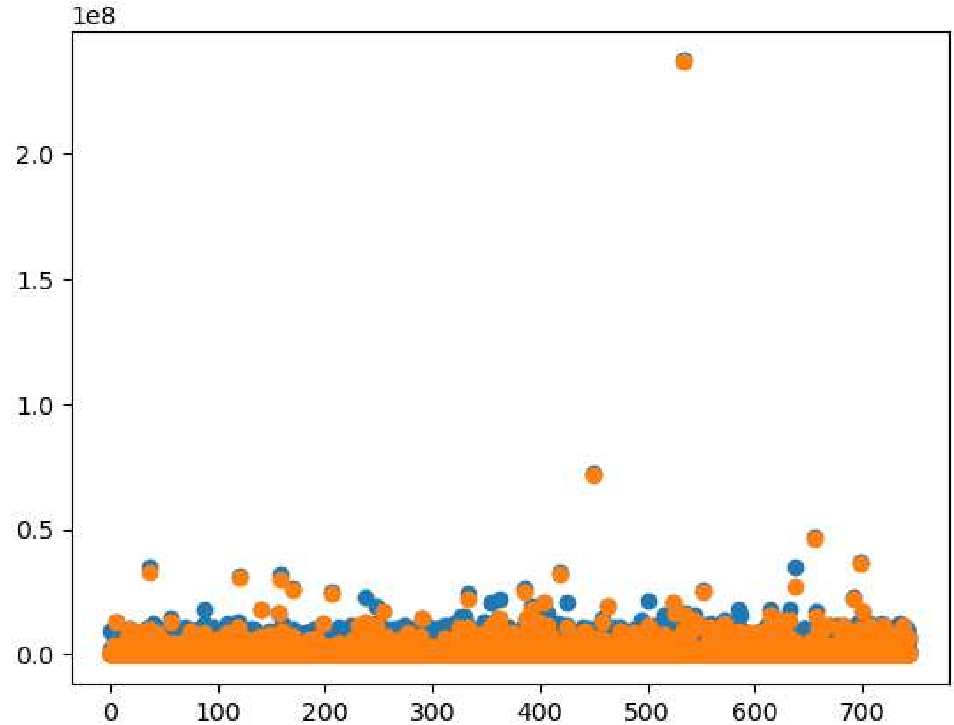
In [23]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(fraud['step'],fraud['oldbalanceOrg'])
ax.scatter(fraud['step'],fraud['oldbalanceDest'])
plt.show()
```



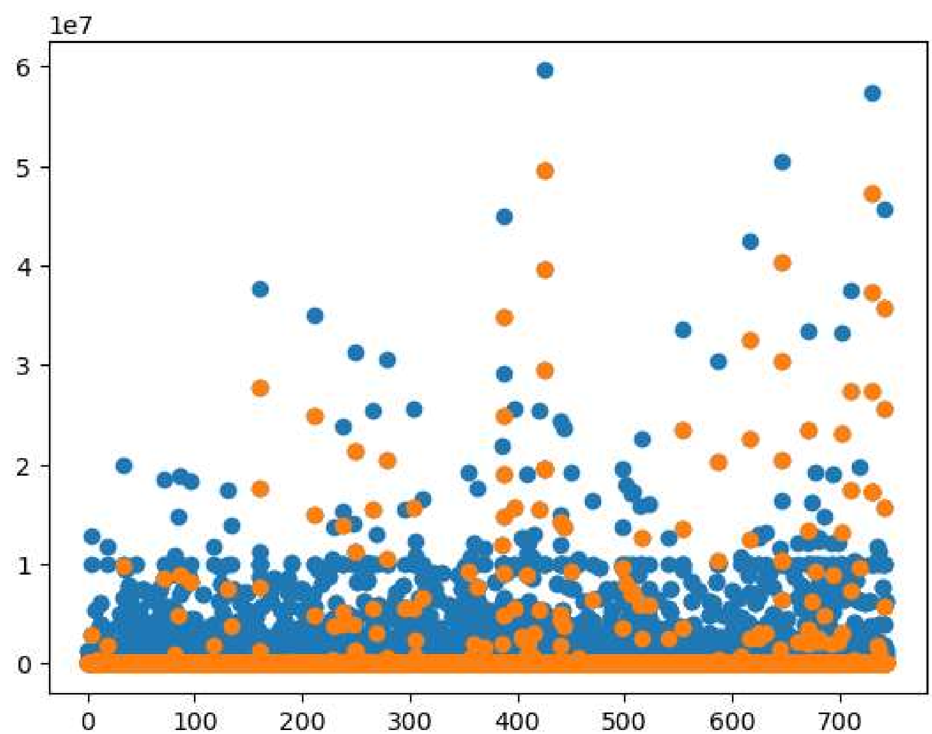
In [24]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(fraud['step'],fraud['newbalanceOrg'])
ax.scatter(fraud['step'],fraud['newbalanceDest'])
plt.show()
```



In [25]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(fraud['step'],fraud['newbalanceDest'])
ax.scatter(fraud['step'],fraud['oldbalanceDest'])
plt.show()
```



In [26]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(fraud['step'],fraud['oldbalanceOrg'])
ax.scatter(fraud['step'],fraud['newbalanceOrg'])
plt.show()
```



Data Cleaning

In [27]:
```
df = df.replace(to_replace={'PAYMENT':1,'TRANSFER':2,'CASH_OUT':3,
                                            'CASH_IN':4,'DEBIT':5,'No':0,'Yes':1})
```

In [28]:
```
df.head()
```

Out[28]:

	step	type	amount	oldbalanceOrg	newbalanceOrg	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	1	9839.64	170136.0	160296.36	0.0	0.0	0	0
1	1	1	1864.28	21249.0	19384.72	0.0	0.0	0	0
2	1	2	181.00	181.0	0.00	0.0	0.0	1	0
3	1	3	181.00	181.0	0.00	21182.0	0.0	1	0
4	1	1	11668.14	41554.0	29885.86	0.0	0.0	0	0

Normalization

In [29]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train_col_sacle = df[['step','type','amount','oldbalanceOrg','newbalanceOrg','oldbalanceDest',
train_scaler_col = scaler.fit_transform(train_col_sacle)
train_scaler_col = pd.DataFrame(train_scaler_col,columns=train_col_sacle.columns)
df['step']= train_scaler_col['step']
df['type']= train_scaler_col['type']
df['amount']= train_scaler_col['amount']
df['oldbalanceOrg']= train_scaler_col['oldbalanceOrg']
df['newbalanceOrg']= train_scaler_col['newbalanceOrg']
df['oldbalanceDest']= train_scaler_col['oldbalanceDest']
df['newbalanceDest']= train_scaler_col['newbalanceDest']
```

Machine Learning Model

In [36]:
```
#### VIF check
###
#########################
# VIF check
#########################
from statsmodels.stats.outliers_influence import variance_inflation_factor

tempMaxVIF = 10 # This VIF variable will be calculated at EVERY iteration in the while loop
maxVIF = 10
trainXCopy = trainX.copy()
counter = 1
highVIFColumnNames = []

while (tempMaxVIF >= maxVIF):

    print(counter)

    # Create an empty temporary df to store VIF values
    tempVIFDf = pd.DataFrame()

    # Calculate VIF using list comprehension
    tempVIFDf['VIF'] = [variance_inflation_factor(trainXCopy.values, i) for i in range(trainX(

    # Create a new column "Column_Name" to store the col names against the VIF values from lis
    tempVIFDf['Column_Name'] = trainXCopy.columns

    # Drop NA rows from the df - If there is some calculation error resulting in NAs
    tempVIFDf.dropna(inplace=True)

    # Sort the df based on VIF values, then pick the top most column name (which has the highe
    tempColumnName = tempVIFDf.sort_values(["VIF"], ascending = False).iloc[0,1]

    # Store the max VIF value in tempMaxVIF
    tempMaxVIF = tempVIFDf.sort_values(["VIF"], ascending = False).iloc[0,0]

    if (tempMaxVIF >= maxVIF): # This condition will ensure that columns having VIF lower than

        # Remove the highest VIF valued "Column" from trainXCopy. As the loop continues this s
        trainXCopy = trainXCopy.drop(tempColumnName, axis = 1)
        highVIFColumnNames.append(tempColumnName)
        print(tempColumnName)

    counter = counter + 1

highVIFColumnNames
```

```
1
newbalanceOrg
2
newbalanceDest
3
```

Out[36]: ['newbalanceOrg', 'newbalanceDest']

In [37]:
```
trainX = trainX.drop(highVIFColumnNames, axis = 1)
testX = testX.drop(highVIFColumnNames, axis = 1)
```

In [38]:
```
trainX.shape
testX.shape
```

Out[38]: (1908786, 6)

Model Building

In [56]:
```
X = df.drop(['isFraud'],axis=1)
y = df[['isFraud']]
```

In [57]:
```
from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.2, random_state = 121
```

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=15)
```

In [59]:
```
if True:
    probabilities = clf.fit(train_X, train_y.values.ravel()).predict(test_X)
```

In [65]:
```
from sklearn.metrics import average_precision_score
if True:
    print(average_precision_score(test_y,probabilities))
    print('Average precision score is 76')
```

```
0.7601969587396279
Average precision score is 76
```

-Here we used Machine learning for the detection of fraud transaction. our model give us 76 accuracy which is fine good. if we want to increase our accuracy so use HyperParameter Tunning method.
-Predictive models produce good precision score and are capable of detection of fraud transaction.

Hyperparameter tunning

Here is used HyperParameter Tuning . in simple word we can say that,this is used to increase or accuracy of our model . in this we also use gridsearch cross validation which help to make or provide best result.

In [70]:
```
from sklearn.model_selection import GridSearchCV
```

In [71]:
```
n_estimators_list=[20,30,50]# No of Trees
max_features_list=[5,9,11]
min_sample_leaf_list=[100,300,400,500,600,700]
```

In [72]:
```
myparamgrid={'n_estimators':n_estimators_list,
             'max_features':max_features_list,
             'min_samples_leaf':min_sample_leaf_list}
```

In [*]:
```
gridSearch=GridSearchCV(estimator=RandomForestClassifier(random_state=2410),
                        param_grid=myparamgrid,scoring='accuracy',cv=5).fit(train_X,train_y)
```

In [ ]:
```
gridSearch.cv_results_
```

In [ ]:
```
gridSearchDf=pd.DataFrame.from_dict(gridSearch.cv_results_)
```

Final Model

In [ ]:
```
FMRF=RandomForestClassifier(random_state=2410,n_estimators=100,
                            max_features=3,min_samples_leaf=75).fit(train_X,train_y)
```

Prediction

In [ ]:
```
final_predict=FMRF.predict(test_X)
```

Evalution

In [ ]:
```
print(classification_report(test_y,final_predict))
```