```
#importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
<frozen importlib._bootstrap>:228: RuntimeWarning: scipy._lib.messagestream.MessageStream size changed, may indicate binary incompat
```

```
#loading dataset
df = pd.read_csv('Fraud.csv')
```

```
#first five rows of the dataset
df.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDes
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M197978715
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M204428222
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C55326406
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C3899701
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M12307017

```
#shape of the dataset
df.shape
```

```
(6362620, 11)
```

```
#information about the columns and their datatypes
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
```

```
df.columns
```

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

```
#checking for any duplicate data
df[df.duplicated()]
```

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDe

```
#checking for any null or misssing values
df.isnull().sum()
```

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

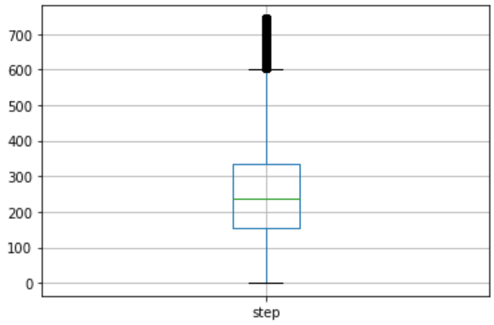
```
isFlaggedFraud    0
dtype: int64
```

```
#information regarding numerical columns
df.describe()
```

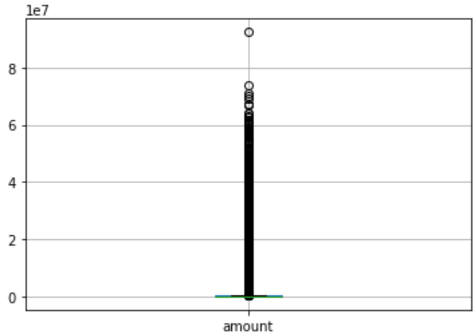
	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newb
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3

```
#checking for outliers using box plot
for col in df.columns:
    if df[col].dtype == 'float64' or df[col].dtype == 'int64':
        print(col)
        df.boxplot(column = col)
        plt.show()
    else:
        pass
```

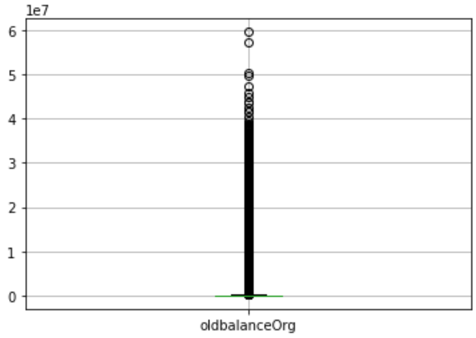
step



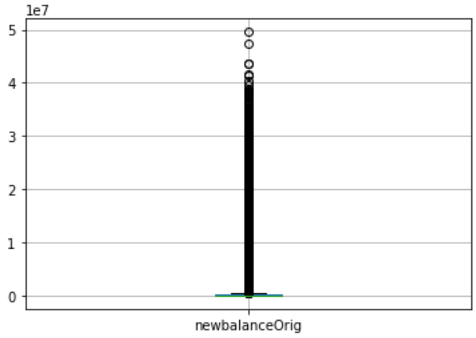
amount



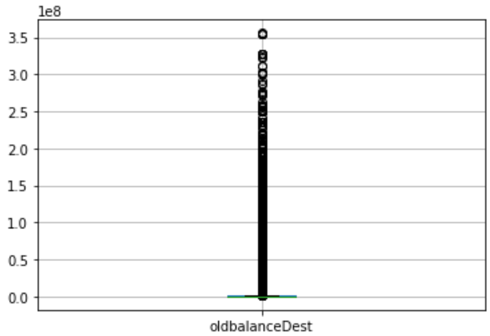
oldbalanceOrg



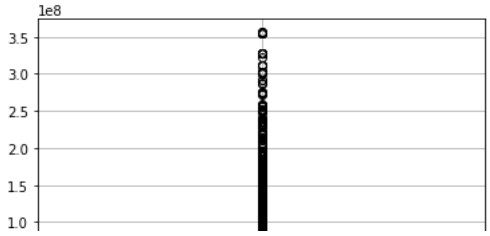
newbalanceOrig

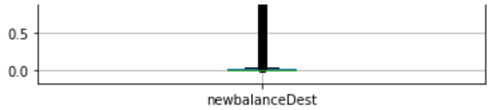


oldbalanceDest

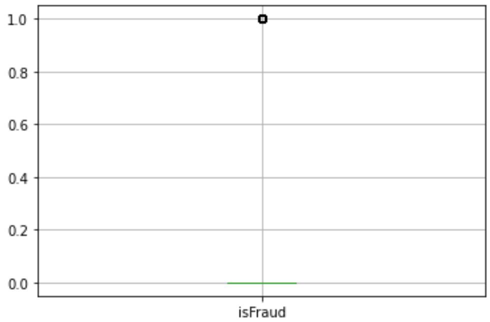


newbalanceDest





isFraud



isFlaggedFraud

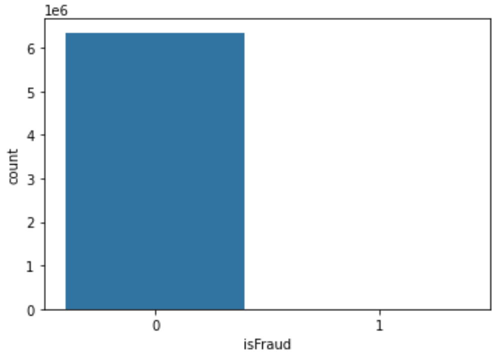

```
#checking for outliers
numerical_columns = ['step','amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']

# Initialize a dictionary to store the number of outliers for each column
outliers_count = {}

for col in numerical_columns:
    # Calculate the IQR for each numerical column
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    # Identify potential outliers using the IQR method
    outliers = ((df[col] < (Q1 - 1.5 * IQR)) |
                (df[col] > (Q3 + 1.5 * IQR)))

    # Count the number of outliers for the current column
    num_outliers = outliers.sum()

    # Store the count in the dictionary
    outliers_count[col] = num_outliers

# Display the number of outliers for each column
for col, count in outliers_count.items():
    print(f"Number of outliers in column '{col}': {count}")
```

```
Number of outliers in column 'step': 102688
Number of outliers in column 'amount': 338078
Number of outliers in column 'oldbalanceOrg': 1112507
Number of outliers in column 'newbalanceOrig': 1053391
Number of outliers in column 'oldbalanceDest': 786135
Number of outliers in column 'newbalanceDest': 738527
```

```
# I believe the reasons for these columns having outliers are legitimate and keeping them as it may provide any hidden patterns.
```

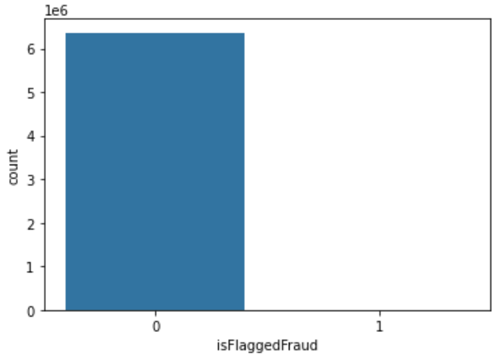
```
sns.countplot(df["isFraud"])  # Zero Establish mean no fraud
df.isFraud.value_counts()      # One Establish mean fraud
```

```
C:\Users\91844\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:
  warnings.warn(
0    6354407
1       8213
Name: isFraud, dtype: int64
```




```
sns.countplot(df["isFlaggedFraud"]) # Zero Establish mean no fraud
df.isFlaggedFraud.value_counts()    # One Establish mean fraud
```

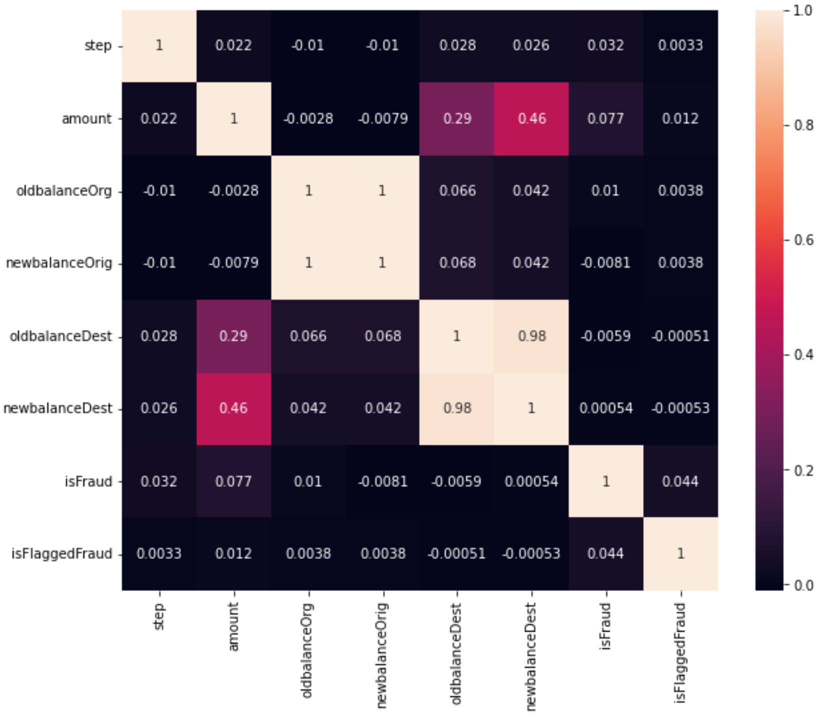
```
C:\Users\91844\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:
  warnings.warn(
0    6362604
1         16
Name: isFlaggedFraud, dtype: int64
```




```
#checking for multicollinearity
df.corr()
```

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

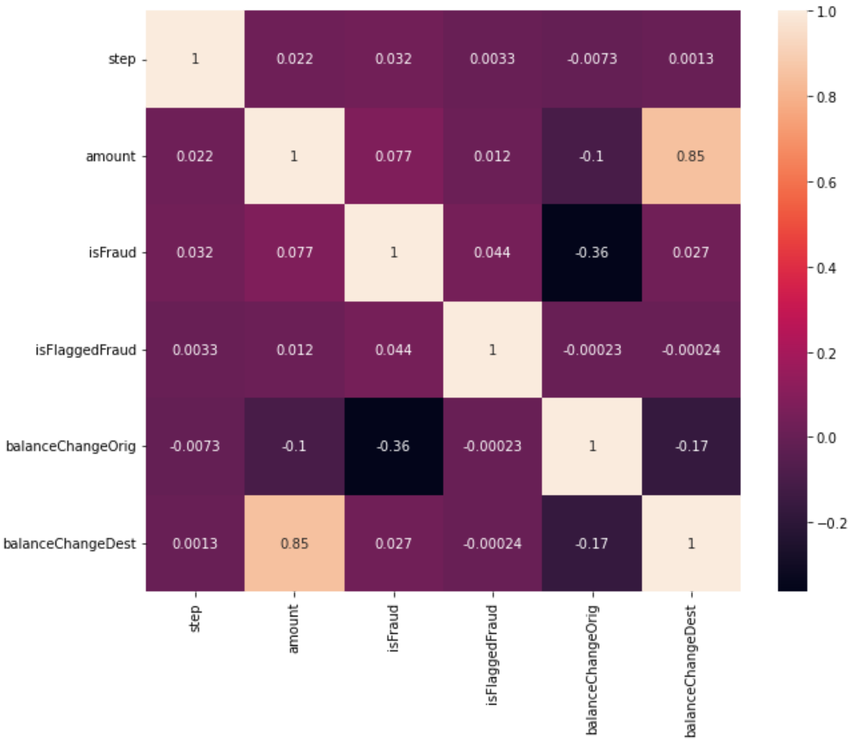
```
correlation_matrix = df.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True)
plt.show()
```




```
#We can see from the above heatmap that oldbalanceOrig and newbalanceOrig have collinearity of 1 we can't define the individual effects
```

```
# Create new columns for balance changes
df['balanceChangeOrig'] = df['newbalanceOrig'] - df['oldbalanceOrg']
df['balanceChangeDest'] = df['newbalanceDest'] - df['oldbalanceDest']

# Drop the original balance columns
df.drop(['oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest'], axis=1, inplace=True)
```

```
correlation_matrix = df.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True)
plt.show()
```




```
df.type.value_counts()
```

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

```
#Data encoding converting categorical columns to numerical
encoded_types = pd.get_dummies(df['type'], prefix='type')
df = pd.concat([df, encoded_types], axis=1)

# Drop the original 'type' column
df.drop(['type'], axis=1, inplace=True)
```

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
df['nameOrig'] = labelencoder.fit_transform(df['nameOrig'])
df['nameDest'] = labelencoder.fit_transform(df['nameDest'])
```

```
from sklearn.model_selection import train_test_split

# Splitting the data into features (X) and target (y)
X = df.drop(['isFraud'], axis=1)  # Features
y = df['isFraud']  # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
```

```
from sklearn.preprocessing import StandardScaler
# Feature Scaling: Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
# Initialize and train the Logistic Regression model
model = LogisticRegression(max_iter=500)
model.fit(X_train_scaled, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=500)
```

```
model.score(X_train_scaled,y_train)
```

```
0.999226733641173
```

```
model.score(X_test_scaled,y_test)
```

```
0.9992518805146308
```

```
from sklearn.tree import DecisionTreeClassifier
model_dt = DecisionTreeClassifier(random_state = 2)
model_dt.fit(X_train_scaled,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=2)
```

```
model_dt.score(X_test_scaled,y_test)
```

```
0.9994059051145597
```

```
feature_importances = model_dt.feature_importances_

# Create a DataFrame to display feature importances
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
})

# Sort features by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Print the feature importance DataFrame
print(feature_importance_df)


# Our fraud detection model is designed to identify potentially fraudulent transactions within a financial company's system. The model u
# The key factors in predicting fraudelent transactions are balanceChangeOrig, balanceChangeDest, step, type_TRANSFER, nameOrig, nameDes
# Balance Changes: Large and unusual balance changes in the accounts of both the transaction initiator and the recipient can indicate fr
# 'TRANSFER' and 'CASH_OUT' transactions because they involve the movement of funds. 'nameOrig' and 'nameDest' are relevant since speci
# 'isFlaggedFraud' flag indicates that the transaction was flagged as potentially fraudulent. 'type_PAYMENT' feature's lower importance
```

```
1              amount    0.040827
```

```
#What are the key factors that predict fraudulent customer?
```

```
10       type_PAYMENT    0.000155
```

```
#Unusual Transaction Frequency:
Fraudulent customers may exhibit an unusually high or low transaction frequency compared to regular customers.

#Unusual Transaction Amounts:
Large or irregular transaction amounts that deviate significantly from a customer's typical spending behavior can be indicative of frauc

#Multiple Account Access:
Frequent access to multiple accounts or a sudden change in login patterns may suggest fraudulent activity.

#Device Anomalies:
Suspicious logins or transactions from new or unfamiliar devices may indicate potential fraud.

#Abnormal Transaction Times:
Transactions occurring at unusual times, especially during non-business hours or holidays, may be considered suspicious.
```