Online Payment Fraud Detection

Data Source: /https://www.kaggle.com/datasets/jainilcoder/online-payment-fraud-detection/data (https://www.kaggle.com/datasets/jainilcoder/online-payment-fraud-detection/data)

Importing Packages

In [1]:
```
import numpy as np
import pandas as pd
import datetime as dt
import warnings
import missingno as msno
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LogisticRegression
```

Ignores all warning messages

In [2]:
```
warnings.filterwarnings("ignore")
```

Reading the csv file

In [3]:
```
df = pd.read_csv("onlinefraud.csv")
# Displaying top 5 rows
df.head()
```

Out[3]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalai
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	181.00	C840083671	181.0	0.00	C38997010	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	

In [4]:
```
# Dispalying rows and columns
df.shape
```

Out[4]:
```
(6362620, 11)
```

In [5]:
```
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 [6]:
```
df.columns
```

Out[6]:
```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
       'isFlaggedFraud'],
      dtype='object')
```

In [7]:
```
df.head().T
```

Out[7]:

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

Data Cleaning

In [8]:
```
# Displaying datatypes
df.dtypes
```

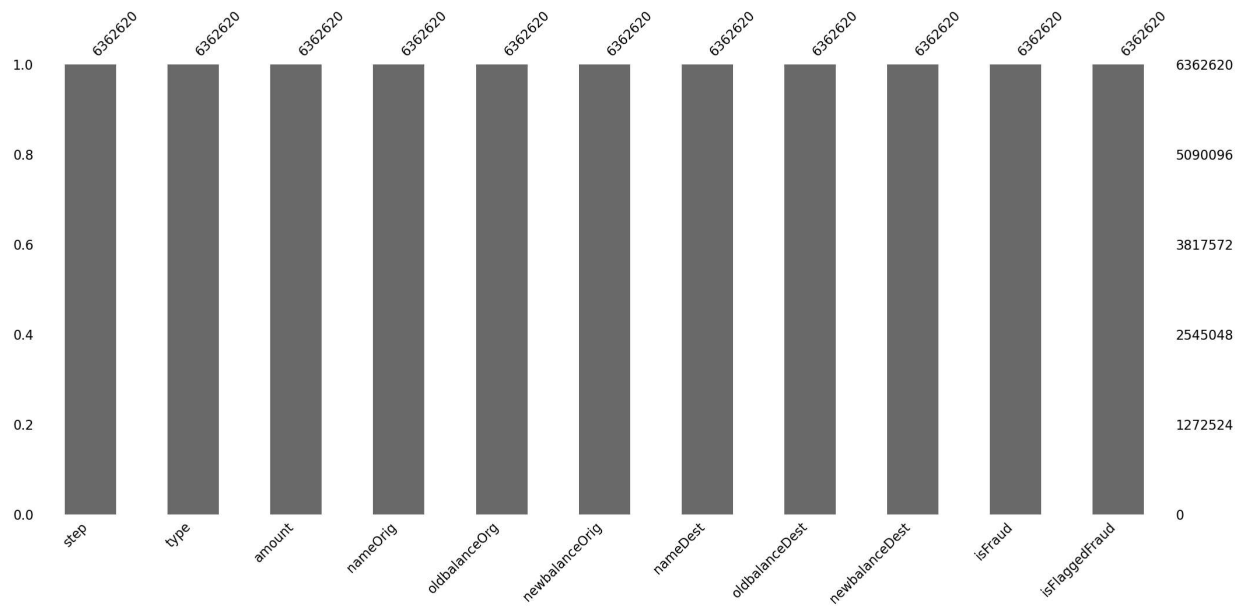
Out[8]:
```
step                int64
type               object
amount            float64
nameOrig           object
oldbalanceOrg     float64
newbalanceOrig    float64
nameDest           object
oldbalanceDest    float64
newbalanceDest    float64
isFraud             int64
isFlaggedFraud      int64
dtype: object
```

In [9]:
```
# Converting datatypes from objects
df = df.convert_dtypes()
df.dtypes
```

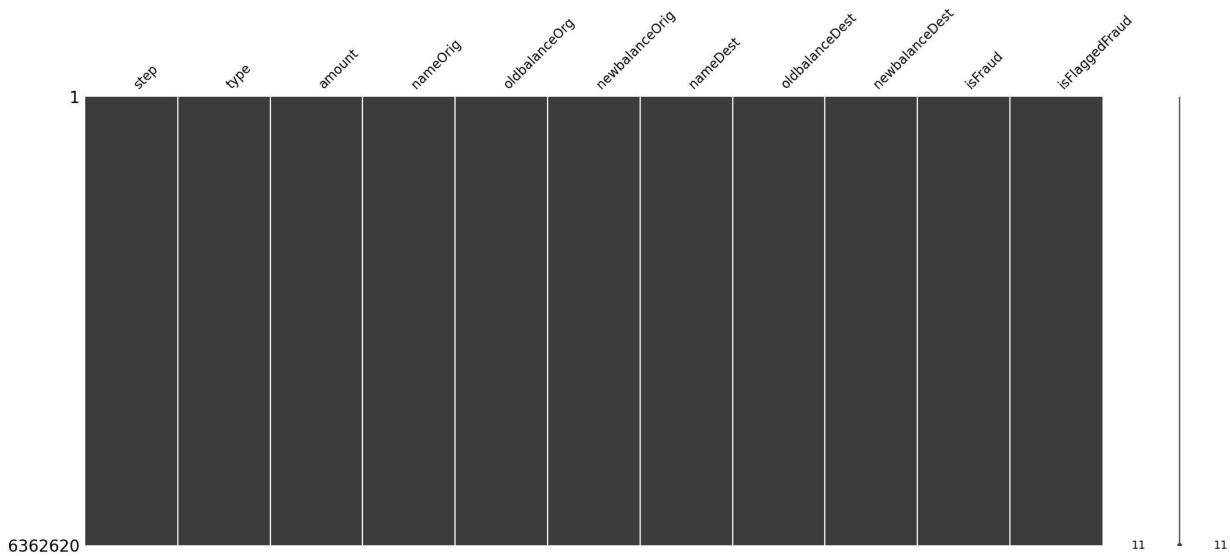
Out[9]:
```
step                       Int64
type              string[python]
amount                   Float64
nameOrig          string[python]
oldbalanceOrg            Float64
newbalanceOrig           Float64
nameDest          string[python]
oldbalanceDest           Float64
newbalanceDest           Float64
isFraud                    Int64
isFlaggedFraud             Int64
dtype: object
```

Analysing missing values

In [10]:
```
# Displaying missing values
msno.bar(df)
plt.show()
```



In [11]:
```
# Displaying missing values
msno.matrix(df)
plt.show()
```



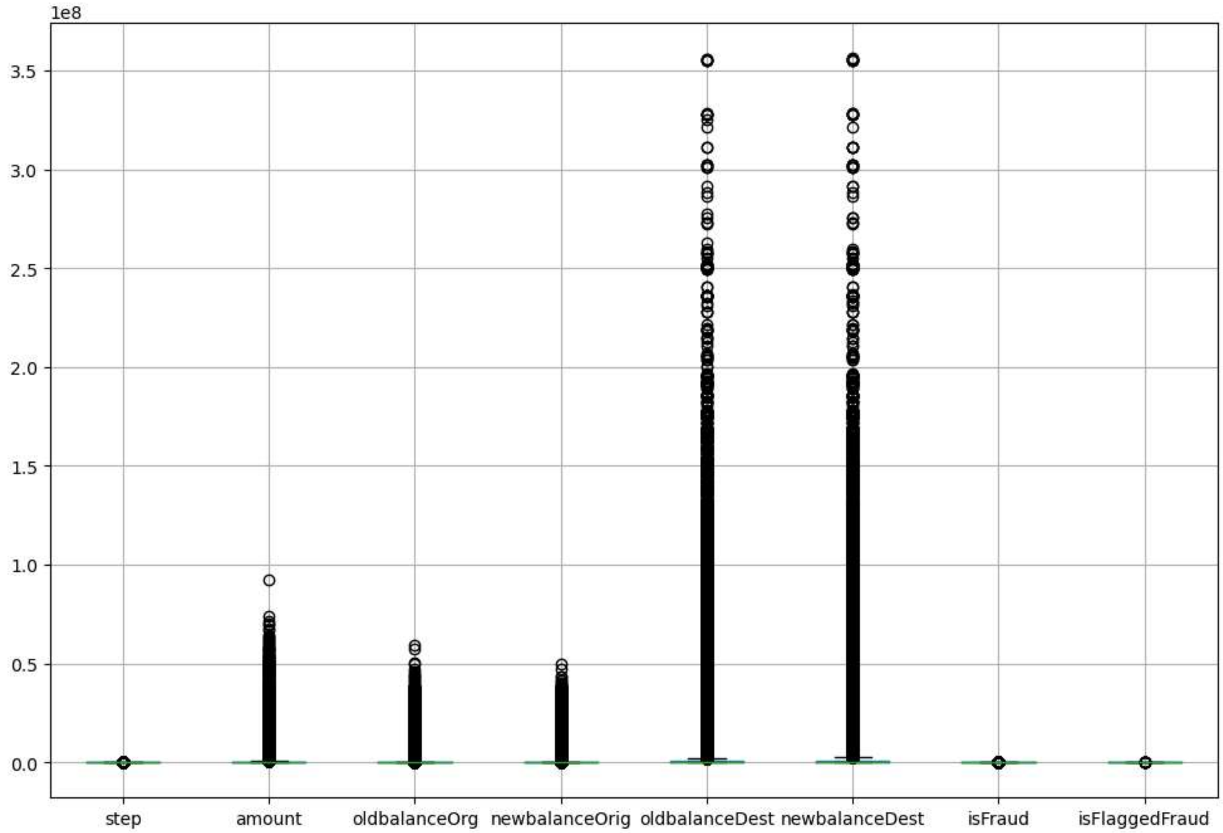
In [12]:
```
df.isnull().sum()
```

Out[12]:
```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

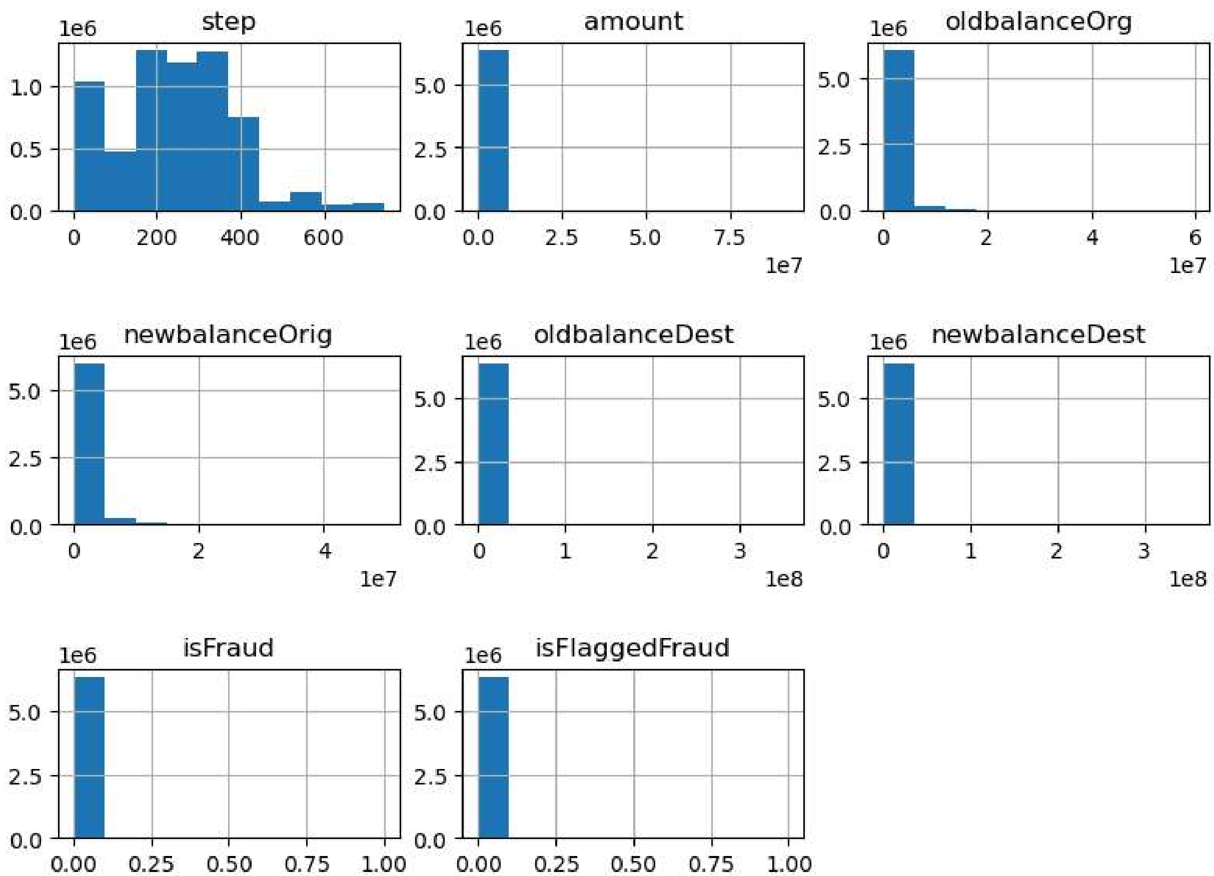
Checking for outliers

In [13]:
```
plt.figure(figsize=(12,8))
df.boxplot()
```

Out[13]: <Axes: >



In [14]:
```
ig, ax = plt.subplots(1, 1, figsize=(8, 6))
df.hist(ax=ax)
plt.tight_layout()
plt.show()
```



Data profile report

In [15]:
```
from ydata_profiling import ProfileReport
profile = ProfileReport(df, title="Profiling Report")
profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	11
Number of observations	6362620
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	582.5 MiB
Average record size in memory	96.0 B

Variable types

Numeric	6
Categorical	3
Text	2

Alerts

`amount` is highly overall correlated with `oldbalanceDest` and 1 other fields (oldbalanceDest, newbalanceDest)	**High correlation**

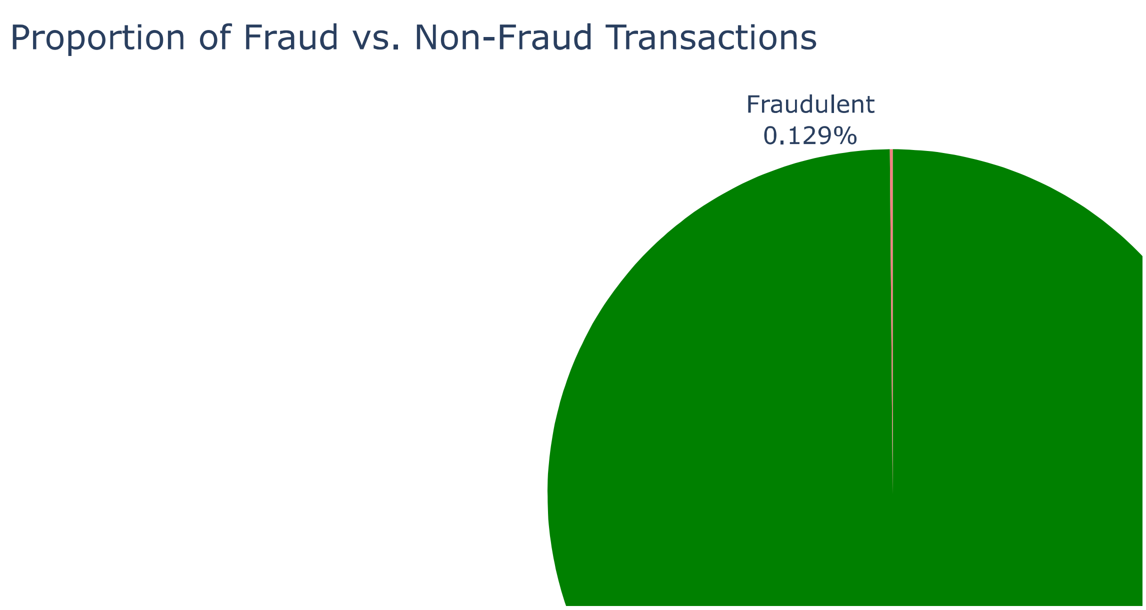
Out[15]:

Exploratory Data Analysis

In [16]:
```
# Displaying the number of Transactions using bar plot
fig = px.histogram(df, x='isFraud', color='isFraud',
                   title='Count Plot of Fraud Transactions',
                   labels={'isFraud': 'Is Fraud'},
                   text_auto=True,
                   color_discrete_sequence=px.colors.sequential.PuBu)
fig.update_layout(
    yaxis_title='Number of Transactions',
    xaxis_title='Is Fraud',
    bargap=0.2,
)
fig.show()
```

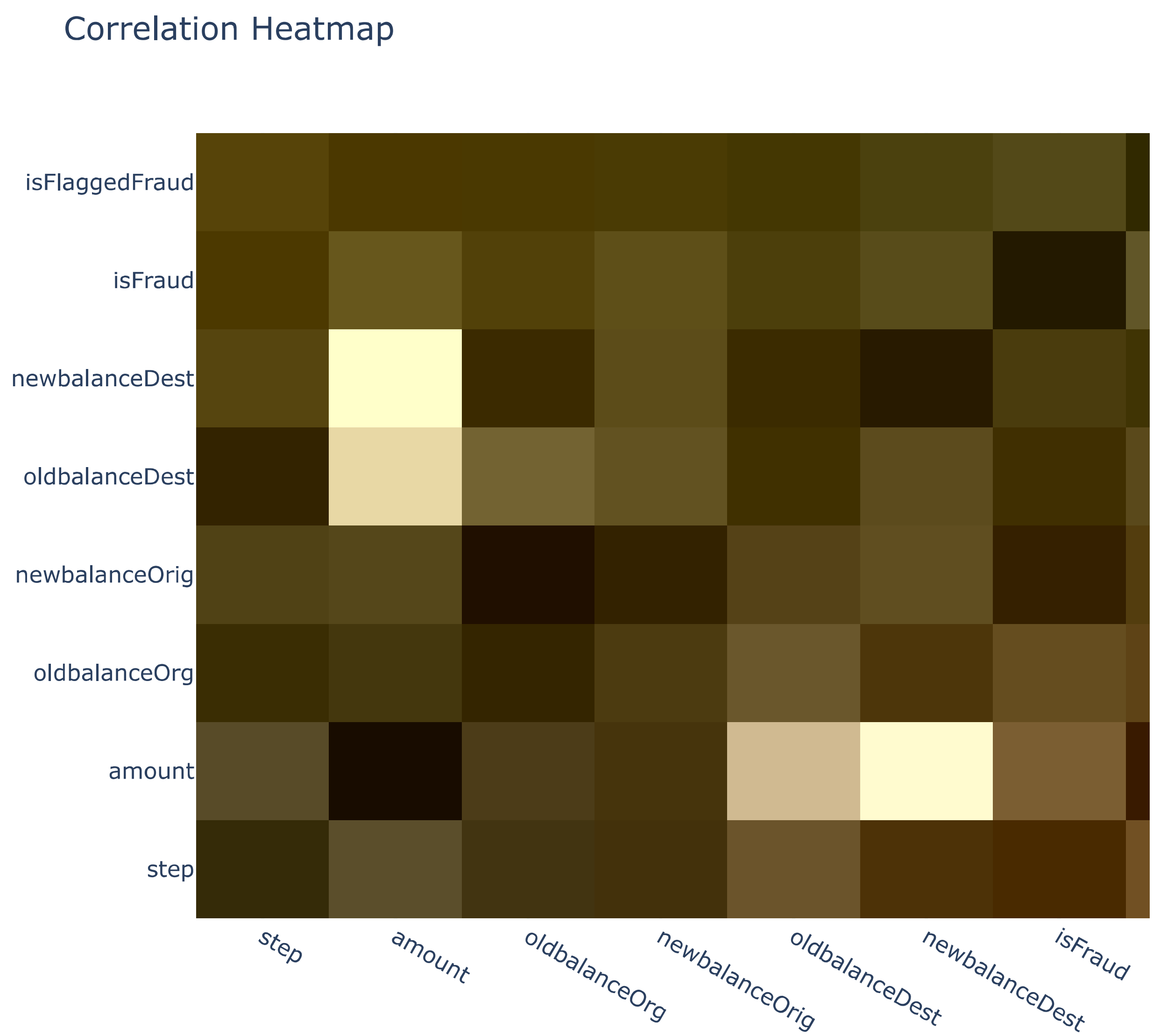
There are very few fraud identified transactions. There is high chances of imbalance class so need to balance the classes using oversampling or undersampling.

In [17]:
```
# Displaying the number of Transactions using pie plot
fraud_counts = df['isFraud'].value_counts()
fraud_df = fraud_counts.reset_index()
fraud_df.columns = ['isFraud', 'Counts']

# Map the 'isFraud' numerical values to more descriptive labels
fraud_df['Type'] = fraud_df['isFraud'].map({0: 'Non-Fraudulent', 1: 'Fraudulent'}

# Now, plot the pie chart using Plotly Express
import plotly.express as px

fig = px.pie(fraud_df, names='Type', values='Counts',
             title='Proportion of Fraud vs. Non-Fraud Transactions',
             color='Type', color_discrete_sequence=['green', 'lightcoral'])

fig.update_traces(textinfo='percent+label')
fig.show()
```



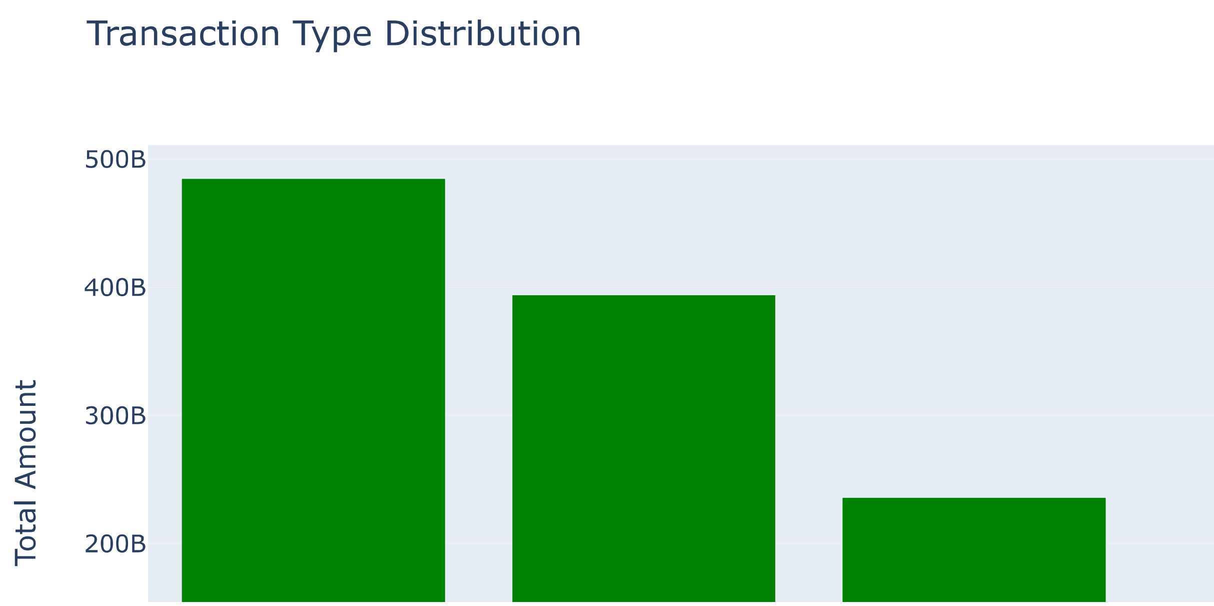
There are very few fraud identified transactions. There is high chances of imbalance class so need to balance the classes using oversampling or undersampling.

In [18]:
```
# Displaying the correlation Heatmap
numeric_df = df.select_dtypes(include=[np.number])
# Calculate the correlation matrix on numeric data only
correlation_matrix = numeric_df.corr()
fig = go.Figure(data=go.Heatmap(
    z=correlation_matrix.values,  # Correlation values
    x=correlation_matrix.columns,  # Feature names for x-axis
    y=correlation_matrix.index,  # Feature names for y-axis
    colorscale='BrBG',  # Valid colorscale for correlation
    colorbar=dict(title='Correlation'),
))

# Update the layout
fig.update_layout(
    title='Correlation Heatmap',
    xaxis=dict(tickmode='linear'),
    yaxis=dict(tickmode='linear'),
    width=800,
    height=600,
)

# Show the plot
fig.show()
```



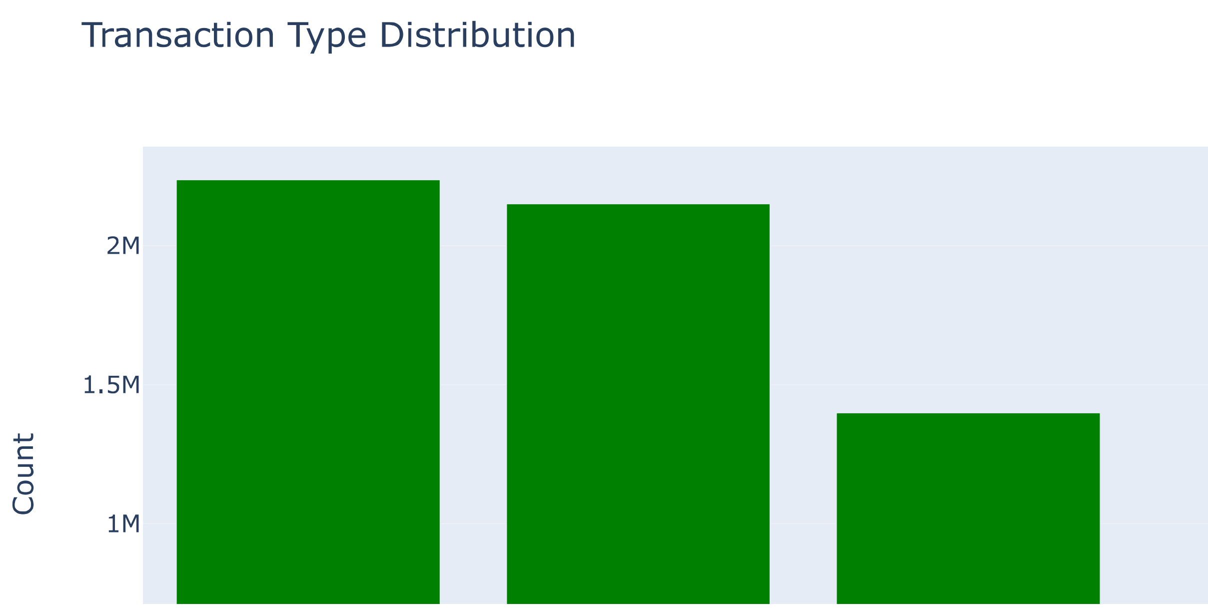
There is a strong corelation between newbalanceOrg and oldbalanceOrg

In [19]:
```
import plotly.express as px

grouped_df = df.groupby('type')['amount'].sum().reset_index()
sorted_grouped_df = grouped_df.sort_values('amount', ascending=False)

# Create a bar chart using Plotly Express, now with the data sorted
fig = px.bar(sorted_grouped_df, x='type', y='amount',
             labels={'type': 'Transaction Type', 'amount': 'Total Amount'},
             title='Transaction Type Distribution',
             color_discrete_sequence=['green'])  # Sets the bars to green

# Customize the chart
fig.update_layout(xaxis_title='Transaction Type',
                  yaxis_title='Total Amount',
                  legend_title='Transaction Type',
                  xaxis=dict(tickangle=45))  # Rotate the x-axis labels for bette

# Show the plot
fig.show()
```



'Transfer' type of transaction has maximum amount of amount processed. Least amount of transaction happend on 'Debit'.

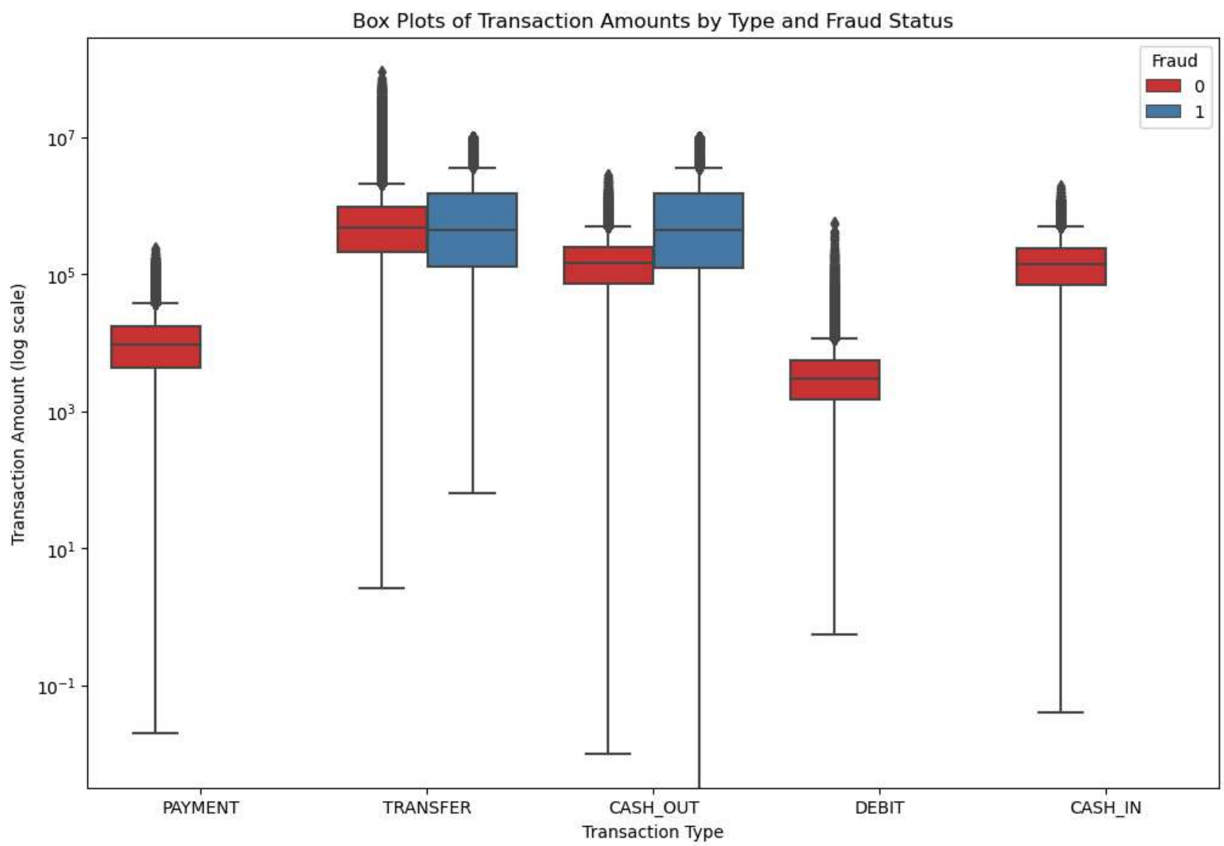
In [20]:
```
import pandas as pd
import plotly.express as px

transaction_type_counts = df['type'].value_counts()

# Convert the Series to a DataFrame for Plotly
transaction_type_counts_df = transaction_type_counts.reset_index()
transaction_type_counts_df.columns = ['Transaction Type', 'Count']

# Create a bar chart using Plotly Express
fig = px.bar(transaction_type_counts_df, x='Transaction Type', y='Count',
             title='Transaction Type Distribution',
             labels={'Count': 'Count', 'Transaction Type': 'Transaction Type'},
             color_discrete_sequence=['green'])  # Sets the bar color

# Customize the chart
fig.update_layout(xaxis_title='Transaction Type',
                  yaxis_title='Count',
                  xaxis=dict(tickangle=45))  # Rotate the x-axis labels for bette

# Show the plot
fig.show()
```



'Cash_out' type of transaction has maximum count of amount processed. Least number of transaction happend on 'Debit'.

Analysing which of Transaction has Fraud transactions

In [21]:
```
plt.figure(figsize=(12, 8))
sns.boxplot(x='type', y='amount', data=df, hue='isFraud', palette='Set1')
plt.yscale('log')
plt.title('Box Plots of Transaction Amounts by Type and Fraud Status')
plt.xlabel('Transaction Type')
plt.ylabel('Transaction Amount (log scale)')
plt.legend(title='Fraud', loc='upper right')
plt.show()
```



There are five types of transactions named Payment, Transfer, Cash_out, Debit and Cash_in. In this only 'Transfer' and 'Cash_out' have fraud transactions.

In [22]:
```
Result = pd.crosstab(index=df.type,columns=df.isFraud)
Result
```

Out[22]:

isFraud	0	1
type		
CASH_IN	1399284	0
CASH_OUT	2233384	4116
DEBIT	41432	0
PAYMENT	2151495	0
TRANSFER	528812	4097

In [23]:
```
transfer_total = 528812+4097
transfer_fraud = 4097/(transfer_total) * 100
transfer_fraud
```

Out[23]: 0.7687991758442811

In [24]:
```
cashout_total=2233384+4116
cashout_fraud= 4116/(cashout_total) * 100
cashout_fraud
```

Out[24]: 0.18395530726256984

> 76% of the fraud transactions happened in 'Transfer' and 18% of the fraud transactions happened in 'Cash_out'.

Calculating the % of Fraud transactions

In [25]:
```
df.isFlaggedFraud.value_counts()
```

Out[25]:
```
isFlaggedFraud
0    6362604
1         16
Name: count, dtype: Int64
```

In [26]:
```
isFraud_flagged_fraud_records = df[(df.isFraud==1) & (df.isFlaggedFraud==1)]
isFraud_flagged_fraud_records
```

Out[26]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest
2736446	212	TRANSFER	4953893.08	C728984460	4953893.08	4953893.08	C639921569
3247297	250	TRANSFER	1343002.08	C1100582606	1343002.08	1343002.08	C1147517658
3760288	279	TRANSFER	536624.41	C1035541766	536624.41	536624.41	C1100697970
5563713	387	TRANSFER	4892193.09	C908544136	4892193.09	4892193.09	C891140444
5996407	425	TRANSFER	10000000.0	C689608084	19585040.37	19585040.37	C1392803603
5996409	425	TRANSFER	9585040.37	C452586515	19585040.37	19585040.37	C1109166882
6168499	554	TRANSFER	3576297.1	C193696150	3576297.1	3576297.1	C484597480
6205439	586	TRANSFER	353874.22	C1684585475	353874.22	353874.22	C1770418982
6266413	617	TRANSFER	2542664.27	C786455622	2542664.27	2542664.27	C661958277
6281482	646	TRANSFER	10000000.0	C19004745	10399045.08	10399045.08	C1806199534
6281484	646	TRANSFER	399045.08	C724693370	10399045.08	10399045.08	C1909486199
6296014	671	TRANSFER	3441041.46	C917414431	3441041.46	3441041.46	C1082139865
6351225	702	TRANSFER	3171085.59	C1892216157	3171085.59	3171085.59	C1308068787
6362460	730	TRANSFER	10000000.0	C2140038573	17316255.05	17316255.05	C1395467927
6362462	730	TRANSFER	7316255.05	C1869569059	17316255.05	17316255.05	C1861208726
6362584	741	TRANSFER	5674547.89	C992223106	5674547.89	5674547.89	C1366804249

In [27]:
```
isFraud_flagged_fraud_records.shape
```

Out[27]: (16, 11)

In [28]:
```
total_fraud= df[df.isFlaggedFraud ==1]
total_fraud = total_fraud.shape[0]
total_fraud
```

Out[28]: 16

In [29]:
```
total_fraud= df[df.isFraud ==1]
total_fraud = total_fraud.shape[0]
total_fraud
```

Out[29]: 8213

In [30]:
```
total_isflaggedFraud= isFraud_flagged_fraud_records.shape[0]
total_isflaggedFraud
```

Out[30]: 16

In [31]:
```
flagged_percent = total_isflaggedFraud/total_fraud * 100
print('Percentage of flagged fraud: ',round(flagged_percent,3))

unflagged_percent= (total_fraud-total_isflaggedFraud)/total_fraud * 100
print('Percentage of incorrectly flagged fraud: ',round(unflagged_percent,3))
```

```
Percentage of flagged fraud:  0.195
Percentage of incorrectly flagged fraud:  99.805
```

> The data reveals a critical challenge in fraud detection, with a mere 0.195% of transactions correctly identified as fraud, against a high 99.805% of transactions that were incorrectly flagged as fraudulent. This significant imbalance suggests the fraud detection mechanism is overly cautious, producing a vast number of false positives. Such inefficiency could strain resources, erode customer trust, and diminish user experience due to unwarranted scrutiny on legitimate transactions.

Fraud amount

In [32]:
```
total_transactions = df.shape[0]
fraud_transaction = df[df.isFraud==1].shape[0]
fraud_percent= fraud_transaction/total_transactions * 100
fraud_percent
```

Out[32]: 0.12908204481801522

In [33]:
```
print('Total transactions: ',total_transactions)
print('Total fraud transactions happened: ',fraud_transaction)
print("Total fraud transaction percent: ",round(fraud_percent,2))
```

```
Total transactions:  6362620
Total fraud transactions happened:  8213
Total fraud transaction percent:  0.13
```

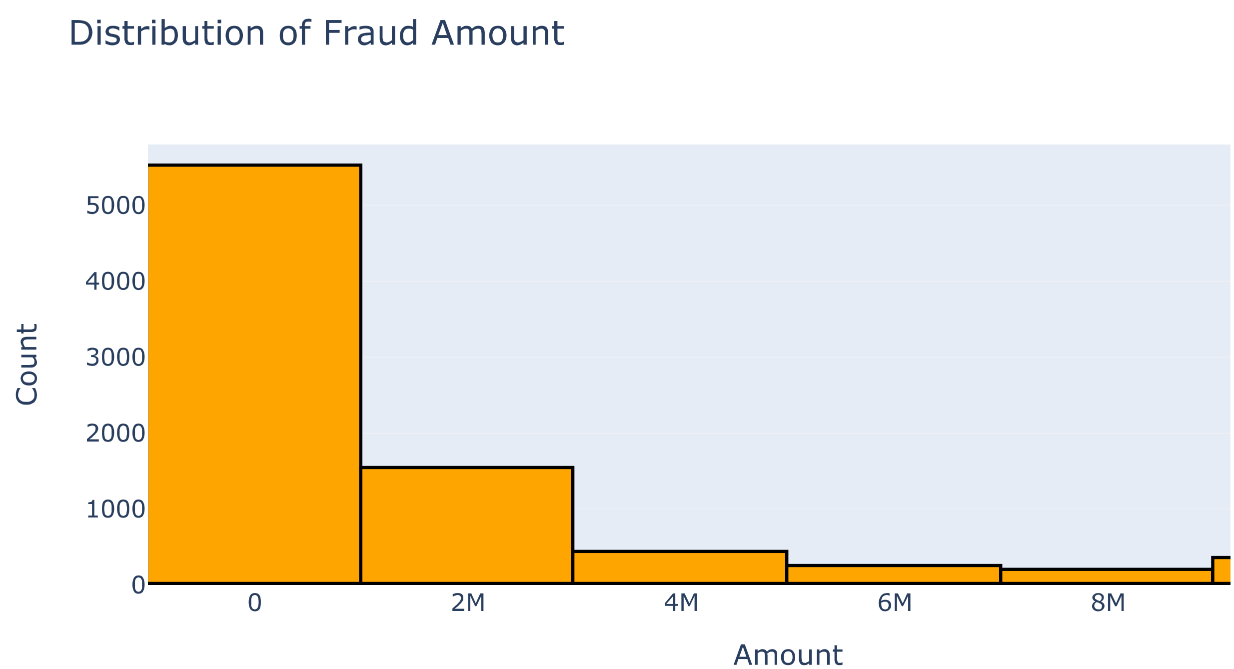
In [34]:
```
fraud_amount= df[df.isFraud==1]
fraud_amount=fraud_amount.sort_values(by=['amount'],ascending=False)
fraud_amount
```

Out[34]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest
3760283	279	CASH_OUT	10000000.0	C1214015158	10000000.0	0.0	C2110157840
5987587	409	CASH_OUT	10000000.0	C97242201	10000000.0	0.0	C786701128
1707592	160	CASH_OUT	10000000.0	C525906402	10000000.0	0.0	C43869769
1707591	160	TRANSFER	10000000.0	C752627210	27670038.08	17670038.08	C1853789265
1707590	160	CASH_OUT	10000000.0	C2068007279	10000000.0	0.0	C836488544
...	...	...	...	...	...	...	...
5996410	425	CASH_OUT	0.0	C69493310	0.0	0.0	C719711728
5996408	425	CASH_OUT	0.0	C832555372	0.0	0.0	C1462759334
6362461	730	CASH_OUT	0.0	C729003789	0.0	0.0	C1388096959
6362463	730	CASH_OUT	0.0	C2088151490	0.0	0.0	C1156763710
3760289	279	CASH_OUT	0.0	C539112012	0.0	0.0	C1106468520

8213 rows × 11 columns

In [35]:
```
import plotly.express as px

# Assuming 'fraud_amount' is a DataFrame with a column named 'amount'
# that you want to plot

# Create a histogram using Plotly Express
fig = px.histogram(fraud_amount, x='amount', nbins=7,
                   title='Distribution of Fraud Amount',
                   labels={'amount': 'Amount'},  # Change 'amount' to your specif
                   color_discrete_sequence=['orange'])  # Sets the bars to orange

# Customize the histogram
fig.update_traces(marker_line_color='black', marker_line_width=1.5)  # Sets the e
fig.update_layout(xaxis_title='Amount', yaxis_title='Count',
                  width=800, height=400)  # Adjusts the size, similar to figsize

# Show the plot
fig.show()
```

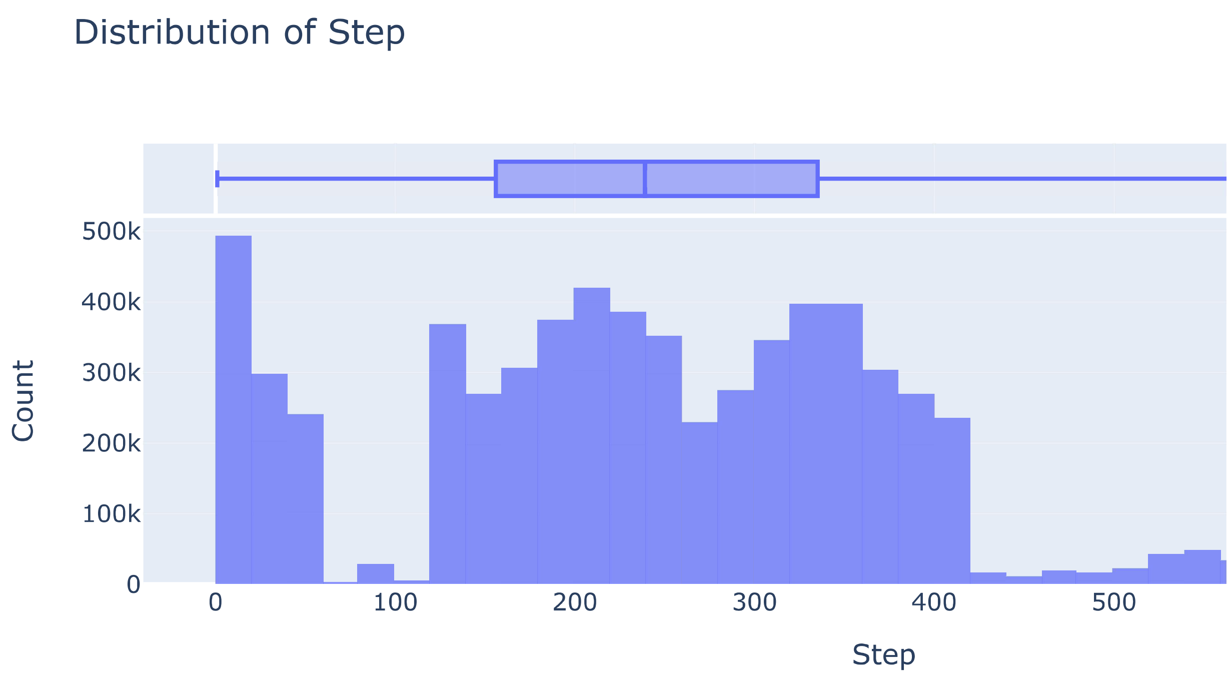


Most of the fraud transaction amount is in between 1 million.

Calculating max frequency of Steps

In [36]:

```
import plotly.express as px

# Assuming df is your DataFrame and 'step' is the column you want to plot

# Create a histogram using Plotly Express
fig = px.histogram(df, x='step', nbins=50,
                   title='Distribution of Step',
                   labels={'step': 'Step'},  # Change 'step' to your specific col
                   opacity=0.75,
                   marginal='box')  # Optional: adds a boxplot alongside the hist

# Customize the histogram appearance
fig.update_layout(xaxis_title='Step', yaxis_title='Count',
                  width=900, height=400)  # Adjusts the size

# Show the plot
fig.show()
```



Maximum distribution are between 150 to 400 of step.

Balancing the data

In [37]:
```
df['isFraud'].value_counts()
```

Out[37]:
```
isFraud
0    6354407
1       8213
Name: count, dtype: Int64
```

OverSampling: SMOTE

In [38]:
```
X = df.drop(columns=['isFraud','type','nameDest','nameOrig'], axis=1)# Remove the
print(X)
```

```
         step      amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
0           1     9839.64       170136.0       160296.36             0.0
1           1     1864.28        21249.0        19384.72             0.0
2           1       181.0          181.0             0.0             0.0
3           1       181.0          181.0             0.0         21182.0
4           1    11668.14        41554.0        29885.86             0.0
...       ...         ...            ...             ...             ...
6362615   743   339682.13      339682.13             0.0             0.0
6362616   743  6311409.28     6311409.28             0.0             0.0
6362617   743  6311409.28     6311409.28             0.0        68488.84
6362618   743   850002.52      850002.52             0.0             0.0
6362619   743   850002.52      850002.52             0.0      6510099.11

         newbalanceDest  isFlaggedFraud
0                   0.0               0
1                   0.0               0
2                   0.0               0
3                   0.0               0
4                   0.0               0
...                 ...             ...
6362615       339682.13               0
6362616             0.0               0
6362617      6379898.11               0
6362618             0.0               0
6362619      7360101.63               0

[6362620 rows x 7 columns]
```

In [39]:
```
Y = df['isFraud']
print(Y)
```

```
0          0
1          0
2          1
3          1
4          0
          ..
6362615    1
6362616    1
6362617    1
6362618    1
6362619    1
Name: isFraud, Length: 6362620, dtype: Int64
```

In [40]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify
```

In [41]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(6362620, 7) (5090096, 7) (1272524, 7)
```

In [42]:
```
from imblearn.over_sampling import SMOTE #SMOTE = synthetic minority oversampling
smote = SMOTE()
```

In [43]:
```
X_train = X_train.astype(float)
```

In [44]:
```
X_train_smote, Y_train_smote = smote.fit_resample(X_train, Y_train)
print(X_train_smote.shape)
print(Y_train_smote.shape)
```

```
(10167052, 7)
(10167052,)
```

UnderSampling

In [45]:
```
legit_txns = df[df.isFraud == 0]
fraud_txns = df[df.isFraud == 1]
```

In [46]:
```
print(legit_txns.shape)
print(fraud_txns.shape)
```

```
(6354407, 11)
(8213, 11)
```

In [47]:
```
legit_sample = legit_txns.sample(n=8213) # Samples 8213 transactions out of the l
undersampled_dataset = pd.concat([legit_sample, fraud_txns], axis=0)
```

In [48]:
```
undersampled_dataset['isFraud'].value_counts()
```

Out[48]:
```
isFraud
0    8213
1    8213
Name: count, dtype: Int64
```

In [49]:
```
X = undersampled_dataset.drop(columns=['isFraud','type','nameDest','nameOrig'], a
print(X)
```

```
         step      amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
125675     11   219285.14       215272.0             0.0         30636.0
1716579   160     14157.4        10927.0             0.0             0.0
5133432   356   231295.34            0.0             0.0       314740.87
954479     44    22553.28            0.0             0.0             0.0
5921102   404     1278.81         9028.0         7749.19             0.0
...       ...         ...            ...             ...             ...
6362615   743   339682.13      339682.13             0.0             0.0
6362616   743  6311409.28     6311409.28             0.0             0.0
6362617   743  6311409.28     6311409.28             0.0        68488.84
6362618   743   850002.52      850002.52             0.0             0.0
6362619   743   850002.52      850002.52             0.0      6510099.11

         newbalanceDest  isFlaggedFraud
125675        249921.14               0
1716579             0.0               0
5133432       546036.22               0
954479              0.0               0
5921102             0.0               0
...                 ...             ...
6362615       339682.13               0
6362616             0.0               0
6362617      6379898.11               0
6362618             0.0               0
6362619      7360101.63               0

[16426 rows x 7 columns]
```

In [50]:
```
Y = undersampled_dataset['isFraud']
print(Y)
```

```
125675     0
1716579    0
5133432    0
954479     0
5921102    0
          ..
6362615    1
6362616    1
6362617    1
6362618    1
6362619    1
Name: isFraud, Length: 16426, dtype: Int64
```

In [51]:
```
X_train_undersampled, X_test_undersampled, Y_train_undersampled, Y_test_undersamp
```

In [52]:
```
print(X.shape, X_train_undersampled.shape, X_test_undersampled.shape)
```

```
(16426, 7) (13140, 7) (3286, 7)
```

Model training

Oversampling

In [53]:
```
scaler = StandardScaler()
X_train_scaled_smote = scaler.fit_transform(X_train_smote)
```

Undersampling

In [54]:
```
X_train_scaled_undersampled = scaler.fit_transform(X_train_undersampled)
X_test_scaled_undersampled = scaler.transform(X_test_undersampled)
```

Creating the Model instances/objects

In [55]:
```
LogisticRegressionModel = LogisticRegression()
```

Oversampling

In [56]:
```
LogisticRegressionModel.fit(X_train_scaled_smote, Y_train_smote)
```

Out[56]:
```
▾ LogisticRegression
LogisticRegression()
```
(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.LogisticRegre

Undersampling

In [57]:
```
LogisticRegressionModel.fit(X_train_scaled_undersampled, Y_train_undersampled)
```

Out[57]:
```
▾ LogisticRegression
LogisticRegression()
```
(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.LogisticRegre

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: