Online Payments Fraud Detection using ML

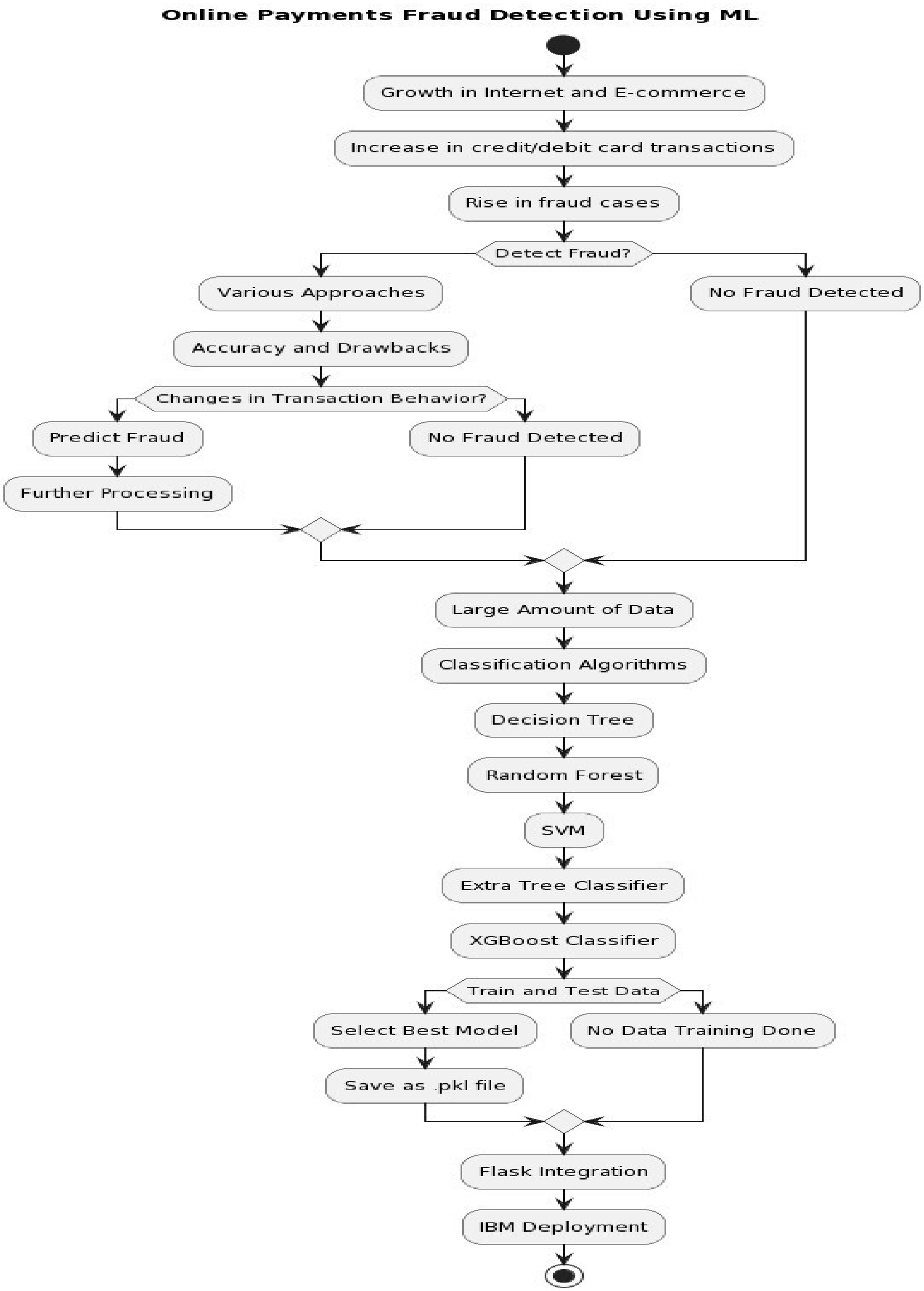
Project Manual

Date	05 November 2023
Team ID	Team-592699
Project Name	Online payment fraud detection using ML

Project Description:

The growth in internet and e-commerce appears to involve the use of online credit/debit card transactions. The increase in the use of credit / debit cards is causing an increase in fraud. The frauds can be detected through various approaches, yet they lag in their accuracy and its own specific drawbacks. If there are any changes in the conduct of the transaction, the frauds are predicted and taken for further process. Due to large amount of data credit / debit card fraud detection problem is rectified by the proposed method We will be using classification algorithms such as Decision tree, Random forest, svm, and Extra tree classifier, xgboost Classifier.We will train and test the data with these algorithms. From this the best model is selected and saved in pkl format. We will be doing flask integration and IBM deployment.

Technical Architecture:



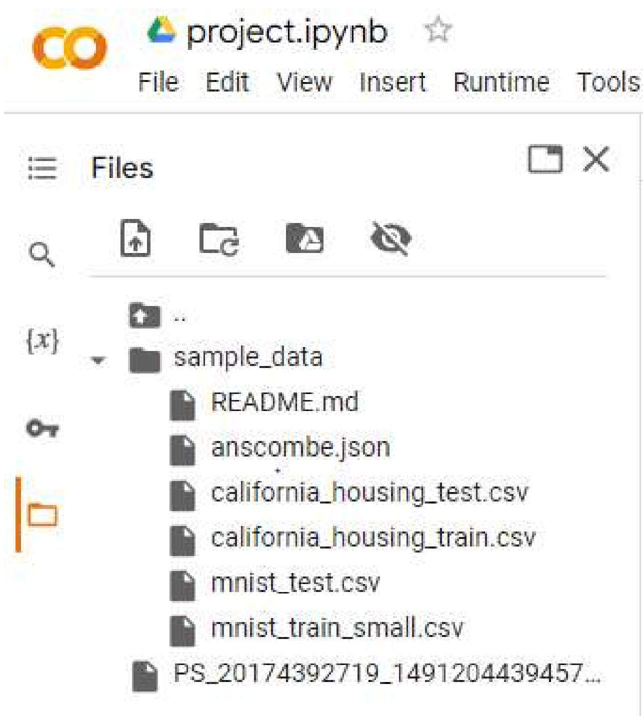
SPRINT 1:

- **PROJECT SCOPE :**
  The project aims to create an efficient online payments fraud detection system using machine learning, featuring classification algorithms such as Decision Tree, Random Forest, SVM, Extra Tree Classifier, and XGBoost. The key objectives include model selection and training, building a user-friendly web interface with Flask, deploying the model on the IBM cloud, and ensuring continuous monitoring for optimal fraud detection. This project seeks to enhance security in e-commerce by accurately identifying and preventing fraudulent credit/debit card transactions, while providing a scalable and accessible solution for real-time fraud detection.

- **PROJECT OBJECTIVE:**
  - Develop an effective fraud detection system that can accurately identify fraudulent online credit/debit card transactions while minimizing false positives.
  - Implement a user-friendly interface for interacting with the fraud detection system, allowing users to submit transaction data and receive real-time fraud detection results.
  - Deploy the best-performing machine learning model on the IBM cloud platform for scalability and accessibility.
  - Continuously monitor the model's performance and ensure that it maintains a high level of accuracy in detecting fraudulent transactions.
  - Provide documentation and training to ensure the successful operation and maintenance of the fraud detection system.

- **Identify potential risks and challenges**
  Developing an online payments fraud detection system using machine learning faces potential challenges, including data quality issues, model overfitting, privacy concerns, operational scalability, and the need for robustness against adversarial attacks. Balancing false positives and false negatives, ensuring regulatory compliance, managing costs, and maintaining user trust are also critical considerations. Addressing these challenges requires a holistic approach, ongoing monitoring, and collaboration across domains to create an effective and reliable fraud detection solution.

- **Identifying and reviewing dataset for analysis**
  It is a dataset from Kaggle for online payment fraud detection.



SPRINT-2 :

- **Clean,transform and prepare data for modeling.**

  Read the csv file

```
df = pd.read_csv('/content/PS_20174392719_1491204439457_log.csv', error_bad_lines=False)
```

Display play first few rows in the dataframe

```
[26] df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0.0	0.0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0.0	0.0
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	1.0	0.0
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	1.0	0.0
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
259701	13	CASH_OUT	381528.10	C1151639555	19050.35	0.00	C1554690148	2692449.40	3073977.50	0.0	0.0
259702	13	CASH_OUT	253846.70	C461636032	20332.00	0.00	C625128530	16564.00	270410.70	0.0	0.0
259703	13	CASH_IN	550193.54	C1640363951	56314.00	606507.54	C1732292969	882690.23	80127.87	0.0	0.0
259704	13	TRANSFER	759982.19	C381497084	606507.54	0.00	C1621805812	1038091.14	1798073.33	0.0	0.0
259705	13	CASH_IN	196627.14	C2004365456	19890.00	216517.14	C2132646226	65135.73	2254.00	NaN	NaN

259706 rows × 11 columns

Generate descriptive statistics of the DataFrame

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

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
count	259706.000000	2.597060e+05	2.597060e+05	2.597060e+05	2.597010e+05	2.597010e+05	259700.000000	259698.0
mean	10.330647	1.830972e+05	8.865923e+05	9.042559e+05	9.552154e+05	1.193862e+06	0.000616	0.0
std	2.007283	3.296686e+05	2.825060e+06	2.861908e+06	2.377506e+06	2.634077e+06	0.024814	0.0
min	1.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.0
25%	9.000000	1.249008e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.0
50%	11.000000	7.192957e+04	1.980900e+04	0.000000e+00	5.714140e+04	1.533787e+05	0.000000	0.0
75%	12.000000	2.329429e+05	1.975310e+05	2.336654e+05	7.996110e+05	1.204577e+06	0.000000	0.0
max	13.000000	1.000000e+07	3.890000e+07	3.890000e+07	3.900000e+07	3.900000e+07	1.000000	0.0

Get the column labels of the DataFrame

```
[28] df.columns

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

Display concise information about the DataFrame, including data types and memory usage

```
[29] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 259706 entries, 0 to 259705
Data columns (total 11 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   step            259706 non-null  int64
 1   type            259706 non-null  object
 2   amount          259706 non-null  float64
 3   nameOrig        259706 non-null  object
 4   oldbalanceOrg   259706 non-null  float64
 5   newbalanceOrig  259706 non-null  float64
 6   nameDest        259702 non-null  object
 7   oldbalanceDest  259701 non-null  float64
 8   newbalanceDest  259701 non-null  float64
 9   isFraud         259700 non-null  float64
 10  isFlaggedFraud  259698 non-null  float64
dtypes: float64(7), int64(1), object(3)
memory usage: 21.8+ MB
```

Display the last few rows of the DataFrame

```
[31] df.tail()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
259701	13	CASH_OUT	381528.10	C1151639555	19050.35	0.00	C1554690148	2692449.40	3073977.50	0.0
259702	13	CASH_OUT	253846.70	C461636032	20332.00	0.00	C625128530	16564.00	270410.70	0.0
259703	13	CASH_IN	550193.54	C1640363951	56314.00	606507.54	C1732292969	882690.23	80127.87	0.0
259704	13	TRANSFER	759982.19	C381497084	606507.54	0.00	C1621805812	1038091.14	1798073.33	0.0
259705	13	CASH_IN	196627.14	C2004365456	19890.00	216517.14	C2132646226	65135.73	2254.00	NaN

Calculate and display the correlation matrix of the DataFrame

```
[32] df.corr()
```

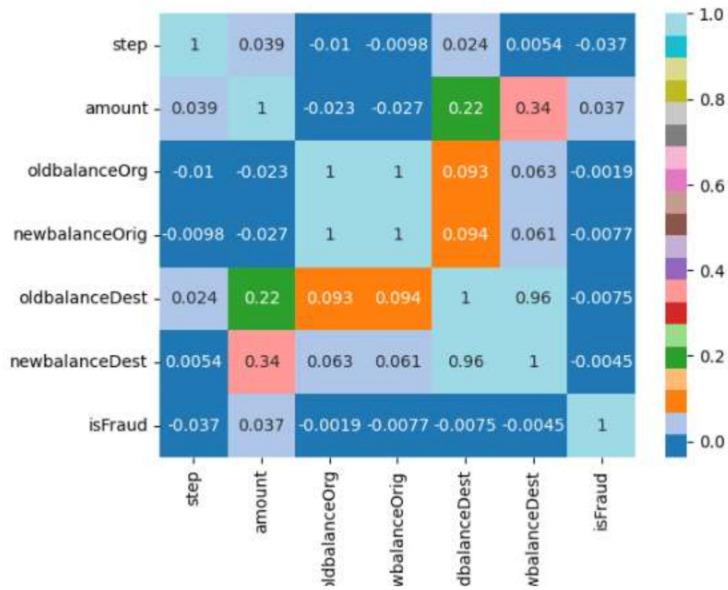
```
<ipython-input-32-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated.
  df.corr()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
step	1.000000	0.038876	-0.010344	-0.009762	0.024330	0.005398	-0.036946
amount	0.038876	1.000000	-0.022522	-0.026537	0.215206	0.339720	0.037125
oldbalanceOrg	-0.010344	-0.022522	1.000000	0.998975	0.092858	0.062516	-0.001933
newbalanceOrig	-0.009762	-0.026537	0.998975	1.000000	0.094374	0.061481	-0.007662
oldbalanceDest	0.024330	0.215206	0.092858	0.094374	1.000000	0.955999	-0.007541
newbalanceDest	0.005398	0.339720	0.062516	0.061481	0.955999	1.000000	-0.004546
isFraud	-0.036946	0.037125	-0.001933	-0.007662	-0.007541	-0.004546	1.000000

Create a heatmap to visualize the correlation matrix of the DataFrame

```
[33] sns.heatmap(df.corr(),annot=True,cmap="tab20")
```

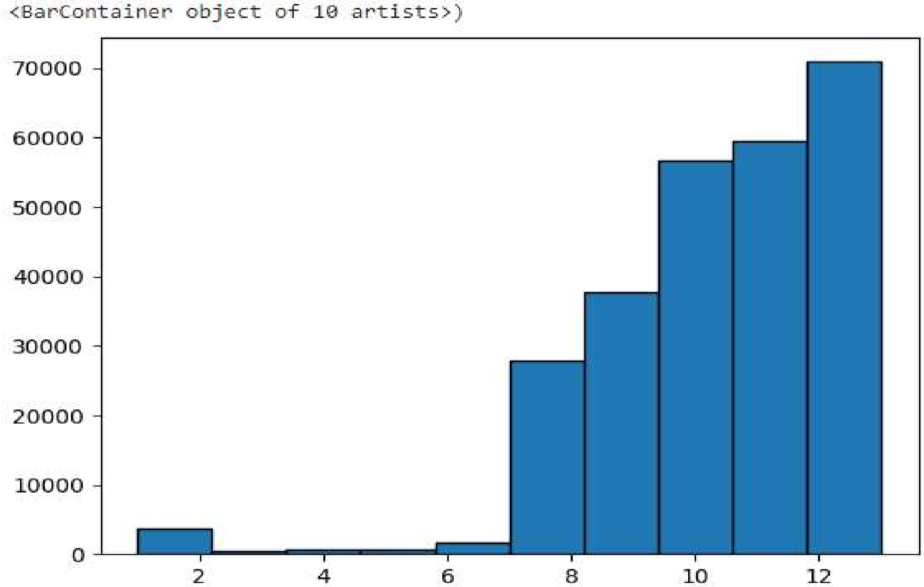
```
<ipython-input-33-dbf7c0edd73f>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated
  sns.heatmap(df.corr(),annot=True,cmap="tab20")
<Axes: >
```



Create a histogram using Matplotlib to visualize the distribution of 'step' column in the DataFrame

```
plt.hist(data=df,x='step', bins=10, edgecolor='black')
```

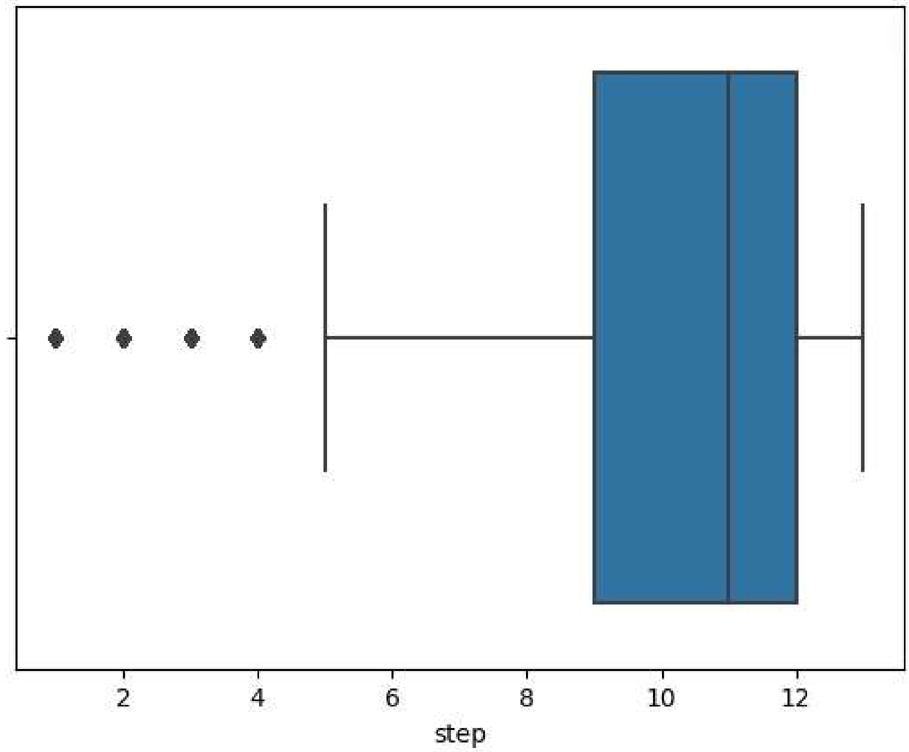
```
(array([ 3722.,   552.,   565.,   665.,  1660., 27933., 37627., 56638.,
        59488., 70856.]),
 array([ 1. ,  2.2,  3.4,  4.6,  5.8,  7. ,  8.2,  9.4, 10.6, 11.8, 13. ]),
 <BarContainer object of 10 artists>)
```



Create a boxplot using Seaborn to visualize the distribution and central tendency of 'step' column in the DataFrame

```
sns.boxplot(data=df,x='step')
```

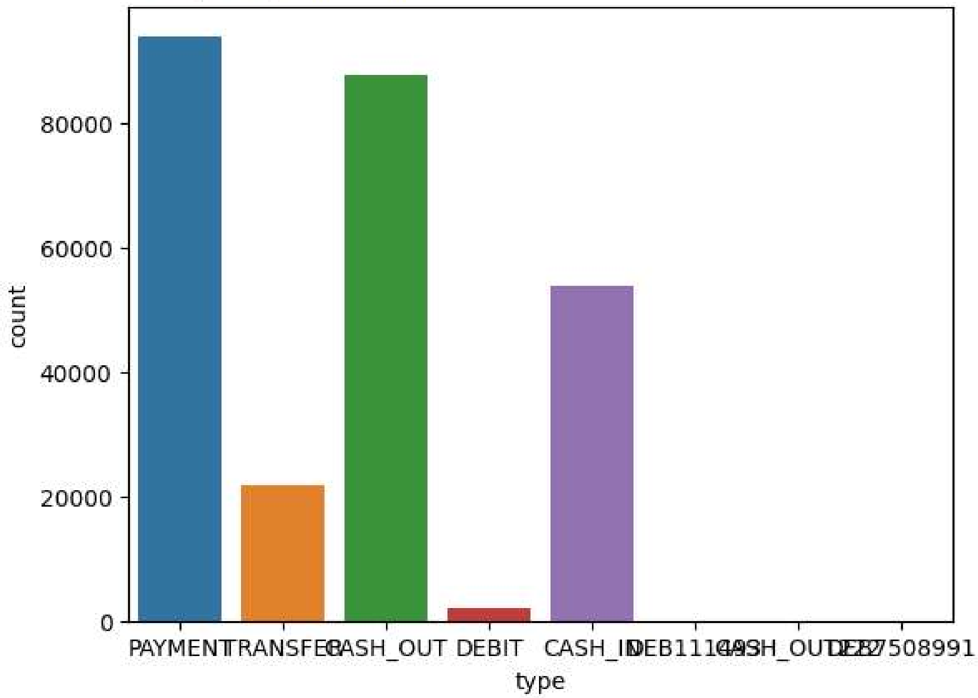
```
<Axes: xlabel='step'>
```



Create a countplot using Seaborn to visualize the distribution of 'type' column in the DataFrame

```
[36] sns.countplot(data=df,x='type')
```

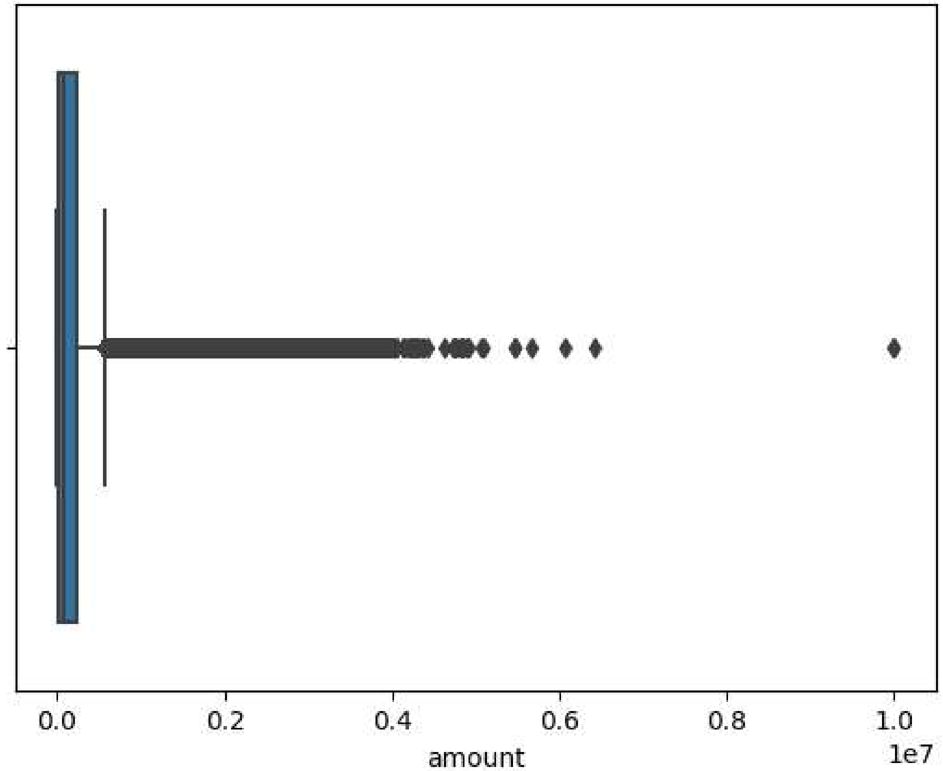
```
<Axes: xlabel='type', ylabel='count'>
```



Create a boxplot using Seaborn to visualize the distribution and central tendency of 'amount' column in the DataFrame

```
sns.boxplot(data=df,x='amount')
```

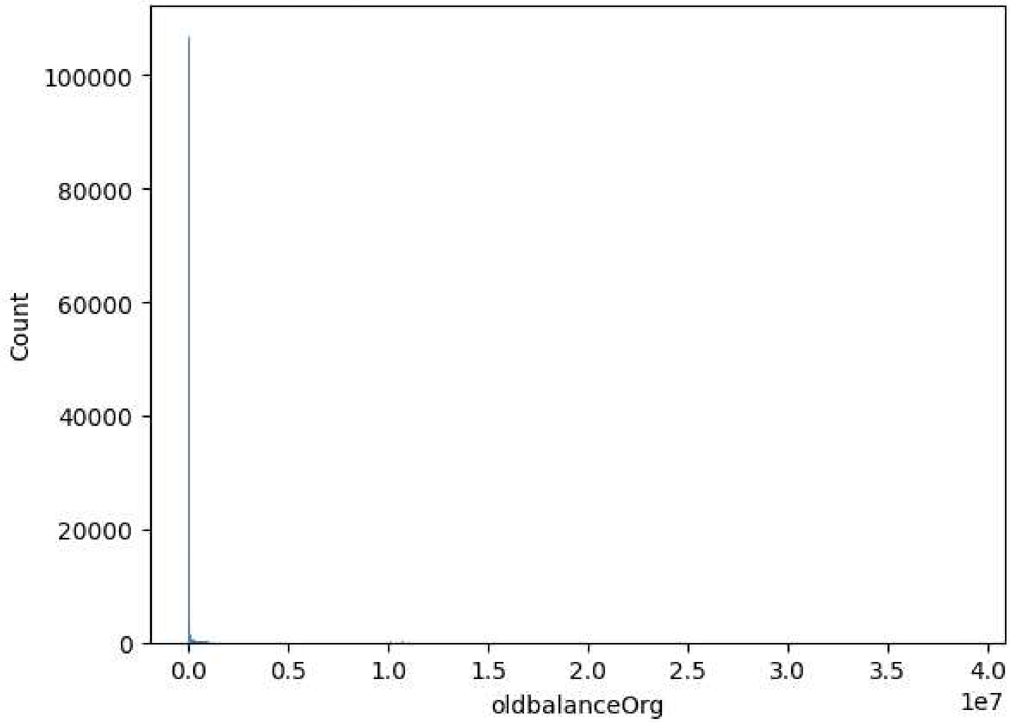
```
<Axes: xlabel='amount'>
```



Create a histogram using Seaborn to visualize the distribution of 'oldbalanceOrg' column in the DataFrame

```
[40] sns.histplot(data=df,x='oldbalanceOrg')
```

```
<Axes: xlabel='oldbalanceOrg', ylabel='Count'>
```



Count the occurrences of each unique value in the 'nameDest' column in the DataFrame

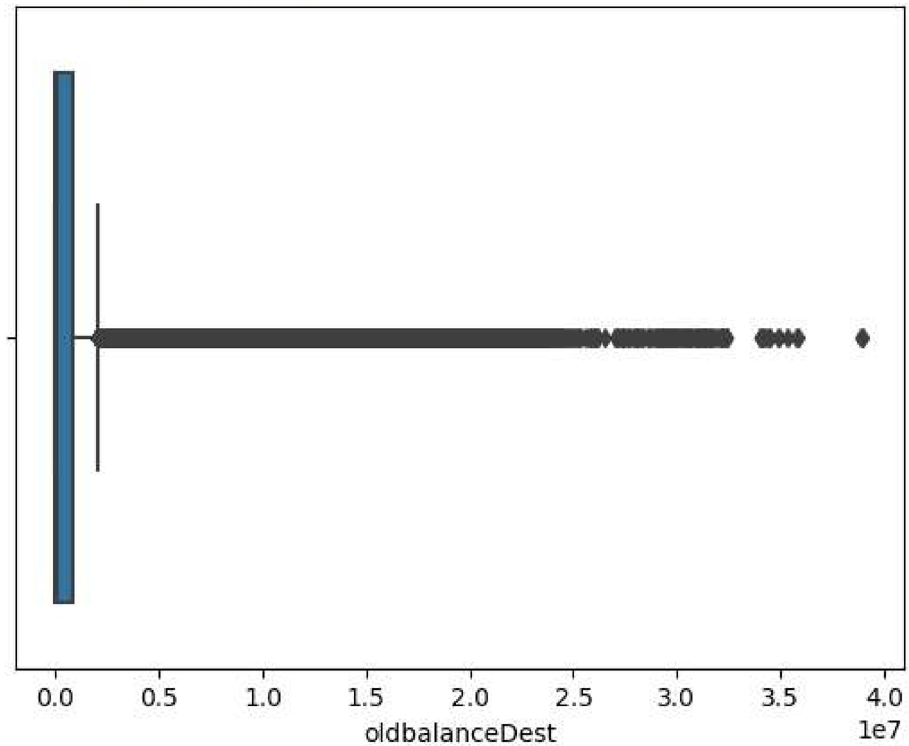
```
[41] df['nameDest'].value_counts()
```

```
C1286084959    87
C985934102     86
C1590550415    79
C2083562754    79
C248609774     79
               ..
M1588897264     1
M1254841124     1
M665734621      1
M844671338      1
C625128530      1
Name: nameDest, Length: 101754, dtype: int64
```

Create a boxplot using Seaborn to visualize the distribution and central tendency of 'oldbalanceDest' column in the DataFrame

```
[42] sns.boxplot(data=df,x='oldbalanceDest')
```

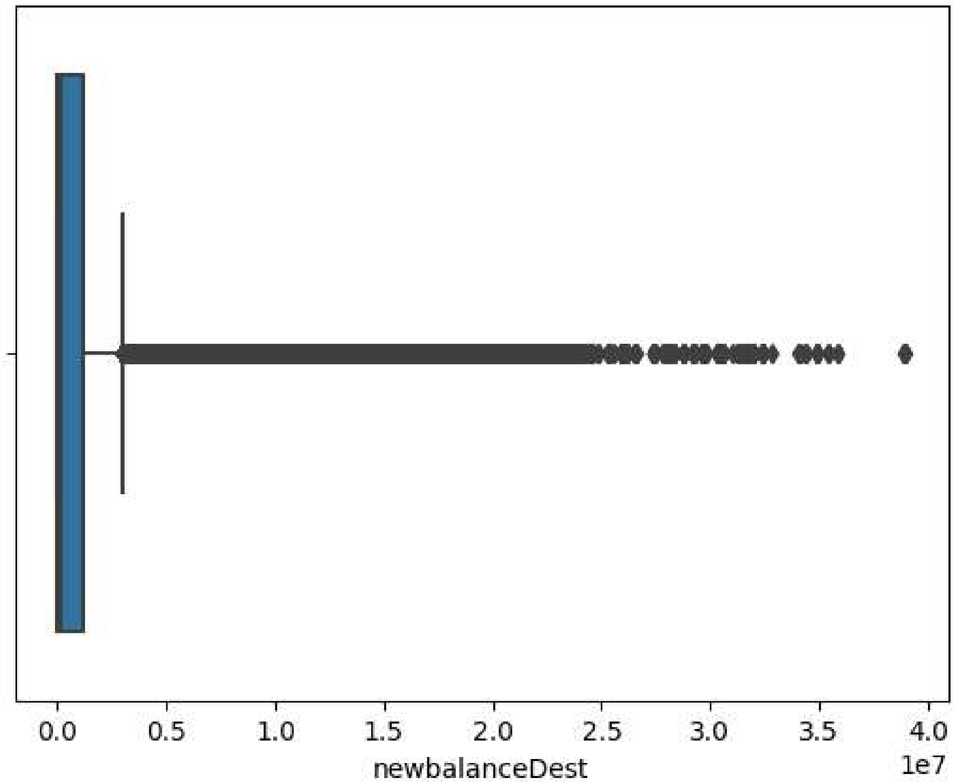
```
<Axes: xlabel='oldbalanceDest'>
```



Create a boxplot using Seaborn to visualize the distribution and central tendency of 'newbalanceDest' column in the DataFrame

```
[43] sns.boxplot(data=df,x='newbalanceDest')
```

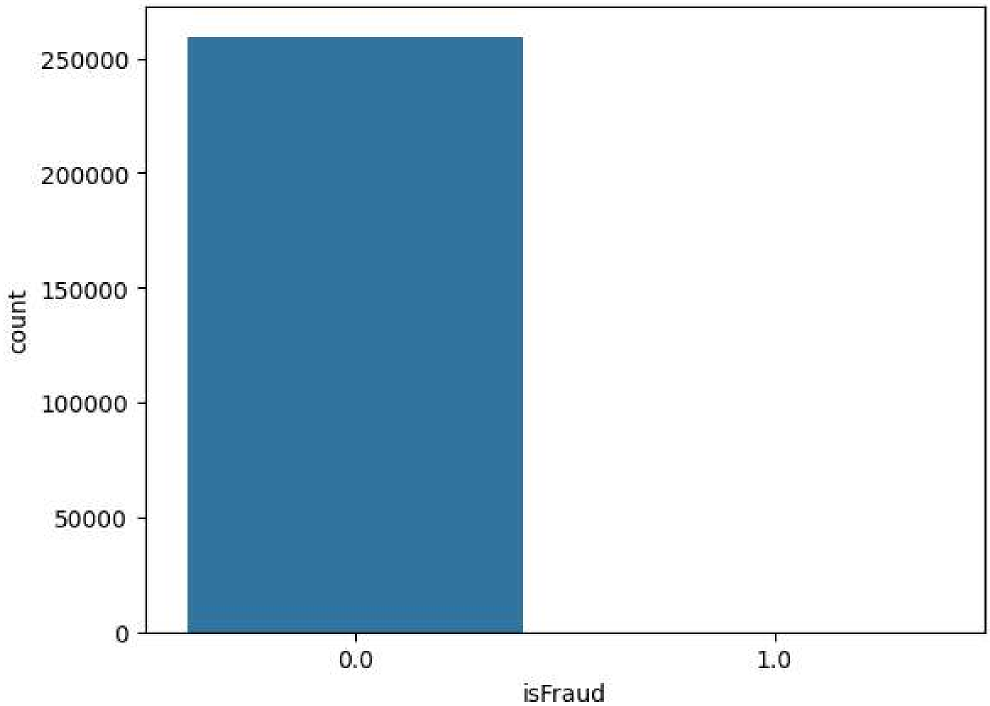
```
<Axes: xlabel='newbalanceDest'>
```



Create a countplot using Seaborn to visualize the distribution of 'isFraud' column in the DataFrame

```
[44] sns.countplot(data=df,x='isFraud')
```

```
<Axes: xlabel='isFraud', ylabel='count'>
```



Create a new column 'isfraud' with labels 'is not fraud' for 0 and 'is fraud' for 1 in the 'isFraud' column

```
df.loc[df['isFraud']==0,'isfraud'] = 'is not fraud'
df.loc[df['isFraud']==1,'isfraud'] = 'is fraud'
df
```

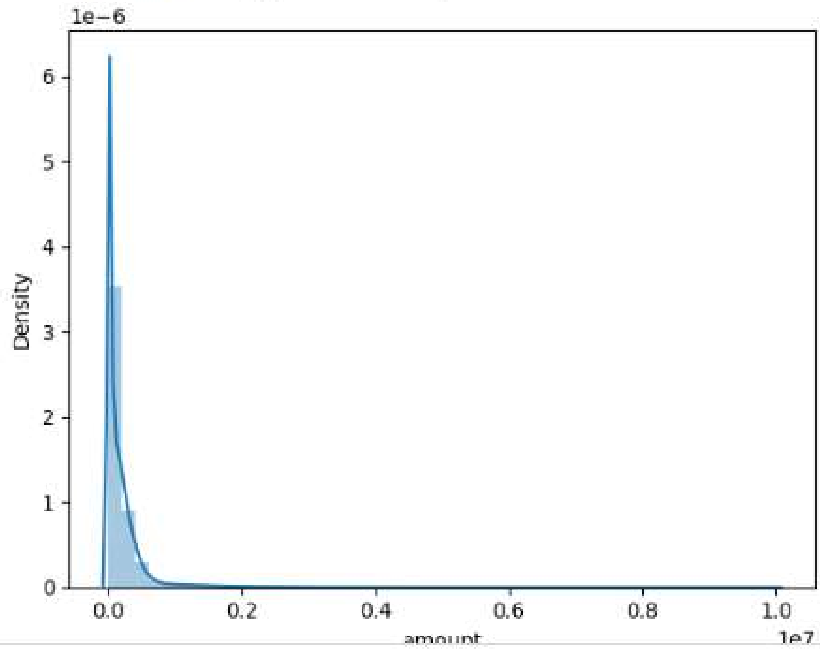
	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isfraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0.0	is not fraud
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0.0	is not fraud
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	1.0	is fraud
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	1.0	is fraud
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0.0	is not fraud
...	...	...	...	...	...	...	...	...	...	...	...
259701	13	CASH_OUT	381528.10	C1151639555	19050.35	0.00	C1554690148	2692449.40	3073977.50	0.0	is not fraud
259702	13	CASH_OUT	253846.70	C461636032	20332.00	0.00	C625128530	16564.00	270410.70	0.0	is not fraud
259703	13	CASH_IN	550193.54	C1640363951	56314.00	606507.54	C1732292969	882690.23	80127.87	0.0	is not fraud
259704	13	TRANSFER	759982.19	C381497084	606507.54	0.00	C1621805812	1038091.14	1798073.33	0.0	is not fraud
259705	13	CASH_IN	196627.14	C2004365456	19890.00	216517.14	C2132646226	65135.73	2254.00	NaN	NaN

259706 rows × 11 columns

Create a distribution plot using Seaborn to visualize the distribution of the 'amount' column in the DataFrame

```
sns.distplot(df.amount)
```

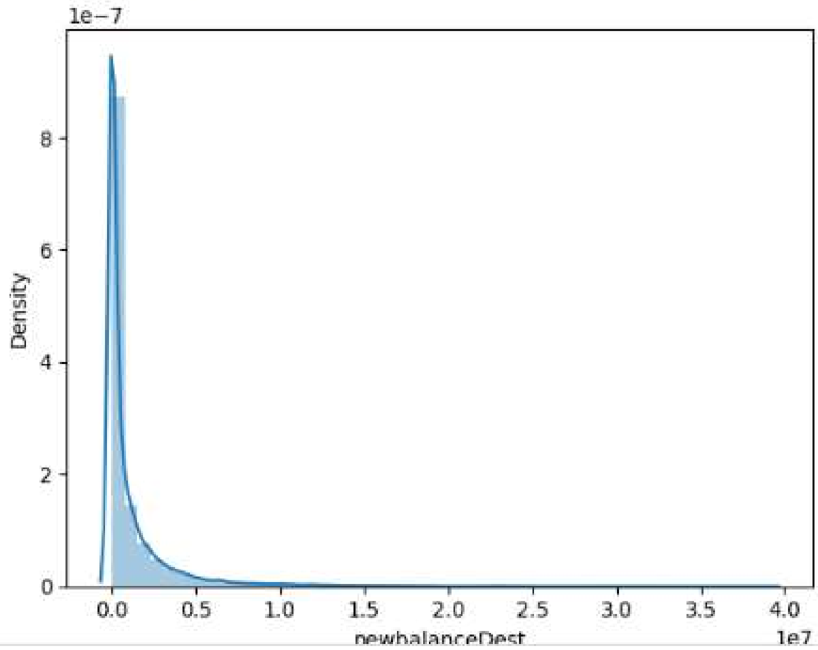
```
<ipython-input-47-dcb24808117d>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.amount)
<Axes: xlabel='amount', ylabel='Density'>
```



Create a distribution plot using Seaborn to visualize the distribution of the 'newbalanceDest' column in the DataFrame

```
[48] sns.distplot(df.newbalanceDest)
```

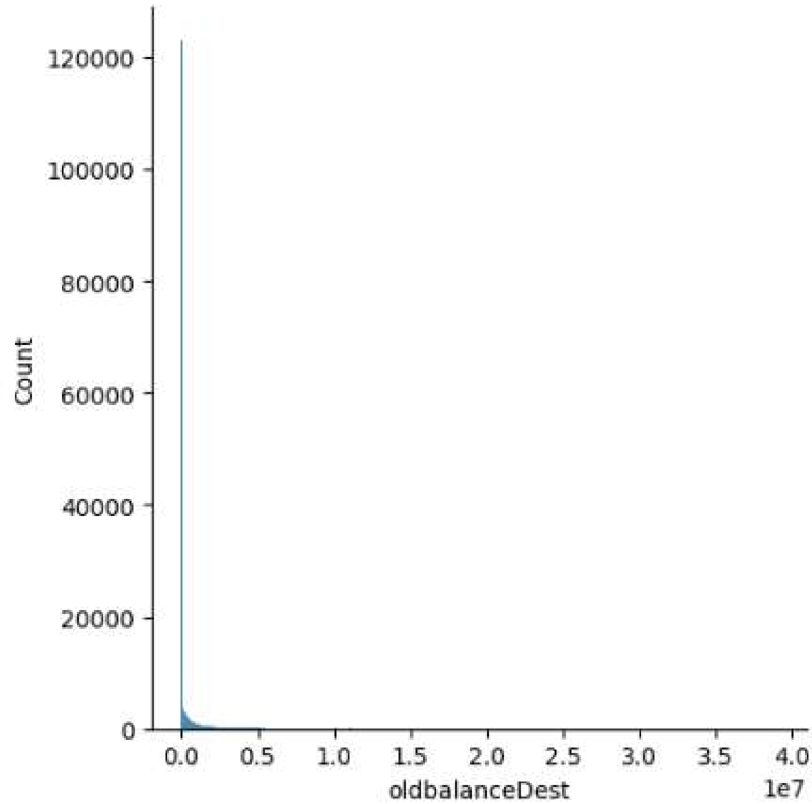
```
<ipython-input-48-1fed35272c49>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.newbalanceDest)
<Axes: xlabel='newbalanceDest', ylabel='Density'>
```



Create a distribution plot using Seaborn to visualize the distribution of the 'oldbalanceDest' column in the DataFrame

```
sns.displot(df.oldbalanceDest)
```

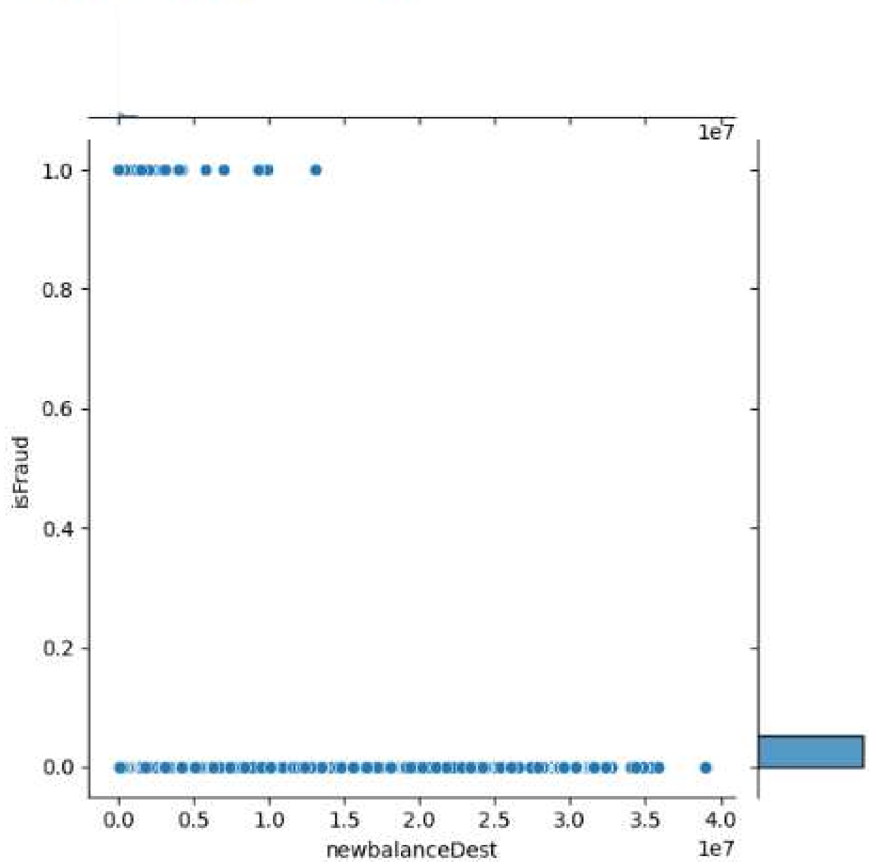
```
<seaborn.axisgrid.FacetGrid at 0x7fd4ae5966b0>
```



Create a joint plot using Seaborn to visualize the relationship between 'newbalanceDest' and 'isFraud' columns in the DataFrame

```
[52] sns.jointplot(data=df,x='newbalanceDest',y='isFraud')
```

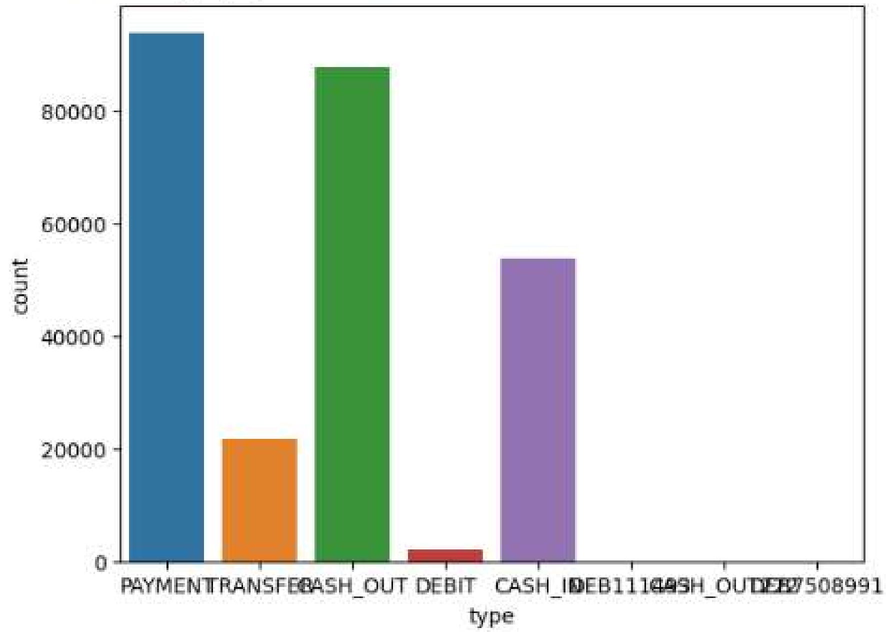
```
<seaborn.axisgrid.JointGrid at 0x7fd4aa053f40>
```



Create a countplot using Seaborn to visualize the distribution of 'type' column in the DataFrame

```
[53] sns.countplot(data=df,x='type')
```

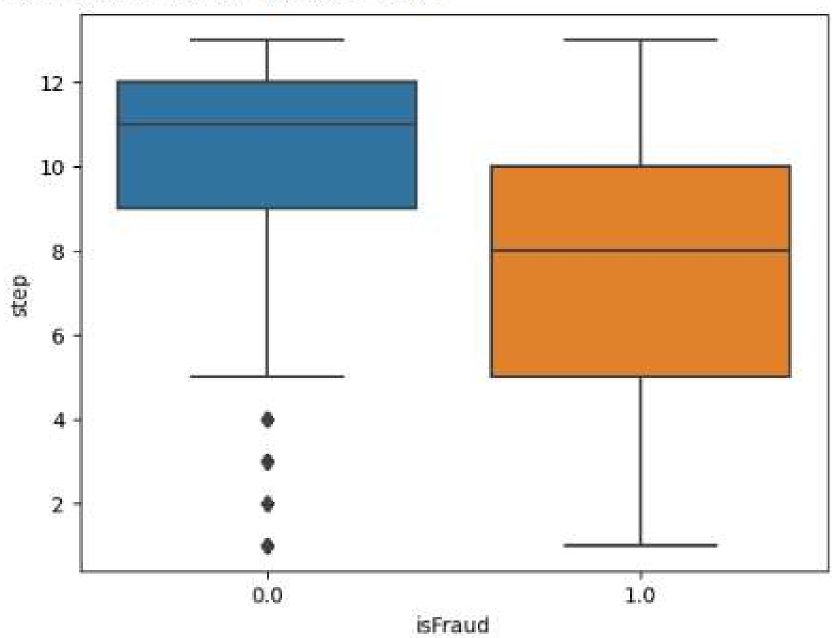
<Axes: xlabel='type', ylabel='count'>



Create a boxplot using Seaborn to visualize the distribution of 'step' for each 'isFraud' category in the DataFrame

```
[54] sns.boxplot(data=df,x='isFraud',y='step')
```

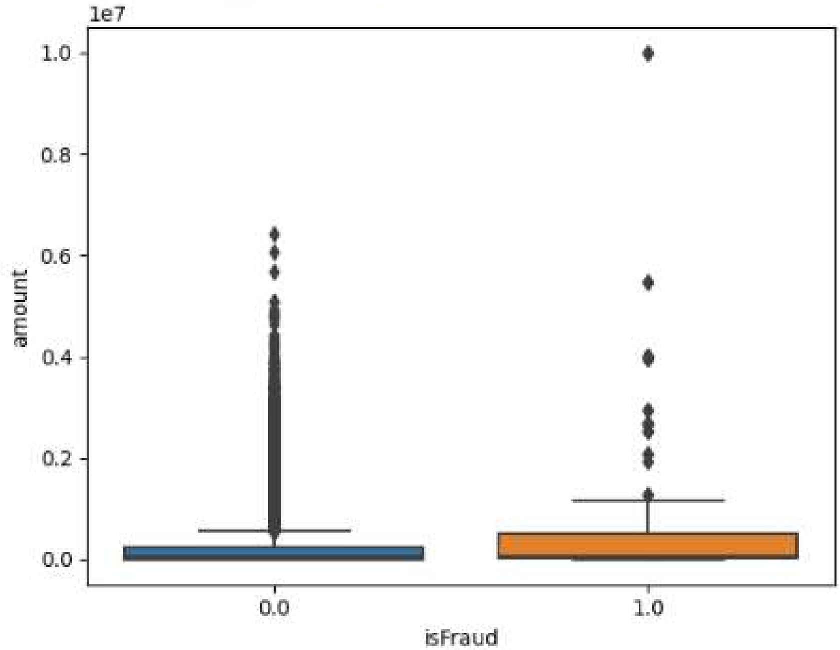
<Axes: xlabel='isFraud', ylabel='step'>



Create a boxplot using Seaborn to visualize the distribution of 'amount' for each 'isFraud' category in the DataFrame

```
[55] sns.boxplot(data=df,x='isFraud',y='amount')
```

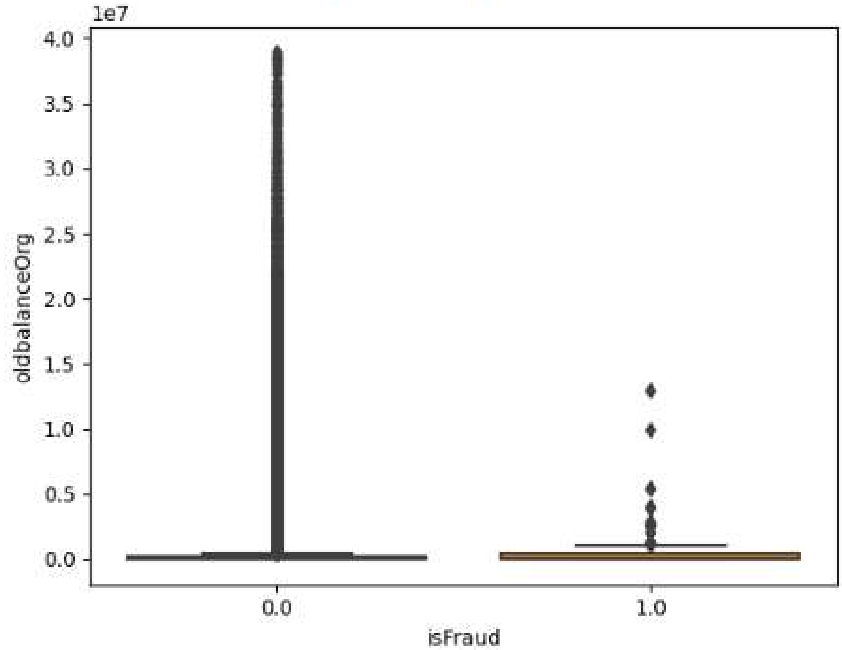
<Axes: xlabel='isFraud', ylabel='amount'>



Create a boxplot using Seaborn to visualize the distribution of 'oldbalanceOrg' for each 'isFraud' category in the DataFrame

```
[56] sns.boxplot(data=df,x='isFraud',y='oldbalanceOrg')
```

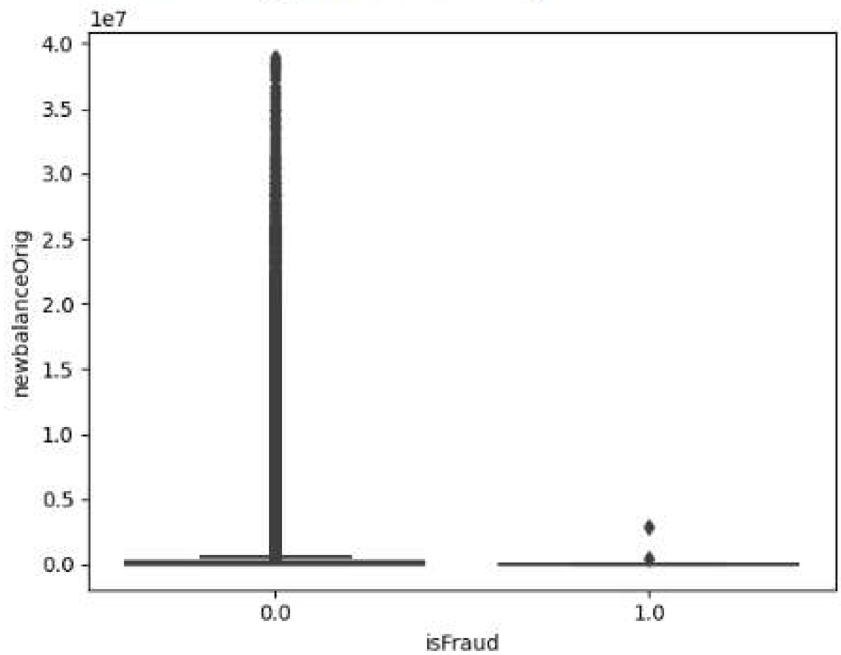
```
<Axes: xlabel='isFraud', ylabel='oldbalanceOrg'>
```



Create a boxplot using Seaborn to visualize the distribution of 'newbalanceOrig' for each 'isFraud' category in the DataFrame

```
[57] sns.boxplot(data=df,x='isFraud',y='newbalanceOrig')
```

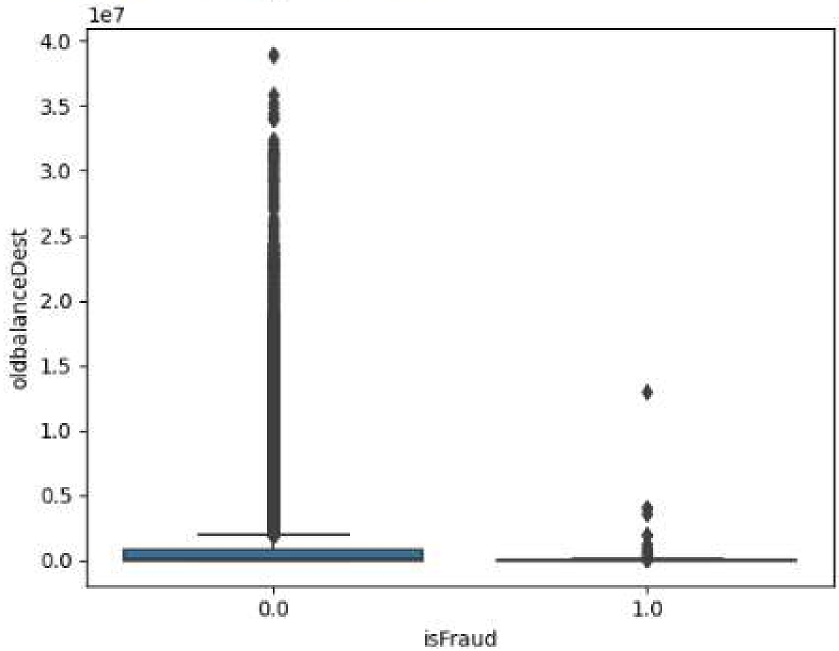
```
<Axes: xlabel='isFraud', ylabel='newbalanceOrig'>
```



Create a boxplot using Seaborn to visualize the distribution of 'oldbalanceDest' for each 'isFraud' category in the DataFrame

```
[58] sns.boxplot(data=df,x='isFraud',y='oldbalanceDest')
```

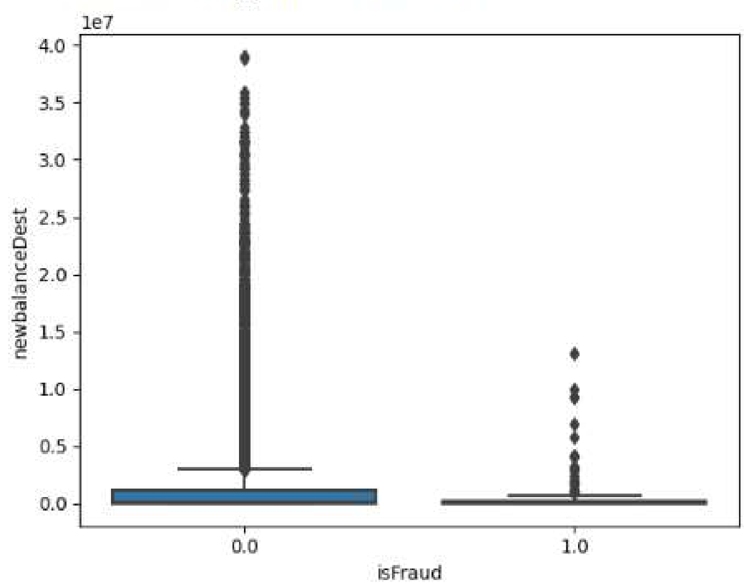
```
<Axes: xlabel='isFraud', ylabel='oldbalanceDest'>
```



Create a boxplot using Seaborn to visualize the distribution of 'newbalanceDest' for each 'isFraud' category in the DataFrame

```
[59] sns.boxplot(data=df,x='isFraud',y='newbalanceDest')
```

```
<Axes: xlabel='isFraud', ylabel='newbalanceDest'>
```



Check and display the sum of missing values in each column of the DataFrame

```
[60] df.isnull().sum()

     step              0
     type              0
     amount            0
     nameOrig          0
     oldbalanceOrg     0
     newbalanceOrig    0
     nameDest          4
     oldbalanceDest    5
     newbalanceDest    5
     isFraud           6
     isfraud           6
     dtype: int64
```

Calculate and print the mode and mean of the 'amount' column in the DataFrame

```
print(stats.mode(df['amount']))
print(np.mean(df['amount']))

ModeResult(mode=2367.99, count=5)
183097.17814210194
```

Calculate the first quartile (Q1), third quartile (Q3), interquartile range (IQR), upper bound, and lower bound for the 'amount' column in the DataFrame

```
q1 = np.quantile(df['amount'],0.25)
q3 = np.quantile(df['amount'],0.75)
IQR = q3-q1
upper_bound = q3+(1.5*IQR)
lower_bound = q1-(1.5*IQR)
```

Define a function to create a side-by-side distribution plot and probability plot for a given feature

```python
def transformedplot(feature):
  plt.figure(figsize=(12,5))
  plt.subplot(1,2,1)
  sns.distplot(feature)
  plt.subplot(1,2,2)
  stats.probplot(feature,plot=p)
```

Use LabelEncoder to encode the 'type' column in the DataFrame and display the value counts

```python
la = LabelEncoder()
df['type'] = la.fit_transform(df['type'])
df['type'].value_counts()
```

```
6    93945
1    87858
0    53905
7    21865
5     2130
3        1
2        1
4        1
Name: type, dtype: int64
```

Separate the features (x) and the target variable (y) in the DataFrame

```python
x
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isfraud
0	1	6	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	is not fraud
1	1	6	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	is not fraud
2	1	7	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	is fraud
3	1	1	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	is fraud
4	1	6	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	is not fraud
...	...	...	...	...	...	...	...	...	...	...
259701	13	1	381528.10	C1151639555	19050.35	0.00	C1554690148	2692449.40	3073977.50	is not fraud
259702	13	1	253846.70	C461636032	20332.00	0.00	C625128530	16564.00	270410.70	is not fraud
259703	13	0	550193.54	C1640363951	56314.00	606507.54	C1732292969	882690.23	80127.87	is not fraud
259704	13	7	759982.19	C381497084	606507.54	0.00	C1621805812	1038091.14	1798073.33	is not fraud
259705	13	0	196627.14	C2004365456	19890.00	216517.14	C2132646226	65135.73	2254.00	NaN

259706 rows × 10 columns

```
[73] y
```

```
0         0.0
1         0.0
2         1.0
3         1.0
4         0.0
         ...
259701    0.0
259702    0.0
259703    0.0
259704    0.0
259705    NaN
Name: isFraud, Length: 259706, dtype: float64
```

```
[74] x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0,test_size=0.2)
```

SPRINT-3

Choose classification algorithms(Decision tree,random forest,svm,Extra tree classifier,and xg boost classifier).

1.Random Forest classifier¶

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)

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

```
0.9958847736625515
```

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

```
1.0
```

```
pd.crosstab(y_test,y_test_predict1)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	232	2
is not Fraud	0	252

```
print(classification_report(y_test,y_test_predict1))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.99      1.00       234
is not Fraud       0.99      1.00      1.00       252

    accuracy                           1.00       486
   macro avg       1.00      1.00      1.00       486
weighted avg       1.00      1.00      1.00       486
```

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

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

```
0.9917695473251029
```

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

```
1.0
```

```
pd.crosstab(y_test,y_test_predict2)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	231	3
is not Fraud	1	251

```
print(classification_report(y_test,y_test_predict2))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.99      0.99       234
is not Fraud       0.99      1.00      0.99       252

    accuracy                           0.99       486
   macro avg       0.99      0.99      0.99       486
weighted avg       0.99      0.99      0.99       486
```

```
from sklearn.ensemble import ExtraTreesClassifier
etc=ExtraTreesClassifier()
etc.fit(x_train,y_train)

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

```
0.9938271604938271
```

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

```
1.0
```

```
pd.crosstab(y_test,y_test_predict3)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	231	3
is not Fraud	0	252

```
print(classification_report(y_test,y_test_predict3))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.99      0.99       234
is not Fraud       0.99      1.00      0.99       252

    accuracy                           0.99       486
   macro avg       0.99      0.99      0.99       486
weighted avg       0.99      0.99      0.99       486
```

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc= SVC()
svc.fit(x_train,y_train)
y_test_predict4=svc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict4)
test_accuracy
```

```
0.7901234567901234
```

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

```
0.8009259259259259
```

```
pd.crosstab(y_test,y_test_predict4)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	132	102
is not Fraud	0	252

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,y_test_predict4))
```

```
              precision    recall  f1-score   support

    is Fraud       1.00      0.56      0.72       234
is not Fraud       0.71      1.00      0.83       252

    accuracy                           0.79       486
   macro avg       0.86      0.78      0.78       486
weighted avg       0.85      0.79      0.78       486
```

```
df.columns
```

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

```
from sklearn.preprocessing import LabelEncoder

la = LabelEncoder()
y_train1 = la.fit_transform(y_train)
```

```
y_test1=la.transform(y_test)
```

```
y_test1=la.transform(y_test)
```

```
y_test1
```

```
array([0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1])
```

```
import xgboost as xgb
xgb1 = xgb.XGBClassifier()
xgb1.fit(x_train, y_train1)

y_test_predict5=xgb1.predict(x_test)
test_accuracy=accuracy_score(y_test1,y_test_predict5)
test_accuracy
```

```
0.9979423868312757
```

```
y_train_predict5=xgb1.predict(x_train)
train_accuracy=accuracy_score(y_train1,y_train_predict5)
train_accuracy
```

```
1.0
```

```
pd.crosstab(y_test1,y_test_predict5)
```

col_0	0	1
row_0		
0	233	1
1	0	252

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test1,y_test_predict5))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       234
           1       1.00      1.00      1.00       252

    accuracy                           1.00       486
   macro avg       1.00      1.00      1.00       486
weighted avg       1.00      1.00      1.00       486
```

Train the selected models with data and evaluate model performance(accuracy,precision).

Compare Models

```
def compareModel():
    print("train accuracy for rfc",accuracy_score(y_train_predict1,y_train))
    print("test accuracy for rfc",accuracy_score(y_test_predict1,y_test))
    print("train accuracy for dtc",accuracy_score(y_train_predict2,y_train))
    print("test accuracy for dtc",accuracy_score(y_test_predict2,y_test))
    print("train accuracy for etc",accuracy_score(y_train_predict3,y_train))
    print("test accuracy for etc",accuracy_score(y_test_predict3,y_test))
    print("train accuracy for svc",accuracy_score(y_train_predict4,y_train))
    print("test accuracy for svcc",accuracy_score(y_test_predict4,y_test))
    print("train accuracy for xgb1",accuracy_score(y_train_predict5,y_train1))
    print("test accuracy for xgb1",accuracy_score(y_test_predict5,y_test1))
```

```
compareModel()
```

```
train accuracy for rfc 1.0
test accuracy for rfc 0.9958847736625515
train accuracy for dtc 1.0
test accuracy for dtc 0.9917695473251029
train accuracy for etc 1.0
test accuracy for etc 0.9938271604938271
train accuracy for svc 0.8009259259259259
test accuracy for svcc 0.7901234567901234
train accuracy for xgb1 1.0
test accuracy for xgb1 0.9979423868312757
```

SPRINT -5 :

Home.html

Predict.html

Submit.html

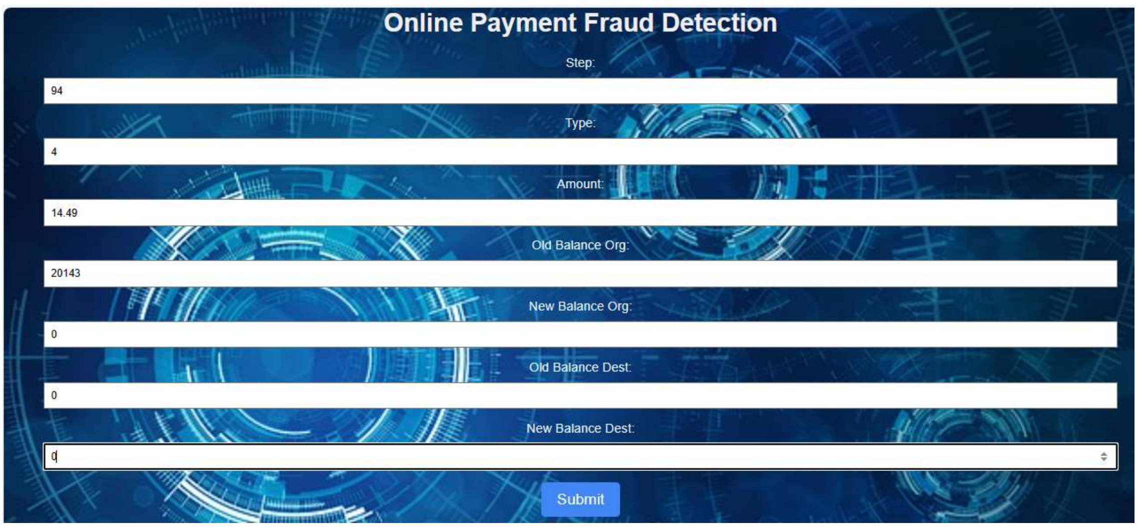

```python
from flask import Flask, render_template, request
import numpy as np
import pickle
import pandas as pd

model = pickle.load(open(r"C:/Users/user/payments.pkl",'rb'))
```

```python
model = pickle.load(open(r"C:/Users/user/payments.pkl",'rb'))

app = Flask(__name__)
```

```python
@app.route("/")
def about():
    return render_template('home.html')

@app.route("/home")
def about1():
    return render_template('home.html')
```

```python
@app.route("/predict")
def home1():
    return render_template('predict.html')

@app.route("/pred", methods=['POST','GET'])
def predict():
   x = [[x for x in request.form.values()]]
   print(x)

   x = np.array(x)
   print(x.shape)


   print(x)
   pred = model.predict(x)
   print(pred[0])
   return render_template('submit.html', prediction_text=str(pred))
```

OUTPUT SCREENSHOTS:




Online Payment Fraud Detection
Step:
36
Type:
3
Amount:
15.76
Old Balance Org:
24000
New Balance Org:
19000
Old Balance Dest:
0
New Balance Dest:
0
Submit


Online Payment Fraud Detection
The predicted fraud for the online payment is ["is not fraud"]