Online Payments Fraud Detection using Machine Learning

By Aravindhan B, Joel Joy Dennis, Saumyarup Guha

Online Payments Fraud Detection using Machine Learning is a proactive approach to identify and prevent fraudulent activities during online transactions. By leveraging historical transaction data, customer behavior patterns, and machine learning algorithms, this project aims to detect potential fraud in real time, ensuring secure and trustworthy online payment experiences for users and businesses alike.

Scenario 1: Real-time Fraud Monitoring

The system continuously monitors online payment transactions in real time. By analyzing transaction features such as transaction amount, location, device information, and user behavior, it can flag suspicious transactions for further investigation, preventing fraudulent activities before they occur.

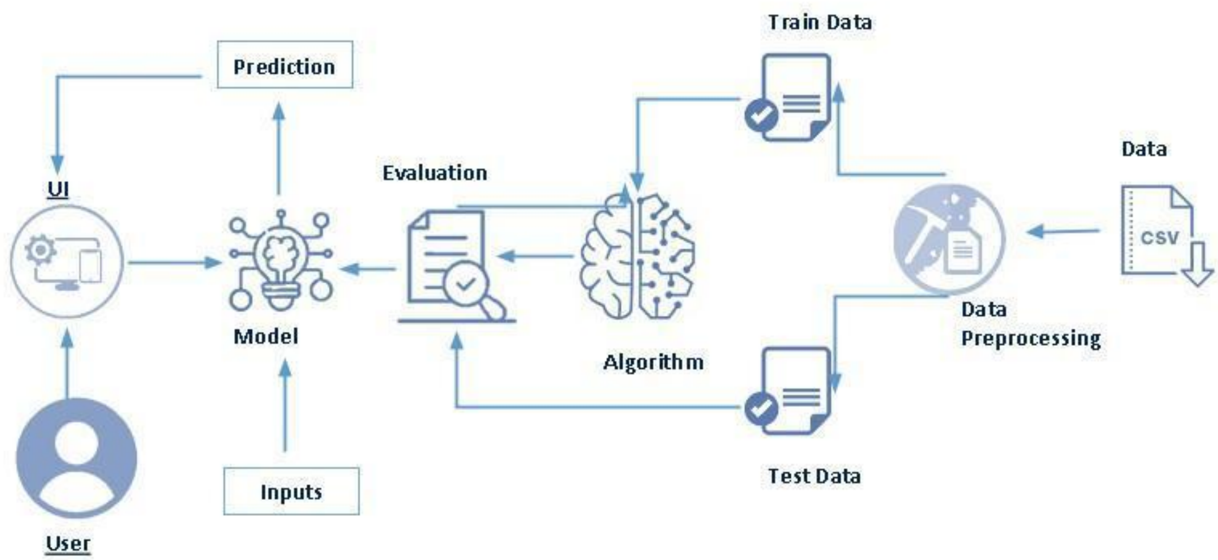
Scenario 2: Fraudulent Account Detection

Machine learning models can detect patterns indicative of fraudulent accounts or activities. By analyzing user behavior over time, such as unusual login times, multiple failed login attempts, or sudden changes in spending patterns, the system can identify and block potentially fraudulent accounts, protecting legitimate users and businesses.

Scenario 3: Adaptive Fraud Prevention

The system adapts and improves its fraud detection capabilities over time. By continuously learning from new data and adjusting its algorithms, it can stay ahead of evolving fraud techniques and trends, providing ongoing protection against online payment fraud for businesses and their customers.

Technical Architecture



Project Flow

- The user interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- The predictions made by the model are showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data collection
  - Collect the dataset or create the dataset
- Data pre-processing
  - Removing unnecessary columns
  - Checking for null values
- Visualizing and analyzing data
  - Univariate analysis
  - Bivariate analysis
  - Descriptive analysis
- Model building
  - Handling categorical values
  - Dividing data into train and test sets
  - Import the model building libraries
  - Comparing the accuracy of various models
  - Hyperparameter tuning of the selected model
  - Evaluating the performance of models
  - Save the model
- Application Building
  - Create an HTML file
  - Build python code

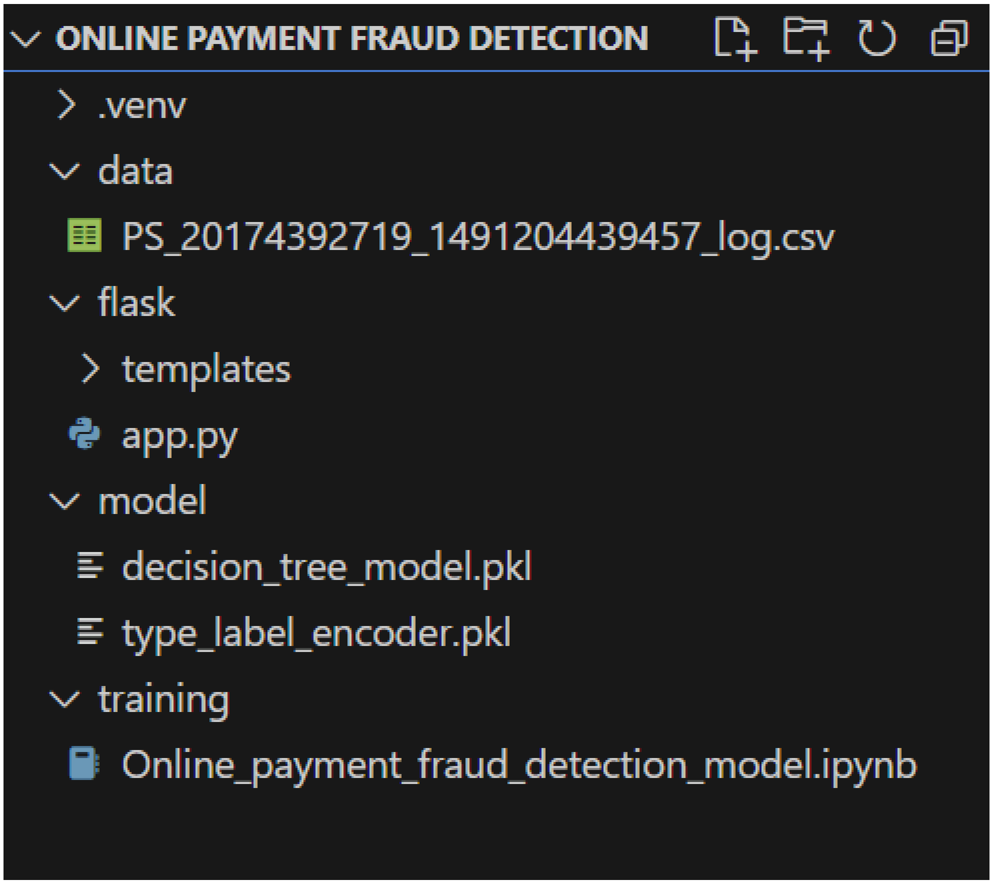
Prior Knowledge

You must have prior knowledge of following topics to complete this project.

- **ML Concepts**
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
  - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classificationalgorithm
  - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
  - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-modelevaluation-error-metrics/
- **Flask Basics**: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- decision_tree_model.pkl is our saved model. Further we will use this model for flask integration.
- Data Folder contains the Dataset used
- The Notebook file contains procedures for building the model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

With the exponential rise in online financial transactions, fraud detection has become a major concern for fintech companies and banks. Manual detection is slow, error-prone, and cannot keep pace with the scale of transactions. The problem is to **accurately and instantly identify fraudulent transactions** in real-time, reducing financial loss and protecting user trust.

Activity 2: Business requirements

To effectively detect fraud in online transactions, the following requirements must be addressed:

- A machine learning model capable of distinguishing between legitimate and fraudulent transactions based on patterns in transaction data.
- A user-friendly web interface that allows input of transaction data and displays instant prediction results.
- Model interpretability and speed to support real-time decision-making.
- Compatibility with structured financial data including transaction amount, balances, and type.

Activity 3: Literature Survey

Various approaches have been proposed for fraud detection:

- Traditional rule-based systems are limited by static logic and high false-positive rates.
- Machine learning techniques like Decision Trees, Random Forests, and XGBoost have been widely adopted due to their ability to learn complex patterns from historical transaction data.

- Research has shown that incorporating features like transaction step, amount, and origin-destination balances significantly improves fraud prediction accuracy.
- Recent studies emphasize the importance of real-time deployment using web applications integrated with trained models.

Activity 4: Social or Business Impact.

The solution provides:

- **Financial security**: Minimizes losses due to fraud, directly benefiting financial institutions and customers.
- **Trust and credibility**: Builds consumer confidence in digital payments.
- **Operational efficiency**: Reduces manual verification workload.
- **Scalability**: Easily extendable to real-world systems with millions of transactions.
- From a social perspective, it promotes **safe adoption of digital finance** in both urban and rural sectors.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used PS_20174392719_1491204439457_logs.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link:https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. Here we have used visualisation style as fivethirtyeight.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,f1_score,classification_report,confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
import xgboost as xgb
import warnings
warnings.filterwarnings('ignore')
import pickle
plt.style.use('ggplot')
```

[57] ✓ 0.0s Python

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```python
data = pd.read_csv(r"D:\Data\Online payment fraud detection\data\PS_20174392719_1491204439457_log.csv")
```

[ ] Python

```python
df = pd.DataFrame(data)

df.head()
```

[66] ✓ 0.0s Python

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

```
df.columns
```
[67] ✓ 0.0s

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

Here, the input features in the dataset are known using the df.columns function.

```
df.drop('isFlaggedFraud', axis=1, inplace=True)
```
[68] ✓ 4.9s

Here, the dataset's superfluous columns are being removed using the drop method.

```
df
```
[69] ✓ 0.0s

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	1
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	1
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0
...	...	...	...	...	...	...	...	...	...	...
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C776919290	0.00	339682.13	1
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C1881841831	0.00	0.00	1
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C1365125890	68488.84	6379898.11	1
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C2080388513	0.00	0.00	1
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C873221189	6510099.11	7360101.63	1

6362620 rows × 10 columns

```
df.head()
```
[70] ✓ 0.0s

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

```
df.head()
```

[70] ✓ 0.0s

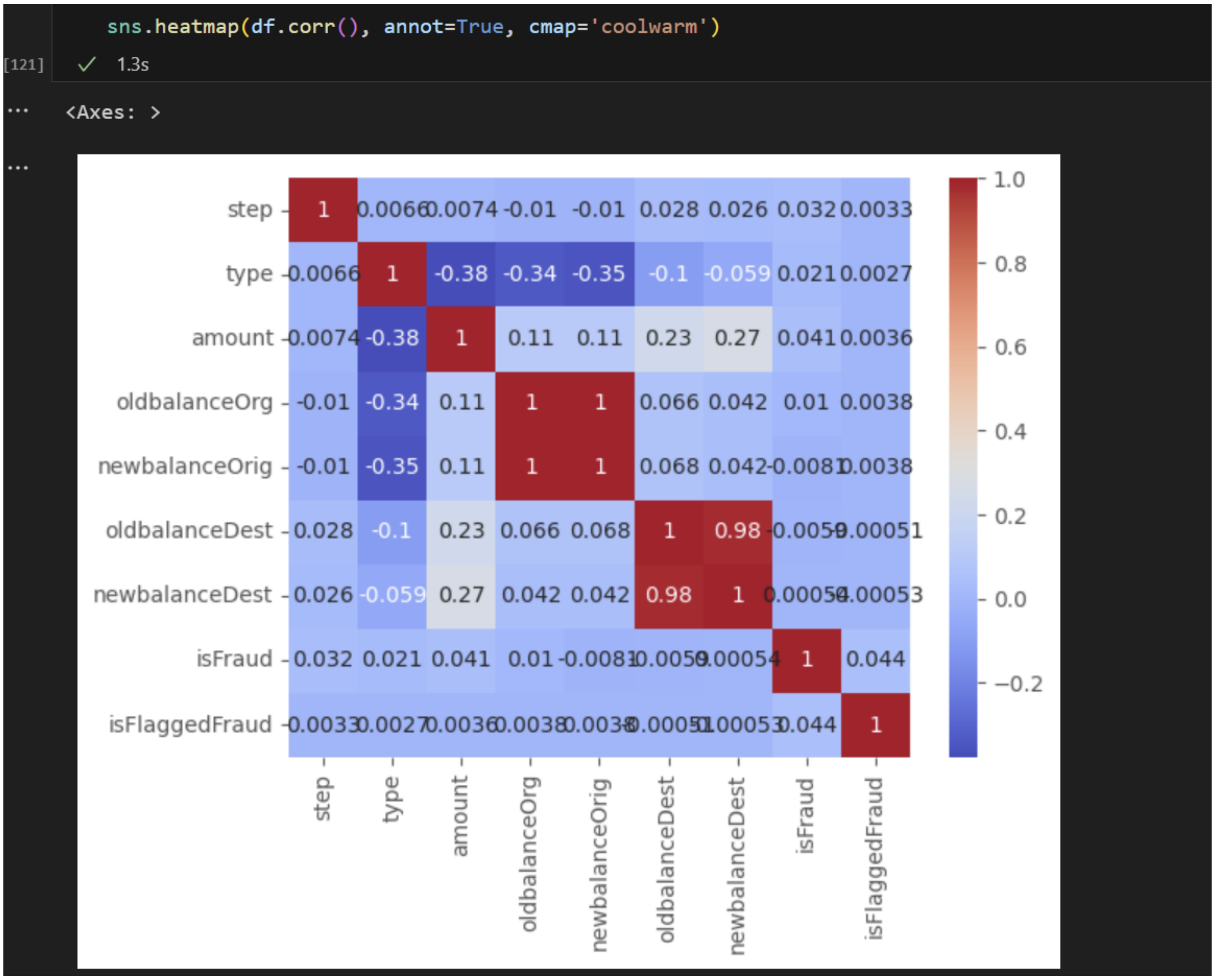
	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0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

```
df.corr()
```

[120] ✓ 0.9s

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
step	1.000000	0.006635	0.007408	-0.010058	-0.010299	0.027665	0.025888	0.031578	0.003277
type	0.006635	1.000000	-0.377490	-0.339760	-0.352758	-0.104679	-0.059364	0.020833	0.002685
amount	0.007408	-0.377490	1.000000	0.106961	0.111432	0.227731	0.265950	0.041085	0.003603
oldbalanceOrg	-0.010058	-0.339760	0.106961	1.000000	0.998803	0.066243	0.042029	0.010154	0.003835
newbalanceOrig	-0.010299	-0.352758	0.111432	0.998803	1.000000	0.067812	0.041837	-0.008148	0.003776
oldbalanceDest	0.027665	-0.104679	0.227731	0.066243	0.067812	1.000000	0.976569	-0.005885	-0.000513
newbalanceDest	0.025888	-0.059364	0.265950	0.042029	0.041837	0.976569	1.000000	0.000535	-0.000529
isFraud	0.031578	0.020833	0.041085	0.010154	-0.008148	-0.005885	0.000535	1.000000	0.044109
isFlaggedFraud	0.003277	0.002685	0.003603	0.003835	0.003776	-0.000513	-0.000529	0.044109	1.000000

utilizing the corr() function to examine the dataset's correlation



Here, a heatmap is used to understand the relationship between the input attributes and the anticipated goal value.

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.
The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling Outliers

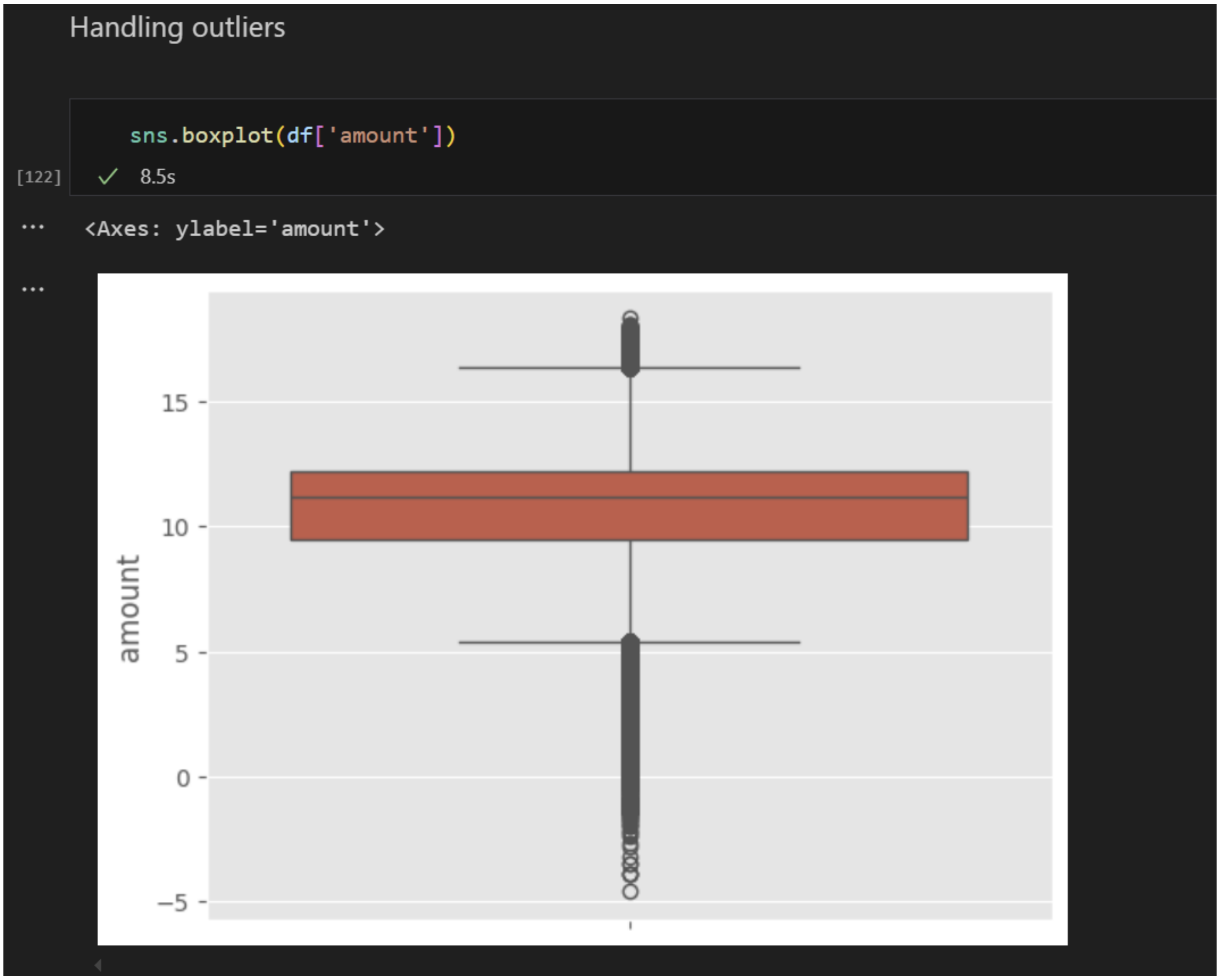
Activity 2.1: Handling missing values

For checking the null values, df.isnull().sum( ) function is used. To sum those null values
We use the .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

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

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

Activity 2.2: Handling Outliers



Here, a box plot is used to identify outliers in the dataset's amount attribute.

To identify and handle outliers in the amount column, the Interquartile Range (IQR) method was used. The first (Q1) and third (Q3) quartiles were computed, and the IQR was calculated as Q3 - Q1. Any transaction amount lying below Q1 - 1.5*IQR or above Q3 + 1.5*IQR was considered an outlier and flagged for further analysis or removal to ensure model robustness.

Removing outliers

```python
from scipy import stats
print(stats.mode(df['amount']))
print(np.mean(df['amount']))
print(np.median(df['amount']))
```

[109] ✓ 0.2s

```
ModeResult(mode=np.float64(10000000.0), count=np.int64(3207))
179861.90354913071
74871.94
```

```python
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)

print(upper_bound)
print(lower_bound)
```

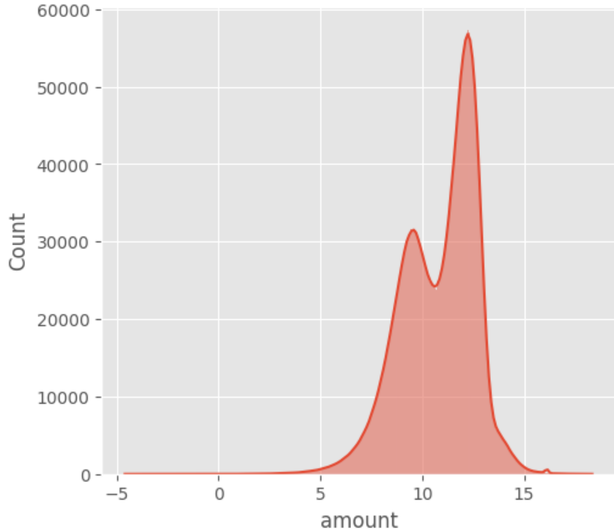
[110] ✓ 0.1s

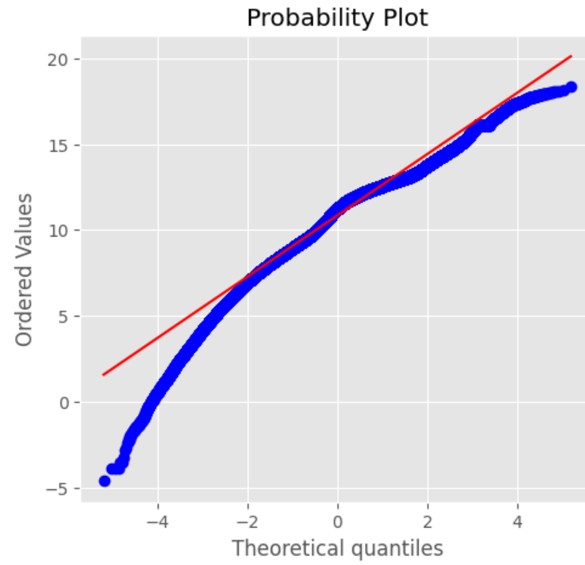
```
501719.33875
-279608.29125
```

```python
def transformationPlot(feature):
    plt.figure(figsize=(12,5))

    plt.subplot(1,2,1)
    sns.histplot(feature, kde=True)

    plt.subplot(1,2,2)
    stats.probplot(feature, plot=plt)

    plt.show()

filtered_amount = df['amount'][df['amount'] > 0]
log_amount = np.log(filtered_amount)

transformationPlot(log_amount)
```

```python
df['amount'] = np.log(df['amount'])
```

✓ 0.0s





Here, transformationPlot is used to plot the data set's outliers for the amount property.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
df.describe(include='all')
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
count	6.362620e+06	6362620	6.362620e+06	6362620	6.362620e+06	6.362620e+06	6362620	6.362620e+06	6.362620e+06	6.362620e+06
unique	NaN	5	NaN	6353307	NaN	NaN	2722362	NaN	NaN	NaN
top	NaN	CASH_OUT	NaN	C1677795071	NaN	NaN	C1286084959	NaN	NaN	NaN
freq	NaN	2237500	NaN	3	NaN	NaN	113	NaN	NaN	NaN
mean	2.433972e+02	NaN	1.798619e+05	NaN	8.338831e+05	8.551137e+05	NaN	1.100702e+06	1.224996e+06	1.290820e-03
std	1.423320e+02	NaN	6.038582e+05	NaN	2.888243e+06	2.924049e+06	NaN	3.399180e+06	3.674129e+06	3.590480e-02
min	1.000000e+00	NaN	0.000000e+00	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.560000e+02	NaN	1.338957e+04	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	0.000000e+00
50%	2.390000e+02	NaN	7.487194e+04	NaN	1.420800e+04	0.000000e+00	NaN	1.327057e+05	2.146614e+05	0.000000e+00
75%	3.350000e+02	NaN	2.087215e+05	NaN	1.073152e+05	1.442584e+05	NaN	9.430367e+05	1.111909e+06	0.000000e+00
max	7.430000e+02	NaN	9.244552e+07	NaN	5.958504e+07	4.958504e+07	NaN	3.560159e+08	3.561793e+08	1.000000e+00

Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

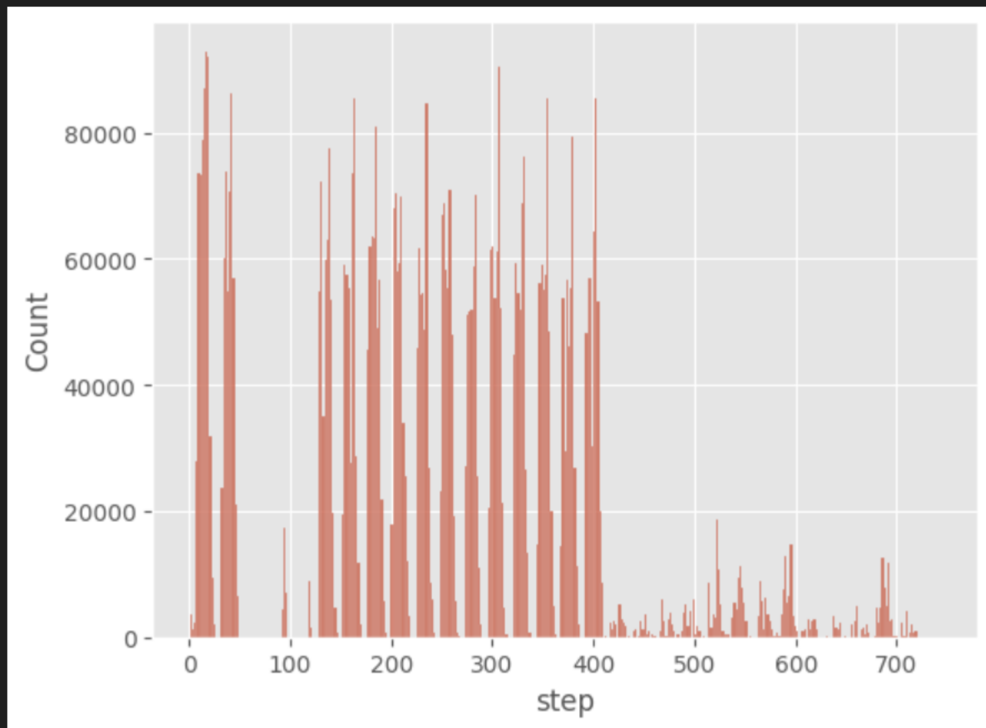
Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature. Here I have displayed the graph such as histplot .

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

[81] ✓ 1.8s

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

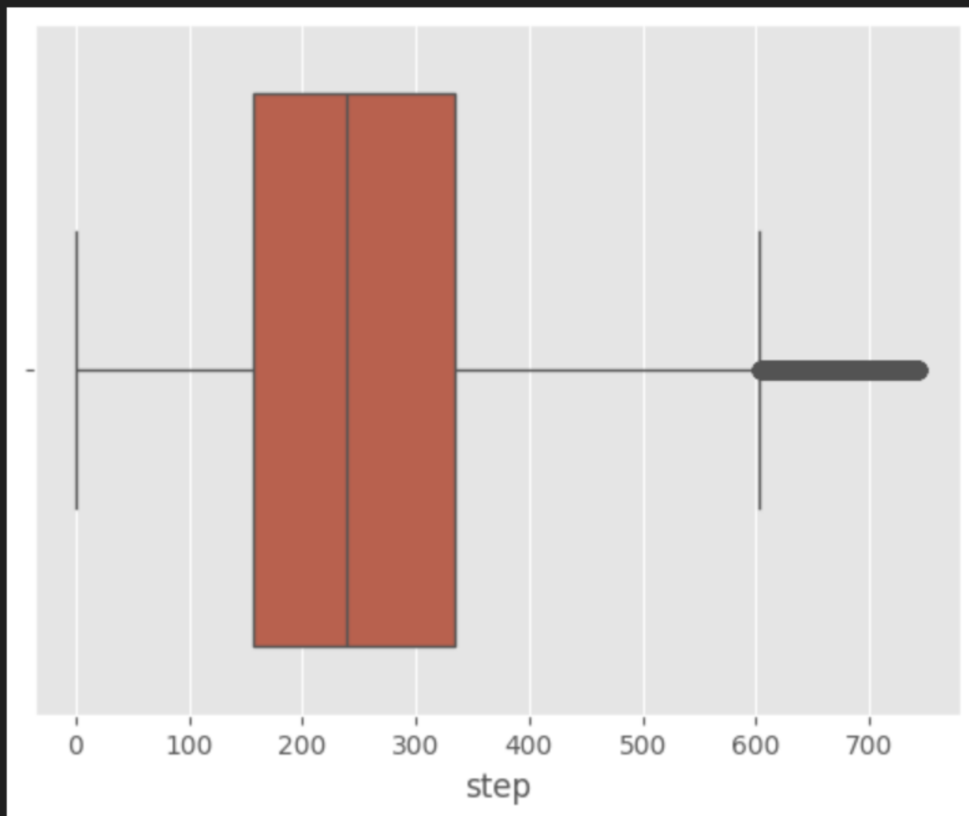


The distribution of one or more variables is represented by a histogram, a traditional visualisation tool, by counting the number of observations that fall within.

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

[82] ✓ 5.1s

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

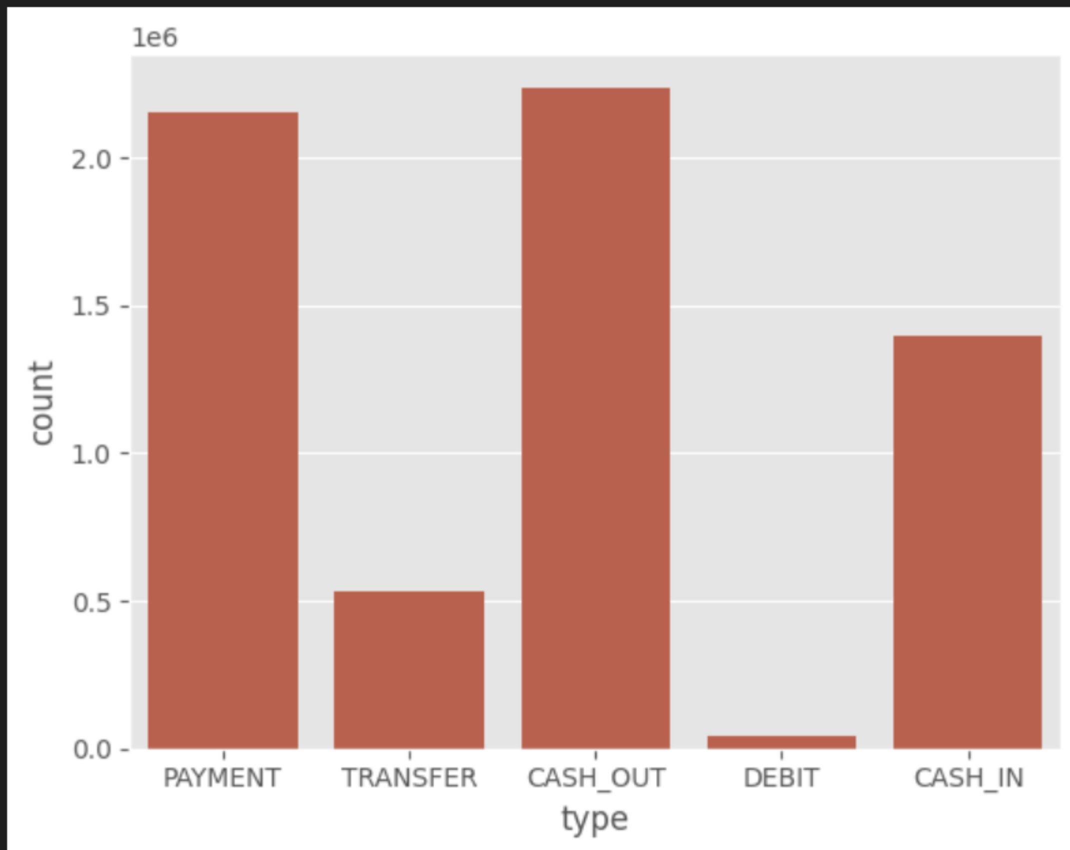


Here, the relationship between the step attribute and the boxplot is visualised.

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

[83] ✓ 5.8s

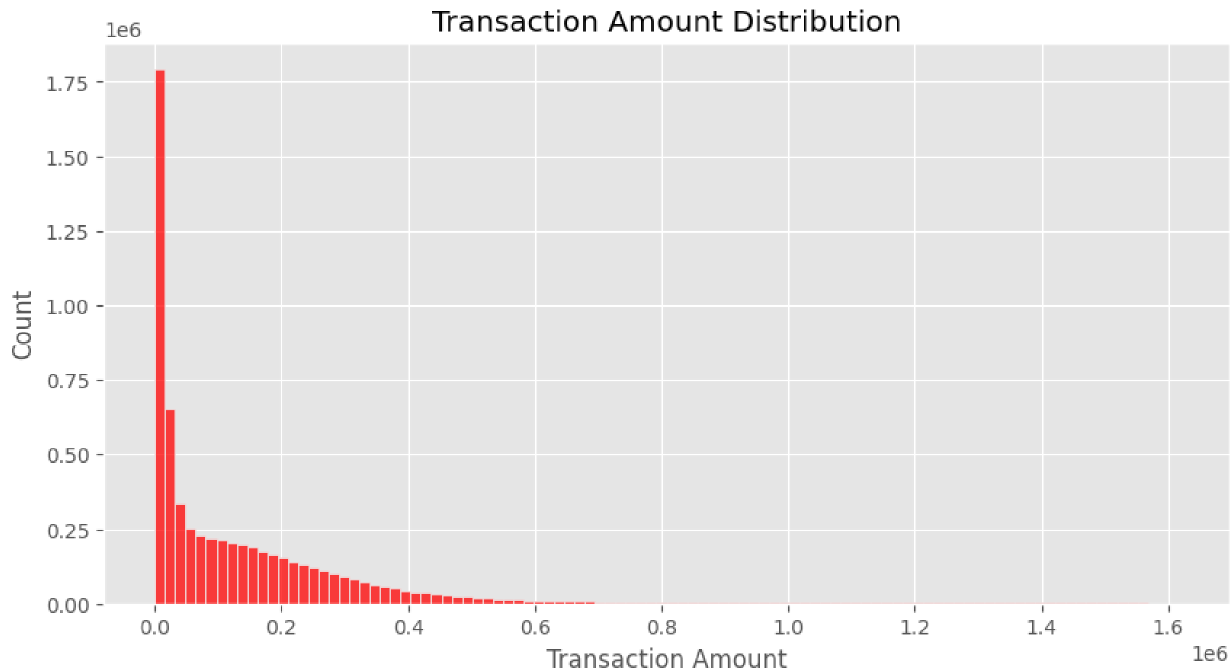
```
<Axes: xlabel='type', ylabel='count'>
```



Here, the counts of observations in the type attribute of the dataset will be displayed using a countplot.

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

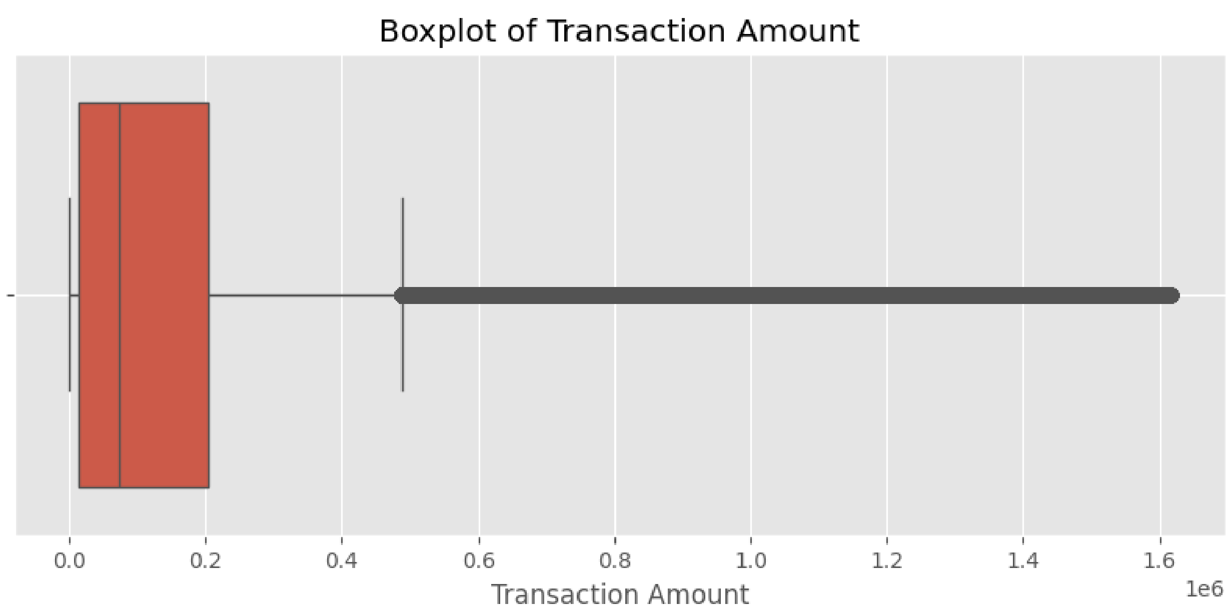
[20] ✓ 2.3s



By creating bins along the data's range and then drawing bars to reflect the number of observations that fall within the amount attribute in the dataset.

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

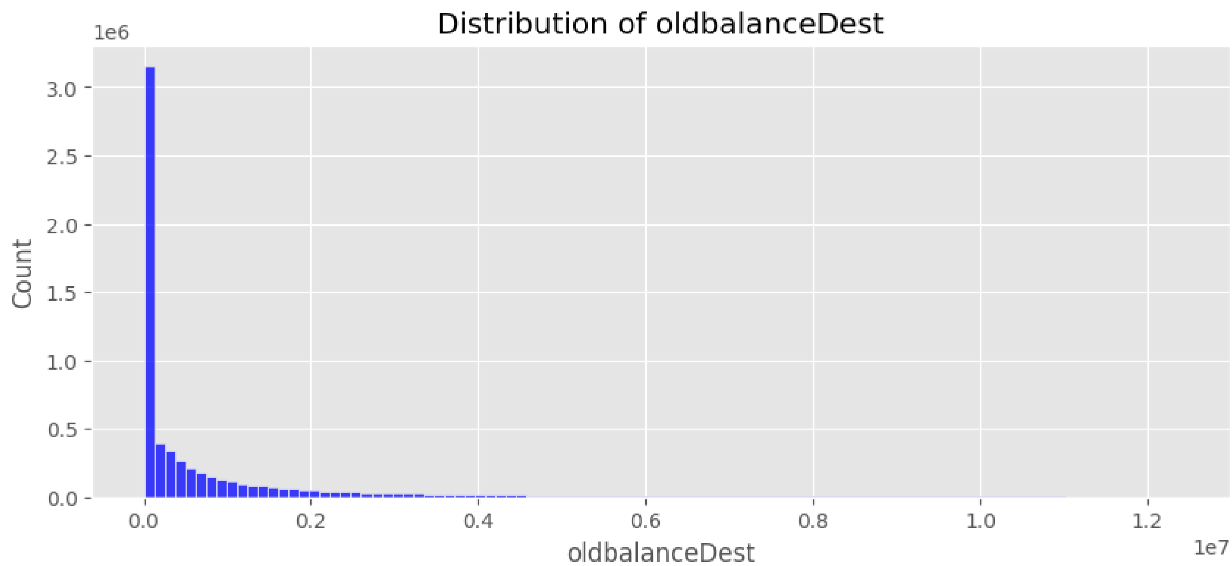
[12] ✓ 6.2s



Here, the relationship between the amount attribute and the boxplot is visualised.

```
sns.histplot(data = df , x = 'oldbalanceDest')
```

[13] ✓ 10.2s



By creating bins along the data's range and then drawing bars to reflect the number of observations that fall within the oldbalanceOrg attribute in the dataset.

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

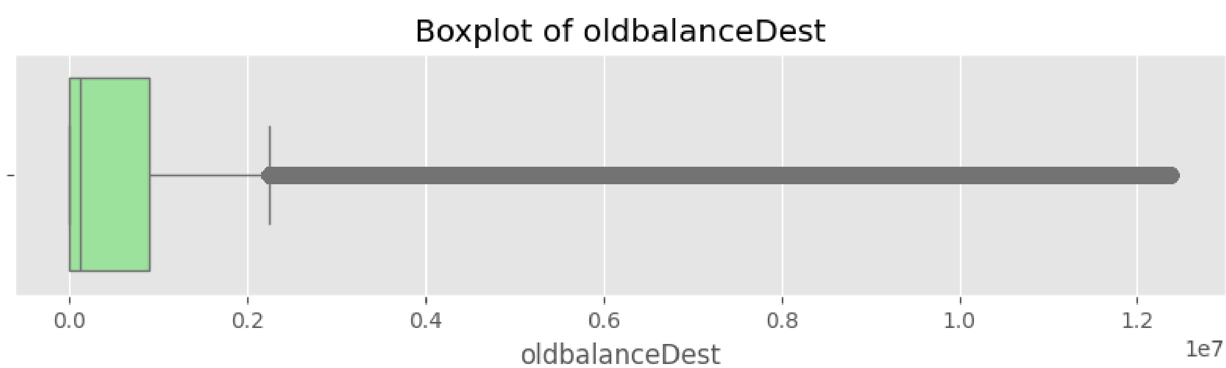
```
[29]  ✓ 3.6s
```

```
nameDest
C1286084959    113
C985934102     109
C665576141     105
C2083562754    102
C248609774     101
                ...
C1049862186      1
C2118381511      1
C2099952089      1
C1027984317      1
C1251365829      1
Name: count, Length: 2722362, dtype: int64
```

utilising the value counts() function here to determine how many times the nameDest column appears.

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

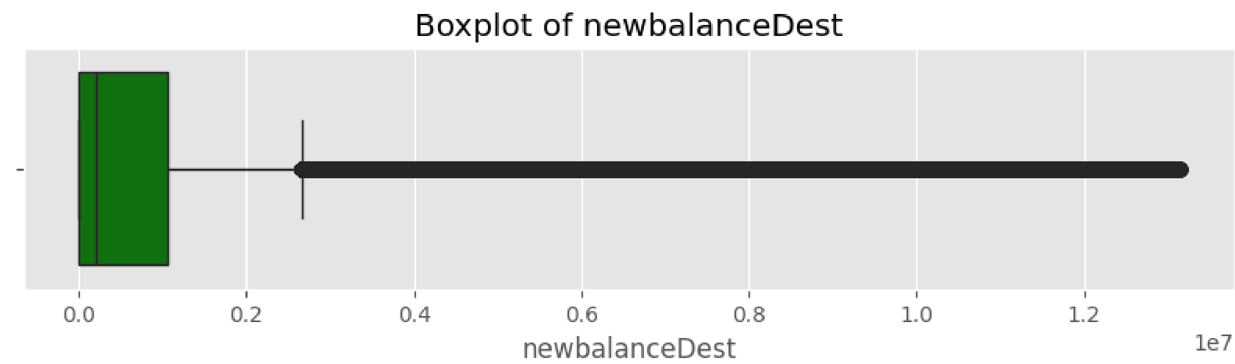
```
✓ 7.4s
```



Here, the relationship between the oldbalanceDest attribute and the boxplot is visualised.

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

✓ 5.9s




```
df.loc[df['isFraud'] == 1,'isFraud'] = 'Fraud'
df.loc[df['isFraud'] == 0,'isFraud'] = 'Not Fraud'
```

[ ]

```
df
```

[ ]

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	Not Fraud
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	Not Fraud
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	Fraud
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	Fraud
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	Not Fraud
...	...	...	...	...	...	...	...	...	...	...
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C776919290	0.00	339682.13	Fraud
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C1881841831	0.00	0.00	Fraud
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C1365125890	68488.84	6379898.11	Fraud
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C2080388513	0.00	0.00	Fraud
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C873221189	6510099.11	7360101.63	Fraud

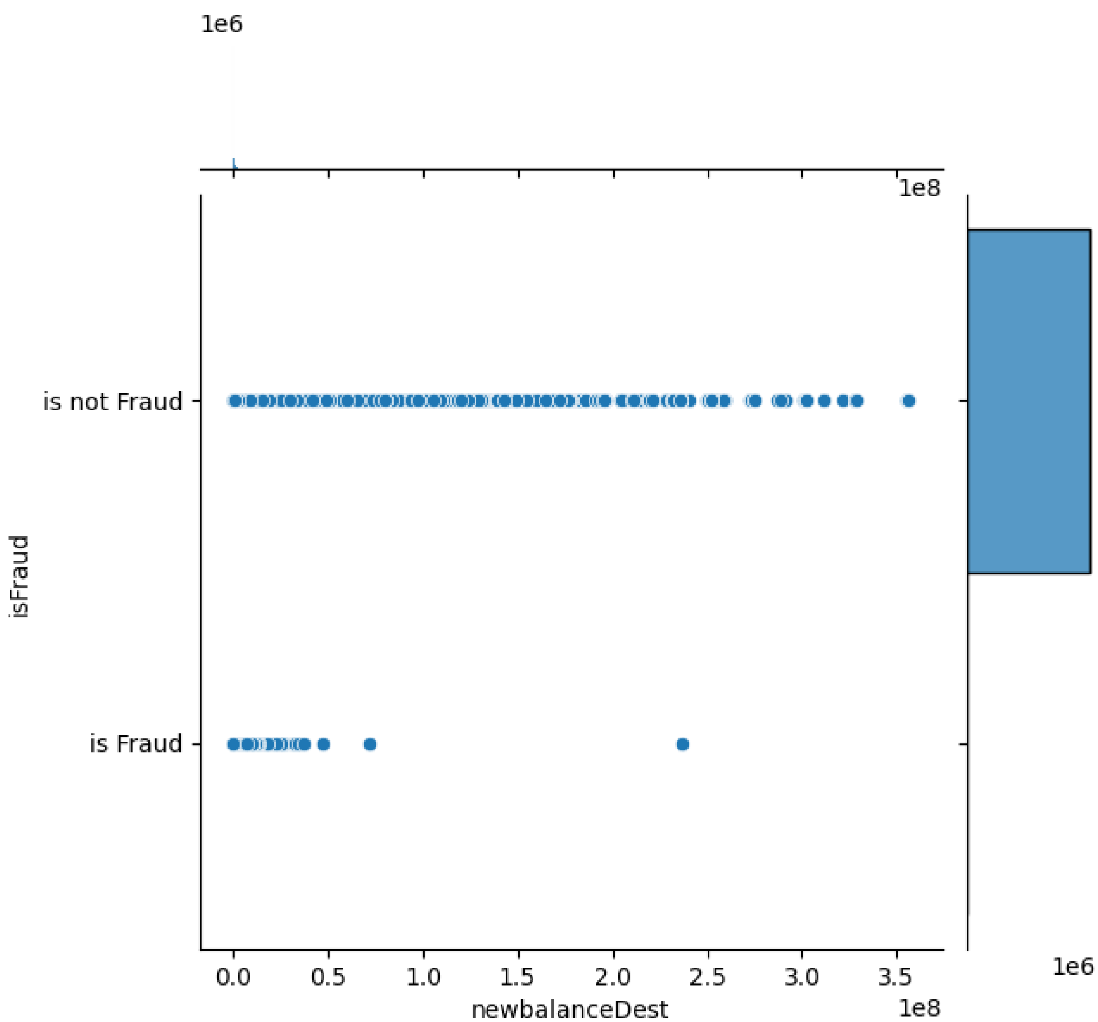
6362620 rows × 10 columns

converting 0-means: is not fraud and 1-means: is fraud using the loc technique here

Activity 2.2: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we are visualising the relationship between newbalanceDest and isFraud. A jointplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

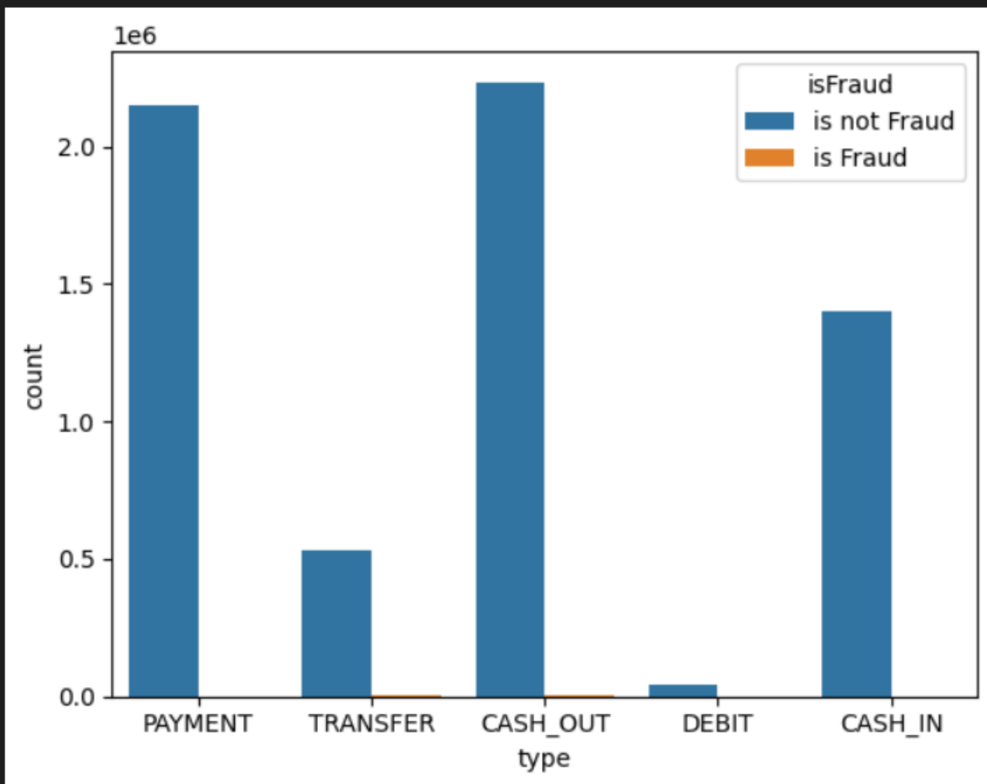
```
sns.jointplot(data=df,x='newbalanceDest',y='isFraud')
```



Here we are visualizing the relationship between type and isFraud.countplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.countplot(data=df,x='type',hue='isFraud')
```

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



Here we are visualizing the relationship between isFraud and step.boxtplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

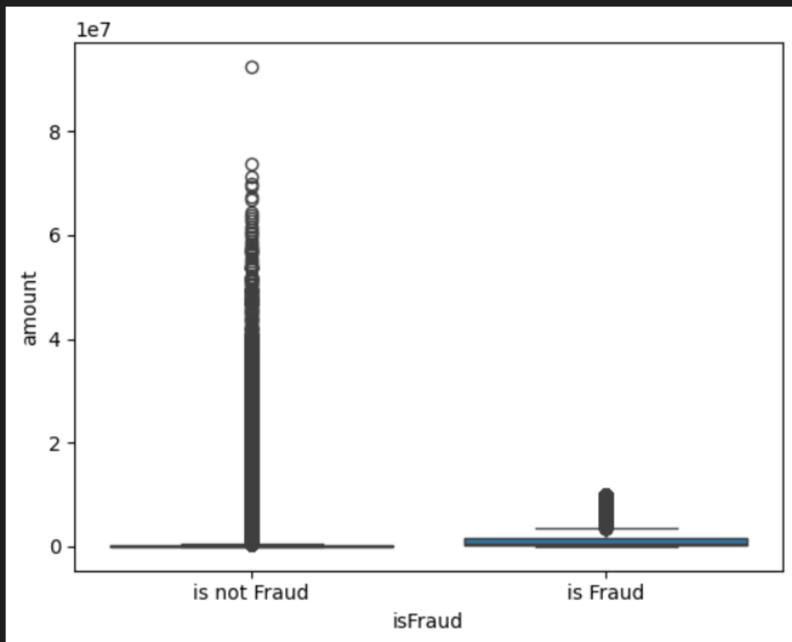
```
sns.boxplot(data=df,x='isFraud',y='step')
```

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

Here we are visualizing the relationship between isFraud and amount. A boxplot is used here. As a 1^{st} parameter we are passing x value and as a 2^{nd} parameter we are passing hue value.

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

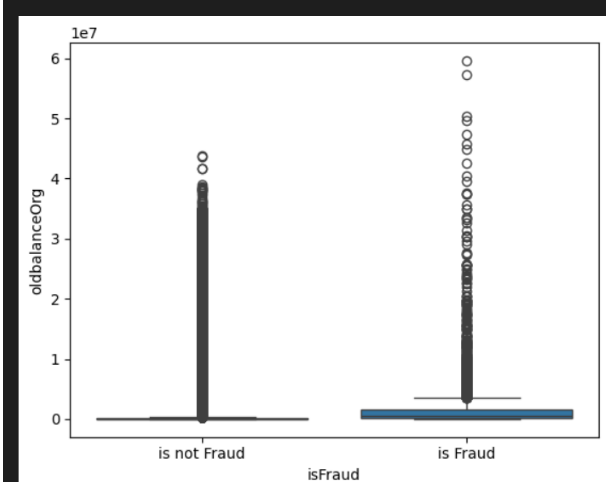
```
<Axes: xlabel='isFraud', ylabel='amount'>
```



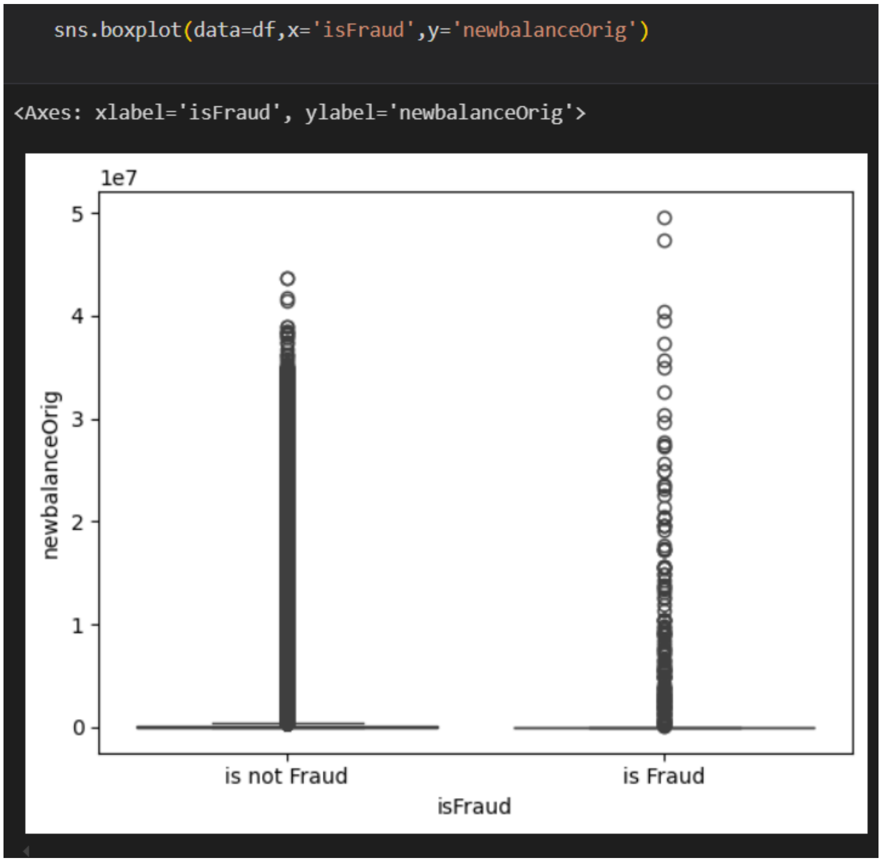
Here we are visualizing the relationship between isFraud and oldbalanceOrg. boxplot is used here. As a 1^{st} parameter we are passing x value and as a 2^{nd} parameter we are passing hue value

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

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



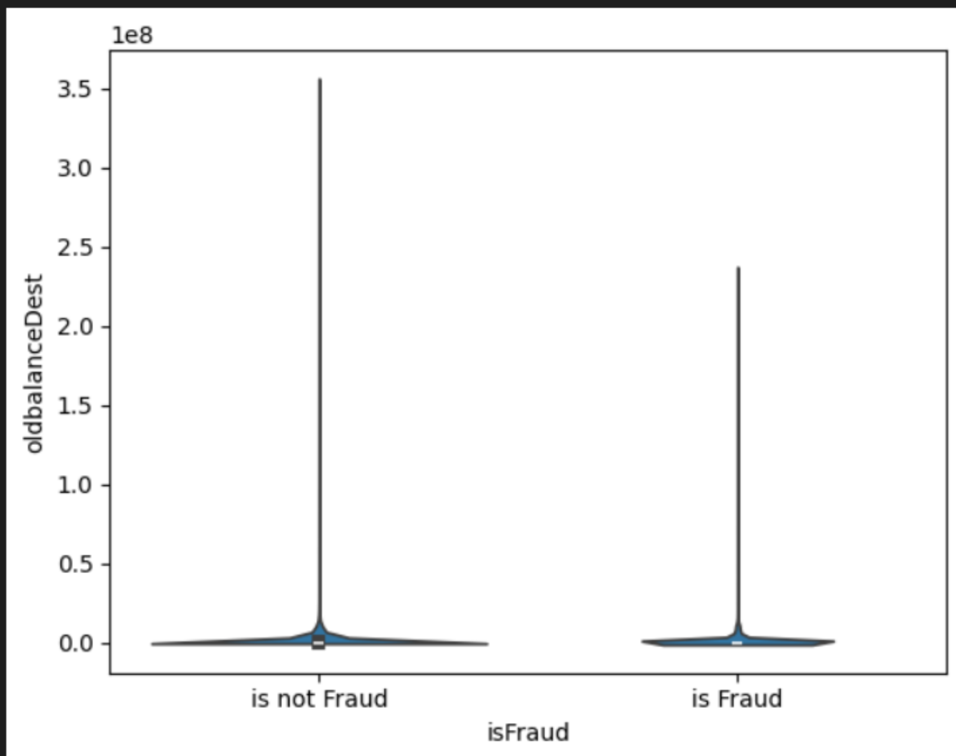
Here we are visualizing the relationship between isFraud and newbalanceOrig. A box plot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.



Here we are visualizing the relationship between isFraud and oldbalanceDest. violinplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.violinplot(data=df,x='isFraud',y='oldbalanceDest')
```

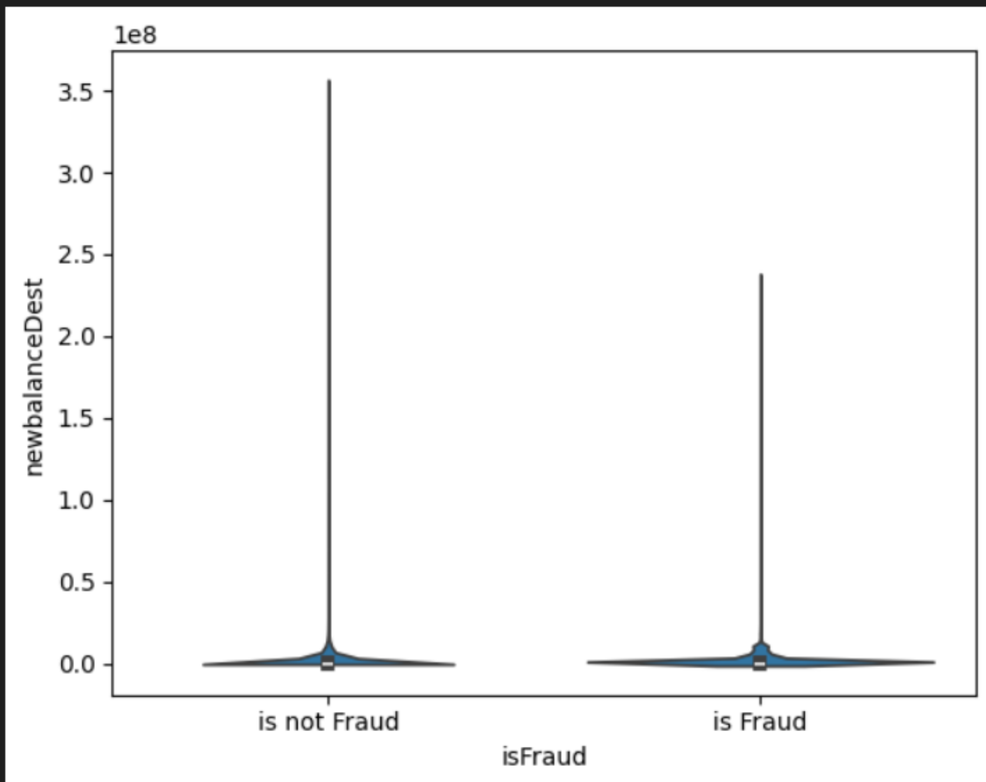
```
<Axes: xlabel='isFraud', ylabel='oldbalanceDest'>
```



Here we are visualizing the relationship between isFraud and newbalanceDest. violinplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.violinplot(data=df,x='isFraud',y='newbalanceDest')
```

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



Activity 3: Data Preprocessing:

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

Handling missing values

Handling Object data label encoding

Splitting dataset into training and test set

```
df.shape
```

```
(6362620, 10)
```

Here, I'm using the shape approach to figure out how big my dataset is.

```
df = df.drop(['nameOrig', 'nameDest'], axis=1)
```

✓ 0.2s

```
df.head()
```

✓ 0.0s

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	170136.0	160296.36	0.0	0.0	is not Fraud
1	1	PAYMENT	1864.28	21249.0	19384.72	0.0	0.0	is not Fraud
2	1	TRANSFER	181.00	181.0	0.00	0.0	0.0	is Fraud
3	1	CASH_OUT	181.00	181.0	0.00	21182.0	0.0	is Fraud
4	1	PAYMENT	11668.14	41554.0	29885.86	0.0	0.0	is not Fraud

Here, the dataset's superfluous columns (nameOrig,nameDest) are being removed using the drop method.

Activity 3.1: Checking for Null Values:

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

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

For checking the null values, data.isnull() function is used. To sum those null values we use the .sum() function to it. From the above image we found that there are no null values present in our dataset.So we can skip handling of missing values step.

```
df.info()
```

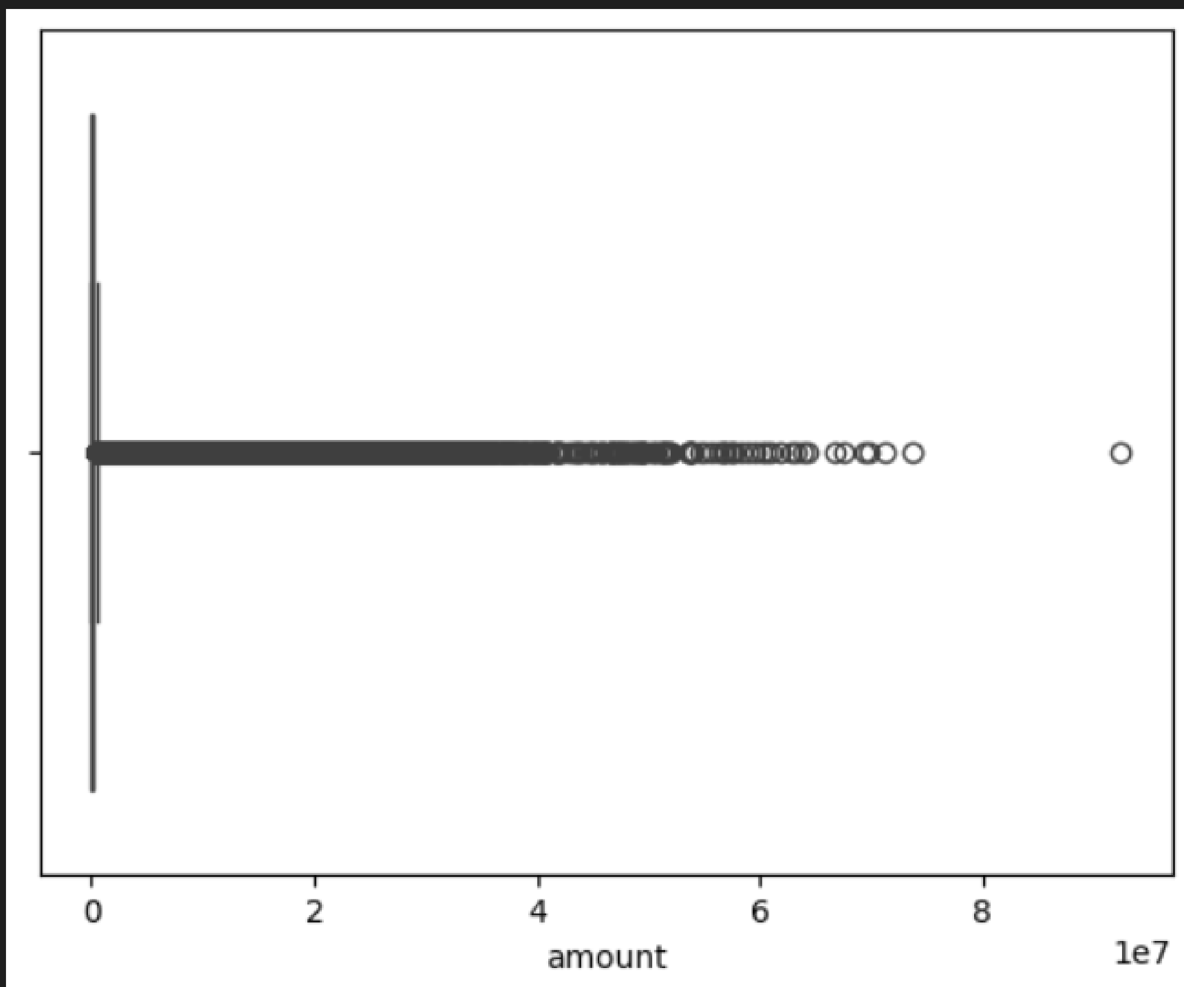
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   Unnamed: 0      int64
 1   step            int64
 2   type            object
 3   amount          float64
 4   nameOrig     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

determining the types of each attribute in the dataset using the info() function

Activity 3.2: Handling Outliers

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

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



Here, a box plot is used to identify outliers in the dataset's amount attribute.
Removing the outliers:

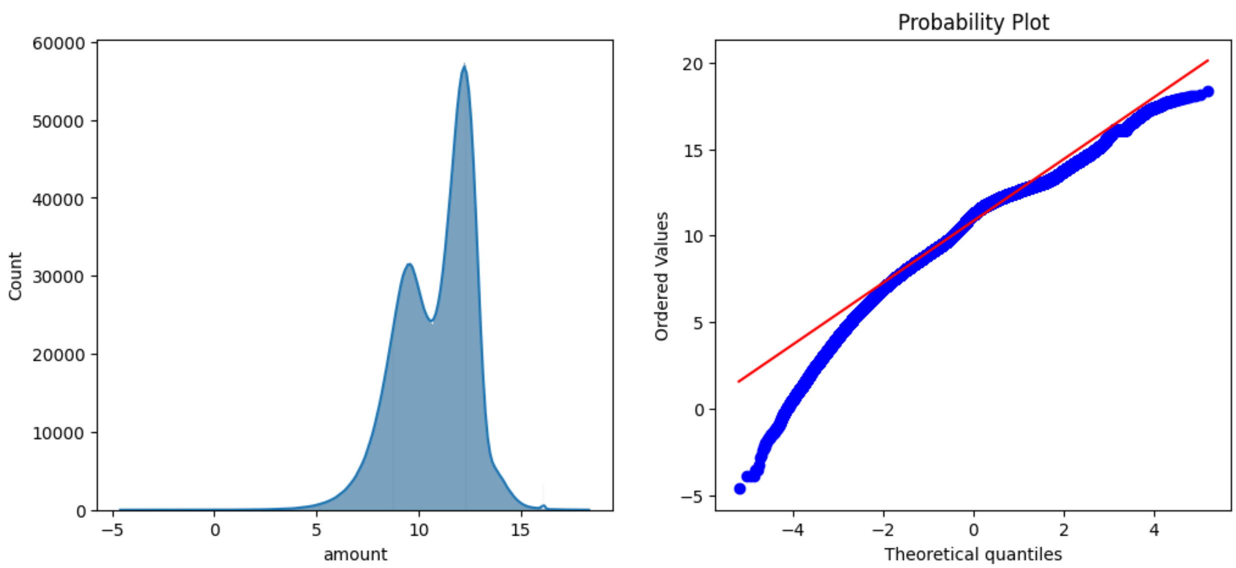
```python
#removing outliers
from scipy import stats
print(stats.mode(df['amount']))
print(np.mean(df['amount']))
```

```
ModeResult(mode=np.float64(10000000.0), count=np.int64(3207))
179861.90354913071
```

```python
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)
print('Q1: ',q1)
print('Q3: ',q3)
print('IQR: ',IQR)
print('Upper Bound: ',upper_bound)
print('Lower Bound: ',lower_bound)
print('Skewed Data: ',len(df[df['amount']>upper_bound]))
print('Skewed Data: ',len(df[df['amount']<lower_bound]))
```

```
Q1:  13389.57
Q3:  208721.4775
IQR:  195331.9075
Upper Bound:  501719.33875
Lower Bound:  -279608.29125
Skewed Data:  338078
Skewed Data:  0
```

```
def transformationPlot(feature):
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    sns.histplot(feature, kde=True)
    plt.subplot(1,2,2)
    stats.probplot(feature, plot=plt)
    plt.show()

filtered_amount = df['amount'][df['amount'] > 0]
log_amount = np.log(filtered_amount)

transformationPlot(log_amount)
```



Here, transformationPlot is used to plot the dataset's outliers for the amount property.

Activity 3.3: Object Data Label Encoding

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

```
type
1    2237500
3    2151495
0    1399284
4     532909
2      41432
Name: count, dtype: int64
```

using labelencoder to encode the dataset's object type

Activity 3.3.1: Dividing the dataset into dependent and independent y and x respectively

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

✓ 0.7s

```
x
```

✓ 0.0s

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

6362620 rows × 7 columns

```
y
✓ 0.0s

0          is not Fraud
1          is not Fraud
2              is Fraud
3              is Fraud
4          is not Fraud
                  ...
6362615        is Fraud
6362616        is Fraud
6362617        is Fraud
6362618        is Fraud
6362619        is Fraud
Name: isFraud, Length: 6362620, dtype: object
```

Activity 3.4: Splitting the dataset into train and test

Now let’s split the Dataset into train and test setsChanges: first split the dataset into x and y and then split the data set.

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And my target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```python
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0, test_size=0.2)

print(x_train.shape)
print(x_test.shape)
print(y_test.shape)
print(y_train.shape)
```

✓ 2.2s

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

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best model is saved based on its performance.

Activity 1.1: Random Forest Classifier

We used a Random Forest Classifier for fraud detection due to its robustness and high accuracy. The model was configured with 100 trees, a maximum depth of 15 to prevent overfitting, and a minimum of 10 samples required to split a node. Training was parallelized across all CPU cores for efficiency, and a fixed random state ensured reproducibility. This setup helped the model effectively learn patterns in the transaction data to detect fraud reliably.

```
rfc = RandomForestClassifier(
    n_estimators=100,
    max_depth=15,
    min_samples_split=10,
    n_jobs=-1,
    random_state=42,
    verbose=1
)
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Activity 1.2: Decision tree classifier

We implemented a **Decision Tree Classifier** as a baseline model for fraud detection. Using a fixed random_state ensured reproducibility. The model was trained on transaction features to learn simple, interpretable decision rules that help distinguish between fraudulent and legitimate activities.

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import numpy as np

dtc = DecisionTreeClassifier(random_state=42)
dtc.fit(X_train, y_train)
y_test_pred2 = dtc.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred2)
print("Accuracy:", accuracy)
```

Activity 1.3: Extra tree classifier

We used an **Extra Trees Classifier** to enhance model robustness and accuracy. With multiple randomized decision trees (n_estimators=50) and parallel processing enabled, the model efficiently captured complex patterns in the transaction data to differentiate between fraudulent and legitimate behavior.

```python
etc = ExtraTreesClassifier(
    n_estimators=50,
    max_depth=None,
    min_samples_split=5,
    n_jobs=-1,
    random_state=42,
    verbose=1
)

etc.fit(X_train, y_train)

y_test_pred3 = etc.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred3)
print("Accuracy:", accuracy)
```

Activity 1.4: Linear SVC

We employed a **Linear Support Vector Classifier (SVC)** to classify fraudulent transactions. Using a linear kernel with a regularization parameter C=1.0 and increased max_iter=10000, the model aimed to find the optimal hyperplane that separates legitimate and fraudulent transactions in high-dimensional space.

```
from sklearn.svm import LinearSVC

svc = LinearSVC(C=1.0, max_iter=10000)
svc.fit(X_train, y_train)
y_test_pred4 = svc.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred4)
print("Accuracy:", accuracy)
```

Activity 1.5: XGBoost

We used an **XGBoost Classifier**, a powerful gradient boosting framework, to enhance prediction accuracy for fraud detection. The model combines multiple weak learners to create a strong predictive model, automatically handling missing values and offering robustness against overfitting.

```
import xgboost as xgb

xgb1 = xgb.XGBClassifier()
xgb1.fit(X_train, y_train)
y_test_pred5 = xgb1.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred5)
print("Accuracy:", accuracy)
```

Activity 2: Testing the model

Here we have tested with the Decision Tree algorithm. You can test with all algorithms. With with the help of the predict() function.

```python
y_train_predict2 = dtc.predict(X_train)
print(classification_report(y_train,y_train_predict2))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   5083503
           1       1.00      1.00      1.00      6593

    accuracy                           1.00   5090096
   macro avg       1.00      1.00      1.00   5090096
weighted avg       1.00      1.00      1.00   5090096
```

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Using multiple evaluation metrics involves assessing a model's performance on the test data through various measurement techniques. This helps gain a more complete picture of the model's strengths and limitations. For classification tasks, we evaluate the model using metrics such as accuracy, precision, recall, support, and the F1-score.

Activity 1.1: Compare the model

Comparing the five models used:

```
# prompt: print all the model accuracy above

print("Random Forest Test Accuracy:", accuracy_score(y_test, y_pred))
print("Decision Tree Test Accuracy:", accuracy_score(y_test, y_test_pred2))
print("Extra Trees Test Accuracy:", accuracy_score(y_test, y_test_pred3))
print("Linear SVC Test Accuracy:", accuracy_score(y_test, y_test_pred4))
print("XGBoost Test Accuracy:", accuracy_score(y_test, y_test_pred5))
```

```
Random Forest Test Accuracy: 0.9991599372585507
Decision Tree Test Accuracy: 0.9997163118338043
Extra Trees Test Accuracy: 0.9996895932807555
Linear SVC Test Accuracy: 0.9991552222197774
XGBoost Test Accuracy: 0.9975882576674389
```

Random Forest

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import numpy as np

rfc = RandomForestClassifier(
    n_estimators=100,       # Number of trees
    max_depth=15,           # Limit depth to speed up and reduce overfitting
    min_samples_split=10,  # Don't split small nodes
    n_jobs=-1,              # Use all CPU cores
    random_state=42,
    verbose=1
)
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  4.5min finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.7s
Accuracy: 0.999684878241982
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    1.6s finished
```

```python
y_train_predict1 = rfc.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_predict1)
print("Train Accuracy:", train_accuracy)
print(classification_report(y_train,y_train_predict1))
```

```
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    3.1s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    7.1s finished
Train Accuracy: 0.9996878251412155
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   5083503
           1       1.00      0.76      0.86      6593

    accuracy                           1.00   5090096
   macro avg       1.00      0.88      0.93   5090096
weighted avg       1.00      1.00      1.00   5090096
```

```python
pd.crosstab(y_test,y_pred)
```

col_0	0	1
isFraud		
0	1270892	12
1	389	1231

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import numpy as np

dtc = DecisionTreeClassifier(random_state=42)
dtc.fit(X_train, y_train)
y_test_pred2 = dtc.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred2)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9997163118338043
```

```
y_train_predict2 = dtc.predict(X_train)
print(classification_report(y_train,y_train_predict2))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   5083503
           1       1.00      1.00      1.00      6593

    accuracy                           1.00   5090096
   macro avg       1.00      1.00      1.00   5090096
weighted avg       1.00      1.00      1.00   5090096
```

```
pd.crosstab(y_test,y_test_pred2)
```

col_0	0	1
isFraud		
0	1270751	153
1	208	1412

Extra Trees

```python
etc = ExtraTreesClassifier(
    n_estimators=50,
    max_depth=None,
    min_samples_split=5,
    n_jobs=-1,
    random_state=42,
    verbose=1
)

etc.fit(X_train, y_train)

y_test_pred3 = etc.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred3)
print("Accuracy:", accuracy)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   39.5s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   45.3s finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    1.4s
Accuracy: 0.9996895932807555
[Parallel(n_jobs=4)]: Done  50 out of  50 | elapsed:    1.6s finished
```

```python
y_train_predict3 = etc.predict(X_train)
print(classification_report(y_train,y_train_predict3))
```

```
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    6.0s
[Parallel(n_jobs=4)]: Done  50 out of  50 | elapsed:    7.0s finished
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   5083503
           1       1.00      0.95      0.97      6593

    accuracy                           1.00   5090096
   macro avg       1.00      0.97      0.99   5090096
weighted avg       1.00      1.00      1.00   5090096
```

```python
pd.crosstab(y_test,y_test_pred3)
```

col_0	0	1
isFraud		
0	1270897	7
1	388	1232

Linear SVC

```
from sklearn.svm import LinearSVC

svc = LinearSVC(C=1.0, max_iter=10000)
svc.fit(X_train, y_train)
y_test_pred4 = svc.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred4)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9991552222197774
```

```
y_train_predict4 = svc.predict(X_train)
print(classification_report(y_train,y_train_predict4))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   5083503
           1       0.96      0.37      0.53      6593

    accuracy                           1.00   5090096
   macro avg       0.98      0.68      0.76   5090096
weighted avg       1.00      1.00      1.00   5090096
```

```
pd.crosstab(y_test,y_test_pred4)
```

col_0	0	1
isFraud		
0	1270878	26
1	1049	571

XG Boost

```python
import xgboost as xgb

xgb1 = xgb.XGBClassifier()
xgb1.fit(X_train, y_train)
y_test_pred5 = xgb1.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred5)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9975882576674389
```

```python
y_train_predict5 = xgb1.predict(X_train)
print(accuracy_score(y_train,y_train_predict5))
print(classification_report(y_train,y_train_predict5))
```

```
0.99761458330059
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   5083503
           1       0.32      0.73      0.44      6593

    accuracy                           1.00   5090096
   macro avg       0.66      0.87      0.72   5090096
weighted avg       1.00      1.00      1.00   5090096
```

```python
pd.crosstab(y_test,y_test_pred5)
```

col_0	0	1
isFraud		
0	1268264	2640
1	429	1191

Activity 2: Comparing model accuracy before & after applying hyperparameter tuning

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after evaluating its performance with various metrics means choosing the one that performs the best overall. This approach helps avoid retraining the model each time it's needed and ensures it can be reused efficiently in the future.

```
# Load the trained model and label encoder
model = pickle.load(open('decision_tree_model.pkl', 'rb'))
type_encoder = pickle.load(open('type_label_encoder.pkl', 'rb'))
```

Activity 2: Integrate with Web Framework

In this section, we'll develop a web application that connects to the machine learning model we previously built. The user interface allows users to input values for prediction. These inputs are passed to the saved model, and the predicted result is displayed on the interface.

This section includes the following steps:

- Creating HTML pages
- Writing the server-side script
- Running the web application

Activity 2.1: Building Html Page:

For this project, we have created the following files and saved them in the templates folder:

- home.html
- submit.html
- predict.html

Activity 2.2: Build Python code:

Importing the libraries:

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

To begin, load the saved model. It's essential to import the Flask module into the project. We create an instance of the Flask class, which serves as our WSGI application. The constructor of the Flask class takes the current module's name (__name__) as its argument.

```python
# Load the trained model and label encoder
model = pickle.load(open('decision_tree_model.pkl', 'rb'))
type_encoder = pickle.load(open('type_label_encoder.pkl', 'rb'))

app = Flask(__name__)
```

Render HTML Pages:

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

@app.route("/predict")
def predict_page():
    return render_template('predict.html')

@app.route("/pred", methods=["POST"])
```

Here, the @app.route("/pred", methods=["POST"]) decorator links the /pred URL to the predict() function, which is triggered when the user submits the form using the POST method. The form values—such as step, type, amount, and balances—are retrieved using request.form and converted to numeric format. The transaction type is encoded using a saved LabelEncoder since the model expects numeric input. All values are then arranged into a NumPy array and passed to the trained model (loaded with pickle) for prediction. Based on the result, a message indicating whether the transaction is legitimate or fraudulent is shown on the submit.html page. Any errors are caught and displayed using a try-except block.

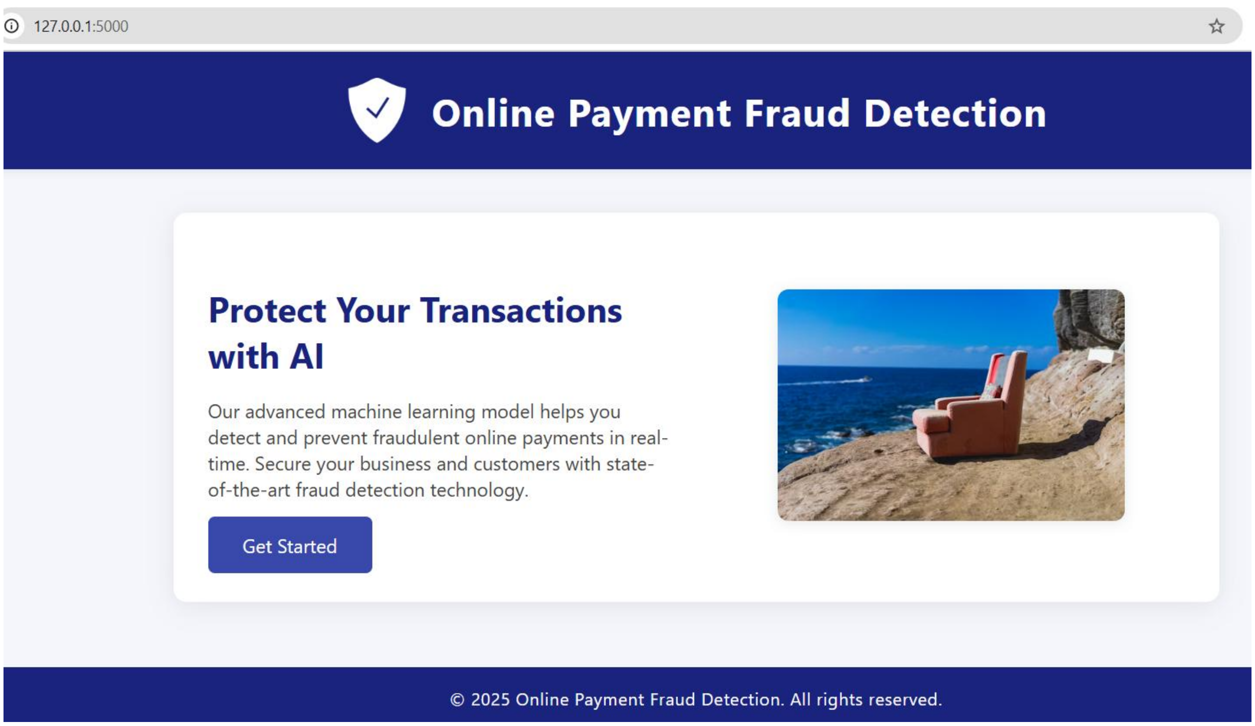
```python
def predict():
    try:
        # Get inputs from form
        step = float(request.form['step'])
        type_str = request.form['type']
        amount = float(request.form['amount'])
        oldbalanceOrg = float(request.form['oldbalanceOrg'])
        newbalanceOrig = float(request.form['newbalanceOrig'])
        oldbalanceDest = float(request.form['oldbalanceDest'])
        newbalanceDest = float(request.form['newbalanceDest'])

        # Encode type using LabelEncoder
        type_encoded = type_encoder.transform([type_str])[0]

        # Arrange input in correct order
        input_data = np.array([[step, type_encoded, amount, oldbalanceOrg,
                                newbalanceOrig, oldbalanceDest, newbalanceDest]])

        # Predict
        prediction = model.predict(input_data)[0]
        result = "fraudulent" if prediction == 1 else "legitimate"

        return render_template("submit.html", prediction_text=f"The transaction is {result}.")

    except Exception as e:
        return f"Error occurred: {str(e)}"
```

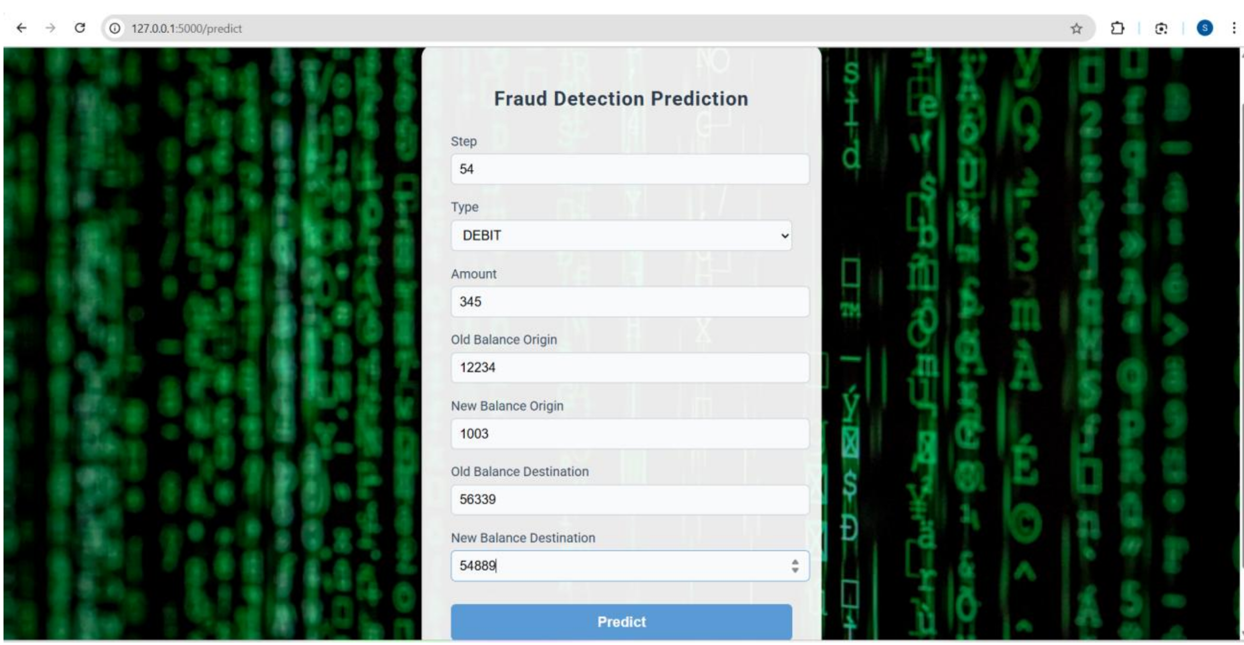
Here we are using the if __name__ == "__main__": block to run our Flask application. This ensures that the app runs only when the script is executed directly, not when it's imported elsewhere. Inside this block, app.run(debug=True, port=5000) starts the local development server on port 5000. The debug=True enables debug mode, which helps in identifying errors and automatically reloads the app when changes are made in the code.

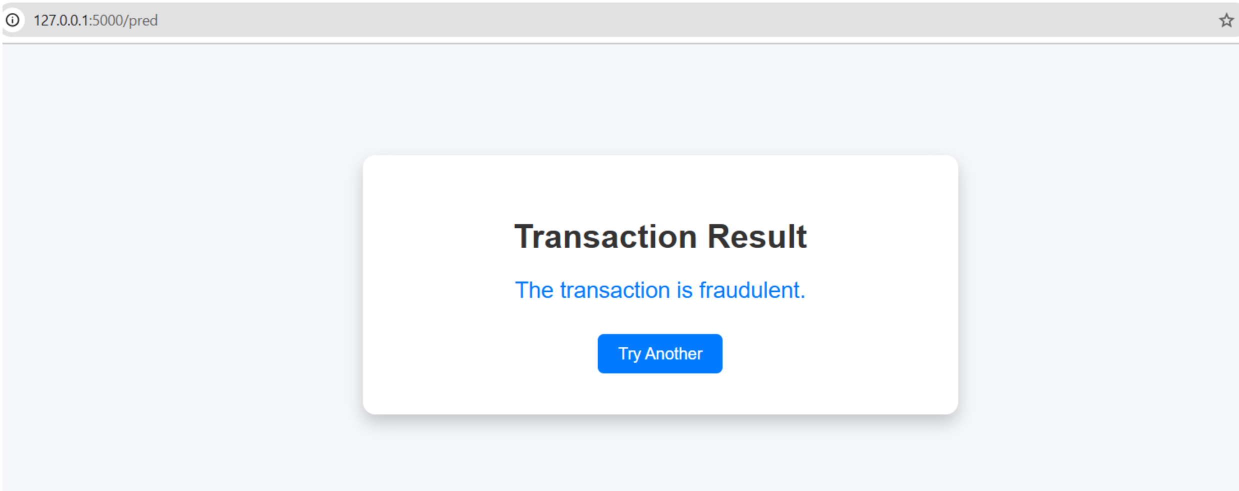
```python
if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

Activity 2.3: Run the web application

```
C:\Users\VICTUS\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator
 LabelEncoder from version 1.6.1 when using version 1.5.2. This might lead to breaking code or invalid results. Use at your own risk. For more info ple
ase refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
C:\Users\VICTUS\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator
 DecisionTreeClassifier from version 1.6.1 when using version 1.5.2. This might lead to breaking code or invalid results. Use at your own risk. For mor
e info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
```







Milestone 7: Project Demonstration & Documentation