Online Payments fraud Detection using Machine Learning

Project Description:

Online payments fraud detection using machine learning is a proactive approach to identify and prevent fraudulent activities during online 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 a like.

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 behaviour, it can flag suspicious transactions for further investigation,preventing fraudulent activities before they occur.

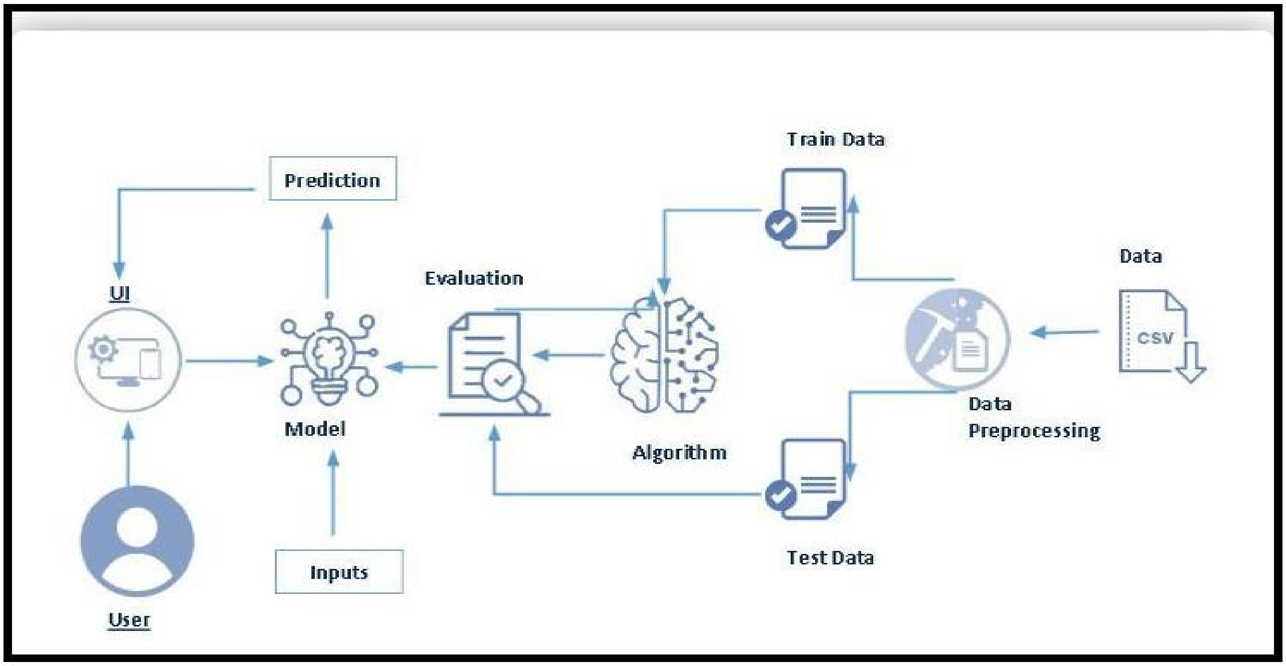
Scenario 2: Fraudulent Account Detection

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.

TechnicalArchitecture:



Pre requisites:

To complete this project, you must require the following software, concepts, and packages

- Anaconda Navigator:

  Refer to the link below to download Anaconda Navigator

- Python packages:

  Open anaconda prompt as administrator

  - Type “pip install numpy” and click enter.
  - Type “pip install pandas” and click enter.
  - Type “pip install scikit-learn” and click enter.
  - Type ”pip install matplotlib” and click enter.
  - Type ”pip install scipy” and click enter.
  - Type ”pip install pickle-mixin” and click enter.
  - Type ”pip install seaborn” and click enter.
  - Type “pip install Flask” and click enter.

Prior Knowledge:

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

- ML Concepts:

  Supervised learning:
  https://www.youtube.com/watch?v=QeKshry8pWQ

  Unsupervised learning:
  https://youtu.be/D6gtZrsYi6c?si=5EOSf7ALg-S3s9K2

  Metrics:
  https://youtu.be/aWAnNHXIKww?si=WoHgrZe0WE2p9mY8

  Flask:
  https://youtu.be/lj4I_CvBnt0?si=2FD9b1DqTr9Oz49r

Project Objectives:

By the end of this project, you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding of data.
- Have knowledge of pre-processing the data/transformation techniques and some visualization concepts before building the model.
- Learn how to build a machine learning model and tune it for better performances.
- Know how to evaluate the model and deploy it using flask.

Project Flow:

- The user interacts with the UI (User Interface) 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

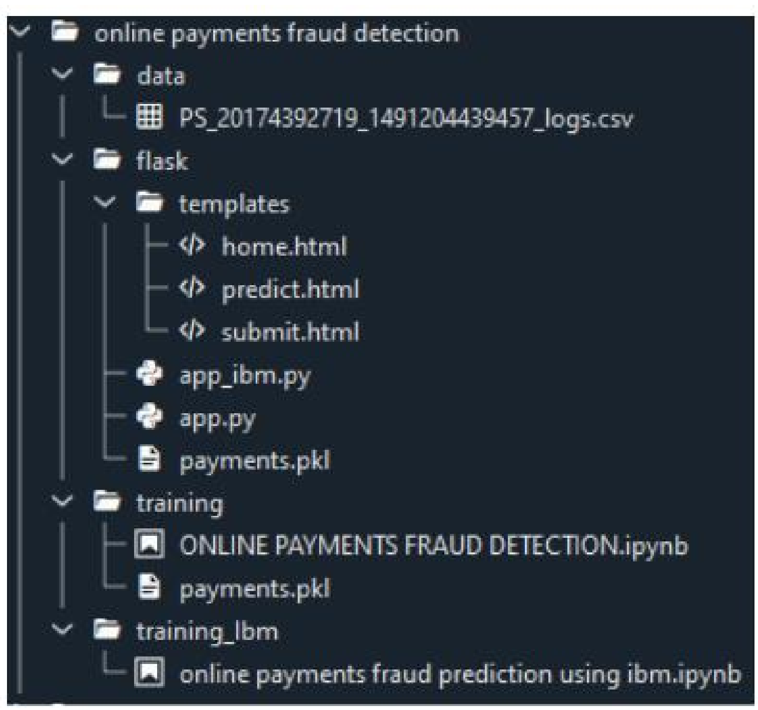
    Handling categorical values
    Dividing data into train and test sets
    Import the model building libraries
    Comparing the accuracy of various models
    Hyperparameter tuning of the selected model
    Evaluating the performance of models
    Save the model

- Application Building

    Create an HTML file
    Build python code

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.
- Model.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains model training files and the training_ibm folder contains IBM deployment files.

Milestone 1: Data Collection

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.

Collect the dataset

Download the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

This dataset contains 12,500 augmented images of blood cells (JPEG) with accompanying cell type labels (CSV). There are approximately 3,000 images for each of 4 different cell types grouped into 4 different folders (according to cell type). The cell types are Eosinophil, Lymphocyte, Monocyte, and Neutrophil.

Link: "C:\Users\midde\Desktop\PS_20174392719_1491204439457_log.csv.zip"

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Milestone 2: Visualizing and Analyzing Data

Activity 2.1: Importing the libraries:

Import the necessary libraries as shown in the image. (optional) here we have used visualisation style as fivethirtyeight.

[8]
0s

```python
# Importing Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import SVC

import xgboost as xgb
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix

import warnings
import pickle
```

Activity 2.2: Read the Dataset:

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

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

```python
# Reading the csv data
df = pd.read_csv('PS_20174392719_1491204439457_log.csv.zip')
```

```python
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
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0	0
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	1	0
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0	0
...	...	...	...	...	...	...	...	...	...	...	...
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C776919290	0.00	339682.13	1	0
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C1881841831	0.00	0.00	1	0
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C1365125890	68488.84	6379898.11	1	0
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C2080388513	0.00	0.00	1	0
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C873221189	6510099.11	7360101.63	1	0

6362620 rows × 11 columns

Terminal . Executing (25m 10s) Python

```python
df.columns
```

```
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)
```

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

```
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	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

About Dataset

The below column reference:

1. step: represents a unit of time where 1 step equals 1 hour
2. type: type of online transaction
3. amount: the amount of the transaction
4. nameOrig: customer starting the transaction
5. oldbalanceOrg: balance before the transaction
6. newbalanceOrig: balance after the transaction
7. nameDest: recipient of the transaction
8. oldbalanceDest: initial balance of recipient before the transaction
9. newbalanceDest: the new balance of recipient after the transaction
10. isFraud: fraud transaction

```
df.head()
```

	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

Above , the dataset’s first five values are loaded using the head method.

```
df.tail()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C776919290	0.00	339682.13	1
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C1881841831	0.00	0.00	1
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C1365125890	68488.84	6379898.11	1
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388513	0.00	0.00	1
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C873221189	6510099.11	7360101.63	1

Above,the datasets last five values are loded using the tail method.

```
plt.style.use('ggplot')
warnings.filterwarnings('ignore')
```

utilising Style use here The Ggplot approach Setting "styles"—basically stylesheets that resemble matplotlibrc files—is a fundamental feature of mpltools. The "ggplot" style, which modifies the style to resemble ggplot, is demonstrated in this dataset.

```
df.corr(numeric_only=True)
```

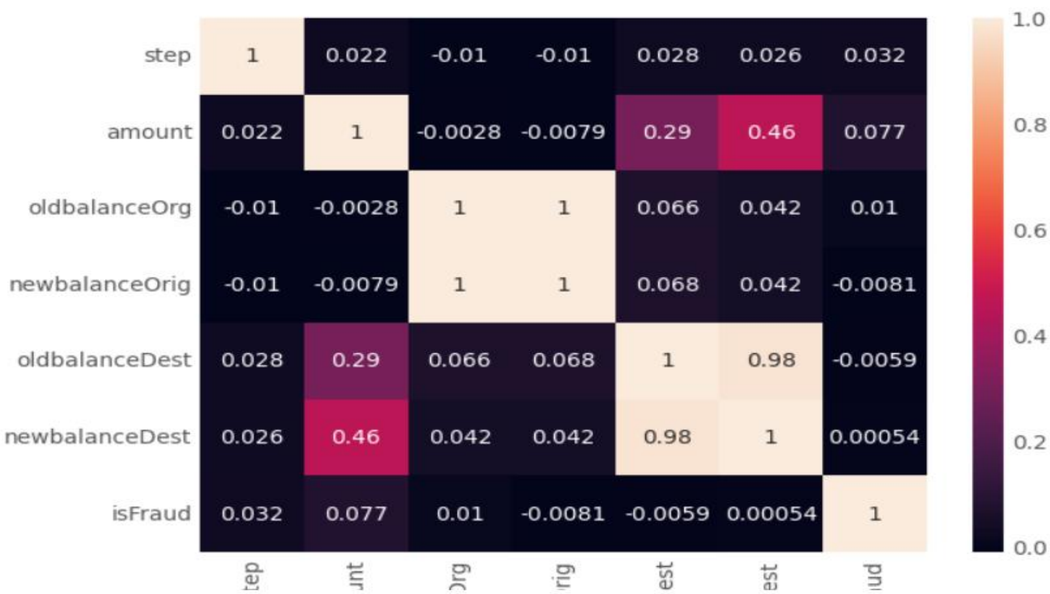
	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
step	1.000000	0.022373	-0.010058	-0.010299	0.027665	0.025888	0.031578
amount	0.022373	1.000000	-0.002762	-0.007861	0.294137	0.459304	0.076688
oldbalanceOrg	-0.010058	-0.002762	1.000000	0.998803	0.066243	0.042029	0.010154
newbalanceOrig	-0.010299	-0.007861	0.998803	1.000000	0.067812	0.041837	-0.008148
oldbalanceDest	0.027665	0.294137	0.066243	0.067812	1.000000	0.976569	-0.005885
newbalanceDest	0.025888	0.459304	0.042029	0.041837	0.976569	1.000000	0.000535
isFraud	0.031578	0.076688	0.010154	-0.008148	-0.005885	0.000535	1.000000

Utilising the corr function to examine the dataset's correlation

Heatmap

```
#heatmap
sns.heatmap(df.corr(numeric_only=True),annot=True)
```

<Axes: >



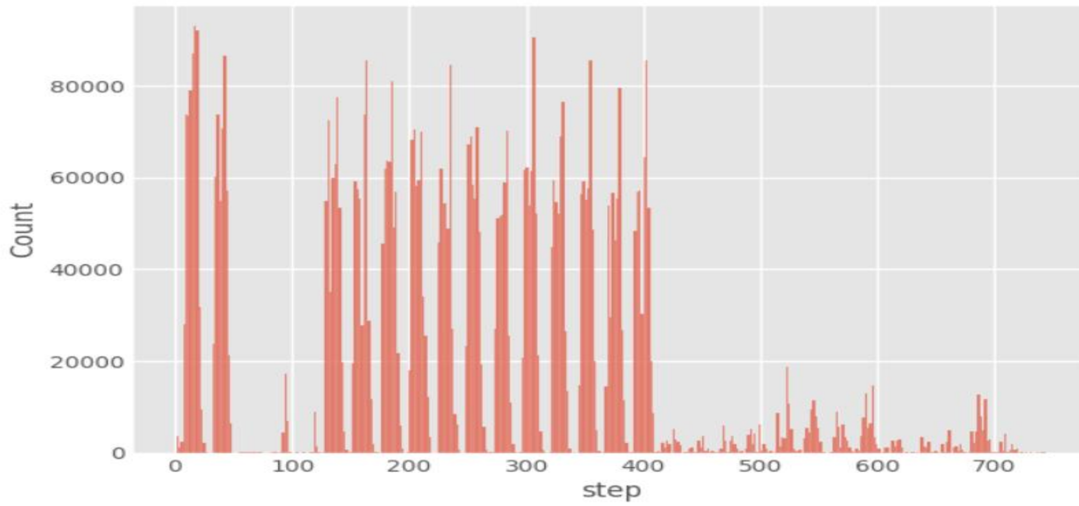
Here,a heatmap is used to understand the relationship between the input attributes and the anticipated goal value.

Activity 2.3 : Univariate Analysis:

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

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

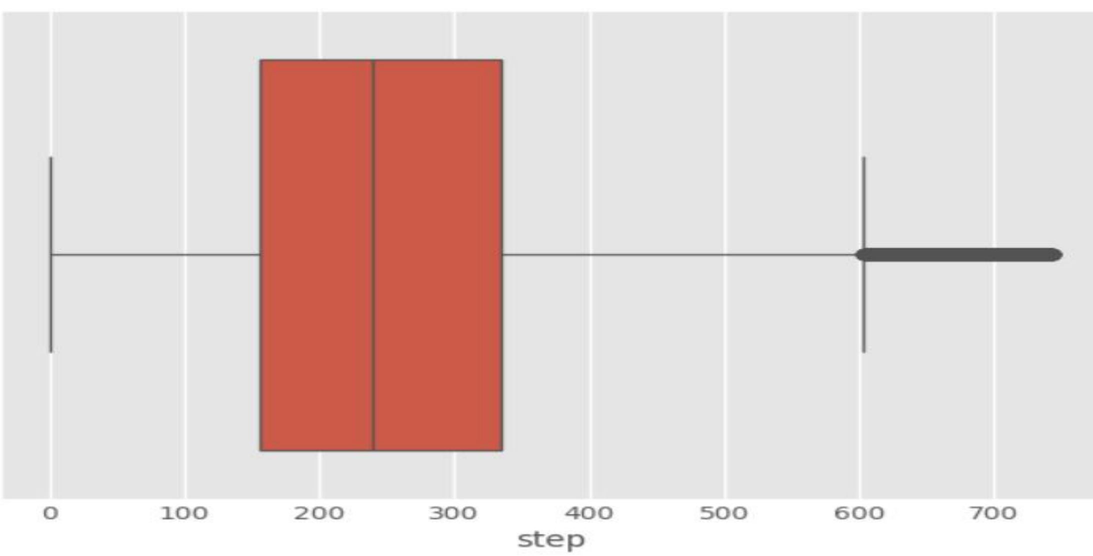
```
<Axes: xlabel='step', ylabel='Count'>
```



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')
```

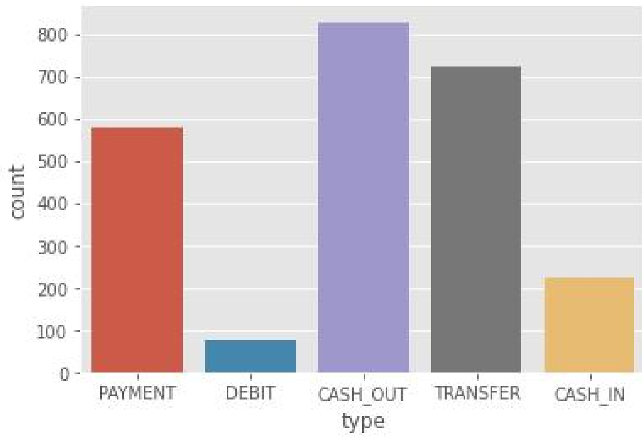
```
<Axes: xlabel='step'>
```



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

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

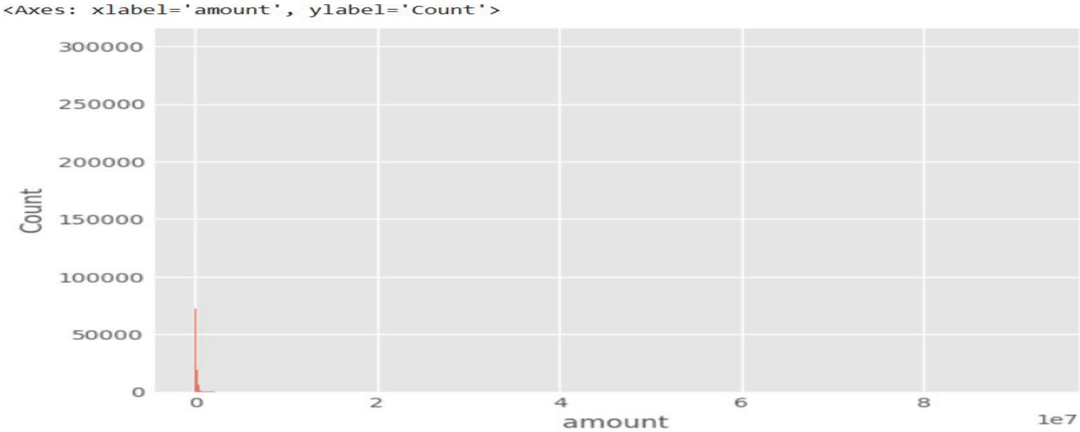
<AxesSubplot:xlabel='type', ylabel='count'>



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

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

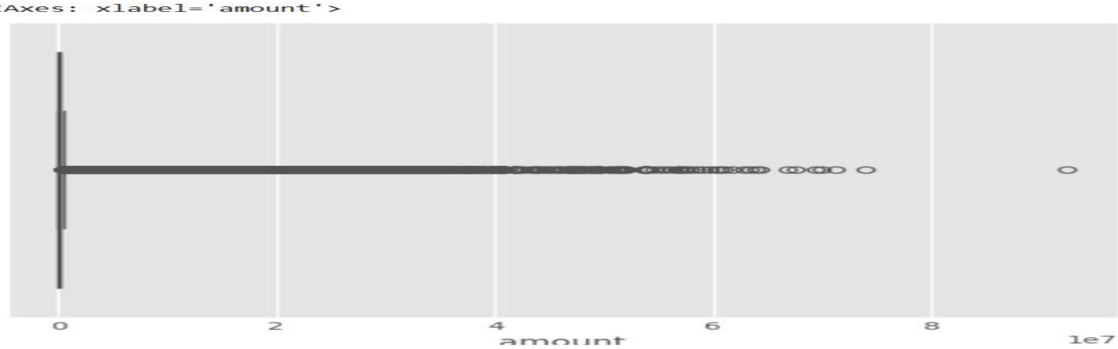
<Axes: xlabel='amount', ylabel='Count'>



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.

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

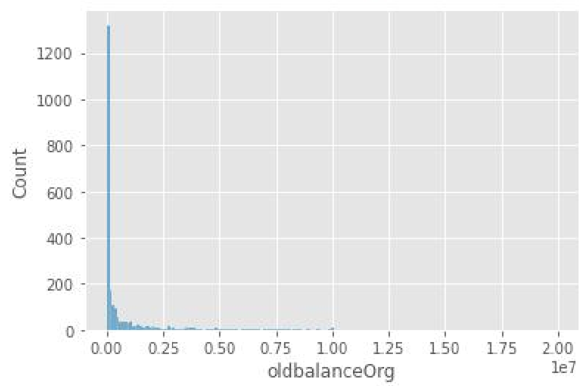
<Axes: xlabel='amount'>



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

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

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



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.

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

nameDest	count
C1286084959	113
C985934102	109
C665576141	105
C2083562754	102
C248609774	101
...	...
M367627425	1
M1902904124	1
M242332837	1
M281573812	1
M1010678443	1

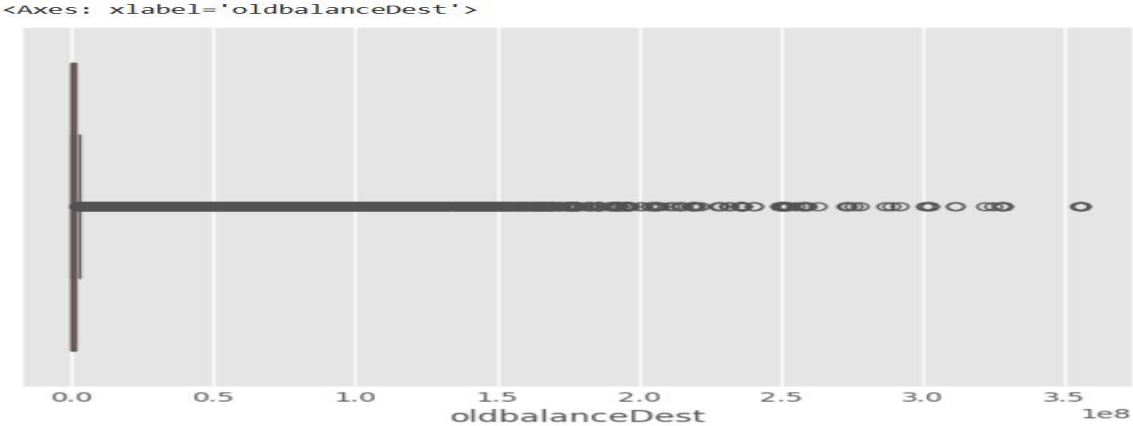
2722362 rows × 1 columns

dtype: int64

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

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

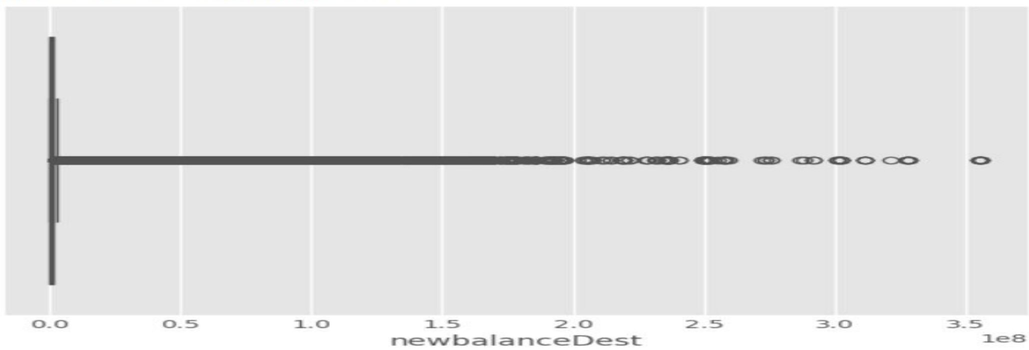
```
<Axes: xlabel='oldbalanceDest'>
```



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

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

<Axes: xlabel='newbalanceDest'>



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

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

<AxesSubplot:xlabel='isFraud', ylabel='count'>

using the countplot approach here to count the number of instances in the dataset's target isFraud column.

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

isFraud	count
0	6354407
1	8213

dtype: int64

Here, we're using the value counts method to figure out how many classes there are in the dataset's target 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
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	is not Fraud
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	is not Fraud
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	is Fraud
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	is Fraud
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	is not Fraud
...	...	...	...	...	...	...	...	...	...	...
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C776919290	0.00	339682.13	is Fraud
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C1881841831	0.00	0.00	is Fraud
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C1365125890	68488.84	6379898.11	is Fraud
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C2080388513	0.00	0.00	is Fraud
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C873221189	6510099.11	7360101.63	is Fraud

6362620 rows × 10 columns

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

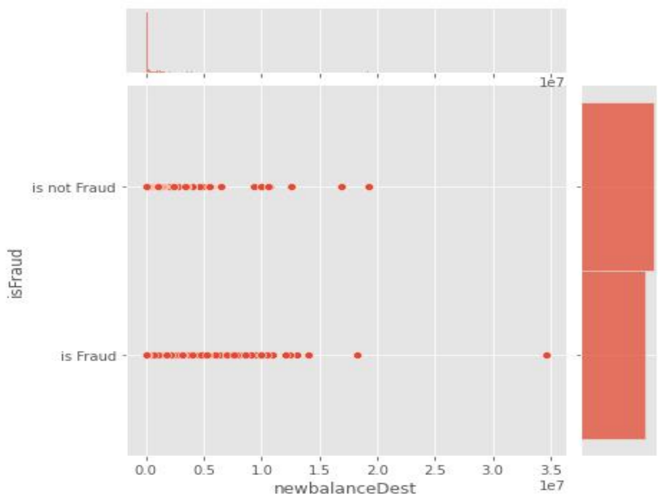
Activity 2.4: Bivariate Analysis:

To find the relation between two features we use bivariate analysis. Here we are visualising the relationship between newbalanceDest and isFraud.

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')
```

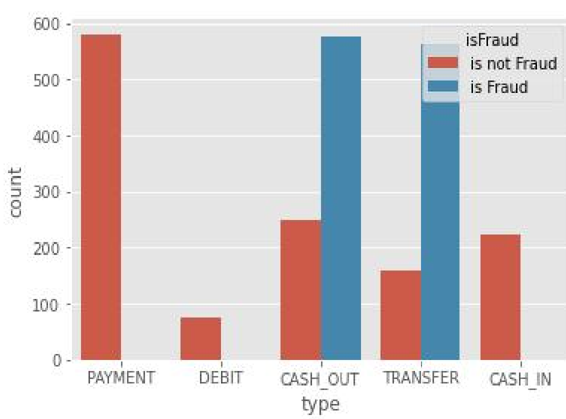
```
<seaborn.axisgrid.JointGrid at 0x15ee667b220>
```



Here we are visualising 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')
```

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



Here we are visualising 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.

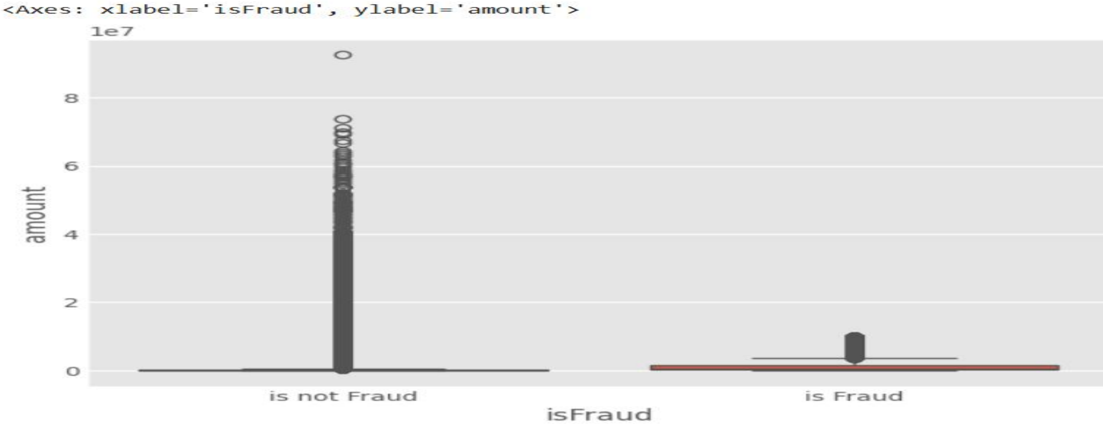
```
sns.boxplot(data=df,x='isFraud',y='step')
```

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

Here we are visualising the relationship between isFraud and amount.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='amount')
```

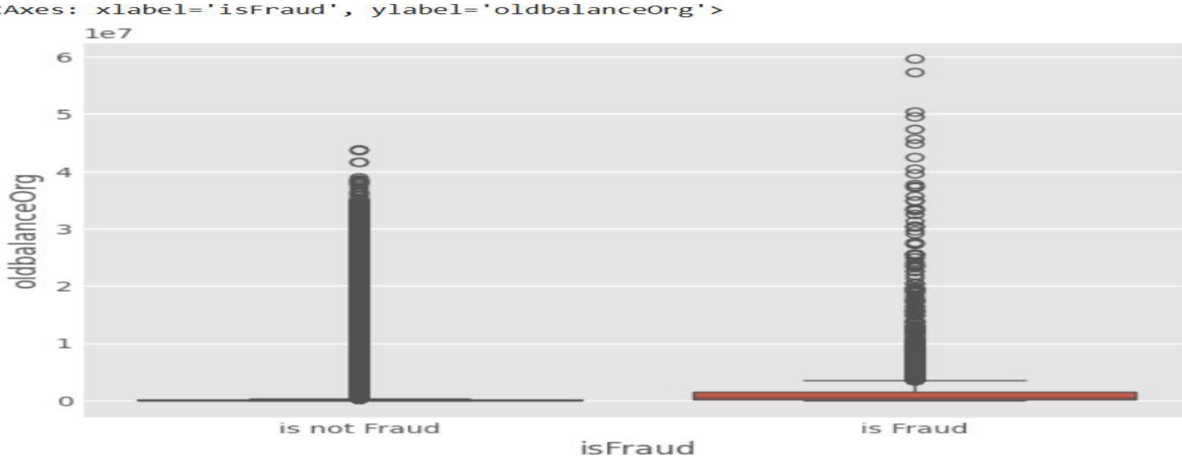
```
<Axes: xlabel='isFraud', ylabel='amount'>
```



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

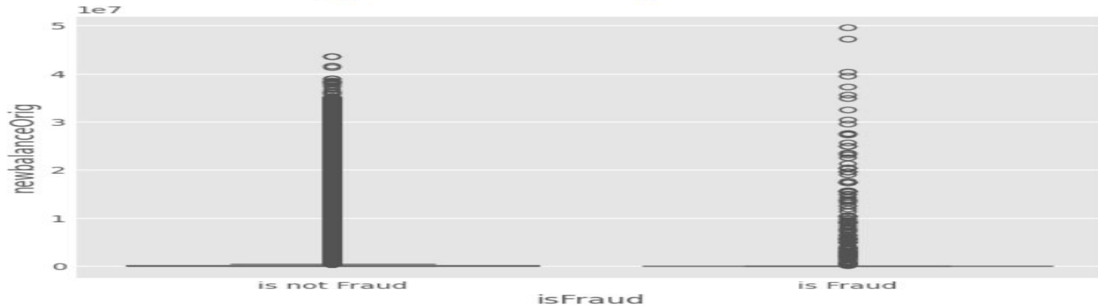
```
sns.boxplot(data=df,x='isFraud',y='oldbalanceOrg')
```

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



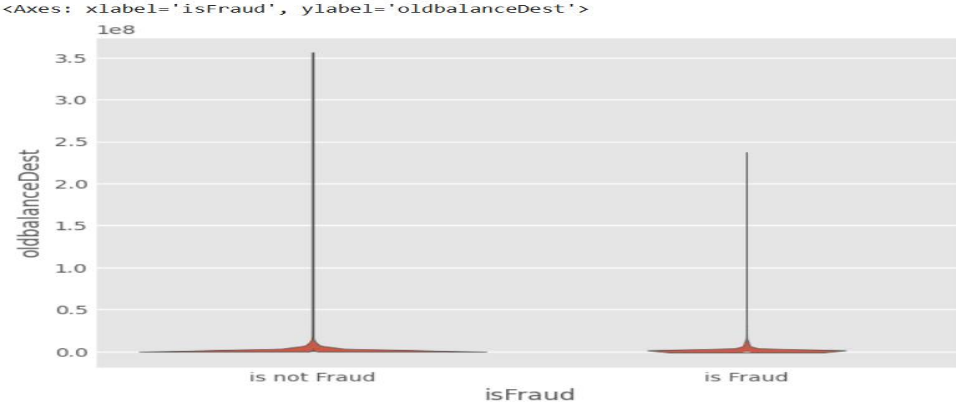
Here we are visualising the relationship between isFraud and newbalanceOrig. boxtplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.boxplot(data=df,x='isFraud',y='newbalanceOrig')
<Axes: xlabel='isFraud', ylabel='newbalanceOrig'>
```



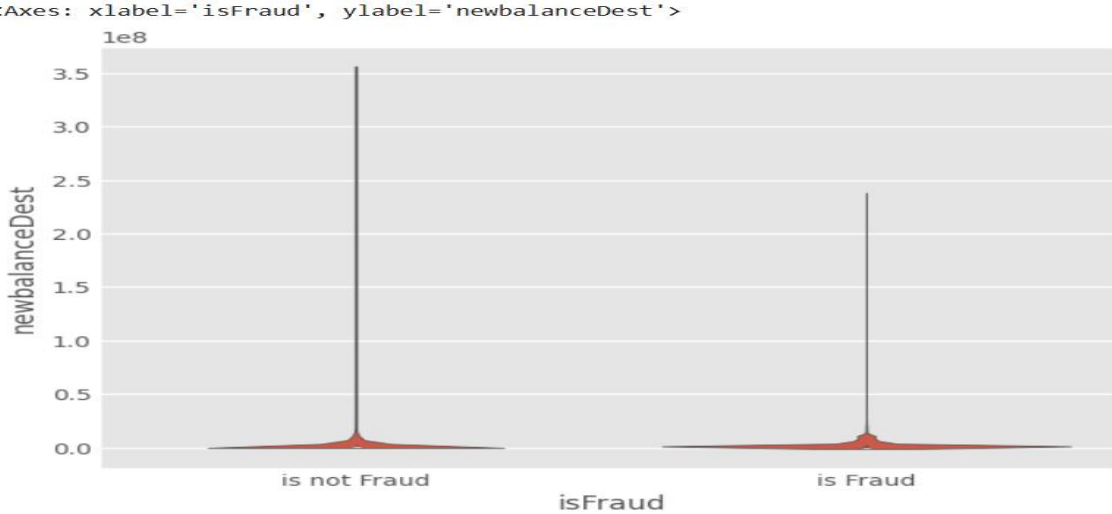
Here we are visualising the relationship between isFraud and oldbalanceDest. violinplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.violinplot(data=df,x='isFraud',y='oldbalanceDest')
<Axes: xlabel='isFraud', ylabel='oldbalanceDest'>
```



Here we are visualising 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 2.5: 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	6362620
unique	NaN	5	NaN	6353307	NaN	NaN	2722362	NaN	NaN	2
top	NaN	CASH_OUT	NaN	C1530544995	NaN	NaN	C1286084959	NaN	NaN	is not Fraud
freq	NaN	2237500	NaN	3	NaN	NaN	113	NaN	NaN	6354407
mean	2.433972e+02	NaN	1.798619e+05	NaN	8.338831e+05	8.551137e+05	NaN	1.100702e+06	1.224996e+06	NaN
std	1.423320e+02	NaN	6.038582e+05	NaN	2.888243e+06	2.924049e+06	NaN	3.399180e+06	3.674129e+06	NaN
min	1.000000e+00	NaN	0.000000e+00	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	NaN
25%	1.560000e+02	NaN	1.338957e+04	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	NaN
50%	2.390000e+02	NaN	7.487194e+04	NaN	1.420800e+04	0.000000e+00	NaN	1.327057e+05	2.146614e+05	NaN
75%	3.350000e+02	NaN	2.087215e+05	NaN	1.073152e+05	1.442584e+05	NaN	9.430367e+05	1.111909e+06	NaN
max	7.430000e+02	NaN	9.244552e+07	NaN	5.958504e+07	4.958504e+07	NaN	3.560159e+08	3.561793e+08	NaN

```
# shape of csv data
df.shape
```

Milestone 3: Data Pre-Processing:

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

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

```
# Shape of csv data
df.shape
```

```
(2430, 10)
```

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

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

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

```
df.head()
```

	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:

Isnull is used (). sum() to check your database for null values. Using the df.info() function, the data type can be determined.

```
# Finding null values
df.isnull().sum()
```

	0
step	0
type	0
amount	0
oldbalanceOrg	0
newbalanceOrig	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()
```

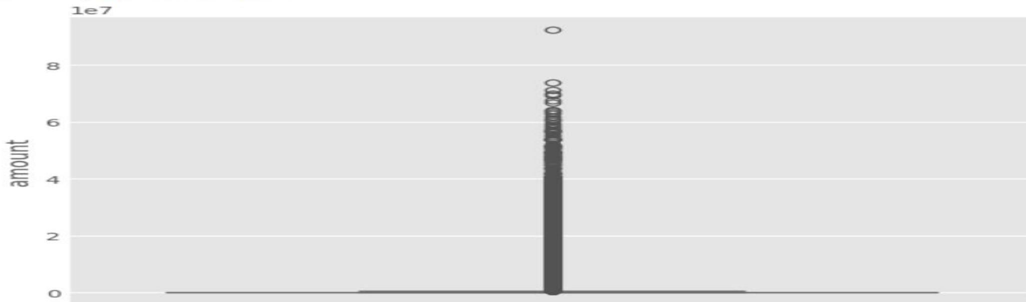
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 8 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   oldbalanceOrg   float64
 4   newbalanceOrig  float64
 5   oldbalanceDest  float64
 6   newbalanceDest  float64
 7   isFraud         object
dtypes: float64(5), int64(1), object(2)
memory usage: 388.3+ MB
```

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

Activity 3.2: Handling Outliers:

```
sns.boxplot(df['amount'])
```

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



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

```
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
```

```
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
```

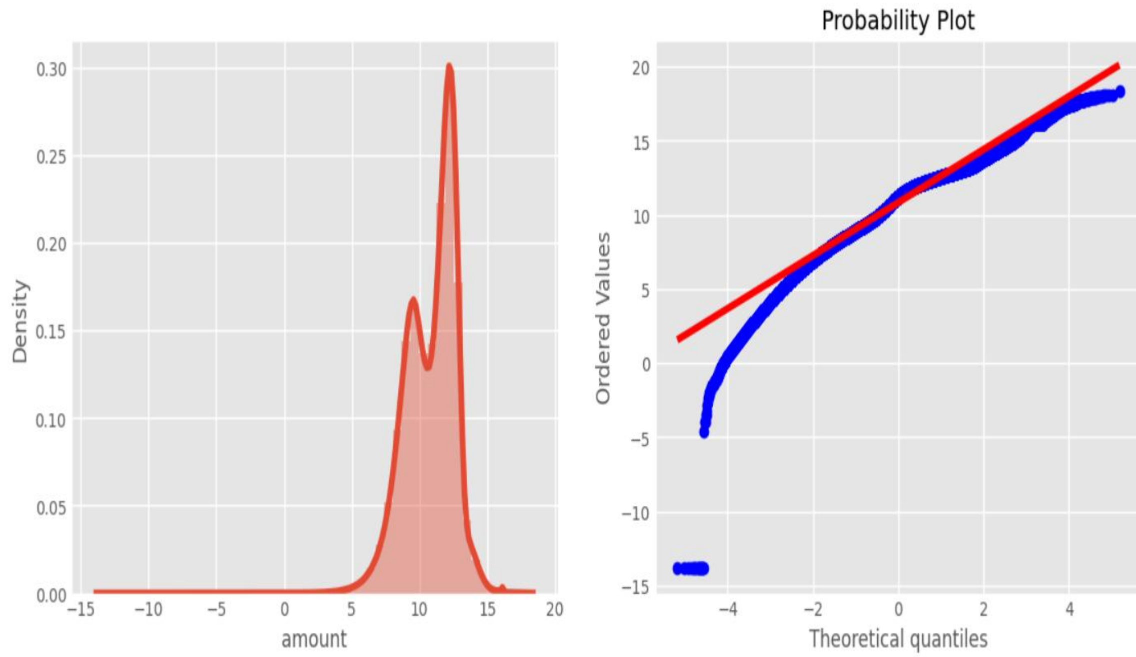
```
# To handle outliers transformation techniques are used.

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

    plt.subplot(1, 2, 1)
    sns.distplot(feature)

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

```
transformationPlot(np.log(df['amount'] + 1e-6))
```




```
df['amount']=np.log(df['amount'] + 1e-6)
```

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

Activity 3.3: Object Data Labelencoding:

```
from sklearn.preprocessing import LabelEncoder

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

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

type	count
1	2237500
3	2151495
0	1399284
4	532909
2	41432

dtype: int64

using labelencoder to encode the dataset's object type.

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

```python
x
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	1	3	9.194174	170136.00	160296.36	0.00	0.00
1	1	3	7.530630	21249.00	19384.72	0.00	0.00
2	1	4	5.198497	181.00	0.00	0.00	0.00
3	1	1	5.198497	181.00	0.00	21182.00	0.00
4	1	3	9.364617	41554.00	29885.86	0.00	0.00
...	...	...	...	...	...	...	...
6362615	743	1	12.735766	339682.13	0.00	0.00	339682.13
6362616	743	4	15.657870	6311409.28	0.00	0.00	0.00
6362617	743	1	15.657870	6311409.28	0.00	68488.84	6379898.11
6362618	743	4	13.652995	850002.52	0.00	0.00	0.00
6362619	743	1	13.652995	850002.52	0.00	6510099.11	7360101.63

6362620 rows × 7 columns

```python
y
```

	isFraud
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

6362620 rows × 1 columns

dtype: object

Activity 3.4: Splitting Data 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.

```
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)
```

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

Milestone 4: Model Building:

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 four classification algorithms. The best model is saved based on its performance.

Activity 4.1: Random Forest Classifier:

A function named RandomForest is created and train and test data are passed as the parameters. Inside the function, the RandomForestClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
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
```

```
0.9997108109552354
```

```
y_train_predict1=rfc.predict(x_train)
train_accuracy=accuracy_score(y_test,y_test_predict1)
train_accuracy
```

```
0.9997108109552354
```

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

col_0 / isFraud	is Fraud	is not Fraud
is Fraud	1299	342
is not Fraud	26	1270857

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

```
              precision    recall  f1-score   support

    is Fraud       0.98      0.79      0.88      1641
is not Fraud       1.00      1.00      1.00   1270883

    accuracy                           1.00   1272524
   macro avg       0.99      0.90      0.94   1272524
weighted avg       1.00      1.00      1.00   1272524
```

Activity 4.2: Decision Tree Classifier:

A function named Decisiontree is created and train and test data are passed as the parameters. Inside the function, the DecisiontreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
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
```

```
0.9997076675960532
```

```
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 / isFraud	is Fraud	is not Fraud
is Fraud	1434	207
is not Fraud	165	1270718

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

```
              precision    recall  f1-score   support

    is Fraud       0.90      0.87      0.89      1641
is not Fraud       1.00      1.00      1.00   1270883

    accuracy                           1.00   1272524
   macro avg       0.95      0.94      0.94   1272524
weighted avg       1.00      1.00      1.00   1272524
```

Activity 4.3: Extra Trees Classifier:

A function named ExtraTree is created and train and test data are passed as the parameters. Inside the function, ExtraTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
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
```

```
0.999704524236871
```

```
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	1279	362
is not Fraud	14	1270869

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

```
              precision    recall  f1-score   support

    is Fraud       0.99      0.78      0.87      1641
is not Fraud       1.00      1.00      1.00   1270883

    accuracy                           1.00   1272524
   macro avg       0.99      0.89      0.94   1272524
weighted avg       1.00      1.00      1.00   1272524
```

Activity 4.4: SupportVectorMachine Classifier:

A function named SupportVector is created and train and test data are passed as the parameters. Inside the function, the SupportVectorClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, confusion matrix and classification report is done.

```
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)
```

preprocessing class of sklearn. LabelEncoder[source] 0 to n classes-1 as the range for the target labels to be encoded. Instead of encoding the input X, the target values, i.e. y, should be encoded using this transformer.

```
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])
```

```
y_train1
```

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

Activity 4.5: Xgboost Classifier:

A function named xgboost is created and train and test data are passed as the parameters. Inside the function, the xgboostClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, confusion matrix and classification report is done.

```
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
```

Activity 4.6:Compare the models:

For comparing the above four models, the compareModel function is defined.

After calling the function, the results of models are displayed as output. From the five models, the svc is performing well. From the below image, We can see the accuracy of the model is 79% accuracy.

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
```

Activity 4.6: Evaluating performance of the model and saving the model:

From sklearn, accuracy_score is used to evaluate the score of the model. On the parameters, we have given svc (model name), x, y, cv (as 5 folds). Our model is performing well. So, we are saving the model is svc by pickle.dump().

```
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
```

```
import pickle
pickle.dump(svc,open('payments.pkl','wb'))
```

Milestone 5: Application Building:

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

Building HTML Pages
Building server side script

Activity 5.1: Building HTML Pages:

For this project create three HTML files namely

- Home.html
- Predict.html
- Submit.html and save them in the templates folder.

Let’s see how our home.html page looks like:

Now when you click on predict button from top right corner you will get redirected to predict.html

Let's look how our predict.html file looks like:

Now when you click on submit button from left bottom corner you will get redirected to submit.html

Let's look how our submit.html file looks like:

Activity 5.2: Build Python Code:

Import the libraries:

```
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'))
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

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

app = Flask(__name__)
```

Render HTML page:

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

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

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.
Retrieves the value from UI:

```
@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))
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

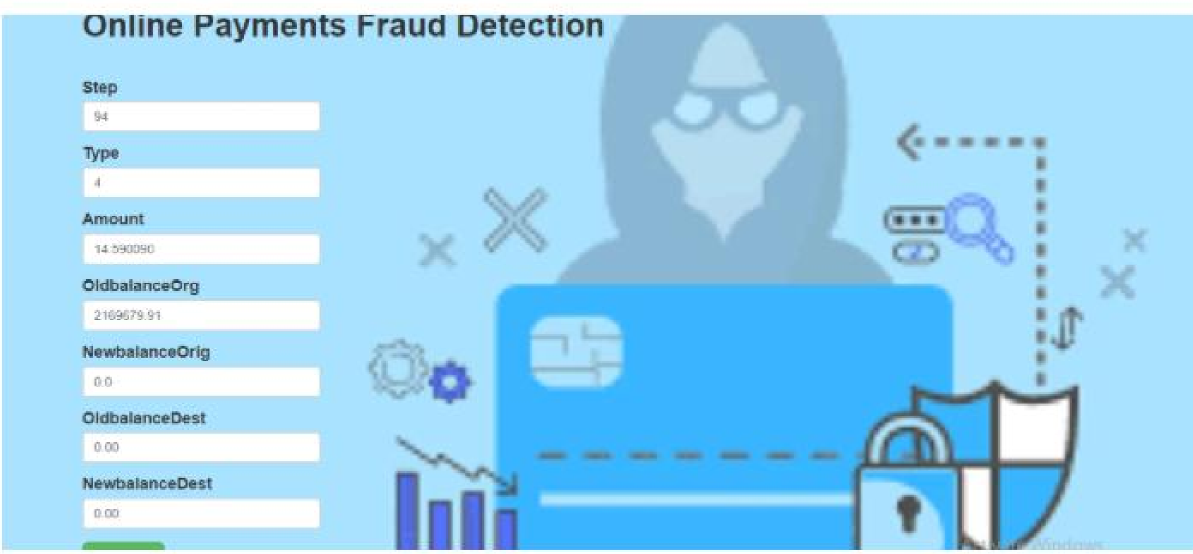
```
if __name__ == "__main__":
 app.run(debug=False)
```

Activity 5.3: Run The Application:

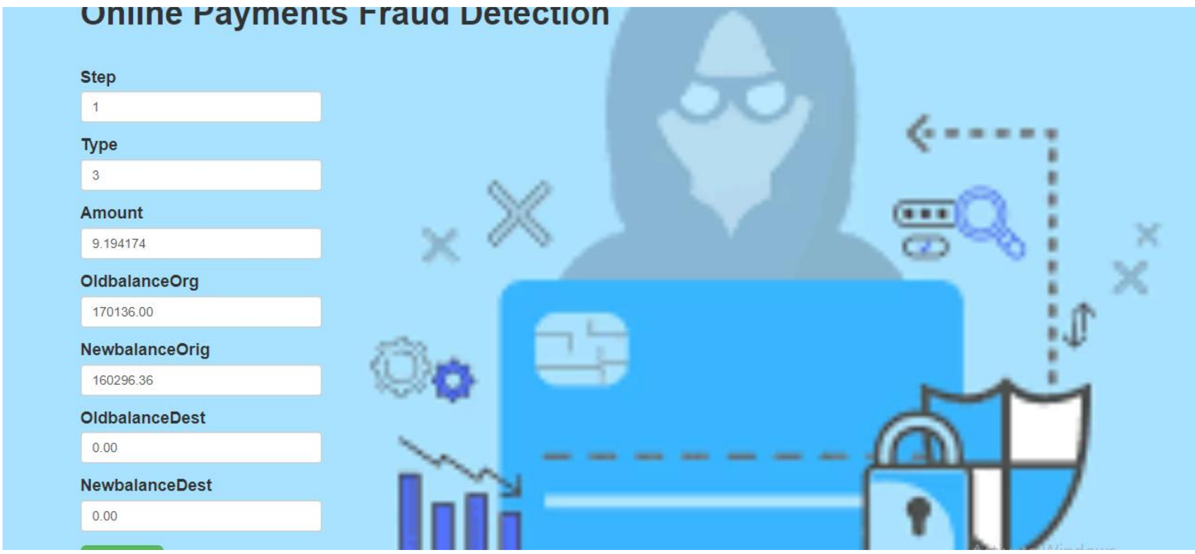
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
In [11]: runfile('C:/Users/user/Desktop/online payments fraud detection/flask/app.py', wdir='C:/Users/user/Desktop/online payments fraud detection/flask')
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

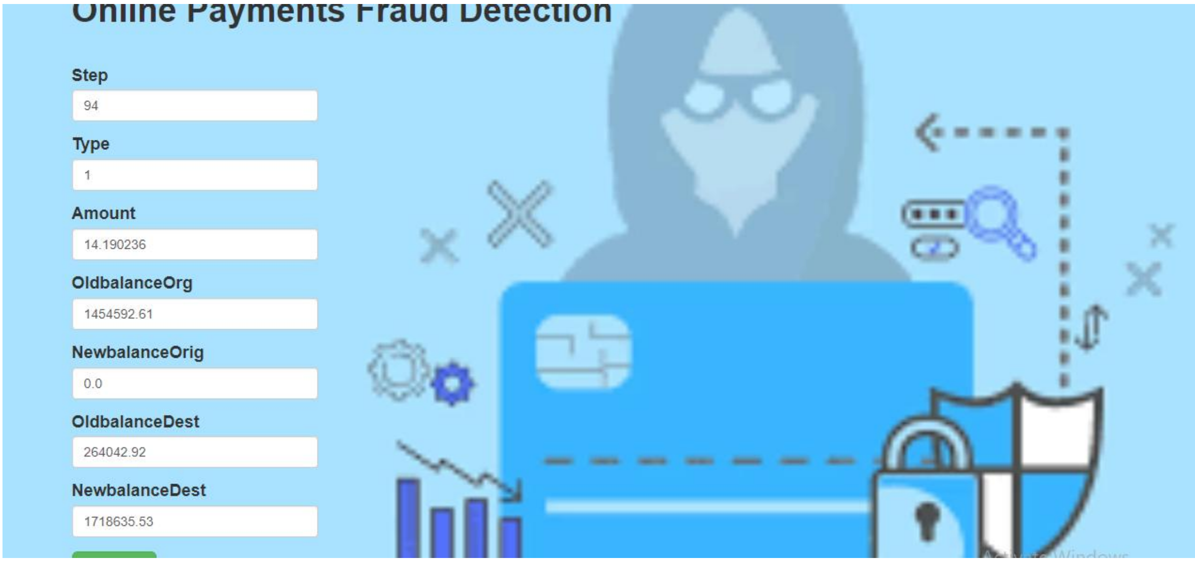
Outputscreensorts:



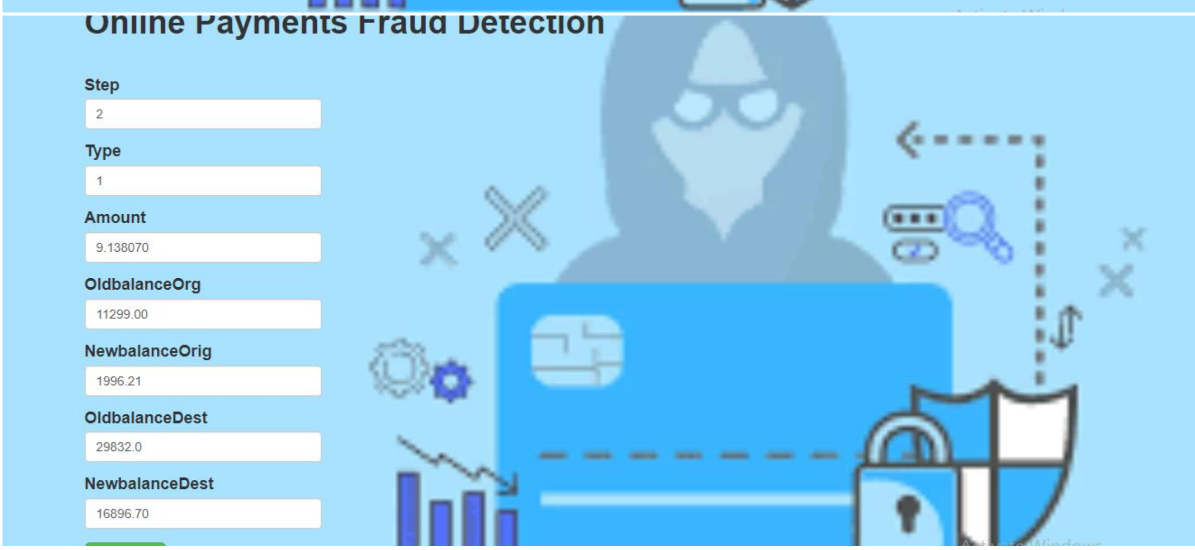

Home
Predict
Online Payments Fraud Detection
The predicted fraud for the online payment is ['is Fraud']


Online Payments Fraud Detection
Step
1
Type
3
Amount
9.194174
OldbalanceOrg
170136.00
NewbalanceOrig
160296.36
OldbalanceDest
0.00
NewbalanceDest
0.00


Home
Predict
Online Payments Fraud Detection
The predicted fraud for the online payment is ['is not Fraud']


Online Payments Fraud Detection
Step
94
Type
1
Amount
14.190236
OldbalanceOrg
1454592.61
NewbalanceOrig
0.0
OldbalanceDest
264042.92
NewbalanceDest
1718635.53


Home
Predict
Online Payments Fraud Detection
The predicted fraud for the online payment is ['is Fraud']


Online Payments Fraud Detection
Step
2
Type
1
Amount
9.138070
OldbalanceOrg
11299.00
NewbalanceOrig
1996.21
OldbalanceDest
29832.0
NewbalanceDest
16896.70


Home
Predict
Online Payments Fraud Detection
The predicted fraud for the online payment is ['is not Fraud']