Online Payments Fraud Detection Using Machine Learning

Project Description:

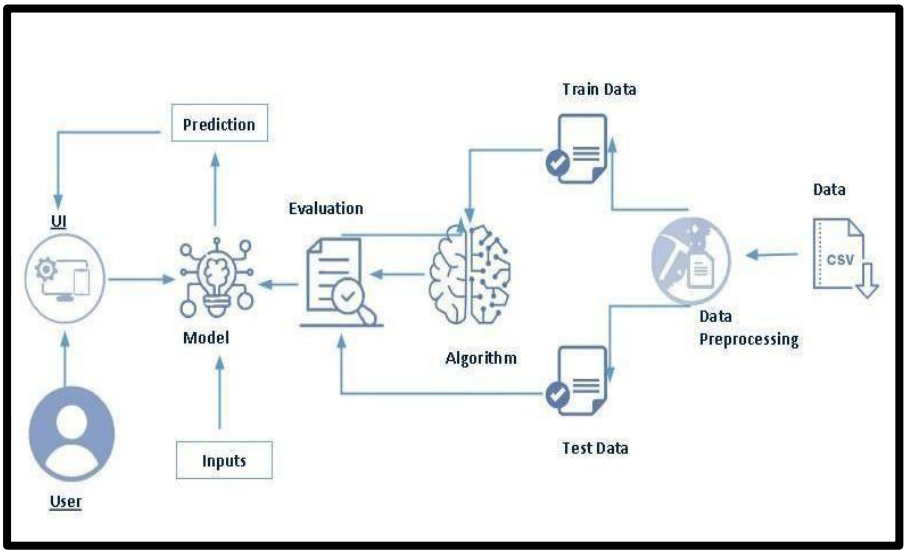
Online Payments Fraud Detection using Machine Learning is a proactive approach to identify and prevent fraudulent activities during online transactions. By leveraging historical transaction data, customer behavior patterns, and machine learning algorithms, this project aims to detect potential fraud 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:



Pre requisites:

To complete this project, you must required following software's, concepts and packages

- **Anaconda navigator and pycharm:**
  - Refer the link below to download anaconda navigator ○ Link : https://youtu.be/1ra4zH2G4o0 • **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 following topics to complete this project.

- **ML Concepts**
  - Supervised learning: https://www.youtube.com/watch?v=QeKshry8pWQ&t=6s This tutorial will comprise of the following topics: Linear Regression ,Logistic Regression ,KNN Naive Bayes
  - Unsupervised learning: https://www.youtube.com/watch?v=D6gtZrsYi6c
  - Xgboost : https://www.analyticsvidhya.com/blog/2018/09/an-end-toend- guide-to-understand-the-math-behind-xgboost/
  - Performance metrics: https://www.youtube.com/watch?v=aWAnNHXIKww

- **Flask Basics**: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on 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 performance.
- Know how to evaluate the model and deploy it using Flask.

Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

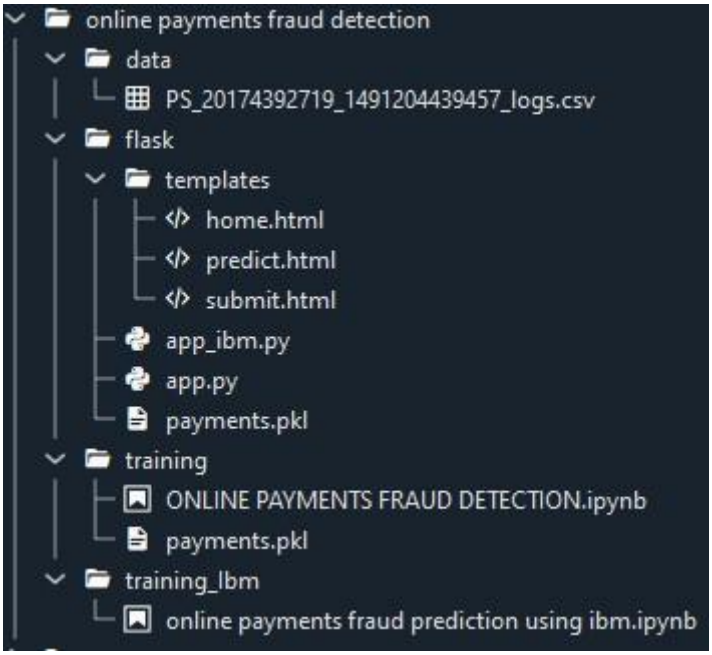
To accomplish this, we have to complete all the activities listed below,

- Data collection
  - Collect the dataset or create the dataset
- Visualizing and analyzing data
  - Univariate analysis
  - Bivariate analysis
  - Multivariate analysis
  - Descriptive analysis
- Data pre-processing
  - Checking for null values
  - Handling outlier
  - Handling categorical data
  - Splitting data into train and test

- Model building ○ Import the model building libraries ○ Initializing the model ○ Training and testing the model ○ Evaluating performance of model ○ 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 training_ibm folder contains IBM deployment files.

Milestone 1: Data Collection

ML depends heavily on data, It is most crucial aspect that makes algorithm training possible.

So this section allows you to download the required dataset.

Activity 1: Download 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. Pleas e refer to the link given below to download the dataset.

Link: link

Milestone 2: Visualizing and analysing the data

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

Activity 1: Importing the libraries

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

```
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, confusion_matrix, classification_report
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: 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 csv file.

```
df = pd.read_csv("/content/PS_20174392719_1491204439457_log.csv")
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0.0	0.0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0.0	0.0
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	1.0	0.0
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	1.0	0.0
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
179009	12	CASH_IN	200271.90	C1829080213	3896164.44	4096436.33	C1769155229	531167.90	690790.44	0.0	0.0
179010	12	CASH_IN	266616.13	C1119089903	4096436.33	4363052.46	C859300166	6064698.49	6080565.41	0.0	0.0
179011	12	CASH_IN	194735.24	C595356999	4363052.46	4557787.70	C803648645	288407.25	152597.27	0.0	0.0
179012	12	CASH_IN	61212.03	C1705359203	4557787.70	4618999.73	C60296041	2492638.18	2620615.37	0.0	0.0
179013	12	CASH_IN	16.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

179014 rows × 11 columns

```
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)
```

```
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	0.0
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
...	...	...	...	...	...	...	...	...	...	...
179009	12	CASH_IN	200271.90	C1829080213	3896164.44	4096436.33	C1769155229	531167.90	690790.44	0.0
179010	12	CASH_IN	266616.13	C1119089903	4096436.33	4363052.46	C859300166	6064698.49	6080565.41	0.0
179011	12	CASH_IN	194735.24	C595356999	4363052.46	4557787.70	C803648645	288407.25	152597.27	0.0
179012	12	CASH_IN	61212.03	C1705359203	4557787.70	4618999.73	C60296041	2492638.18	2620615.37	0.0
179013	12	CASH_IN	16.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN

179014 rows × 10 columns

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

```
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	0.0
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

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
179009	12	CASH_IN	200271.90	C1829080213	3896164.44	4096436.33	C1769155229	531167.90	690790.44	0.0
179010	12	CASH_IN	266616.13	C1119089903	4096436.33	4363052.46	C859300166	6064698.49	6080565.41	0.0
179011	12	CASH_IN	194735.24	C595356999	4363052.46	4557787.70	C803648645	288407.25	152597.27	0.0
179012	12	CASH_IN	61212.03	C1705359203	4557787.70	4618999.73	C60296041	2492638.18	2620615.37	0.0
179013	12	CASH_IN	16.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN

above, the dataset's last five values are loaded using the tail method.

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

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

label_encoder = LabelEncoder()
df_numeric_for_corr['type'] = label_encoder.fit_transform(df_numeric_for_corr['type'])

correlation_matrix= df_numeric_for_corr.corr()
correlation_matrix
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
step	1.000000	-0.087140	0.053716	-0.003387	-0.002984	0.020696	0.007400	-0.042704
type	-0.087140	1.000000	0.096818	-0.357027	-0.371258	-0.156384	-0.114242	0.013833
amount	0.053716	0.096818	1.000000	-0.019359	-0.023891	0.223895	0.349275	0.036239
oldbalanceOrg	-0.003387	-0.357027	-0.019359	1.000000	0.998970	0.097961	0.066721	-0.002824
newbalanceOrig	-0.002984	-0.371258	-0.023891	0.998970	1.000000	0.099503	0.065720	-0.008649
oldbalanceDest	0.020696	-0.156384	0.223895	0.097961	0.099503	1.000000	0.949306	-0.008282
newbalanceDest	0.007400	-0.114242	0.349275	0.066721	0.065720	0.949306	1.000000	-0.005282
isFraud	-0.042704	0.013833	0.036239	-0.002824	-0.008649	-0.008282	-0.005282	1.000000

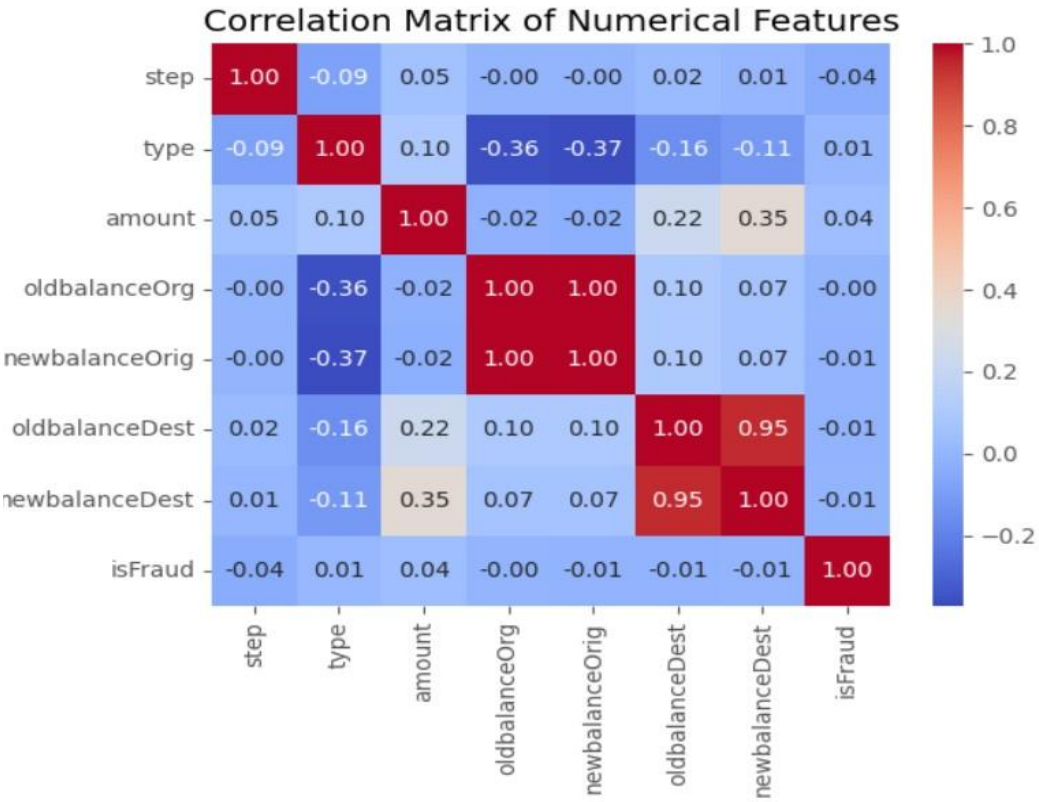
utilising Style use here The Ggplot approach Setting "styles"—basically stylesheets that resemble matplotlibrc files—is a fundamental feature of mpltools. The "ggplot" style, which modifies the style to resemble ggplot, is demonstrated in this dataset.

utilising the corr function to examine the dataset's correlation.

HeatMap :

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

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



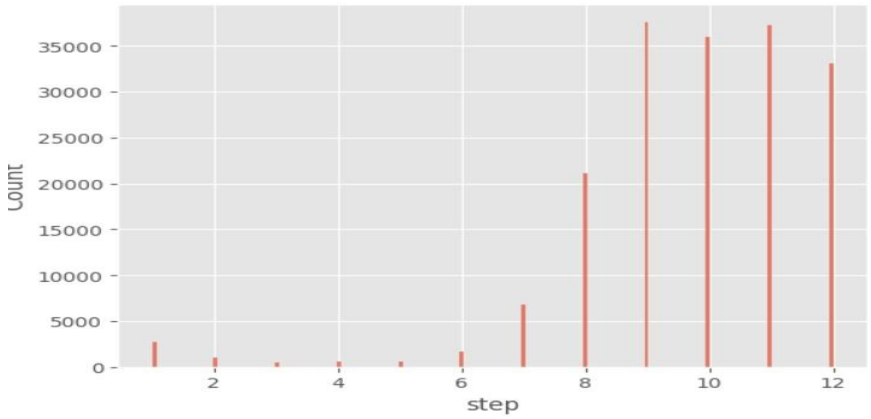
Activity 3: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed three different graphs such as histplot , boxplot and countplot.

- Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

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

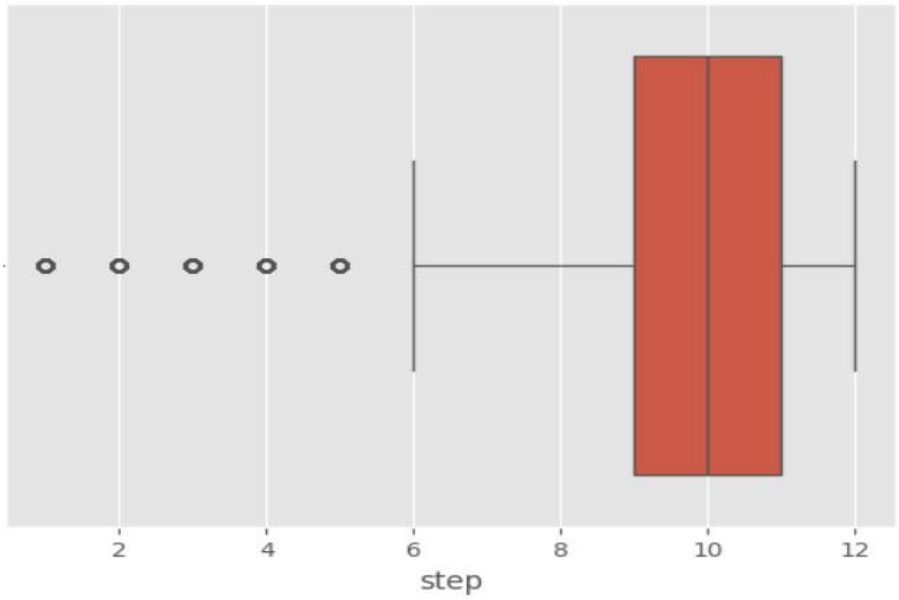
```
<Axes: xlabel='step', ylabel='Count'>
```



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')
```

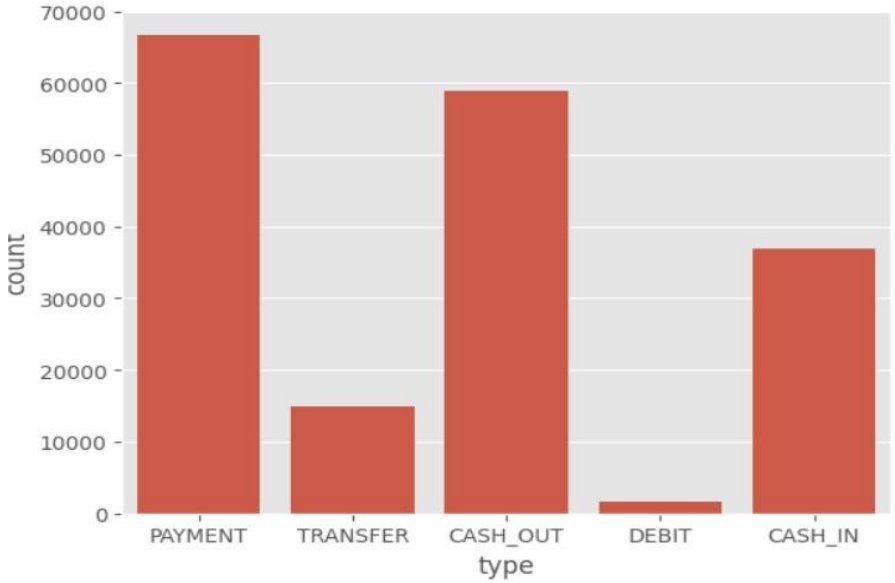
```
<Axes: xlabel='step'>
```



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

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

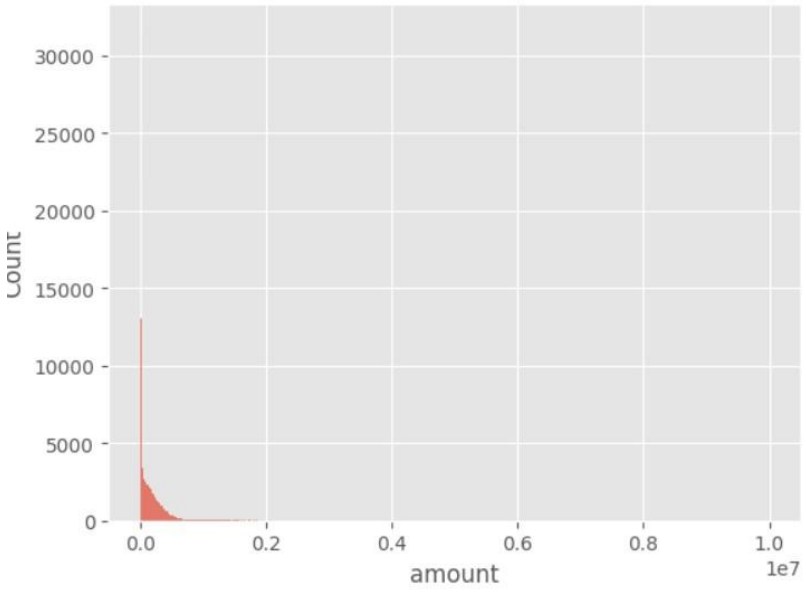
```
<Axes: xlabel='type', ylabel='count'>
```



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

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

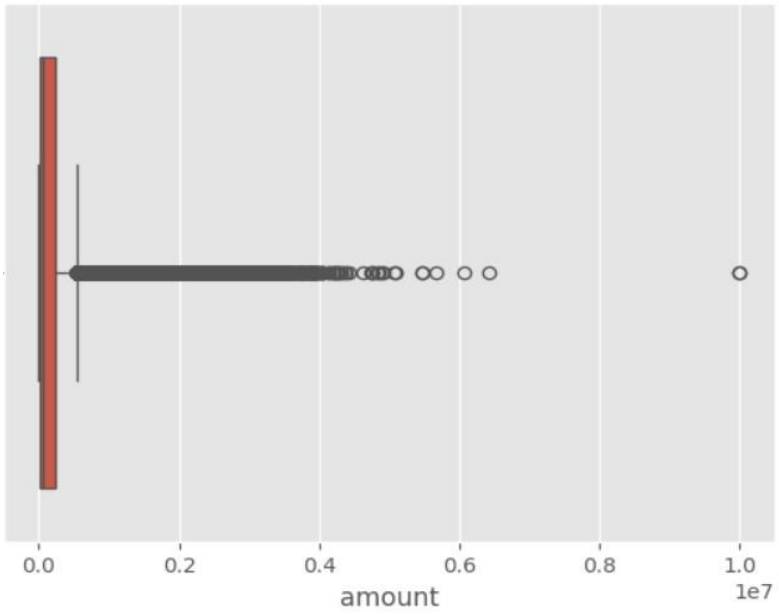
```
<Axes: xlabel='amount', ylabel='Count'>
```



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.

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

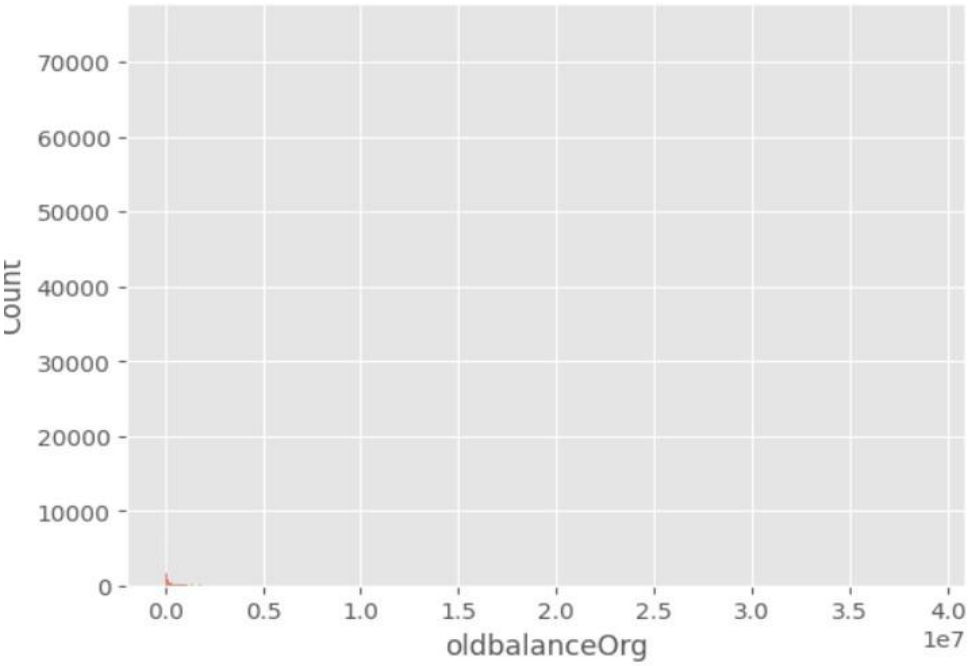
```
<Axes: xlabel='amount'>
```



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

```
sns.histplot(df,x="oldbalanceOrg")
```

```
<Axes: 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.

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

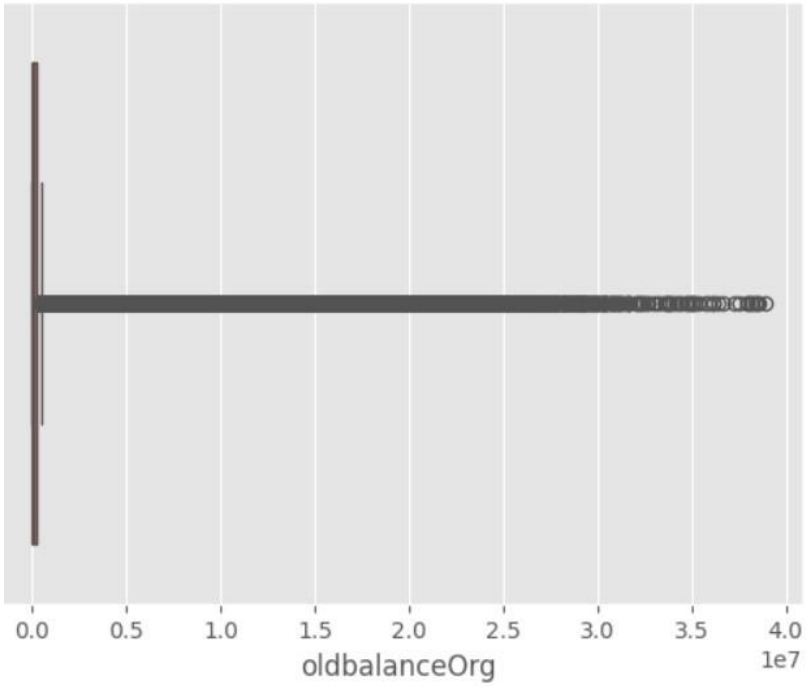
nameDest	count
C985934102	81
C1286084959	80
C248609774	75
C1590550415	74
C2083562754	74
...	...
C856483013	1
M229502821	1
M1246844215	1
M553851243	1
M1119685331	1

36417 rows × 1 columns

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

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

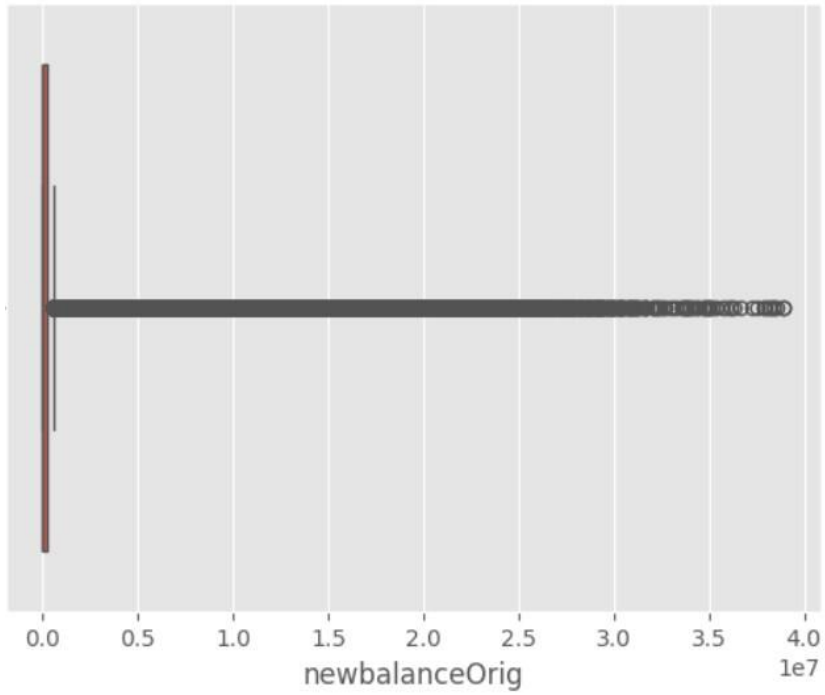
```
<Axes: xlabel='oldbalanceOrg'>
```



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

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

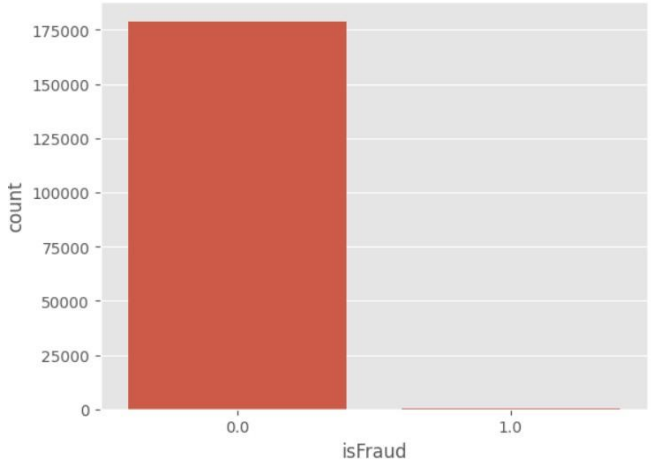
```
<Axes: xlabel='newbalanceOrig'>
```



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

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

```
<Axes: 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
0.0	178874
1.0	139

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
...	...	...	...	...	...	...	...	...	...	...
179009	12	CASH_IN	200271.90	C1829080213	3896164.44	4096436.33	C1769155229	531167.90	690790.44	is not fraud
179010	12	CASH_IN	266616.13	C1119089903	4096436.33	4363052.46	C859300166	6064698.49	6080565.41	is not fraud
179011	12	CASH_IN	194735.24	C595356999	4363052.46	4557787.70	C803648645	288407.25	152597.27	is not fraud
179012	12	CASH_IN	61212.03	C1705359203	4557787.70	4618999.73	C60296041	2492638.18	2620615.37	is not fraud
179013	12	CASH_IN	16.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN

179014 rows × 10 columns

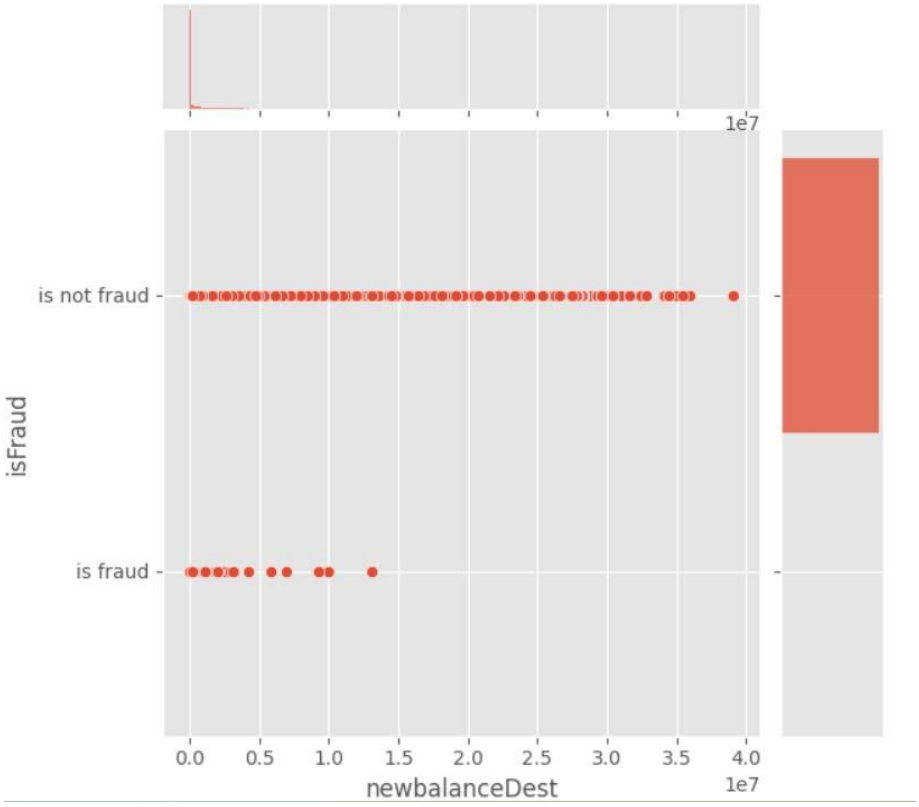
converting 0-means: is not fraud and 1-means: is fraud using the loc technique here

Activity 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')
```

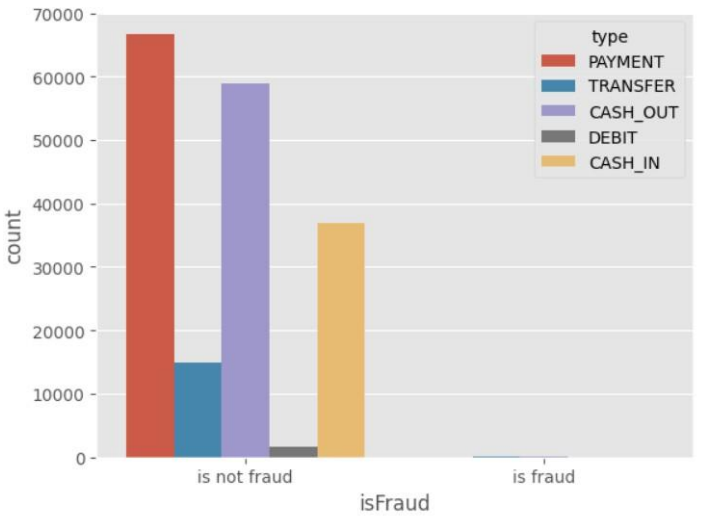
```
<seaborn.axisgrid.JointGrid at 0x7c1d71e094f0>
```



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

```
<Axes: xlabel='isFraud', 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')
```

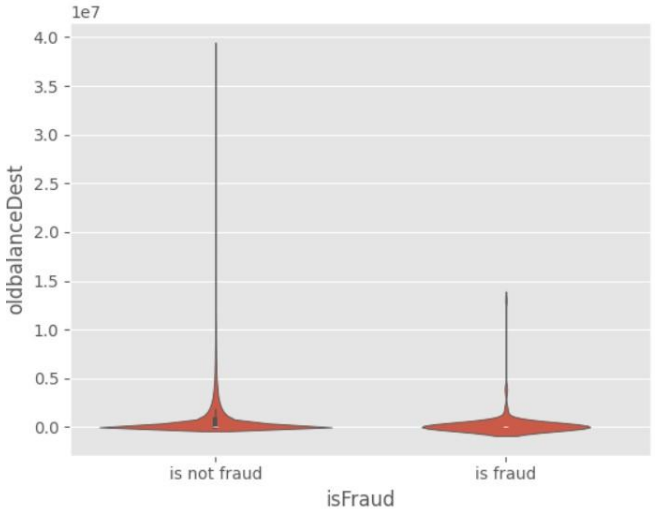
```
<Axes: xlabel='isFraud', ylabel='newbalanceOrig'>
```

Violin Plot:

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')
```

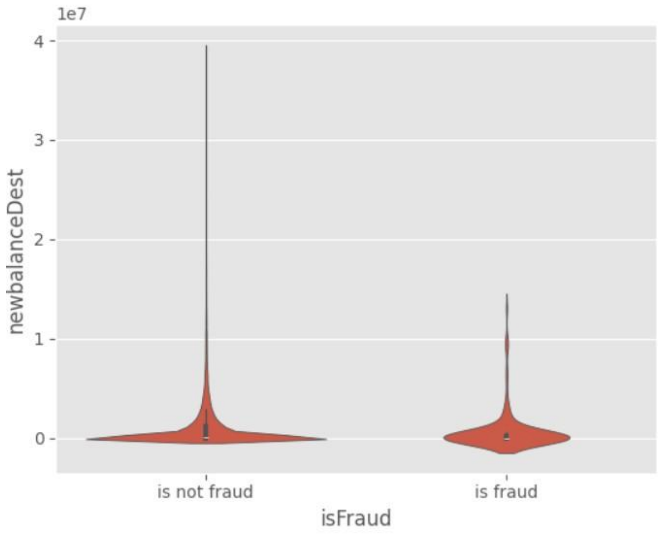
```
<Axes: xlabel='isFraud', ylabel='oldbalanceDest'>
```



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 6: 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	179014.000000	179014	1.790140e+05	179013	1.790130e+05	1.790130e+05	179013	1.790130e+05	1.790130e+05	179013
unique	NaN	5	NaN	179010	NaN	NaN	86417	NaN	NaN	2
top	NaN	PAYMENT	NaN	C44568807	NaN	NaN	C985934102	NaN	NaN	is not fraud
freq	NaN	66741	NaN	2	NaN	NaN	81	NaN	NaN	178874
mean	9.739222	NaN	1.801320e+05	NaN	8.915856e+05	9.089488e+05	NaN	9.272605e+05	1.188120e+06	NaN
std	1.999448	NaN	3.335828e+05	NaN	2.814423e+06	2.851534e+06	NaN	2.374230e+06	2.676059e+06	NaN
min	1.000000	NaN	3.200000e-01	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	NaN
25%	9.000000	NaN	1.166942e+04	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	NaN
50%	10.000000	NaN	6.566817e+04	NaN	1.976200e+04	0.000000e+00	NaN	4.233860e+04	1.163345e+05	NaN
75%	11.000000	NaN	2.268392e+05	NaN	1.935541e+05	2.276798e+05	NaN	7.298561e+05	1.166438e+06	NaN
max	12.000000	NaN	1.000000e+07	NaN	3.893942e+07	3.894623e+07	NaN	3.903958e+07	3.904248e+07	NaN

Milestone 3: Data Pre-processing

As we have understood how the data is lets pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of 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.

Activity 1: Checking for null values

- Let's find the shape of our dataset first, To find the shape of our data, df.shape method is used. To find the data type, df.info() function is used.

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

	0
step	0
type	0
amount	0
nameOrig	1
oldbalanceOrg	1
newbalanceOrig	1
nameDest	1
oldbalanceDest	1
newbalanceDest	1
isFraud	1

dtype: int64

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

```
df.info()
```

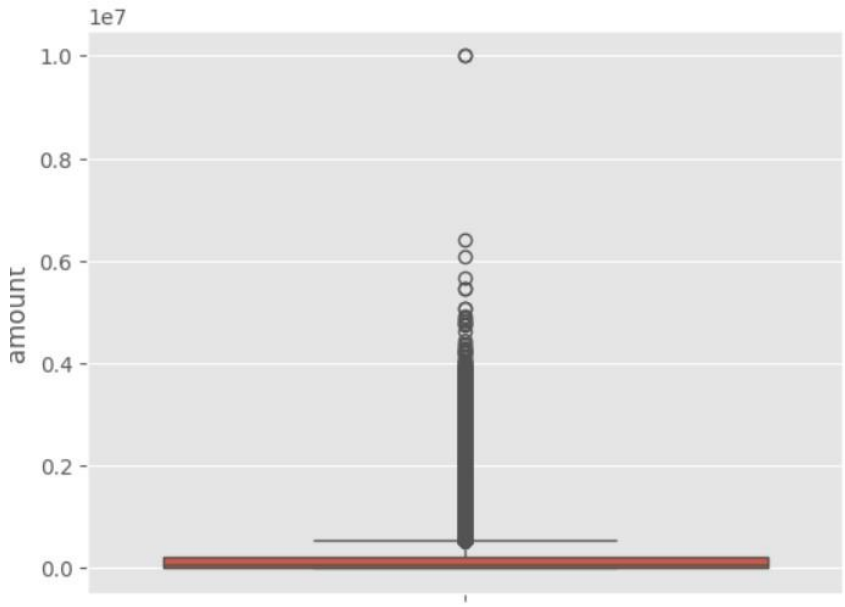
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 179014 entries, 0 to 179013
Data columns (total 10 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   step            179014 non-null  int64
 1   type            179014 non-null  object
 2   amount          179014 non-null  float64
 3   nameOrig        179013 non-null  object
 4   oldbalanceOrg   179013 non-null  float64
 5   newbalanceOrig  179013 non-null  float64
 6   nameDest        179013 non-null  object
 7   oldbalanceDest  179013 non-null  float64
 8   newbalanceDest  179013 non-null  float64
 9   isFraud         179013 non-null  object
dtypes: float64(5), int64(1), object(4)
memory usage: 13.7+ MB
```

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

Activity 2: Handling Outliers

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

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



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

Remove Outliers:

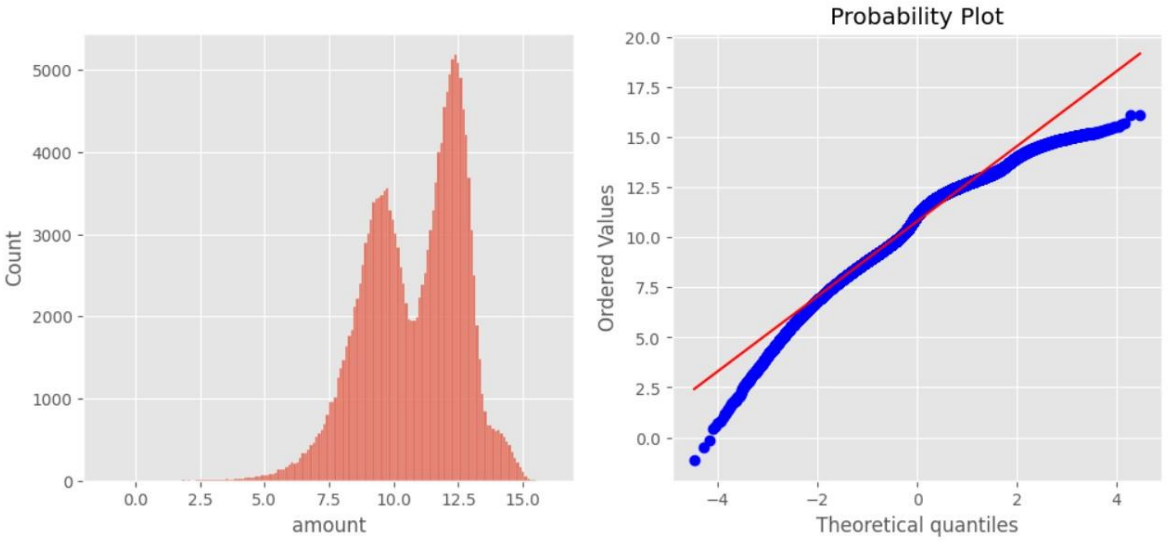
```
print(stats.mode(df['amount']))
print(np.mean(df['amount']))
```

```
lodeResult(mode=np.float64(2367.99), count=np.int64(3))
.80133.012693715
```

```
upperbound: 549593.805
lower bound: -311084.075
q1: 11670.13
q3: 226839.6
iqr: 215169.47
skewed data: 10832
skewed data: 0
```

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

```
transformationplot(np.log(df['amount']))
```



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

Activity 3:Object Data Label Encoding:

using labelencoder to encode the dataset's object type

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

```
df['type']=label_encoder.fit_transform(df['type'])
```

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

type	count
3	66741
1	58910
0	36871
4	14917
2	1574

dtype: int64

Dividing dataset into dependent and independent y and x respectively.

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

Activity 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.

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

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((143210, 9), (35803, 9), (143210,), (35803,))
```

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 1: Random forest model

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.

```
model = RandomForestClassifier()

# Drop non-numeric columns 'nameOrig' and 'nameDest' from x_train and x_test
x_train_processed = x_train.drop(['nameOrig', 'nameDest'], axis=1)
x_test_processed = x_test.drop(['nameOrig', 'nameDest'], axis=1)

model.fit(x_train_processed, y_train)

y_pred = model.predict(x_test_processed)

test_accuracy_rfc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
ccuracy: 0.9995

lassification Report:
              precision    recall  f1-score   support

    is fraud       0.87      0.43      0.58        30
s not fraud       1.00      1.00      1.00     35773

    accuracy                           1.00     35803
   macro avg       0.93      0.72      0.79     35803
eighted avg       1.00      1.00      1.00     35803


onfusion Matrix:
[   13    17]
 [    2 35771]]
```

```
y_train_predict_rfc=model.predict(x_train_processed)
train_accuracy_rfc=accuracy_score(y_train,y_train_predict_rfc)
train_accuracy_rfc
```

```
1.0
```

Activity 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.

```
dtc=DecisionTreeClassifier()
dtc.fit(x_train_processed,y_train)
y_test_predict_dtc=dtc.predict(x_test_processed)
test_accuracy_dtc=accuracy_score(y_test,y_test_predict_dtc)
print(test_accuracy_dtc)
```

```
0.9994693182135576
```

```
y_train_predict_dtc=dtc.predict(x_train_processed)
train_accuracy_dtc=accuracy_score(y_train,y_train_predict_dtc)
train_accuracy_dtc
```

```
1.0
```

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

col_0	is fraud	is not fraud
isFraud		
is fraud	18	12
is not fraud	7	35766

```
print(classification_report(y_test,y_test_predict_dtc))
```

```
              precision    recall  f1-score   support

    is fraud       0.72      0.60      0.65        30
is not fraud       1.00      1.00      1.00     35773

    accuracy                           1.00     35803
   macro avg       0.86      0.80      0.83     35803
weighted avg       1.00      1.00      1.00     35803
```

Activity 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.

```
etc=ExtraTreesClassifier()
etc.fit(x_train_processed,y_train)
y_test_predict_etc=etc.predict(x_test_processed)
test_accuracy_etc=accuracy_score(y_test,y_test_predict_etc)
print(test_accuracy_etc)
```

```
0.9993855263525403
```

```
y_train_predict_etc=etc.predict(x_train_processed)
train_accuracy_etc=accuracy_score(y_train,y_train_predict_etc)
train_accuracy_etc
```

```
1.0
```

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

col_0	is fraud	is not fraud
isFraud		
is fraud	8	22
is not fraud	0	35773

```
print(classification_report(y_test,y_test_predict_etc))
```

```
              precision    recall  f1-score   support

    is fraud       1.00      0.27      0.42        30
is not fraud       1.00      1.00      1.00     35773

    accuracy                           1.00     35803
   macro avg       1.00      0.63      0.71     35803
weighted avg       1.00      1.00      1.00     35803
```

Activity 4: Support Vector Machine 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

```
svc=SVC()
svc.fit(x_train_processed,y_train)
y_test_predict_svc=svc.predict(x_test_processed)
test_accuracy_svc=accuracy_score(y_test,y_test_predict_svc)
print(test_accuracy_svc)
```

```
0.9991620813898276
```

```
y_train_predict_svc=svc.predict(x_train_processed)
train_accuracy_svc=accuracy_score(y_train,y_train_predict_svc)
train_accuracy_svc
```

```
0.999259828224286
```

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

col_0	is not fraud
isFraud	
is fraud	30
is not fraud	35773

```
print(classification_report(y_test,y_test_predict_svc))
```

```
              precision    recall  f1-score   support

    is fraud       0.00      0.00      0.00        30
is not fraud       1.00      1.00      1.00     35773

    accuracy                           1.00     35803
   macro avg       0.50      0.50      0.50     35803
weighted avg       1.00      1.00      1.00     35803
```

```
df.columns
```

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

```
la=LabelEncoder()
y_train1=la.fit_transform(y_train)
y_test1=la.fit_transform(y_test)
y_test1
```

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

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_train1
```

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

Activity 5: Xgboost model

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

```
xgb1=xgb.XGBClassifier()
xgb1.fit(x_train_processed,y_train1)
y_test_predict_xgb=xgb1.predict(x_test_processed)
test_accuracy_xgb=accuracy_score(y_test1,y_test_predict_xgb)
print(test_accuracy_xgb)
```

```
0.999608971315253
```

```
y_train_predict_xgb=xgb1.predict(x_train_processed)
train_accuracy_xgb=accuracy_score(y_train1,y_train_predict_xgb)
train_accuracy_xgb
```

```
0.999993017247399
```

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

col_0	0	1
row_0		
0	18	12
1	2	35771

```
print(classification_report(y_test1,y_test_predict_xgb))
```

```
              precision    recall  f1-score   support

           0       0.90      0.60      0.72        30
           1       1.00      1.00      1.00     35773

    accuracy                           1.00     35803
   macro avg       0.95      0.80      0.86     35803
weighted avg       1.00      1.00      1.00     35803
```

Activity 5: Compare the model

For comparing the above four models compareModel function is defined.

```
def compareModel():
  print("train accuracy for rfc",train_accuracy_rfc)
  print("test accuracy for rfc",test_accuracy_rfc)
  print("train accuracy for dtc",train_accuracy_dtc)
  print("test accuracy for dtc",test_accuracy_dtc)
  print("train accuracy for etc",train_accuracy_etc)
  print("test accuracy for etc",test_accuracy_etc)
  print("train accuracy for svc",train_accuracy_svc)
  print("test accuracy for svc",test_accuracy_svc)
  print("train accuracy for xgb",train_accuracy_xgb)
  print("test accuracy for xgb",test_accuracy_xgb)
```

```
compareModel()
```

```
train accuracy for rfc 1.0
test accuracy for rfc 0.9994693182135576
train accuracy for dtc 1.0
test accuracy for dtc 0.9994693182135576
train accuracy for etc 1.0
test accuracy for etc 0.9993855263525403
train accuracy for svc 0.999259828224286
test accuracy for svc 0.9991620813898276
train accuracy for xgb 0.999993017247399
test accuracy for xgb 0.999608971315253
```

After calling the function, the results of models are displayed as output. From the four model Xgboost is performing well. From the below image, We can see the accuracy of the model is 99.96% accuracy.

Activity 6: Evaluating performance of the model and saving the model

From sklearn, cross_val_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 by pickle.dump().

```
svc=SVC()
svc.fit(x_train_processed,y_train)
y_test_predict_svc=svc.predict(x_test_processed)
test_accuracy_svc=accuracy_score(y_test,y_test_predict_svc)
print(test_accuracy_svc)
```

```
0.9991620813898276
```

```
y_train_predict_svc=svc.predict(x_train_processed)
train_accuracy_svc=accuracy_score(y_train,y_train_predict_svc)
train_accuracy_svc
```

```
0.999259828224286
```

```
pickle.dump(svc,open('model.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 serverside script

Activity1: Building Html Pages:

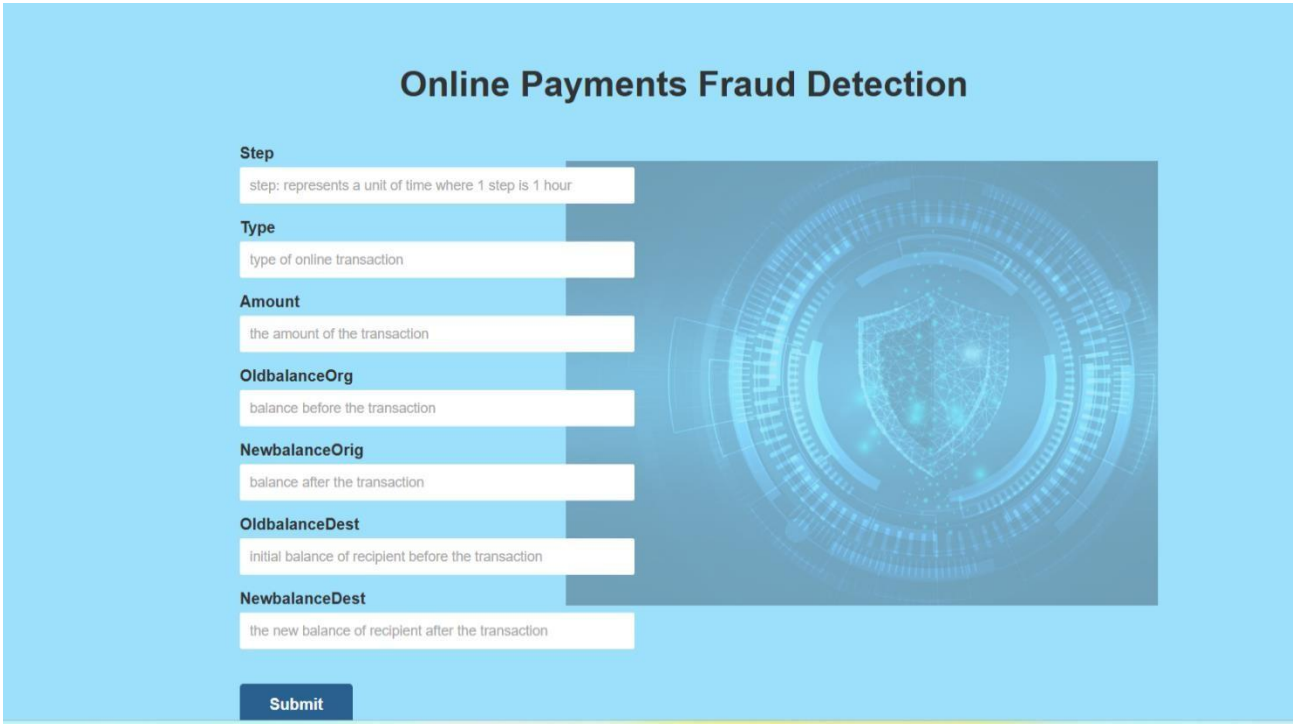
For this project create three HTML files namely

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

and save them in 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 2: Build Python code:

Import the libraries

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

Load the saved model. Importing 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('model.pkl', 'rb'))
app = Flask(__name__)
```

Render HTML page:

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

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

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

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in 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 predict():
    return render_template('predict.html')

@app.route('/predict', methods=['POST', 'GET'])
def predict_data():
    x=[[x for x in request.form.values()]]
    print(x)
    x=np.array(x)
    print(x.shape)
    prediction=model.predict(x)
    print(prediction[0])
    return render_template('submit.html', prediction_result=str(prediction))
```

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 rendered to the text that we have mentioned in the submit.html page earlier. Main Function:

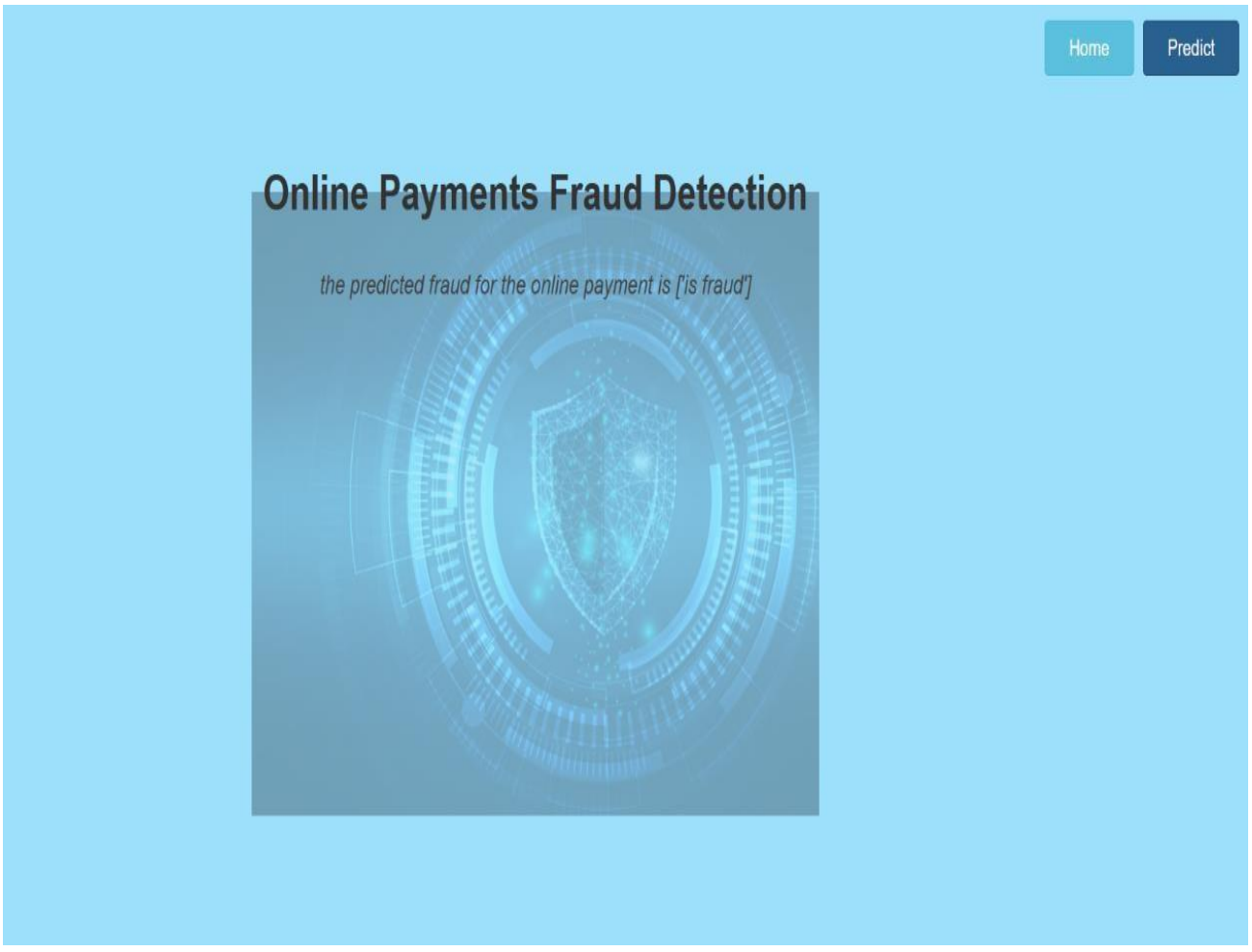
```
if __name__ == '__main__':
    app.run(debug=False)
```

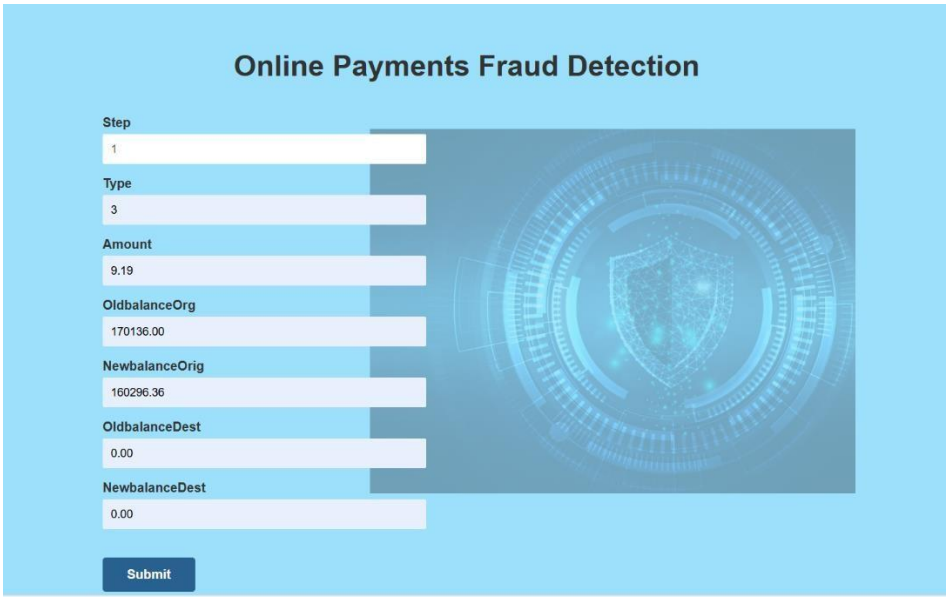
Activity 3: Run the application

- 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 top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
[Running] python -u "c:\Users\anush\Downloads\online-payment-fraud-detection-using-ml-main\online-payment-fraud-detection-using-ml-main\app.py"
 * Serving Flask app 'app'
 * Debug mode: off
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
```

Output Screenshots:





Github link: https://github.com/thotadivya28/Online-payment-fraud-detection-using-machine-learning

Video link: https://drive.google.com/file/d/1gh2cmbHaWOmTokQHV5ENBuiOJd4MdP8e/view?usp=drive_link