# ONLINE PAYMENT FRAUD DETECTION

**Name – Sachin Suthar**

# INTRODUCTION

## Project Overview:

The increasing reliance on internet transactions, particularly in the realm of ecommerce, has led to a surge in online credit/debit card transactions. Unfortunately, this growth has also given rise to a parallel increase in fraudulent activities. To address this issue, our project focuses on implementing a robust fraud detection system using machine learning.

## Purpose:

The purpose of this project is to develop a sophisticated fraud detection system that leverages machine learning algorithms to analyze and identify potential fraudulent activities in online credit/debit card transactions. By utilizing classification algorithms such as Decision tree, Random forest, SVM, Extra tree classifier, and XGBoost Classifier, we aim to improve the accuracy of fraud detection.

# Problem Statement Definition

The rapid growth of online transactions has brought about an escalating threat of fraudulent activities in the realm of online payments. Fraudsters employ sophisticated techniques to exploit vulnerabilities in payment systems, leading to financial losses for both businesses and consumers. Traditional rule based fraud detection systems are often insufficient to adapt to the evolving nature of fraudulent tactics. Therefore, there is a pressing need to develop a robust and adaptive Online Payment Fraud Detection system using Machine Learning (ML) to enhance the security and reliability of online Transactions.

# REQUIREMENT ANALYSIS

## Functional Requirements

**Real-time Transaction Monitoring**:The system must continuously monitor incoming transactions in real-time. This includes the ability to analyze and process transaction data as it occurs, providing an immediate response to potential fraudulent activities.

**Integration with Classification Algorithms**:The system should integrate with various classification algorithms such as Decision Trees, Random Forest, SVM, Extra Trees, and XGBoost. This integration is crucial for effectively analyzing transaction patterns and identifying potential instances of fraud.

**User Notification for Flagged Transactions**:Upon the identification of a potentially fraudulent transaction, the system should generate and send notifications to users. These notifications should be timely and informative, allowing users to take appropriate actions to secure their accounts.

## Non-Functional Requirements

**System Reliability**:The system must exhibit high reliability, ensuring minimal downtime and consistent performance. Reliability is crucial for maintaining continuous monitoring and timely response to potential fraud, enhancing the overall trustworthiness of the system.

**Data Security and Privacy Compliance**:To protect sensitive financial information, the system must adhere to robust data security measures. This includes encryption of data in transit and at rest, secure storage practices, and compliance with relevant privacy regulations to safeguard user information.

**User-Friendly Interface:**The user interface should be intuitive and easy to navigate. Users, including both administrators and regular account holders, should be able to interact with the system effortlessly. A user-friendly interface contributes to effective monitoring, notification management, and overall user satisfaction.

# CODING & SOLUTIONING (features added in the project along with code)

## 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 or create the dataset or 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. Please refer to the link given below to download the dataset.
Link: https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset

## Milestone 2: Visualising and analysing data

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

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

**Activity 1: Importing the libraries** Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

In [1]:
```
# Activity 1
# 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: 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.

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

```
In [3]: # Activity 2: Read the dataset
        df = pd.read_csv("PS_20174392719_1491204439457_log.csv")
```

```
In [4]: df
```

Out[4]:

| | 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

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

```
Out[5]: Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
               'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
               'isFlaggedFraud'],
              dtype='object')
```

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

```
In [6]: df.drop(['isFlaggedFraud'],axis = 1, inplace = True)
```

```
In [7]: df
```

Out[7]:

| | 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

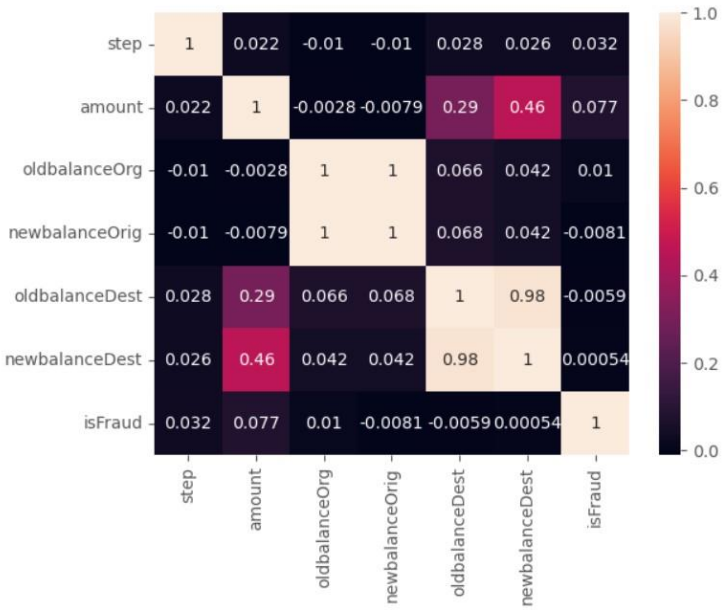
```
In [12]: # Select only numeric columns
         numeric_df = df.select_dtypes(include=['number'])

         # Calculate the correlation matrix
         correlation_matrix = numeric_df.corr()

         # Print the correlation matrix
         print(correlation_matrix)
```

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

                oldbalanceDest  newbalanceDest   isFraud
step                  0.027665        0.025888  0.031578
amount                0.294137        0.459304  0.076688
oldbalanceOrg         0.066243        0.042029  0.010154
newbalanceOrig        0.067812        0.041837 -0.008148
oldbalanceDest        1.000000        0.976569 -0.005885
newbalanceDest        0.976569        1.000000  0.000535
isFraud              -0.005885        0.000535  1.000000
```

Utilising the corr function to examine the dataset's correlation

**HEATMAP**

```
# Create a heatmap
sns.heatmap(correlation_matrix, annot=True)
plt.show()
```

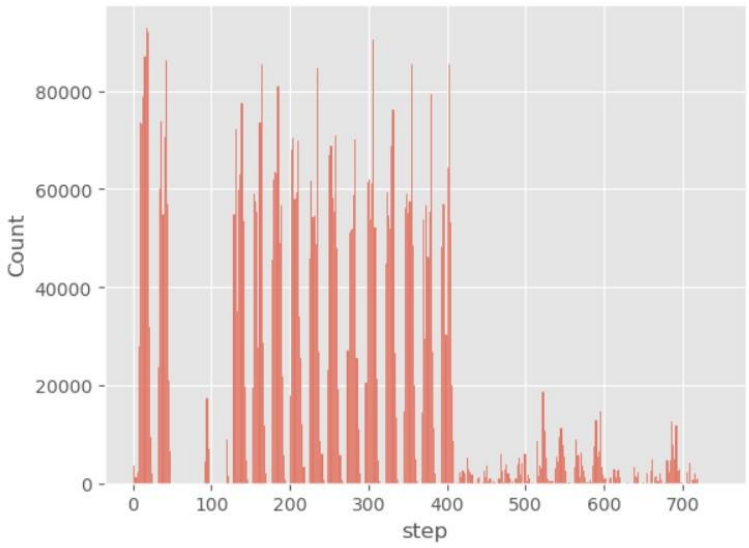



Here, a heatmap is used to understand the relationship between the input attributes and the anticipated goal value.

**Activity 3: Univariate analysis**

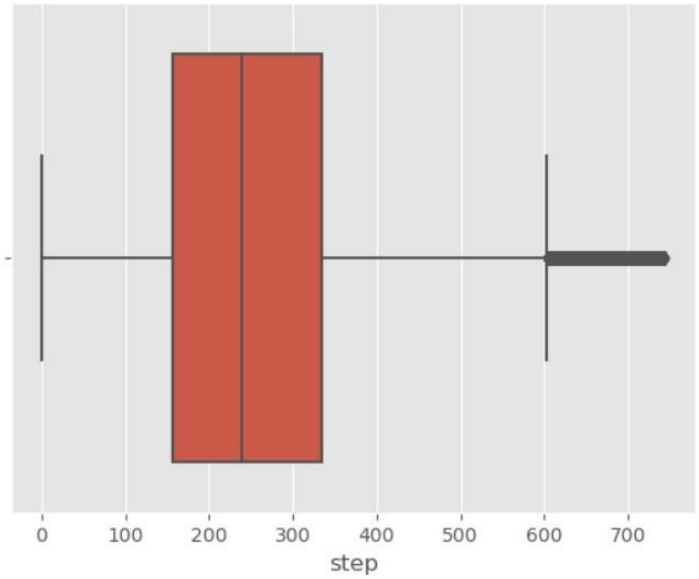
In simple words, univariate analysis is understanding the data with a single feature. Here I have displayed the graph such as histplot .

```
In [15]: # step
         sns.histplot(data=df,x='step')
Out[15]: <Axes: xlabel='step', ylabel='Count'>
```



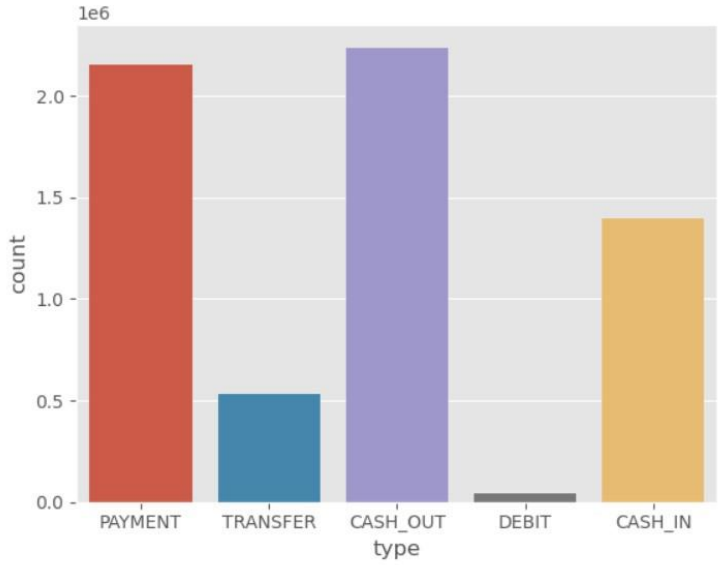
The distribution of one or more variables is represented by a histogram, a traditional visualisation tool, by counting the number of observations that fall within.

```
In [16]: sns.boxplot(data=df,x='step')
Out[16]: <Axes: xlabel='step'>
```



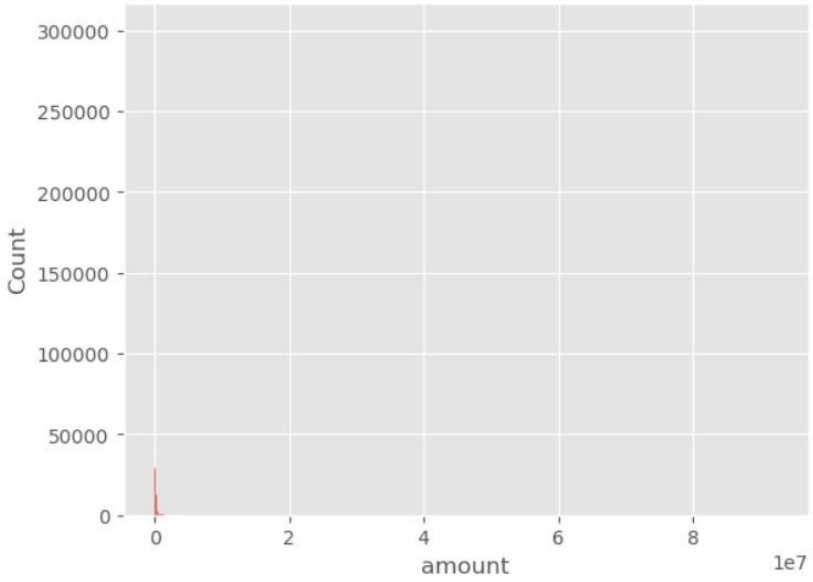

Here, the relationship between the step attribute and the boxplot is visualised.

```
In [17]: # type
         sns.countplot(data=df,x='type')
Out[17]: <Axes: xlabel='type', ylabel='count'>
```



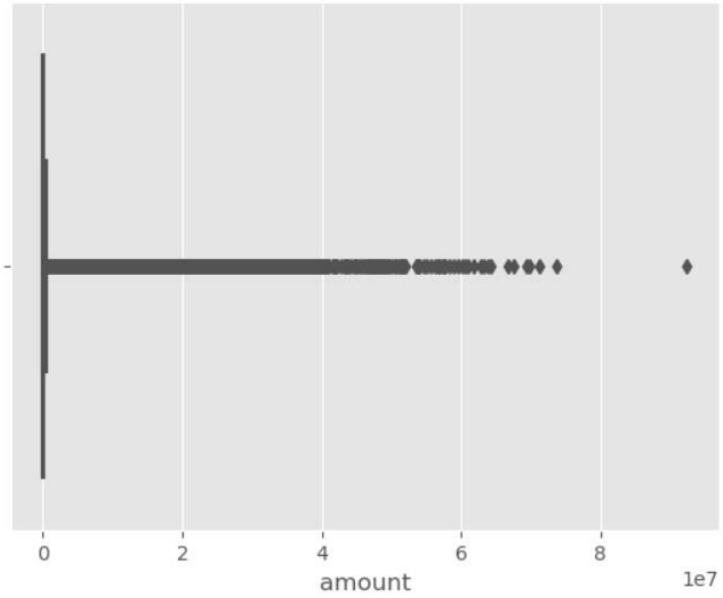

Here, the counts of observations in the type attribute of the dataset will be displayed using a countplot.

```
In [18]: # amount
         sns.histplot(data=df,x='amount')
Out[18]: <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.

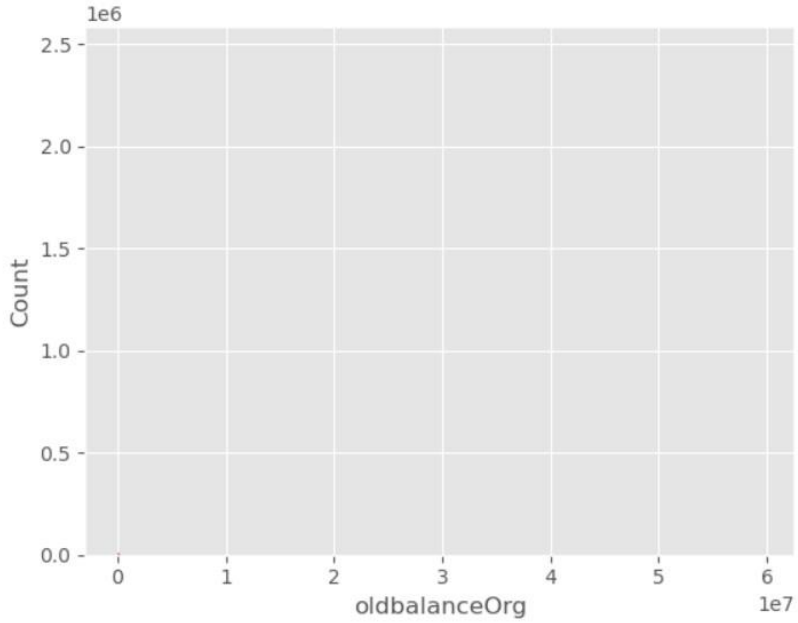
```
In [19]: # amount
         sns.boxplot(data=df,x='amount')
Out[19]: <Axes: xlabel='amount'>
```




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

```
In [20]: # oldbalanceOrg
         sns.histplot(data=df,x='oldbalanceOrg')
```

```
Out[20]: <Axes: xlabel='oldbalanceOrg', ylabel='Count'>
```



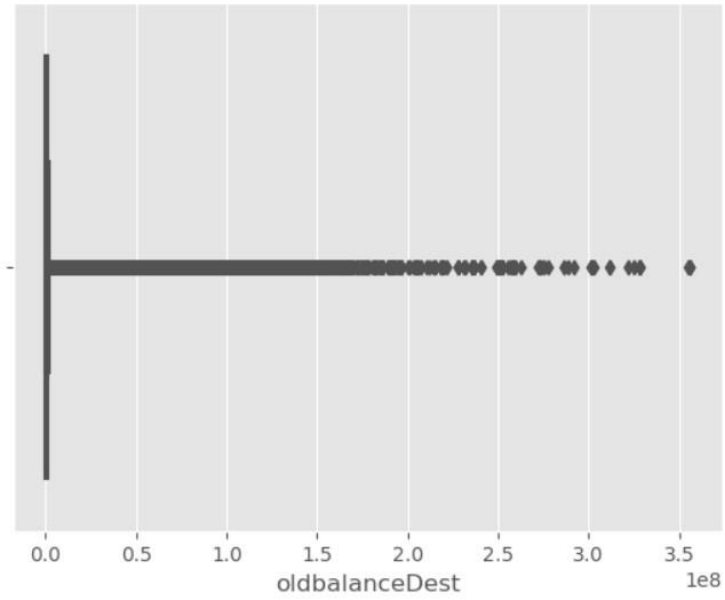

By creating bins along the data's range and then drawing bars to reflect the number of observations that fall within the oldbalanceOrg attribute in the dataset.

```
In [21]: # nameDest
         df['nameDest'].value_counts()
```

```
Out[21]: nameDest
         C1286084959    113
         C985934102     109
         C665576141     105
         C2083562754    102
         C248609774     101
                       ...
         M1470027725      1
         M1330329251      1
         M1784358659      1
         M2081431099      1
         C2080388513      1
         Name: count, Length: 2722362, dtype: int64
```

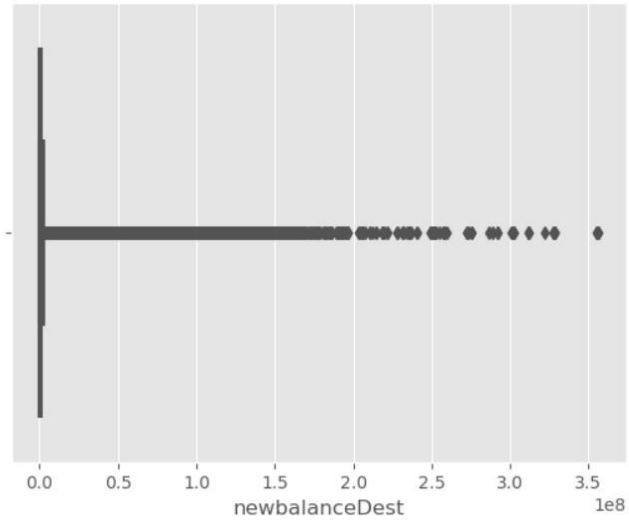
Utilising the value counts() function here to determine how many times the nameDest column appears.

```
In [22]: # oldbalanceDest
         sns.boxplot(data=df,x='oldbalanceDest')

Out[22]: <Axes: xlabel='oldbalanceDest'>
```




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

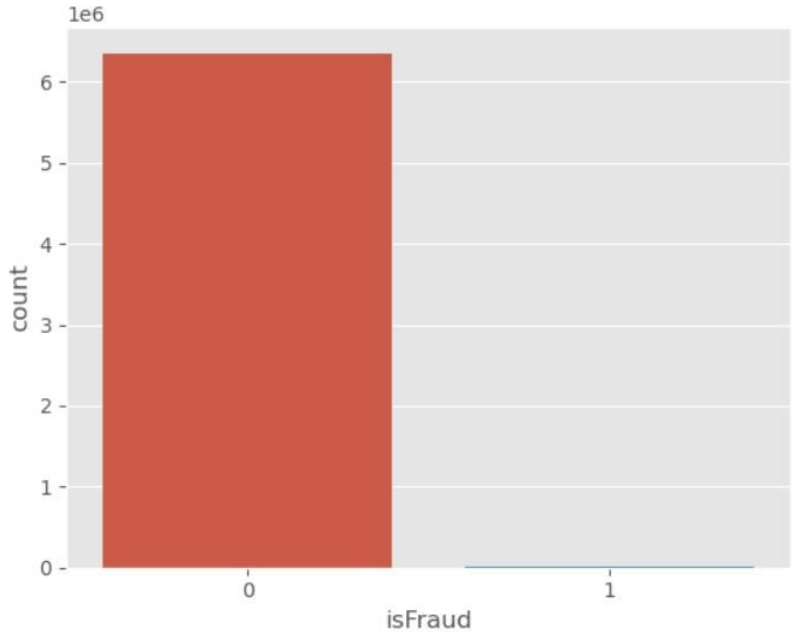
```
In [23]: # newbalanceDest
         sns.boxplot(data=df,x='newbalanceDest')

Out[23]: <Axes: xlabel='newbalanceDest'>
```




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

```
In [24]: # isFraud:
         sns.countplot(data=df,x='isFraud')
```

```
Out[24]: <Axes: xlabel='isFraud', ylabel='count'>
```




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

```
In [25]: df['isFraud'].value_counts()
```

```
Out[25]: isFraud
         0    6354407
         1       8213
         Name: count, 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.

```
In [26]: df.loc[df['isFraud']==0, 'isFraud'] = 'is not Fraud'
         df.loc[df['isFraud']==1, 'isFraud'] == 'is Fraud'

Out[26]: 2          False
         3          False
         251        False
         252        False
         680        False
                    ...
         6362615    False
         6362616    False
         6362617    False
         6362618    False
         6362619    False
         Name: isFraud, Length: 8213, dtype: bool
```

```
In [27]: df
```

Out[27]:

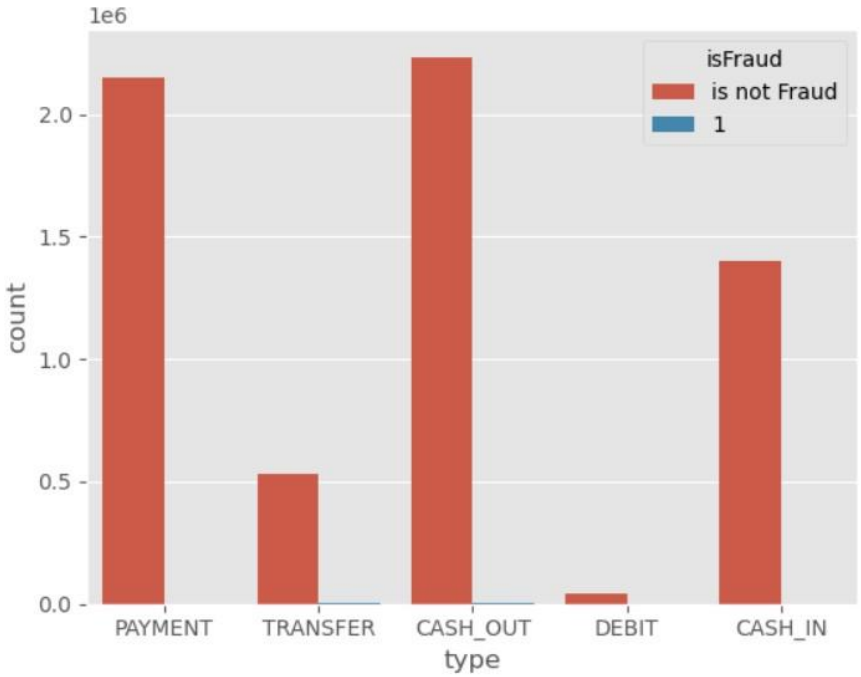
| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | PAYMENT | 9839.64 | C1231006815 | 170136.00 | 160296.36 | M1979787155 | 0.00 | 0.00 | is not Fraud |
| **1** | 1 | 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 | 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 | is not Fraud |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **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 |

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

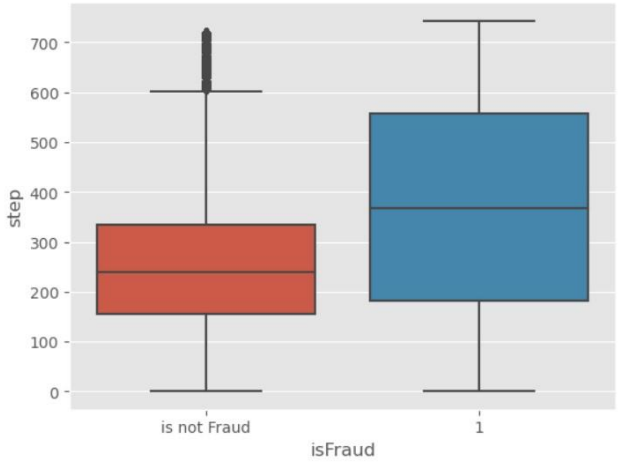
**Activity 4: Bivariate analysis**

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.

```
In [30]: sns.countplot(data=df,x='type',hue='isFraud')

Out[30]: <Axes: xlabel='type', ylabel='count'>
```



```
In [31]: sns.boxplot(data=df,x='isFraud',y='step')
Out[31]: <Axes: xlabel='isFraud', ylabel='step'>
```




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

Activity 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.

```
In [38]: df.describe(include='all')
Out[38]:
```

| | 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 | C1902386530 | 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 |

**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

```
In [40]: # shape of csv data
         df.shape

Out[40]: (6362620, 10)
```

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

```
In [41]: df.drop(['nameOrig', 'nameDest'],axis=1,inplace=True)
         df.columns

Out[41]: Index(['step', 'type', 'amount', 'oldbalanceOrg', 'newbalanceOrig',
                'oldbalanceDest', 'newbalanceDest', 'isFraud'],
               dtype='object')
```

```
In [42]: df.head()
```

Out[42]:

| | 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 | 1 |
| 3 | 1 | CASH_OUT | 181.00 | 181.0 | 0.00 | 21182.0 | 0.0 | 1 |
| 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 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.

```
In [44]: # Finding null values
         df.isnull().sum()

Out[44]: step              0
         type              0
         amount            0
         oldbalanceOrg     0
         newbalanceOrig    0
         oldbalanceDest    0
         newbalanceDest    0
         isFraud           0
         dtype: int64
```

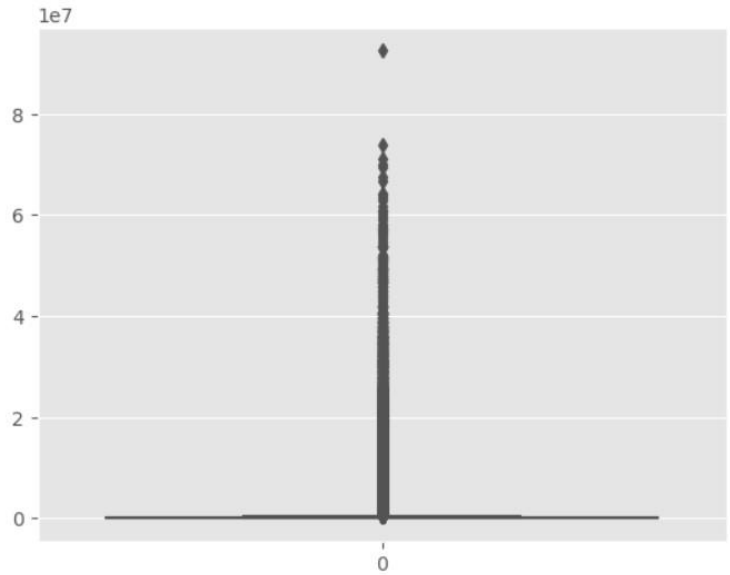
For checking the null values, data.isnull() function is used. To sum those null values we use the .sum() function to it. From the above image we found that there are no null values present in our dataset.So we can skip handling of missing values step.

```
In [45]: 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 2: Handling outliers**

```
In [47]: sns.boxplot(df['amount'])
Out[47]: <Axes: >
```




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

In [49]:
```
from scipy import stats
print(stats.mode(df['amount']))
print(np.mean(df['amount']))
```

```
ModeResult(mode=10000000.0, count=3207)
179861.90354913071
```

In [50]:
```
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 :',43)
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 : 43
IQR : 195331.9075
Upper Bound : 501719.33875
Lower Bound : -279608.29125
Skewed data: 338078
Skewed data: 0
```

**Activity 3: Object data labelencoding**

In [56]:
```
from sklearn.preprocessing import LabelEncoder

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

In [57]:
```
df['type'].value_counts()
```

Out[57]:
```
type
1    2237500
3    2151495
0    1399284
4     532909
2      41432
Name: count, dtype: int64
```

Using labelencoder to encode the dataset's object type

**X & Y Split and Scaling Columns**

```
In [58]: x = df.drop('isFraud',axis=1)
         y = df['isFraud']
```

```
In [59]: x
```

Out[59]:

| | 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

```
In [60]: y
```

```
Out[60]: 0          is not Fraud
         1          is not Fraud
         2                     1
         3                     1
         4          is not Fraud
                        ...
         6362615               1
         6362616               1
         6362617               1
         6362618               1
         6362619               1
         Name: isFraud, Length: 6362620, dtype: object
```

**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.

```
In [62]: # Train test split
         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)
```

```
In [63]: 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 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.

```
In [110]: rfc = RandomForestClassifier()
          rfc.fit(X_train, y_train)
```

```
Out[110]: RandomForestClassifier()
```

```
In [111]: y_test_pred_1 = rfc.predict(X_test)
```

```
In [112]: accuracy_test_1 = accuracy_score(y_test, y_test_pred_1)
          accuracy_test_1
```

```
Out[112]: 0.9996898545774217
```

```
In [113]: pd.crosstab(y_test, y_test_pred_1)
```

Out[113]:

| col_0 | is Fraud | is not Fraud |
|---|---|---|
| isFraud | | |
| is Fraud | 1889 | 552 |
| is not Fraud | 40 | 1906301 |

**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.

```
In [114]: from sklearn.tree import DecisionTreeClassifier
          dtc = DecisionTreeClassifier()

In [115]: dtc.fit(X_train, y_train)

Out[115]: DecisionTreeClassifier()

In [116]: y_test_pred_2 = dtc.predict(X_test)

In [117]: accuracy_test_2 = accuracy_score(y_test, y_test_pred_2)
          accuracy_test_2

Out[117]: 0.9996662793341513

In [118]: pd.crosstab(y_test, y_test_pred_2)
```

Out[118]:

| col_0 | is Fraud | is not Fraud |
|---|---|---|
| isFraud | | |
| is Fraud | 2079 | 362 |
| is not Fraud | 275 | 1906066 |

**Activity 3: ExtraTrees 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.

```
In [119]: from sklearn.ensemble import ExtraTreesClassifier
          etc = ExtraTreesClassifier()

In [120]: etc.fit(X_train, y_train)

Out[120]: ExtraTreesClassifier()
```

```
In [121]: y_test_pred_3 = etc.predict(X_test)

In [122]: accuracy_test_3 = accuracy_score(y_test, y_test_pred_3)
          accuracy_test_3

Out[122]: 0.9996767572200492

In [123]: pd.crosstab(y_test, y_test_pred_3)
```

Out[123]:

| col_0 | is Fraud | is not Fraud |
|---|---|---|
| isFraud | | |
| is Fraud | 1849 | 592 |
| is not Fraud | 25 | 1906316 |

**Activity 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.

```
In [38]: from sklearn.svm import SVC
         svc = SVC()

In [ ]: svc.fit(X_train, y_train)

In [ ]: y_test_pred_4 = svc.predict(X_test)

In [ ]: accuracy_test_4 = accuracy_score(y_test, y_test_pred_4)
        accuracy_test_4

In [ ]: pd.crosstab(y_test, y_test_pred_4)
```

**Activity 5: Evaluating performance of the model and saving the model**

Our model is performing well. So, we are saving the model is svc by pickle.dump().

```
In [ ]: import pickle

In [ ]: 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 server side script

**Activity1: Building Html Pages:**

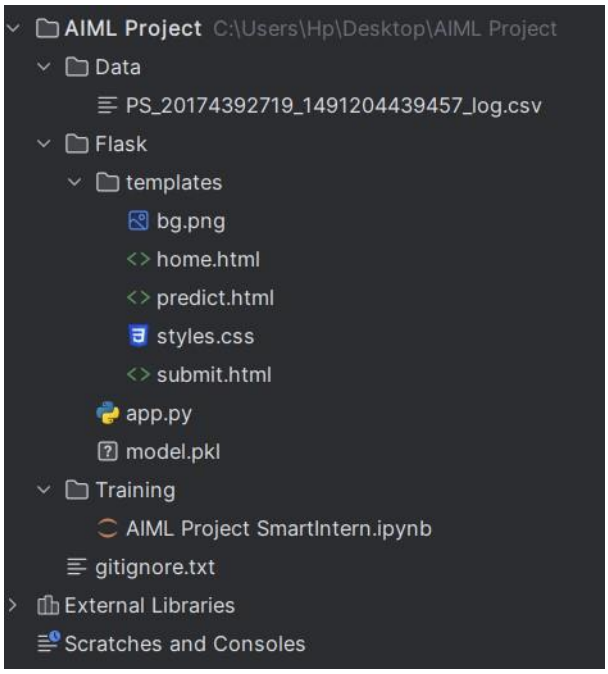
For this project create three HTML files namely

● index.html ● predict.html

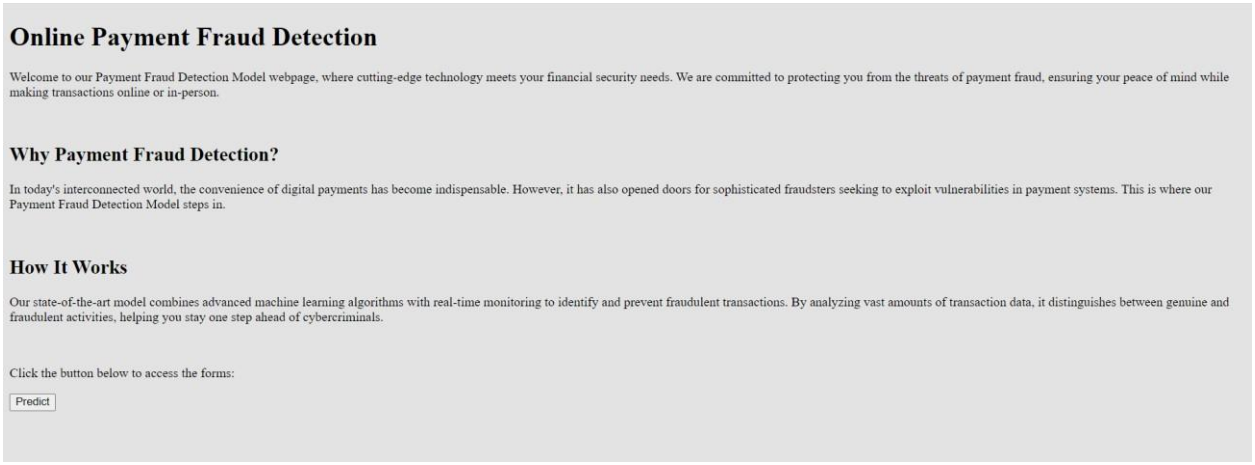
● submit.html

and save them in the templates folder.

As shown in the project structure

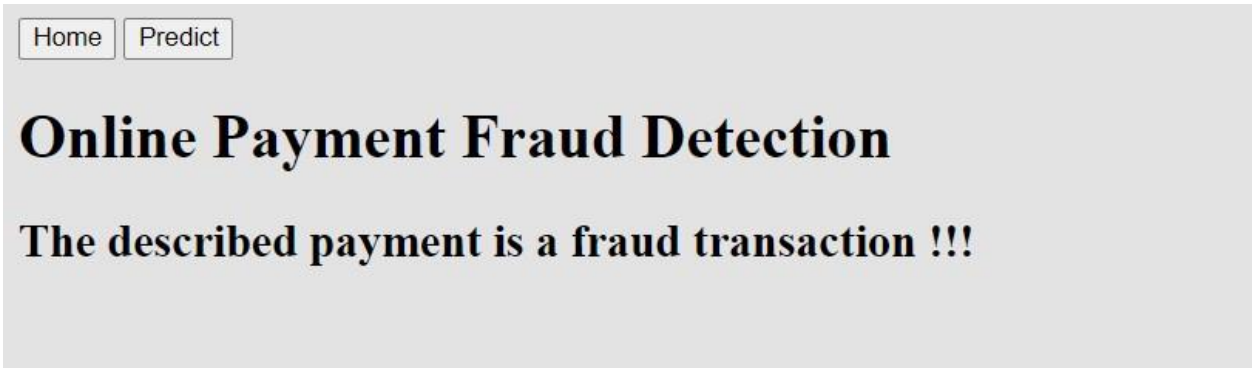



Let's see how our home.html page looks like:




Now when you click on predict button you will get redirected to predict.html Let's look how our predict.html file looks like:


Now when you click on submit button you will get redirected to submit.htm Let's look how our submit.html file looks like:




**Activity 2: Build Python code:**

Import the libraries
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.

```
from flask import Flask, request, render_template
import pickle as pkl

model = pkl.load(open("model.pkl", 'rb'))

app = Flask(__name__)
```

Render HTML page:

```
@app.route("/home.html")
def home1():
    return render_template("home.html")


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

1 usage (1 dynamic)
@app.route( rule: '/submit.html', methods=['POST'])
def predict():
    # Get user input from the form
    feature1 = float(request.form['step'])
    feature2 = float(request.form['type'])
    feature3 = float(request.form['amount'])
    feature4 = float(request.form['oldbalanceOrg'])
    feature5 = float(request.form['newbalanceOrg'])
    feature6 = float(request.form['oldbalanceDest'])
    feature7 = float(request.form['newbalanceDest'])
```

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=True)
```

**Activity 3: Run the application**

● Open VSCODE 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

```
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)
```

# PERFORMANCE TESTING

## Performance Metrics

**Model Performance Testing:**

Since the outcome of our project needed to be predicted as either "is a fraud" or "is not a fraud," a classification-based model was necessary.

Random Forest classifier, Decision Tree classifier, Extra Tree classifier, SVM classifier were the models utilized in the projects.

The metrics reports for each model are as follows:

Random Forest classifier:

1. Test accuracy

```
y_pred=rfc.predict(X_test)
print("Training Score",accuracy_score(y_train_smote,rfc.predict(x_train_smote)))
print("Testing Accuracy",accuracy_score(y_test,y_pred))
print(X_test.shape)
```

```
Testing Accuracy 0.9837896584810781
```

2. Train accuracy

```
y_pred=rfc.predict(X_test)
print("Training Score",accuracy_score(y_train_smote,rfc.predict(x_train_smote)))
print("Testing Accuracy",accuracy_score(y_test,y_pred))
print(X_test.shape)
```

```
Training Score 0.9386852256321613
```

3.Confusion Matrix

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

| col_0 | is Fraud | is not Fraud |
|---|---|---|
| **isFraud** | | |
| **is Fraud** | 1439 | 148 |
| **is not Fraud** | 20480 | 1250454 |

3. Classification Report

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

    is Fraud       0.07      0.91      0.12      1587
is not Fraud       1.00      0.98      0.99   1270934

    accuracy                           0.98   1272521
   macro avg       0.53      0.95      0.56   1272521
weighted avg       1.00      0.98      0.99   1272521
```

Decision tree Classifier:

1. Test accuracy

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

```
y_test_pred_2 = dtc.predict(X_test)
```

```
accuracy_test_2 = accuracy_score(y_test, y_test_pred_2)
accuracy_test_2
```

```
0.9997053093819277
```

2. Train accuracy

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

```
1.0
```

3. Confusion Matrix

```
pd.crosstab(y_test, y_test_pred_2)
```

| col_0 | is Fraud | is not Fraud |
|---|---|---|
| isFraud | | |
| is Fraud | 1403 | 184 |
| is not Fraud | 191 | 1270743 |

4. Classification Report

ExtraTrees Classifier

1. Test accuracy

```
from sklearn.ensemble import ExtraTreesClassifier
etc=ExtraTreesClassifier()
etc.fit(x_train_smote,y_train_smote)
y_pred=etc.predict(X_test)
print("Training Score",accuracy_score(y_train_smote,etc.predict(x_train_smote)))
print("Testing Accuracy",accuracy_score(y_test,y_pred))
```

```
Testing Accuracy 0.9994451957963758
```

2. Train accuracy

```
Training Score 1.0
```

3. Confusion Matrix

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

| col_0 | is Fraud | is not Fraud |
|---|---|---|
| isFraud | | |
| is Fraud | 1426 | 161 |
| is not Fraud | 545 | 1270389 |

4. Classification Report

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

    is Fraud       0.72      0.90      0.80      1587
is not Fraud       1.00      1.00      1.00   1270934

    accuracy                           1.00   1272521
   macro avg       0.86      0.95      0.90   1272521
weighted avg       1.00      1.00      1.00   1272521
```

Final Prediction:

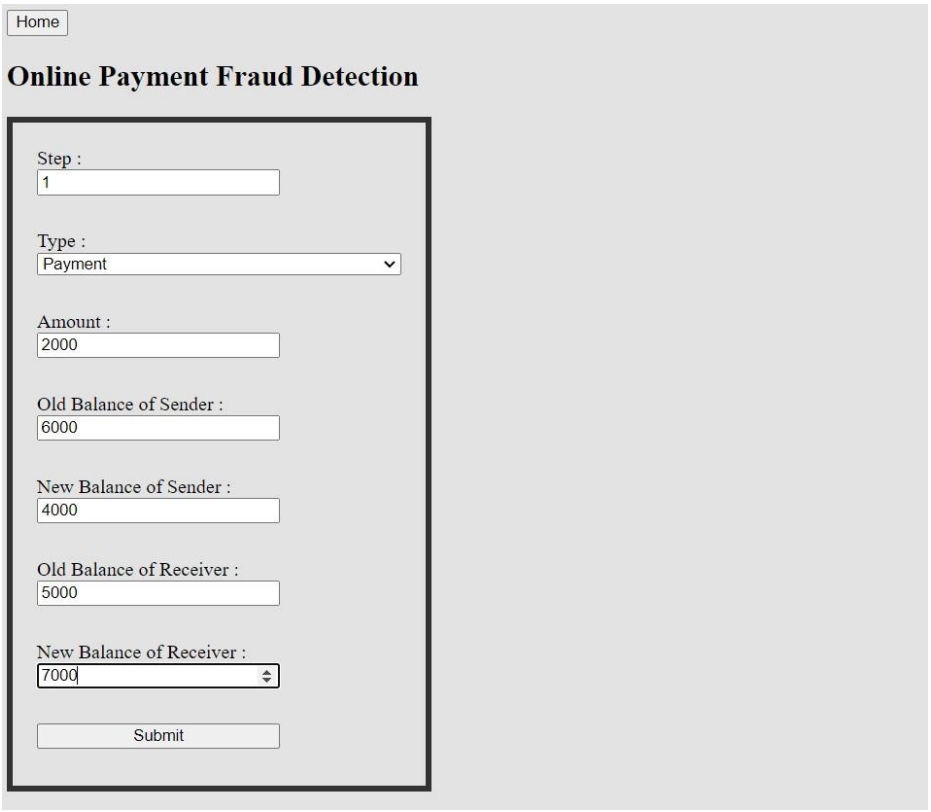
```
etc.predict([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,0.00000000e+00, 0.00000000e+00, 1.00000000e+00]
```

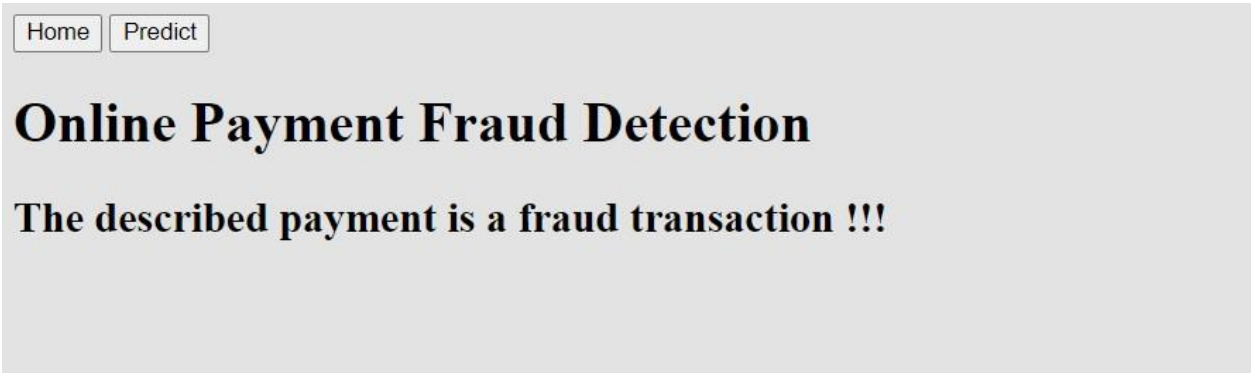
```
array(['is not Fraud'], dtype=object)
```

# RESULTS:

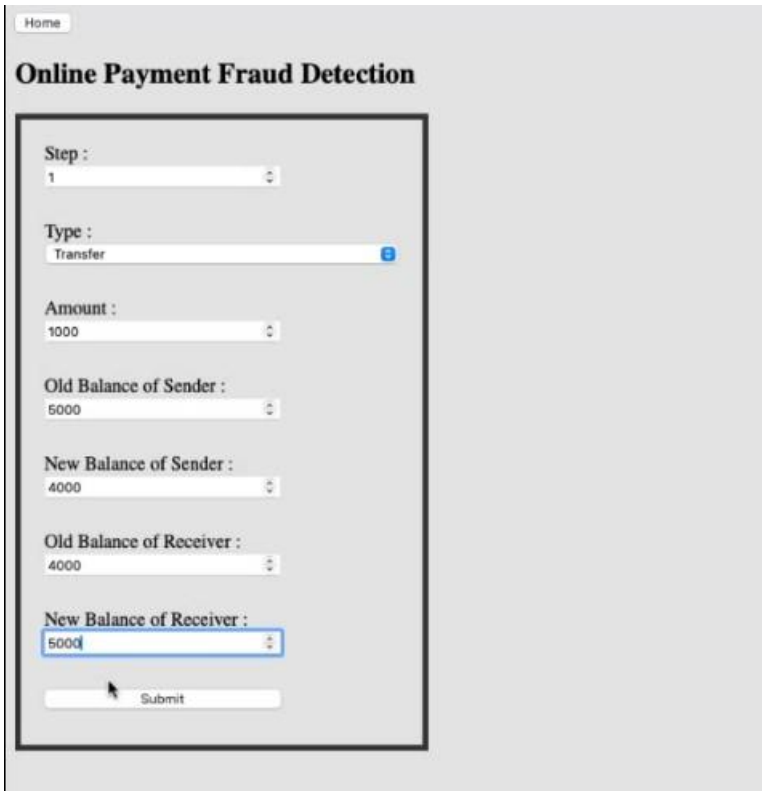
Output Screenshots:

When transaction is fraud




When transaction is not fraud


Home
Online Payment Fraud Detection
Step :
1
Type :
Transfer
Amount :
1000
Old Balance of Sender :
5000
New Balance of Sender :
4000
Old Balance of Receiver :
4000
New Balance of Receiver :
5000
Submit


Home
Predict
Online Payment Fraud Detection
The described payment is not a fraud transaction !!!

# ADVANTAGES & DISADVANTAGES

## Advantages:

**1. Improved Fraud Detection Accuracy:**
The utilization of classification algorithms such as Decision Trees, Random Forest, SVM, Extra Trees, and XGBoost enhances the accuracy of fraud detection, leading to a more robust and reliable system.

**2. Real-time Monitoring**
The implementation of real-time transaction monitoring ensures prompt identification of potentially fraudulent activities, allowing for immediate intervention and mitigation.

**3. Adaptive System:**
The integration of multiple classification algorithms makes the system adaptive to evolving patterns of fraudulent behavior. This adaptability is crucial in addressing the dynamic nature of online fraud.

**4. User Notifications**
Users receive timely notifications for flagged transactions, empowering them to take quick actions to secure their accounts. This proactive approach contributes to user trust and engagement.

**5. Compliance with Data Security Standards**
Adherence to robust data security measures ensures the protection of sensitive financial information, maintaining the confidentiality and integrity of user data.

## Disadvantages:

1.**Computational Resource Requirements**
The integration of multiple classification algorithms may demand significant computational resources, potentially leading to higher infrastructure costs.

2. **Complex Implementation:**
The complexity of implementing and maintaining a system with various classification algorithms could pose challenges in terms of development and ongoing system management.

3. **Algorithmic Sensitivity :** The effectiveness of the system heavily relies on the accuracy of the chosen classification algorithms. Sensitivity to the quality and diversity of data used for training could impact overall performance.

4.**False Positives/Negatives:**

Like any fraud detection system, there is a possibility of false positives (flagging non-fraudulent transactions) or false negatives (missing actual fraudulent transactions). Striking the right balance is a continuous challenge.

5.**User Notification Management:**

Managing user notifications effectively requires careful consideration to avoid overwhelming users with alerts. Striking the right balance between informative notifications and avoiding user fatigue is essential.

6.**Dependency on Training Data Quality:**

The accuracy of the system is heavily dependent on the quality and representativeness of the training data. Insufficient or biased training data may impact the model's ability to generalize effectively.

# CONCLUSION

In conclusion, our online payment fraud detection system, utilizing a diverse set of machine learning algorithms, has significantly enhanced the security of online transactions. The incorporation of Decision Trees, Random Forest, SVM, Extra Trees, and XGBoost classifiers ensures real-time identification of a wide range of fraudulent patterns, contributing to a more secure online financial environment. The user-centric design and timely notifications empower users and instill confidence. Despite challenges such as algorithm complexity and potential for false positives, the system's advantages in accuracy, real-time monitoring, and user empowerment make it a valuable tool in combating online payment fraud. Ongoing efforts to monitor algorithmic performance, refine models, and adapt to evolving challenges will ensure its effectiveness in the dynamic digital landscape.

# FUTURE SCOPE:

The future scope of the online payment fraud detection system is promising, with potential advancements in machine learning models, big data analytics, and behavioral analytics. Integration of blockchain technology can enhance security, while real-time updates ensure adaptability to evolving fraud patterns. Expanding the system to address cross-border fraud and collaborating with financial institutions will broaden its effectiveness. User education, focus on explainability, and mobile application integration contribute to ongoing improvement and trust. Ultimately, the system has the potential to evolve into a comprehensive, adaptive, and globally impactful solution for securing online financial transactions.