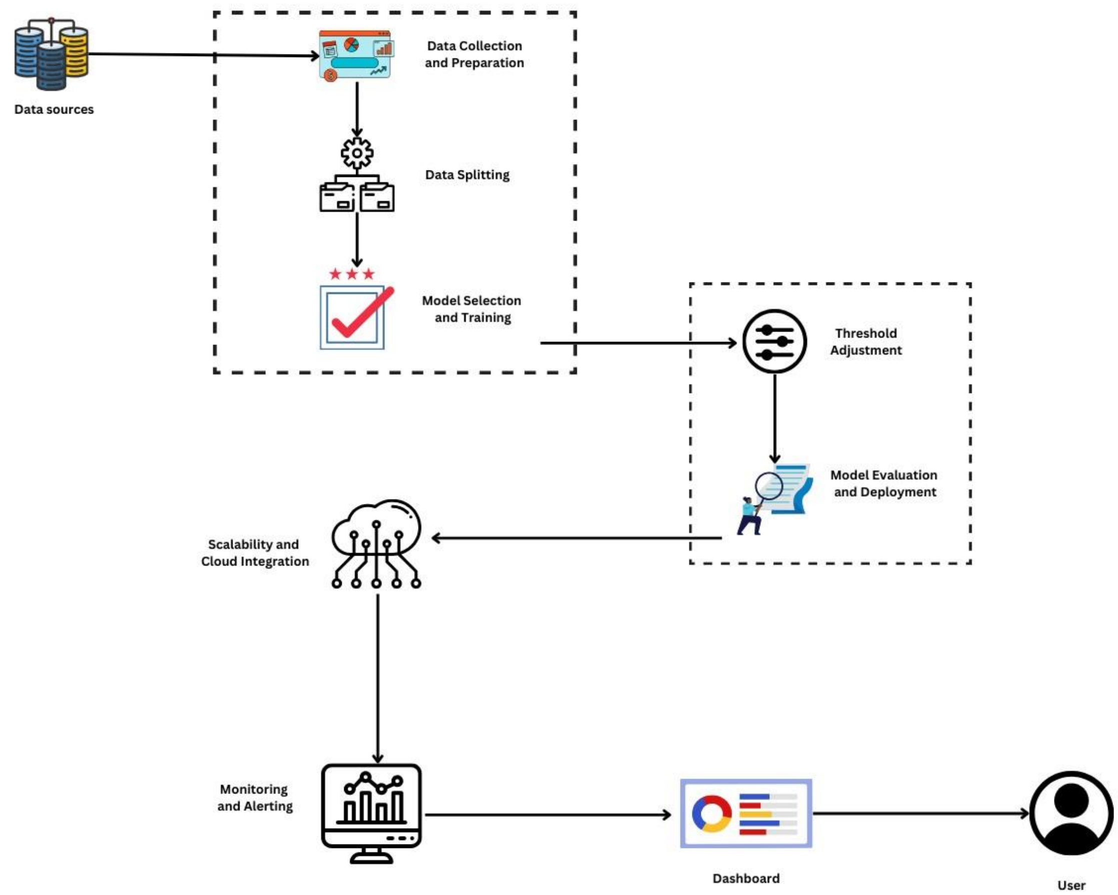
Online Payments Fraud Detection using ML

Project Description:

The rise of the internet and e-commerce has increased reliance on online credit/debit card transactions. However, this increase in utilisation has also resulted in an increase in fraud instances. Detecting these scams is difficult, as different approaches have variable degrees of accuracy and their own set of limitations. Monitoring transaction behaviour for any deviations is necessary for predicting and resolving fraud. Given the large quantity of data involved, the suggested solution addresses the credit/debit card fraud detection issue.

Classification methods such as Decision Trees, Random Forest, SVM, Extra Tree Classifier, and XGBoost Classifier are used in our strategy. We intend to determine the most successful model through rigorous training and testing. Once the best model has been found, it will be stored in pkl format. Flask integration enabling easy web application development and deployment on IBM infrastructure are the next steps.

Technical Architecture:



Pre requisites:

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

- Anaconda Navigator and VS code.
- Python packages
  - o Open VS code→view→terminal→and ensure following steps
  - o Type “pip install numpy”and click enter.
  - o Type “pip install pandas”and click enter.
  - o Type “pip install scikit-learn”and click enter.
  - o Type ”pip install matplotlib” and click enter.
  - o Type ”pip install scipy” and click enter.
  - o Type ”pip install pickle-mixin ” and click enter.
  - o Type ”pip install seaborn” and click enter.
  - o Type “pip install Flask” and click enter.

Prior Knowledge:

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

- Ml concepts(algos)
  - o Supervised Learning
  - o Unsupervised Learning
  - o Regression and Classification
  - o Decision Tree
  - o Random Forest
  - o XGboost Classifier
  - o SVM
  - o Extra tree classifier
  - o Evaluation metrics
  - o Flask Basics

Project Objectives:

When you finish this project, you will:

1. Develop a thorough understanding of key machine learning ideas and techniques.
2. Gain a thorough grasp of data, including preparation and transformation techniques, with an emphasis on dealing with outliers and visualisation principles.

In addition to these learning objectives, the project seeks to accomplish the following objectives in the field of Online Payment Fraud Detection using ML:

1. Use core machine learning techniques to detect and respond to fraudulent behaviour in online credit/debit card transactions.
2. To improve the accuracy and reliability of the fraud detection model, use data pretreatment and transformation methods.
3. Learn how to handle outliers in a dataset, resulting in a more robust fraud detection system.
4. Investigate visualisation approaches to explain patterns and abnormalities in transaction data effectively.
5. Integrate classification methods such as Decision Trees, Random Forest, SVM, Extra Tree Classifier, and XGBoost Classifier successfully to determine the most efficient model for fraud detection.
6. For practical use, save and deploy the chosen model in pkl format.
7. Create a Flask-based web application that allows users to engage with the fraud detection system in a natural way.
8. Deploy the project on IBM infrastructure to ensure real-world accessibility and scalability.

Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed 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

o Collect the dataset or create the dataset

● Visualising and analysing data

o Importing the libraries

o Read the Dataset

o Univariate analysis

o Bivariate analysis

o Descriptive analysis

● Data pre-processing

o Checking for null values

o Handling outlier

o Handling categorical(object) data

o Splitting data into train and test

● Model building

o Import the model building libraries

o Initialising the model

o Training and testing the model

o Evaluating performance of model

o Save the model

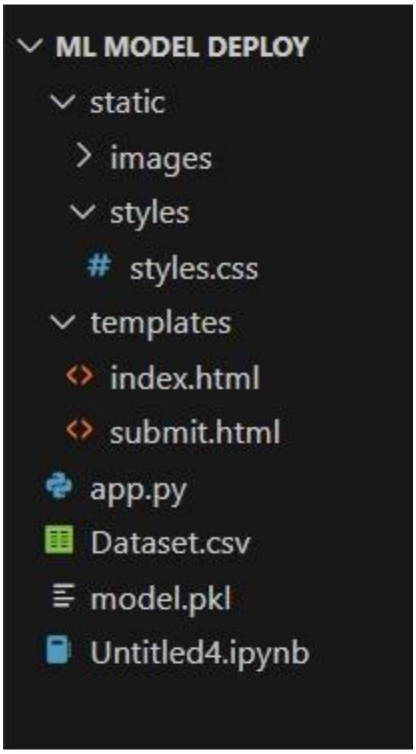
● Application Building

o Create an HTML file

o 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.

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.

```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    import seaborn as sns
    import warnings
    warnings.filterwarnings('ignore')
```

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.

```
df = pd.read_csv("C:\ML MODEL DEPLOY\Dataset.csv")
df
```

[4] Python

	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

```
df.columns
```

[5] Python

```
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.

```python
df=df.drop(columns=['isFlaggedFraud'])
df.head()
```

[6] Python

	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

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

B. oldbalanceDest: initial balance of recipient before the transaction

9. newbalanceDest: the new balance of recipient after the transaction

10. isFraud: fraud transaction

```python
df.corr(numeric_only=True)['isFraud'].sort_values(ascending=False)
```

[8] Python

```
isFraud           1.000000
amount            0.076688
step              0.031578
oldbalanceOrg     0.010154
newbalanceDest    0.000535
oldbalanceDest   -0.005885
newbalanceOrig   -0.008148
Name: isFraud, dtype: float64
```

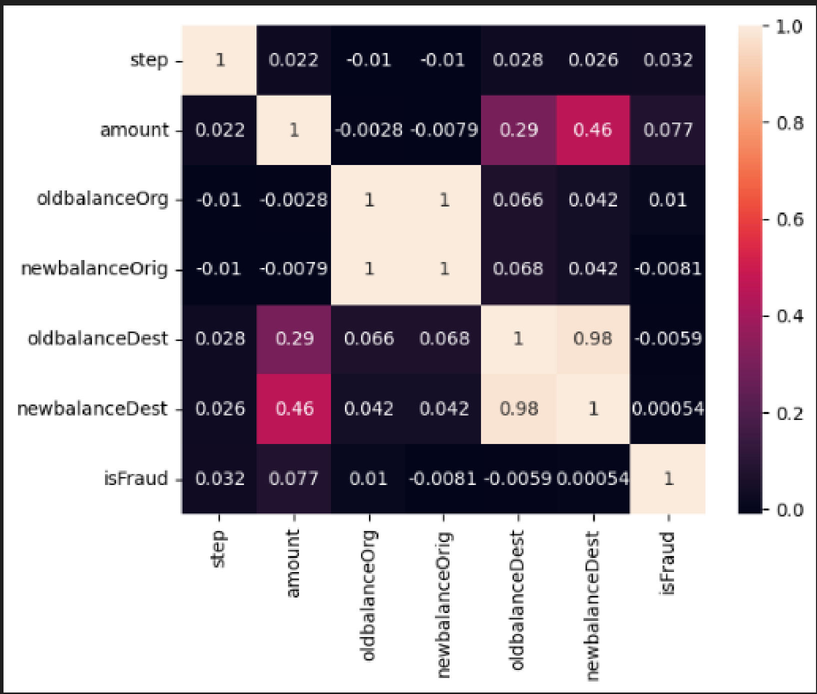
Utilising the corr function to examine the dataset's correlation

HEATMAP

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

[11]

```
<Axes: >
```



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

Activity 3: Univariate analysis

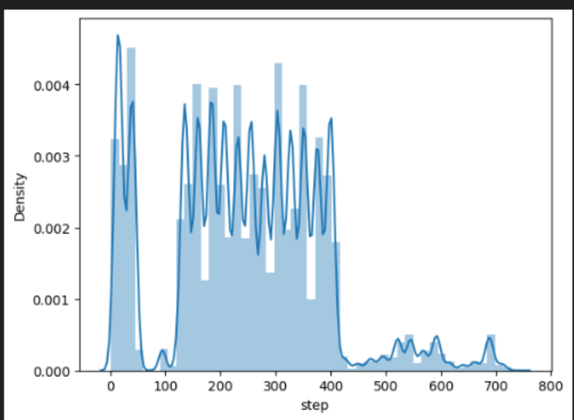
In simple words, univariate analysis is understanding the data with a single feature. Here I have displayed the graph such as distplot.

Univariate Analysis

```
sns.distplot(df['step'])
```

[12] Python

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

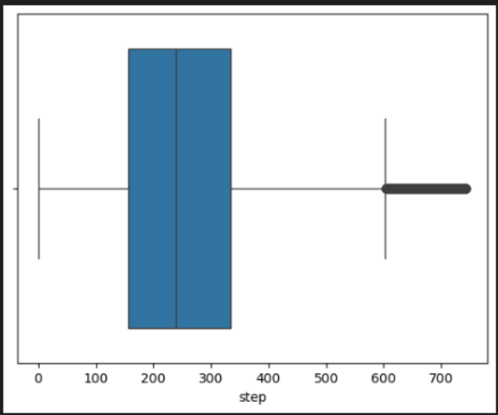


The distribution of one or more variables is represented by a histogram, a traditional visualisation tool, by counting the number of observations that fall within.

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

[14] Python

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

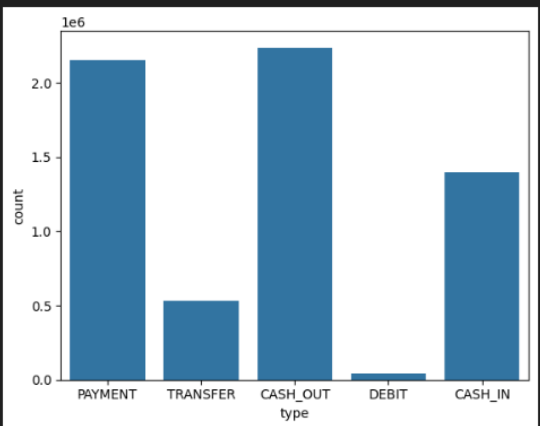


Here, the relationship between the step attribute and the boxplot is visualised.

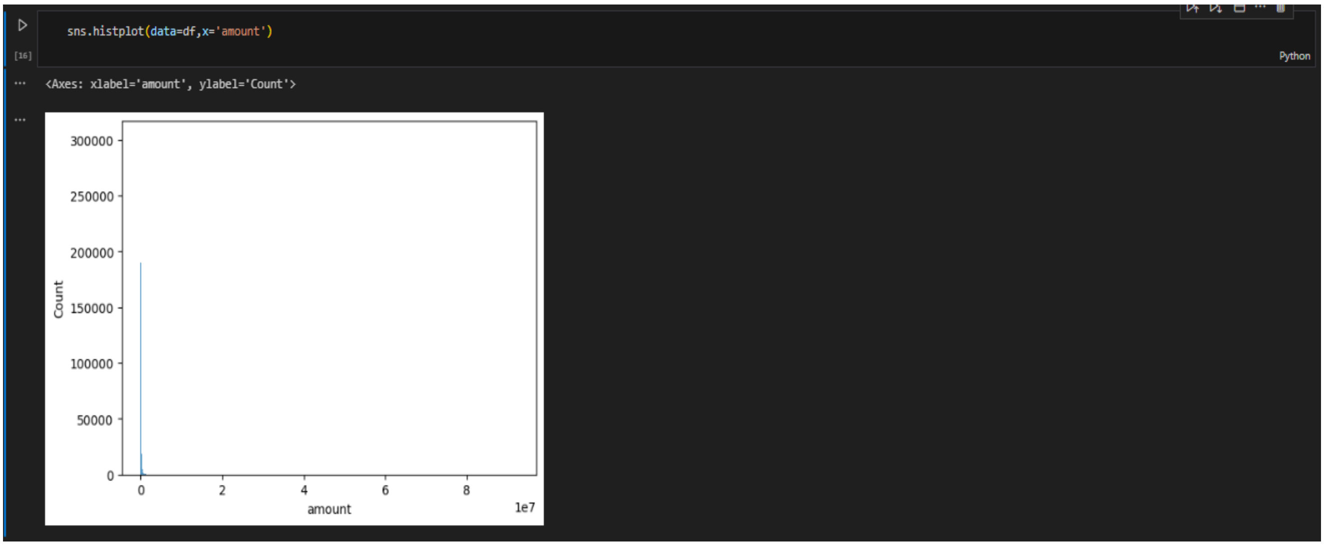
```
sns.countplot(data=df,x='type')
```

[17] Python

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

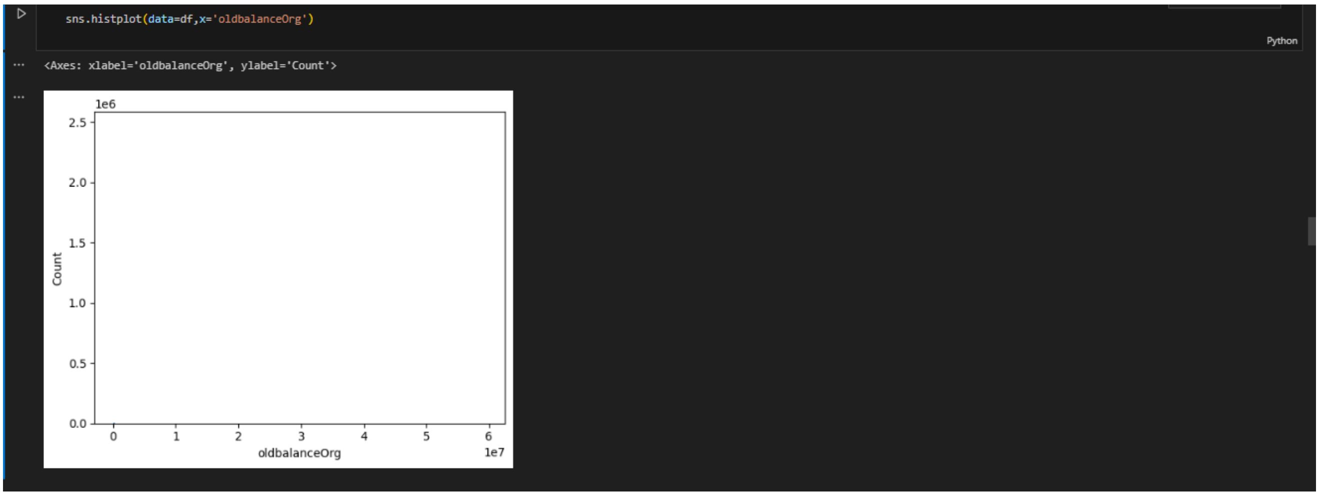


Here, the counts of observations in the type attribute of the dataset will be displayed using a countplot.



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.

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



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.

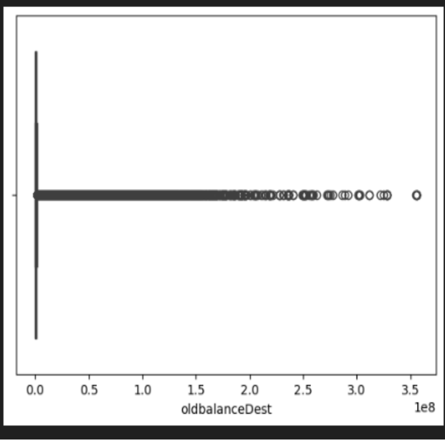
```
df['nameDest'].value_counts()
```

```
nameDest
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.

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

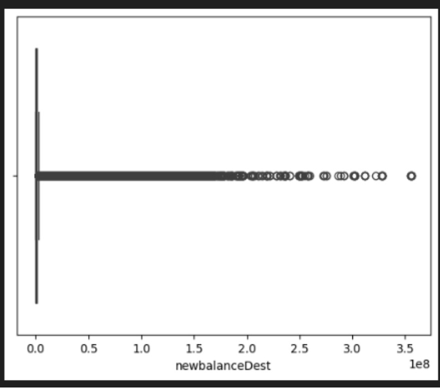
```
<Axes: xlabel='oldbalanceDest'>
```



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

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

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

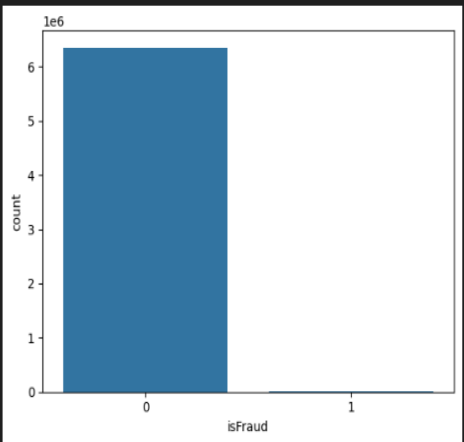


Here, the relationship between the newbalanceDest attribute and the boxplot is visualised.

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

[22] Python

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



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

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

[23] Python

```
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.

```python
df.loc[df['isFraud']==0,'isFraud']='is not Fraud' ...
```

```python
df.head()
```

[27] Python

	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	is not Fraud
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	is not Fraud
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	is Fraud
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	is Fraud
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	is not Fraud

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

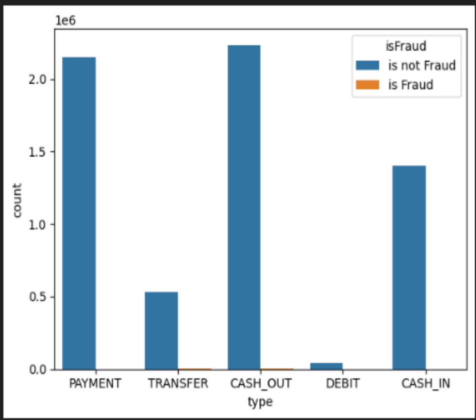
Activity 4: Bivariate analysis

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.

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

[28] Python

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



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.

```
df.describe()
```

[9] Python

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+06	1.290820e-03
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06	3.590480e-02
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+05	0.000000e+00
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+06	0.000000e+00
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+08	1.000000e+00

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

```python
df.shape
```

```
(6362620, 10)
```

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

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

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.

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

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

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

```python
df.info()
```

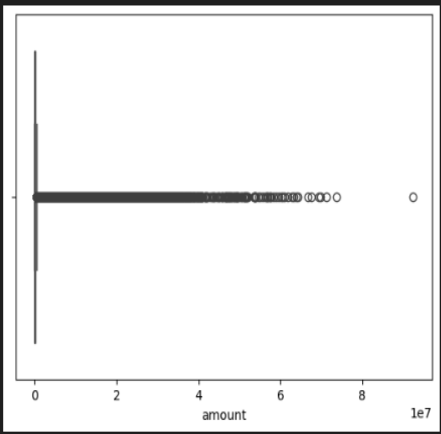
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 10 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
dtypes: float64(5), int64(2), object(3)
memory usage: 485.4+ MB
```

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

Activity 2: Handling outliers

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

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



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

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

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

```python
q1 = df['amount'].quantile(0.25)
q3 = df['amount'].quantile(0.75)
IQR = q3-q1
whisker_width = 1.5
lower_whisker = q1 -(whisker_width*IQR)
upper_whisker = q3 + (whisker_width*IQR)
df['amount']=np.where(df['amount']>upper_whisker,upper_whisker,np.where(df['amount']<lower_whisker,lower_whisker,df['amount']))
```

```python
print('Q1 = ', q1)
print('Q3 = ', q3)
print('IQR = ', IQR)
print('Upper Bound = ', upper_whisker)
print('Lower Bound = ', lower_whisker)
```

```
Q1 =  13389.57
Q3 =  208721.4775
IQR =  195331.9075
Upper Bound =  501719.33875
Lower Bound =  -279608.29125
```

Activity 3: Object data labelencoding

```python
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
```

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

```
array([3, 4, 1, 2, 0])
```

Using labelencoder to encode the dataset's object type

X & Y Split and Scaling Columns

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

```
x_scaled.head()
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	0.0	0.75	0.778262	0.002855	0.003233	0.000000	0.0
1	0.0	0.75	0.684441	0.000357	0.000391	0.000000	0.0
2	0.0	1.00	0.552912	0.000003	0.000000	0.000000	0.0
3	0.0	0.25	0.552912	0.000003	0.000000	0.000059	0.0
4	0.0	0.75	0.787875	0.000697	0.000603	0.000000	0.0

```
y.head()
```

```
0    is not Fraud
1    is not Fraud
2        is Fraud
3        is Fraud
4    is not Fraud
Name: isFraud, 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.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_scaled,y, random_state=50, test_size=0.3)
```

```
X_train.shape
```

```
(4453822, 7)
```

```
X_test.shape
```

```
(1908782, 7)
```

```
y_train.shape
```

```
(4453822,)
```

```
y_test.shape
```

```
(1908782,)
```

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: Logistic Regression

Logistic regression is a statistical method for predicting the probability of a binary outcome (e.g., yes or no, spam or not spam). It is a popular classification algorithm due to its simplicity and interpretability.The basic idea behind logistic regression is to fit a linear equation to the data and then use the sigmoid function to transform the linear output into a probability

Logistic Regression

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
y=df['isFraud']
X=df.drop('isFraud',axis=1)
X_scaled=pd.DataFrame(scale.fit_transform(X),columns=X.columns)
X_scaled.head()
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.2,random_state=42)
```

```
from imblearn.over_sampling import SMOTE
y_train.value_counts()
sm=SMOTE(random_state=42)
x_train_smote,y_train_smote=sm.fit_resample(X_train,y_train)
print("The values after scaling are")
y_train_smote.value_counts()
```

```
The values after scaling are

isFraud
is not Fraud    5083473
is Fraud        5083473
Name: count, dtype: int64
```

Between 0 and 1. The sigmoid function is a mathematical function that squashes any real number to a value between 0 and 1.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report,roc_auc_score,roc_curve
lr=LogisticRegression()
lr.fit(x_train_smote,y_train_smote)
y_pred=lr.predict(X_test)
print("Training Score",accuracy_score(y_train_smote,lr.predict(x_train_smote)))
print("Testing Accuracy",accuracy_score(y_test,y_pred))
```

```
Training Score 0.8966881500108292
Testing Accuracy 0.9662284551689128
```

```python
confusion_matrix(y_test,y_pred)
```

```
array([[   1312,     275],
       [  42700, 1228234]], dtype=int64)
```

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

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

```
              precision    recall  f1-score   support

    is Fraud       0.03      0.83      0.06      1587
is not Fraud       1.00      0.97      0.98   1270934

    accuracy                           0.97   1272521
   macro avg       0.51      0.90      0.52   1272521
weighted avg       1.00      0.97      0.98   1272521
```

Activity 2: 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.

Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier
rfc =RandomForestClassifier(criterion='entropy', max_depth=5,
                            min_samples_leaf=10, min_samples_split=10,
                            n_estimators=50)
rfc.fit(x_train_smote,y_train_smote)
```

```python
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
Testing Accuracy 0.9837896584810781
(1272521, 7)
```

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

```python
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
```

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

Activity 3: 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.

Decision Tree Classifier

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

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

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

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

```
0.9997053093819277
```

```
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

Activity 4: 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.

Extra tree Classifier

```python
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))
```

```
Training Score 1.0
Testing Accuracy 0.9994451957963758
```

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

```python
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
```

```python
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)
```

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().

```python
import pickle
pickle.dump(rfc, 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:

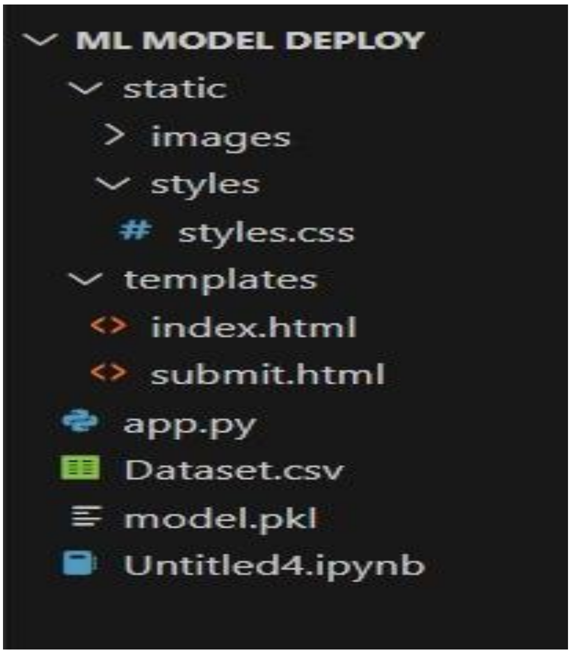
For this project create three HTML files namely

● index.html

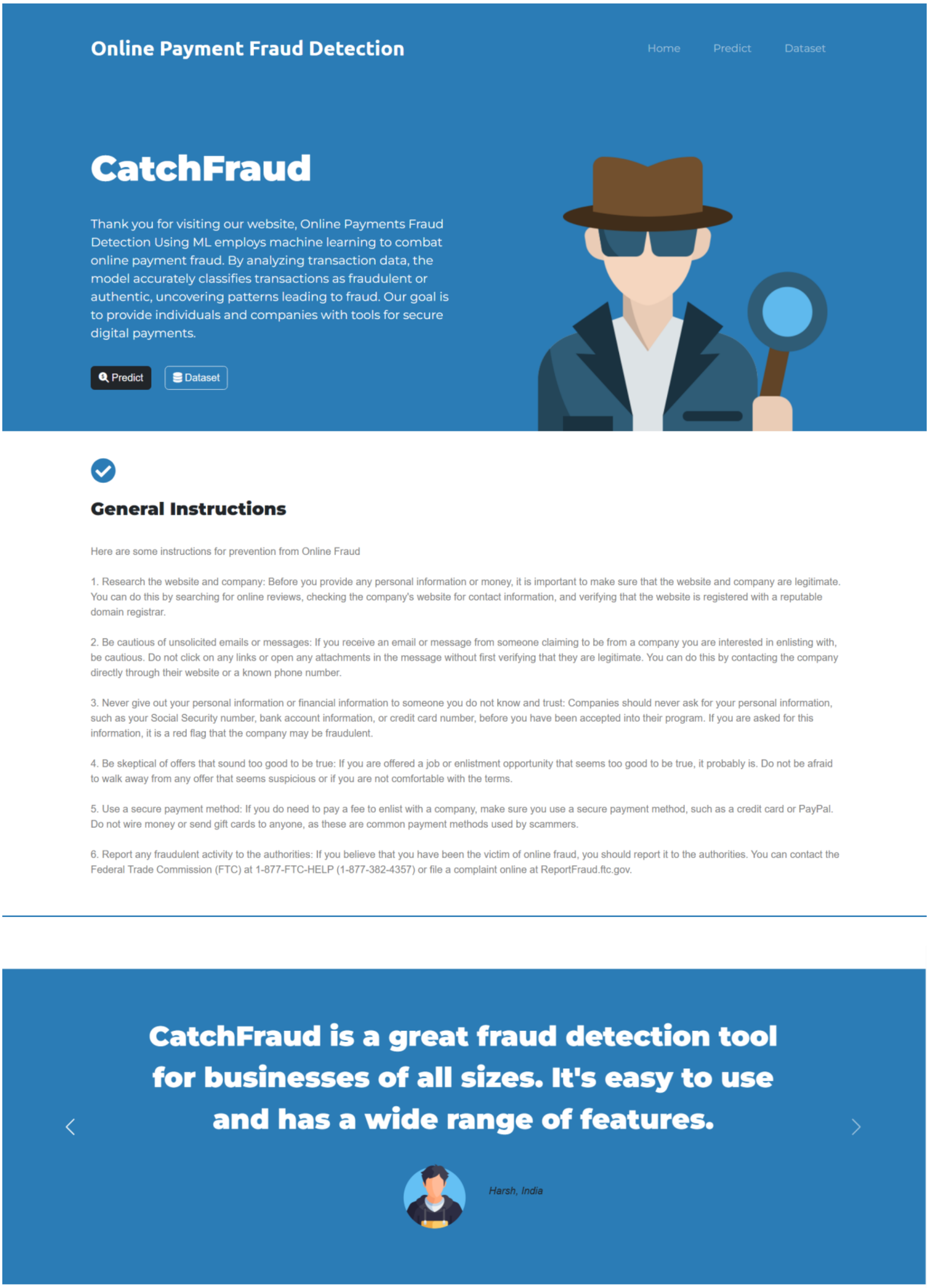
● submit.html

and save them in the templates folder.

As shown in the project structure

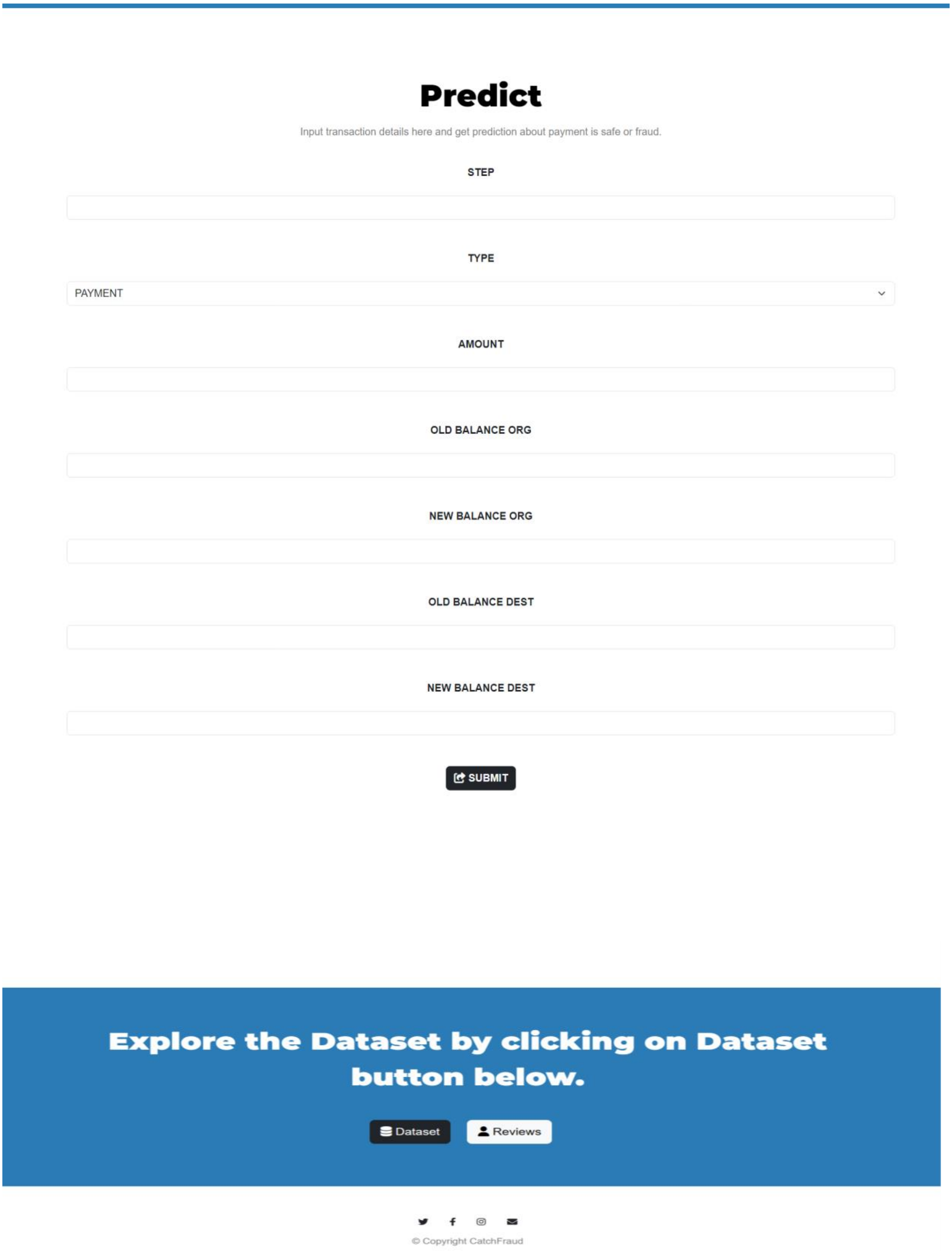


Let’s see how our home.html page looks like:



Now when you click on predict button 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

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,render_template,request
import numpy as np
import pickle
import pandas as pd


model=pickle.load(open("C:\ML MODEL DEPLOY\model.pkl",'rb'))
app=Flask(__name__)
```

Render HTML page:

```
dict_val= {'PAYMENT':0, 'TRANSFER':1 ,'CASH_OUT':2 ,'DEBIT':3 ,'CASH_IN':4}
@app.route("/")
def start():
    return render_template('index.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:

```
def predict():
    return render_template('index.html')

@app.route("/Home")
def index():
    return render_template('index.html')

@app.route("/login",methods=['POST'])
def login():
    x=[[x for x in request.form.values()]]
    x=np.array(x)
    print(x)
    # pred=model.predict(x)
    x[0][1]=dict_val[x[0][1]]
    print(x)
    x=x.astype(float)
    output=model.predict(x)
    val=""
    if(output[0]==0):
        val="Not Fraud"
    else:
        val="Fraud"
    return render_template('submit.html',y=val)
```

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.

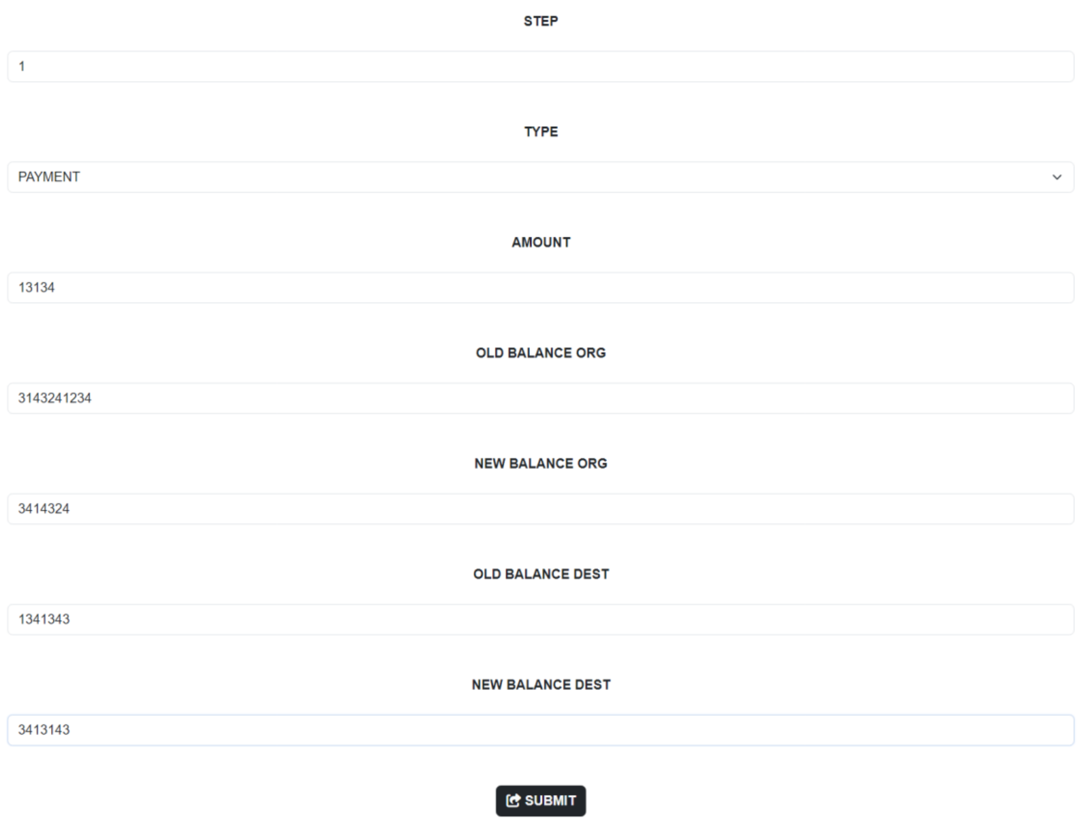
● Click on the predict button from the top right corner, enter the inputs,

click on the submit button, and see the result/prediction on the web.

```
PS C:\ML MODEL DEPLOY> python -u "c:\ML MODEL DEPLOY\app.py"
 * 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 stat
 * Debugger is active!
 * Debugger PIN: 749-454-911
127.0.0.1 - - [09/Nov/2023 12:39:45] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Nov/2023 12:39:45] "GET /static/styles/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [09/Nov/2023 12:39:45] "GET /static/images/boy.png HTTP/1.1" 304 -
127.0.0.1 - - [09/Nov/2023 12:39:45] "GET /static/images/detective.png HTTP/1.1" 304 -
```

Output screenshots:

When transaction is fraud



Online Payment Fraud Detection

Home Dataset About

CatchFraud

Your transaction is Fraud

If you want to predict again then click on the Home button below

Home Dataset

When transaction is not fraud

STEP

1

TYPE

PAYMENT

AMOUNT

1312

OLD BALANCE ORG

132

NEW BALANCE ORG

14334

OLD BALANCE DEST

0

NEW BALANCE DEST

0

SUBMIT


Online Payment Fraud Detection
Home
Dataset
About
CatchFraud
Your transaction is not Fraud
If you want to predict again then click on the Home button below
Home
Dataset