Online Payments Fraud Detection using ML

Introduction:-

The rapid growth of internet usage and the widespread adoption of e-commerce platforms have ushered in a corresponding surge in online credit and debit card transactions. However, this increase in digital transactions has also brought about a rise in fraudulent activities. As a response to this evolving landscape, the proposed project focuses on the development of an advanced credit/debit card fraud detection system. This system aims to address the limitations of existing approaches by leveraging a combination of classification algorithms, including Decision Trees, Random Forest, SVM, Extra Tree Classifier, and XGBoost Classifier.

The key motivation behind this initiative is the recognition that while various methods for fraud detection exist, they often fall short in terms of accuracy and may be accompanied by specific drawbacks. The proposed solution seeks to overcome these challenges by harnessing the strengths of diverse classification algorithms, each offering unique perspectives on transaction data.

To execute this project, a comprehensive dataset will be compiled, capturing a broad spectrum of credit/debit card transactions. This dataset will serve as the foundation for training and testing the selected classification algorithms.

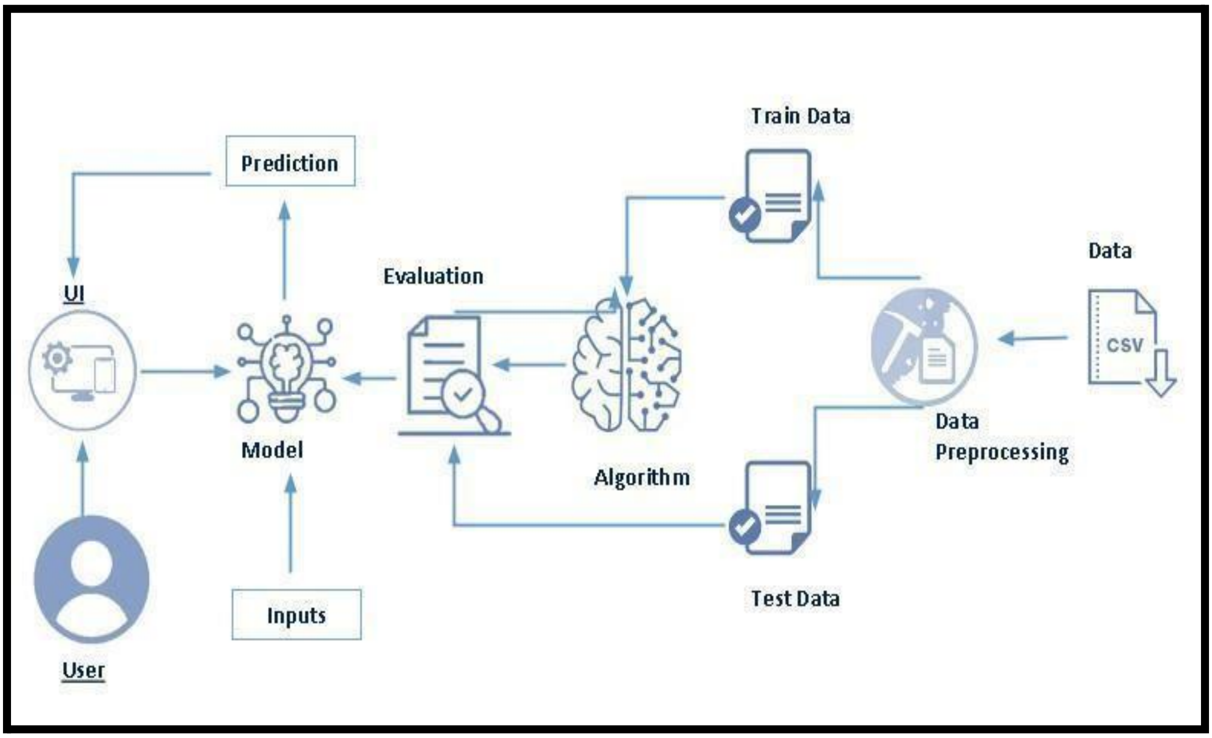
The aim is to identify the most effective model in accurately distinguishing between legitimate and fraudulent transactions.

The selected models, including Decision Trees, Random Forest, SVM, Extra Tree Classifier, and XGBoost Classifier, will undergo rigorous training and testing phases. The performance of each algorithm will be assessed using established metrics to determine the most reliable and efficient model for fraud detection.

Upon identifying the optimal model, it will be saved in a pkl format for future use. The integration of Flask will enable seamless deployment of the selected model, facilitating real-time fraud detection in online credit and debit card transactions

The overarching goal of this project is to contribute to the enhancement of security measures in the realm of online transactions, providing a robust and adaptable fraud detection system. The subsequent sections will delve into the detailed methodology, model selection criteria, and the steps involved in Flask integration.

Technical Architecture:



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,

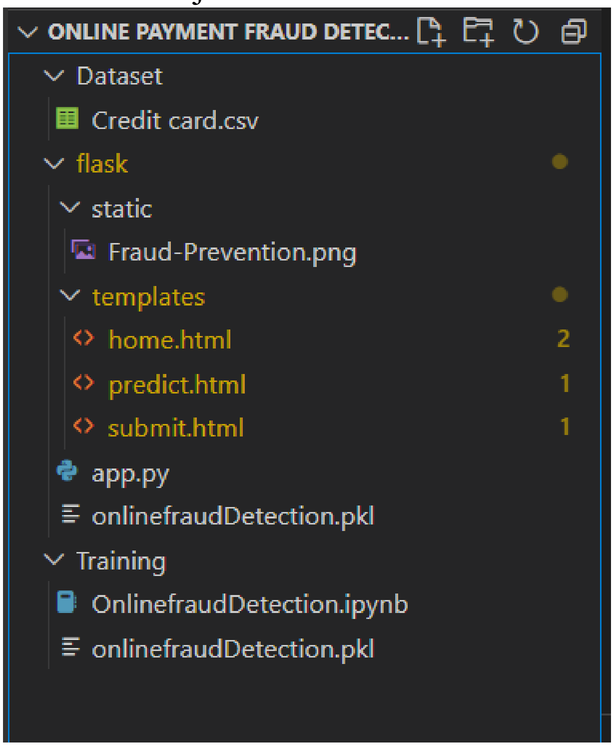
- Data collection
  - Collect the dataset or create the dataset
- Visualising and analysing data
  - Importing the libraries
  - Read the Dataset
  - Univariate analysis
  - Bivariate analysis
  - Descriptive analysis
- Data pre-processing
  - Checking for null values
  - Handling outlier

- Handling categorical(object) data
- Splitting data into train and test

- Model building
  - Import the model building libraries
  - Initialising 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 the training_ibm folder contains IBM deployment files.

Data Collection

Acquiring the Dataset for Machine Learning:

The foundational element of any machine learning endeavor is the dataset. It serves as the bedrock for training algorithms and deriving meaningful insights. To fulfill this requirement, various sources of data can be explored, with popular options including platforms such as Kaggle.com, the UCI repository, and more.

In the context of this project, the dataset utilized is named "PS_20174392719_1491204439457_logs.csv." This dataset was obtained from Kaggle.com, a renowned hub for open datasets and machine learning resources.

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

Visualising and analysing data

```python
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
```

Read the Dataset

In pandas we have a function called read_csv() to read the dataset.

```python
#Reading the csv file
df=pd.read_csv(r'Credit card.csv')
```

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

```
df.columns
```

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

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

Below, the dataset's first five and last five values are loaded using the head and tail method.

```
df.head()
```

Python

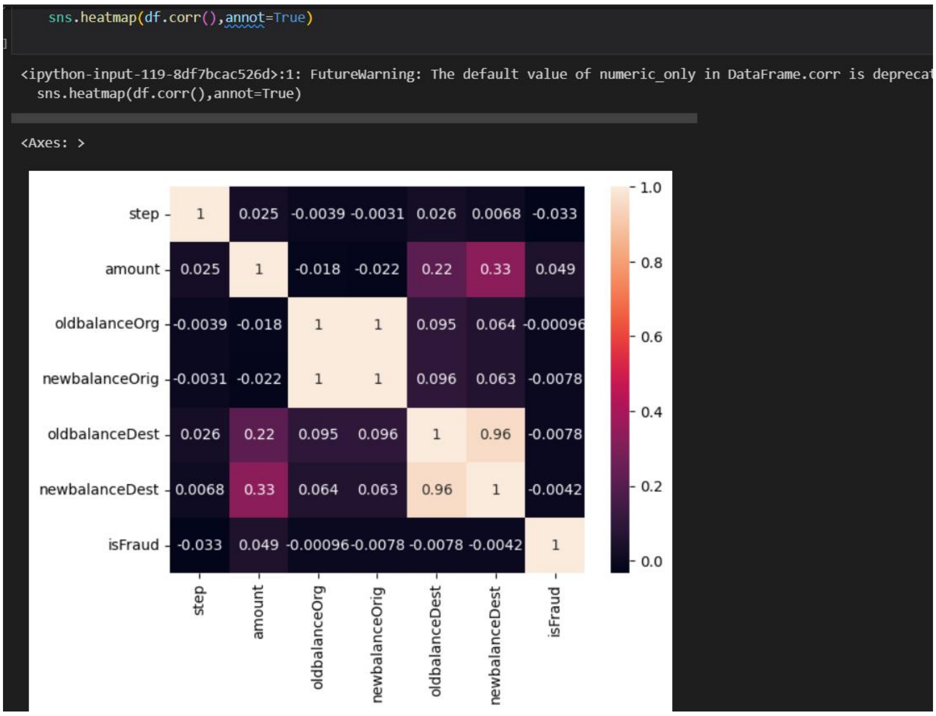
	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0.0	0.0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0.0	0.0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1.0	0.0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1.0	0.0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0.0	0.0

```
df.tail()
```

Python

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
260504	14	CASH_OUT	178684.97	C8826189	21907.00	0.0	C1656139345	2842641.84	3116495.42	0.0	0.0
260505	14	CASH_OUT	118586.47	C185704945	35509.00	0.0	C726864847	39.00	0.00	0.0	0.0
260506	14	CASH_OUT	308503.35	C1324237571	123433.72	0.0	C500437472	97892.36	458476.08	0.0	0.0
260507	14	CASH_OUT	310920.52	C1355560664	0.00	0.0	C625901069	1499435.86	1876515.71	0.0	0.0
260508	14	CASH_OUT	56206.95	C1695876425	0.00	0.0	NaN	NaN	NaN	NaN	NaN

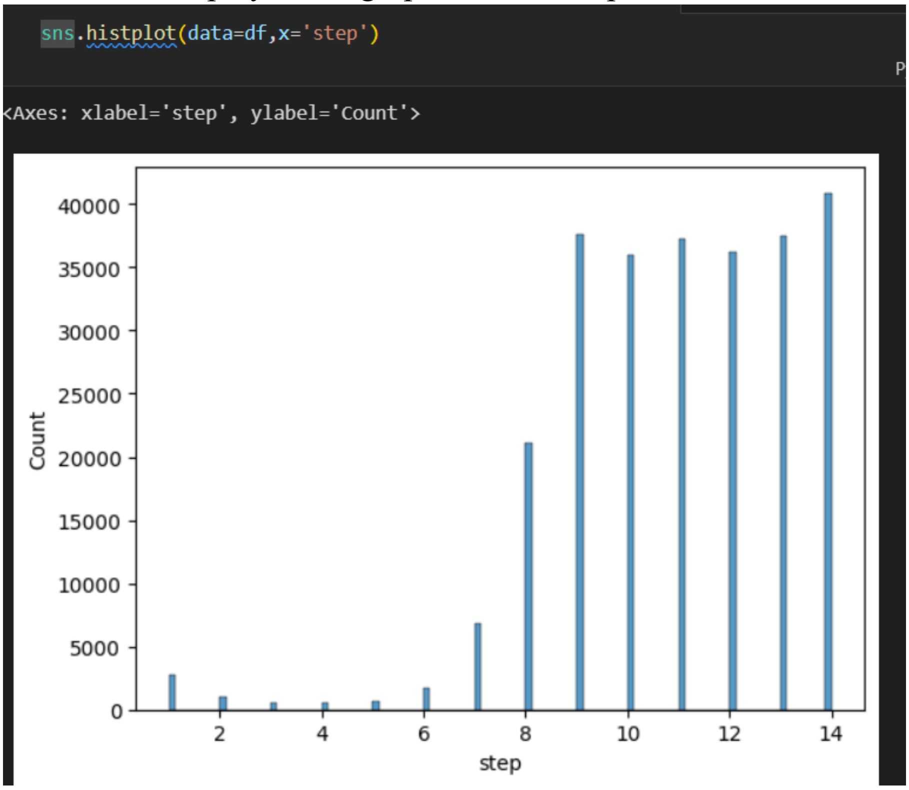
HeatMap:-



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

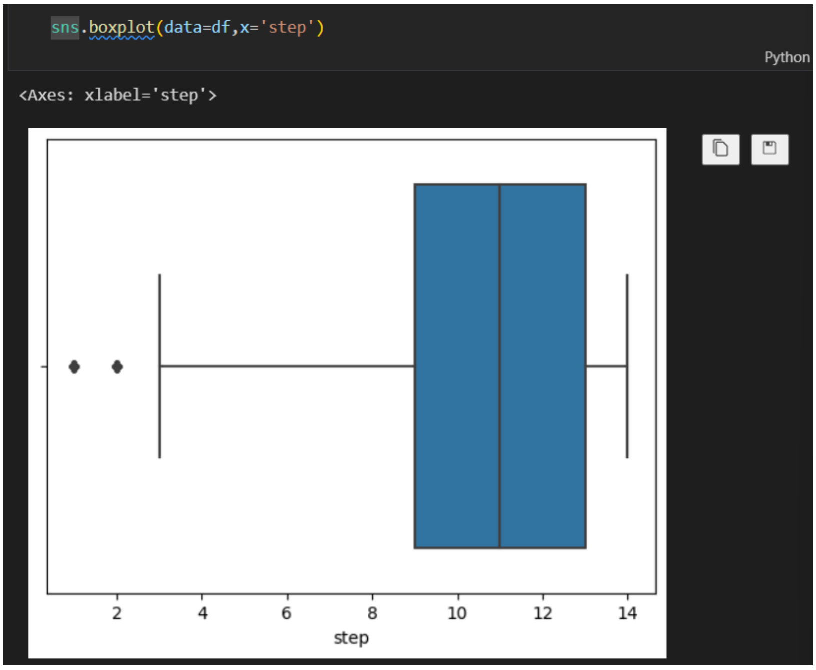
Hist Plot:-

Here I have displayed the graph such as histplot .



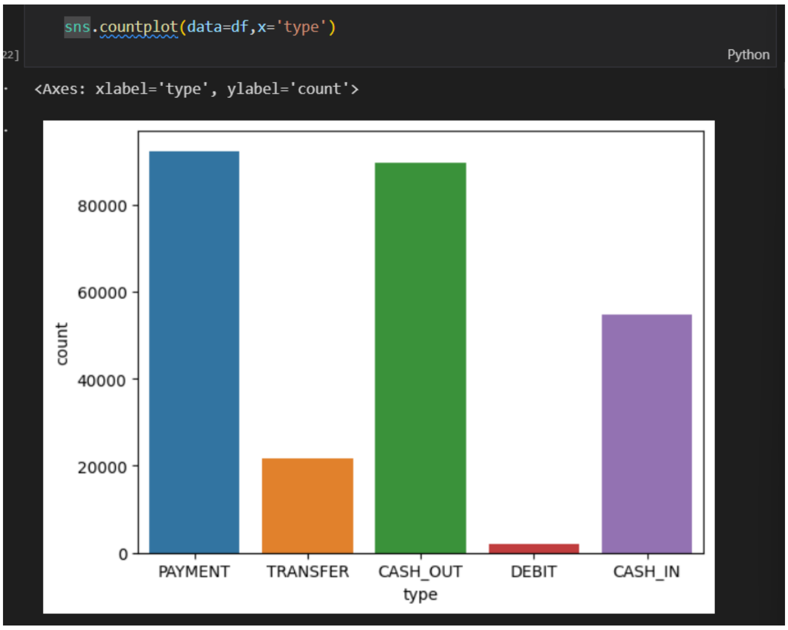
BoxPlot:-

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



Count Plot:-

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

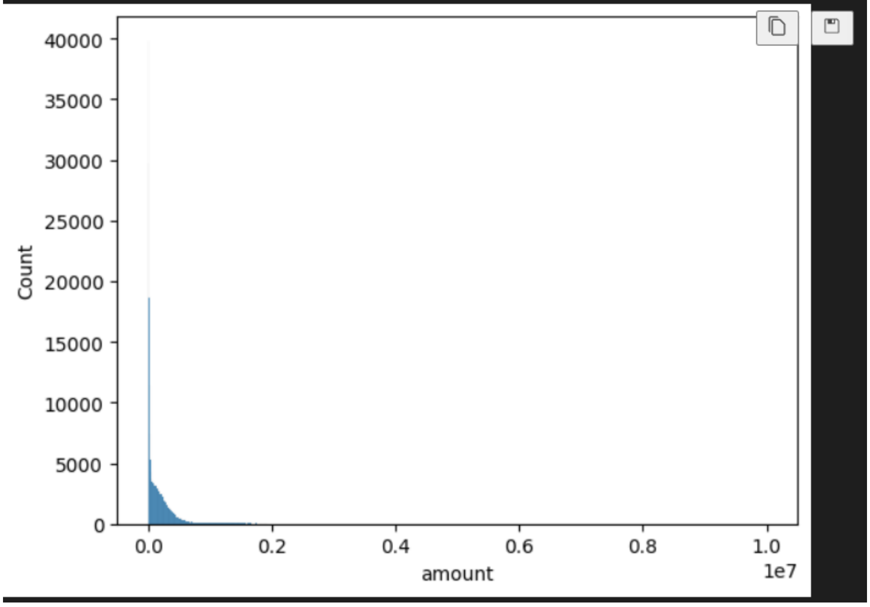


HistPlot:-

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

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



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

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

```
0.0    260341
1.0       167
Name: isFraud, dtype: int64
```

Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process.

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

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
count	260509.000000	260509	2.605090e+05	260509	2.605090e+05	2.605090e+05	260508	2.605080e+05	2.605080e+05	260508.000000
unique	NaN	5	NaN	260501	NaN	NaN	119861	NaN	NaN	NaN
top	NaN	PAYMENT	NaN	C260230637	NaN	NaN	C985934102	NaN	NaN	NaN
freq	NaN	92331	NaN	2	NaN	NaN	85	NaN	NaN	NaN
mean	10.904337	NaN	1.784775e+05	NaN	8.841306e+05	9.027391e+05	NaN	9.611855e+05	1.194126e+06	0.000641
std	2.414674	NaN	3.123849e+05	NaN	2.820156e+06	2.857684e+06	NaN	2.366205e+06	2.612450e+06	0.025311
min	1.000000	NaN	3.000000e-01	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	0.000000
25%	9.000000	NaN	1.246374e+04	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	0.000000
50%	11.000000	NaN	7.483560e+04	NaN	1.878000e+04	0.000000e+00	NaN	6.998200e+04	1.682056e+05	0.000000
75%	13.000000	NaN	2.316096e+05	NaN	1.846340e+05	2.230885e+05	NaN	8.215832e+05	1.225196e+06	0.000000
max	14.000000	NaN	1.000000e+07	NaN	3.893942e+07	3.894623e+07	NaN	4.133844e+07	4.138365e+07	1.000000

Data Pre-processing

Now that we have acquired the dataset, it's imperative to pre-process the data to ensure its suitability for training machine learning models. The downloaded dataset may exhibit randomness and imperfections that could impede the efficacy of the model. The pre-processing activities encompass several crucial steps

- Handling Missing values
- Handling Object Data Label Encoding
- Splitting Dataset into Training and Test Set

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

Data Preprocessing

```
[126]
df.shape
```

```
(260509, 10)
```

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

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

```
df.columns
```

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

```
df.head()
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	170136.0	160296.36	0.0	0.0	0.0
1	1	PAYMENT	1864.28	21249.0	19384.72	0.0	0.0	0.0
2	1	TRANSFER	181.00	181.0	0.00	0.0	0.0	1.0
3	1	CASH_OUT	181.00	181.0	0.00	21182.0	0.0	1.0
4	1	PAYMENT	11668.14	41554.0	29885.86	0.0	0.0	0.0

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.

```
#checking for null values
df.isnull().sum()
```

```
step              0
type              0
amount            0
oldbalanceOrg     0
newbalanceOrig    0
oldbalanceDest    1
newbalanceDest    1
isFraud           1
dtype: int64
```

Handling the Null values:- [Imputing the null values using mean]

#Handling null values

```
df['newbalanceDest'].fillna(df['newbalanceDest'].mean(), inplace=True)
```

Python

```
df['oldbalanceOrg'].fillna(df['oldbalanceOrg'].mean(), inplace=True)
```

Python

```
df['newbalanceOrig'].fillna(df['newbalanceOrig'].mean(), inplace=True)
```

Python

```
df['oldbalanceDest'].fillna(df['oldbalanceDest'].mean(), inplace=True)
```

Python

```
df['isFraud'].fillna(df['isFraud'].mean(), inplace=True)
```

Python

Again Checking the null values:-

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

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

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260509 entries, 0 to 260508
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   step            260509 non-null  int64
 1   type            260509 non-null  object
 2   amount          260509 non-null  float64
 3   oldbalanceOrg   260509 non-null  float64
 4   newbalanceOrig  260509 non-null  float64
 5   oldbalanceDest  260509 non-null  float64
 6   newbalanceDest  260509 non-null  float64
 7   isFraud         260509 non-null  float64
dtypes: float64(6), int64(1), object(1)
memory usage: 15.9+ MB
```

Handling outliers

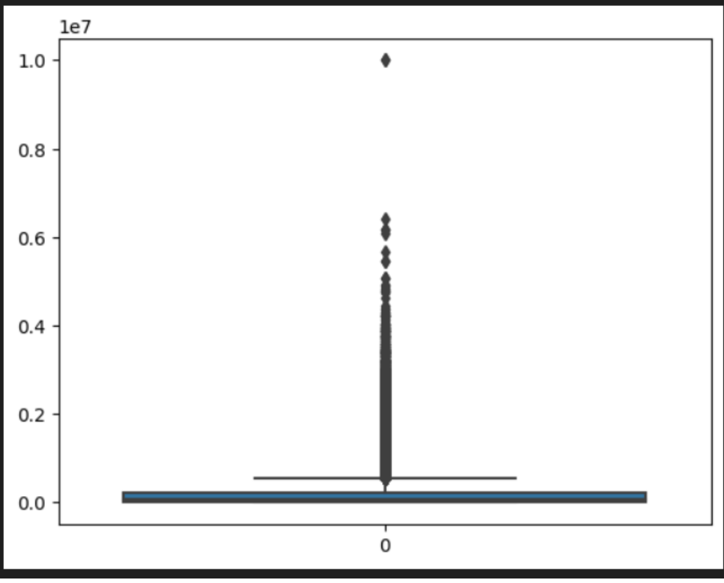
Here, a boxplot is used to identify outliers in the dataset's amount attribute.

Handling outliers

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

[138] Python

```
<Axes: >
```



Removing the outliers:-

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

[139]

```
ModeResult(mode=706.25, count=4)
178477.54881048258
```

```python
q1=np.quantile(df['amount'],0.25)
q3=np.quantile(df['amount'],0.75)
```

[140]

```python
IQR =q3-q1
upper_bound =q3+(1.5*IQR)
lower_bound =q1-(1.5*IQR)
print('q1',q1)
print('q3:',q3)
print("IQR:",IQR)
print("Upper Bound:",upper_bound)
print("Lower Bound:'", lower_bound)
print('Skewed data', len(df[df['amount']>upper_bound]))
print('Skewed data:', len(df[df['amount']<lower_bound]))
```

[141]

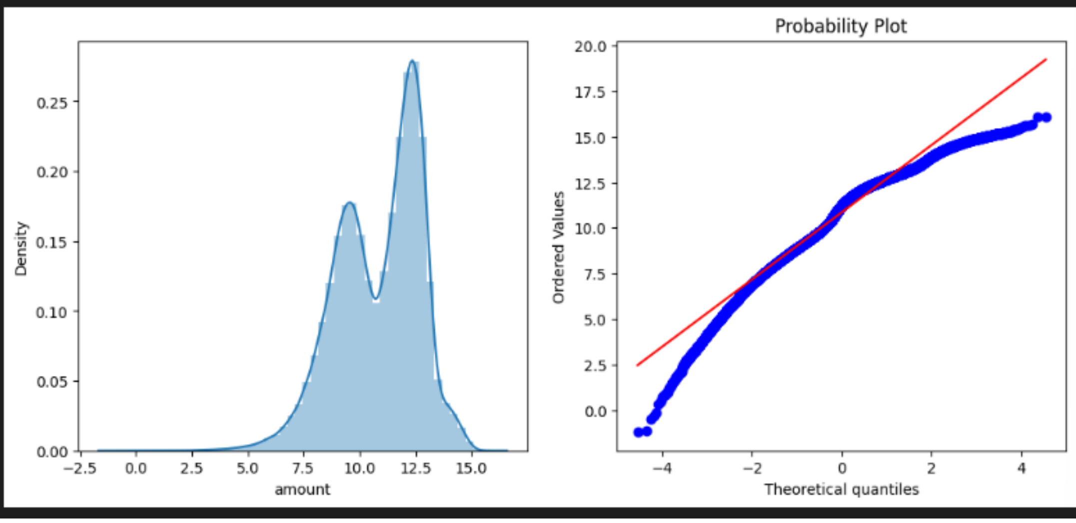
```
q1 12463.74
q3: 231609.62
IQR: 219145.88
Upper Bound: 560328.44
Lower Bound:' -316255.08
Skewed data 14703
Skewed data: 0
```

```python
#to handle the null values we use transformation techniques
def transformationPlot(feature):
  plt.figure(figsize=(12,5))
  plt.subplot(1,2,1)
  sns.distplot(feature)
  plt.subplot(1,2,2)
  stats.probplot(feature, plot=plt)
```
Python

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

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

```python
sns.distplot(feature)
```



Label Encoding:
using labelencoder to encode the dataset's object type

```python
from sklearn.preprocessing import LabelEncoder
lb =LabelEncoder()
df['type']= lb.fit_transform(df['type'])
```

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

```
3    92331
1    89666
0    54729
4    21782
2     2001
Name: type, dtype: int64
```

Splitting the Dataset:-

```
Add Code Cell
#dividing the dataset
x=df.drop('isFraud',axis=1)
y=df['isFraud']
```

```
x.head()
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	1	3	9839.64	170136.0	160296.36	0.0	0.0
1	1	3	1864.28	21249.0	19384.72	0.0	0.0
2	1	4	181.00	181.0	0.00	0.0	0.0
3	1	1	181.00	181.0	0.00	21182.0	0.0
4	1	3	11668.14	41554.0	29885.86	0.0	0.0

```
y.head()
```

```
0    0.0
1    0.0
2    1.0
3    1.0
4    0.0
Name: isFraud, dtype: float64
```

Splitting data into train and test

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.

```
#splitting the data
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)
```

Python

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

Python

```
(208407, 7)
(52102, 7)
(52102,)
(208407,)
```

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.

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. Test data is predicted. For evaluating the model, a confusion matrix and classification report is done.

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Python

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
from sklearn.metrics import mean_squared_error
y_test_predict1=rfc.predict(x_test)
```

Python

```
y_test = np.array(y_test)
```

Python

```
y_test = np.round(y_test).astype(int)
```

Python

```
print(set(y_test))
print(set(y_test_predict1))
```

```
test_accuracy = accuracy_score(y_test, y_test_predict1)
```
Python

```
test_accuracy
```
Python

```
0.9996545238186634
```

```
y_train_predict1=rfc.predict(x_train)
train_accuracy= accuracy_score(y_train,y_train_predict1)
train_accuracy
```
Python

```
1.0
```

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

col_0	0.0	1.0
row_0		
0	52066	3
1	15	18

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

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     52069
           1       0.86      0.55      0.67        33

    accuracy                           1.00     52102
   macro avg       0.93      0.77      0.83     52102
weighted avg       1.00      1.00      1.00     52102
```

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. Test data is predicted with. For evaluating the model, a confusion matrix and classification report is done.

Decision Tree Classifier

```python
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(x_train, y_train)
y_test_predict2 =dtc.predict(x_test)
test_accuracy= accuracy_score(y_test,y_test_predict2)
test_accuracy
```

[162] Python

```
0.9994817857279951
```

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

[163] Python

```
1.0
```

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

[164] Python

col_0	0.0	1.0
row_0		
0	52066	3
1	15	18

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

[165] Pytho

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     52069
           1       0.86      0.55      0.67        33

    accuracy                           1.00     52102
   macro avg       0.93      0.77      0.83     52102
weighted avg       1.00      1.00      1.00     52102
```

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. Test data is predicted. For evaluating the model, a confusion matrix and classification report is done.

Extra Trees Classifier

```python
from sklearn.tree import ExtraTreeClassifier
etc=DecisionTreeClassifier()
etc.fit(x_train, y_train)

y_test_predict3 =etc.predict(x_test)
test_accuracy= accuracy_score(y_test,y_test_predict3)
test_accuracy
```

[166] Python

```
0.9995201719703658
```

```python
y_train_predict3=etc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict3)
train_accuracy
```

[167] Python

```
1.0
```

```python
pd.crosstab(y_test,y_test_predict3)
```

[168] Python

col_0	0.0	1.0
row_0		
0	52058	11
1	14	19

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

[169] Python

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     52069
           1       0.63      0.58      0.60        33

    accuracy                           1.00     52102
   macro avg       0.82      0.79      0.80     52102
weighted avg       1.00      1.00      1.00     52102
```

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. Test data is predicted and saved in a new variable. For evaluating the model, confusion matrix and classification report is done

Support Vector Machine

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc=SVC()
svc.fit(x_train,y_train)

y_test_predict4 =svc.predict(x_test)
test_accuracy= accuracy_score(y_test,y_test_predict4)
test_accuracy
```

```
0.9993666270008829
```

```python
y_train_predict4=svc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict4)
train_accuracy
```

```
0.9994242036016064
```

```python
pd.crosstab(y_test,y_test_predict4)
```

col_0	0.0
row_0	
0	52069
1	33

```python
print(classification_report(y_test,y_test_predict4))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     52069
           1       0.00      0.00      0.00        33

    accuracy                           1.00     52102
   macro avg       0.50      0.50      0.50     52102
weighted avg       1.00      1.00      1.00     52102
```

preprocessing class of sklearn. LabelEncoder[source] 0 to n classes-1 as the range for the target labels to be encoded.

```
df.columns
[174]                                                        Python

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

from sklearn.preprocessing import LabelEncoder
la=LabelEncoder()
y_train1=la.fit_transform(y_train)
[175]                                                        Python
```

Instead of encoding the input X, the target values, i.e. y, should be encoded using this transformer.

```
y_test1=la.transform(y_test)
]                                                            Python

y_test1=la.transform(y_test)
]                                                            Python

y_test1
]                                                            Python
array([0, 0, 0, ..., 0, 0, 0])

y_train1
]                                                            Python
array([0, 0, 0, ..., 0, 0, 0])
```

xgboost Classifier¶

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

Xgboost Classifier

```python
import xgboost as xgb
xgb1=xgb.XGBClassifier()
xgb1.fit(x_train, y_train1)

y_test_predict5=xgb1.predict(x_test)
test_accuracy=accuracy_score(y_test1,y_test_predict5)
test_accuracy
```

[180] Python

```
0.9996929100610341
```

```python
y_train_predict5=xgb1.predict(x_train)
train_accuracy=accuracy_score(y_train1,y_train_predict5)
train_accuracy
```

[181] Python

```
0.9999904033933601
```

```python
pd.crosstab(y_test1,y_test_predict5)
```

[182] Python

col_0	0	1
row_0		
0	52066	3
1	13	20

```python
print(classification_report(y_test1,y_test_predict5))
```

[183] Python

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     52069
           1       0.87      0.61      0.71        33

    accuracy                           1.00     52102
   macro avg       0.93      0.80      0.86     52102
weighted avg       1.00      1.00      1.00     52102
```

6: Compare the model

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

After calling the function, the results of models are displayed as output. From the five models, the svc is performing well. From the below image, We can see the accuracy of the model is 99.93% accuracy. .

```python
def compareModel():
    print("train accuracy for rfc", accuracy_score(y_train_predict1,y_train))
    print("test accuracy for rfc", accuracy_score(y_test_predict1,y_test))
    print("train accuracy for dtc", accuracy_score(y_train_predict2,y_train))
    print("test accuracy for dtc",accuracy_score(y_test_predict2,y_test))
    print("train accuracy for etc", accuracy_score(y_train_predict3,y_train))
    print("test accuracy for etc", accuracy_score(y_test_predict3,y_test))
    print("train accuracy for svc", accuracy_score(y_train_predict4,y_train))
    print("test accuracy for svce", accuracy_score(y_test_predict4,y_test))
    print("train accuracy for xgb1",accuracy_score(y_train_predict5,y_train1))
    print("test accuracy for xgbi", accuracy_score(y_test_predict5,y_test1))
```

Python

```python
compareModel()
```

Python

```
train accuracy for rfc 1.0
test accuracy for rfc 0.9996545238186634
train accuracy for dtc 1.0
test accuracy for dtc 0.9994817857279951
train accuracy for etc 1.0
test accuracy for etc 0.9995201719703658
train accuracy for svc 0.9994242036016064
test accuracy for svce 0.9993666270008829
train accuracy for xgb1 0.9999904033933601
test accuracy for xgbi 0.9996929100610341
```

7: Evaluating performance of the model and saving the model

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

Evaluating the model

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
SVC= SVC()
svc.fit(x_train,y_train)
y_test_predict4=svc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict4)
test_accuracy
```

Python

```
0.9993666270008829
```

```python
y_train_predict4=svc.predict(x_train)
train_accuracy-accuracy_score(y_train,y_train_predict4)
train_accuracy
```

Python

```
0.9999904033933601
```

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

Python

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

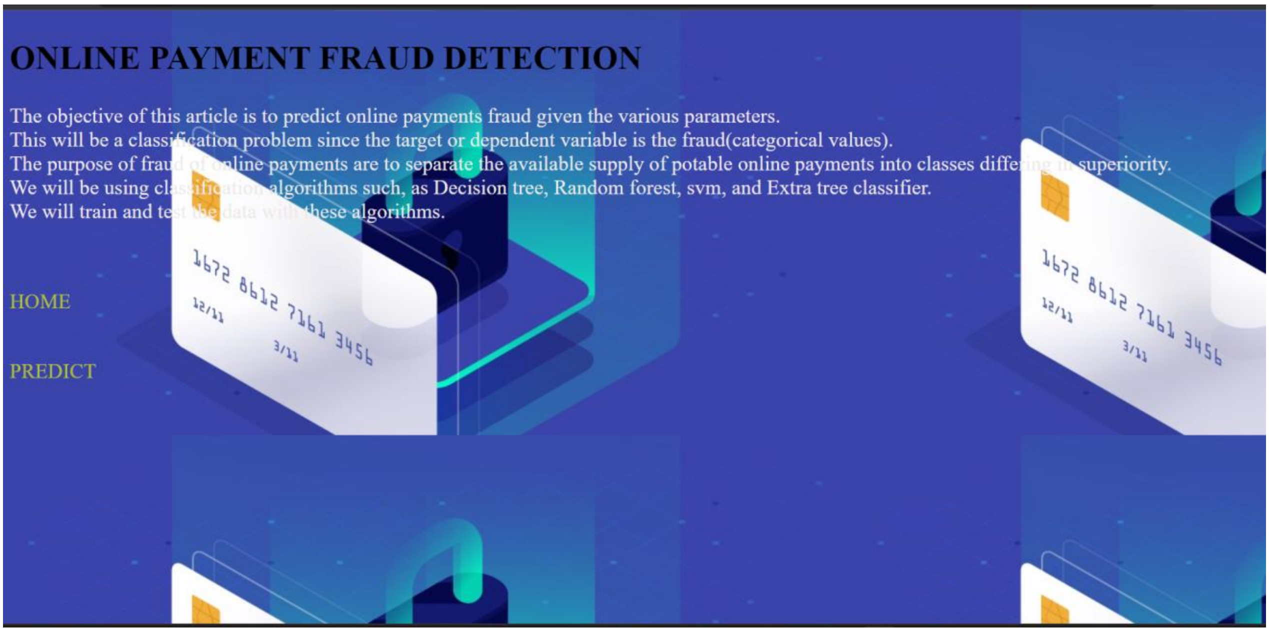
Building server side script

Building Html Pages:

For this project create three HTML files namely

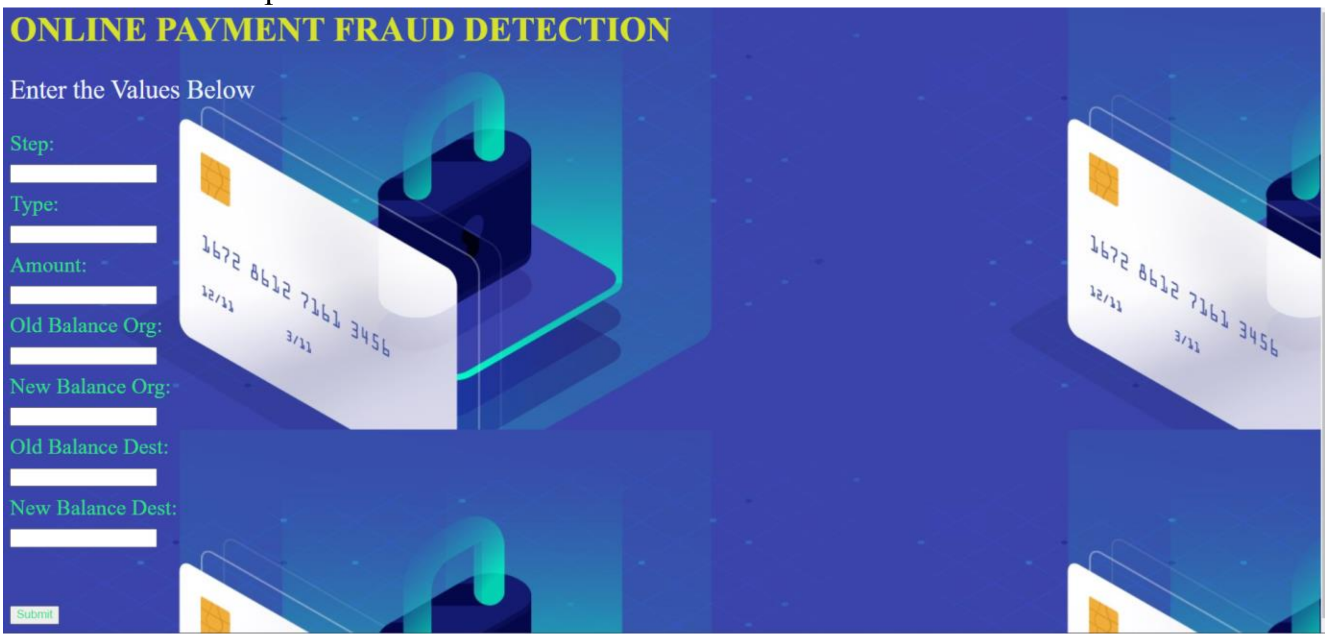
- home.html
- predict.html
- submit.html and save them in the templates folder.

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



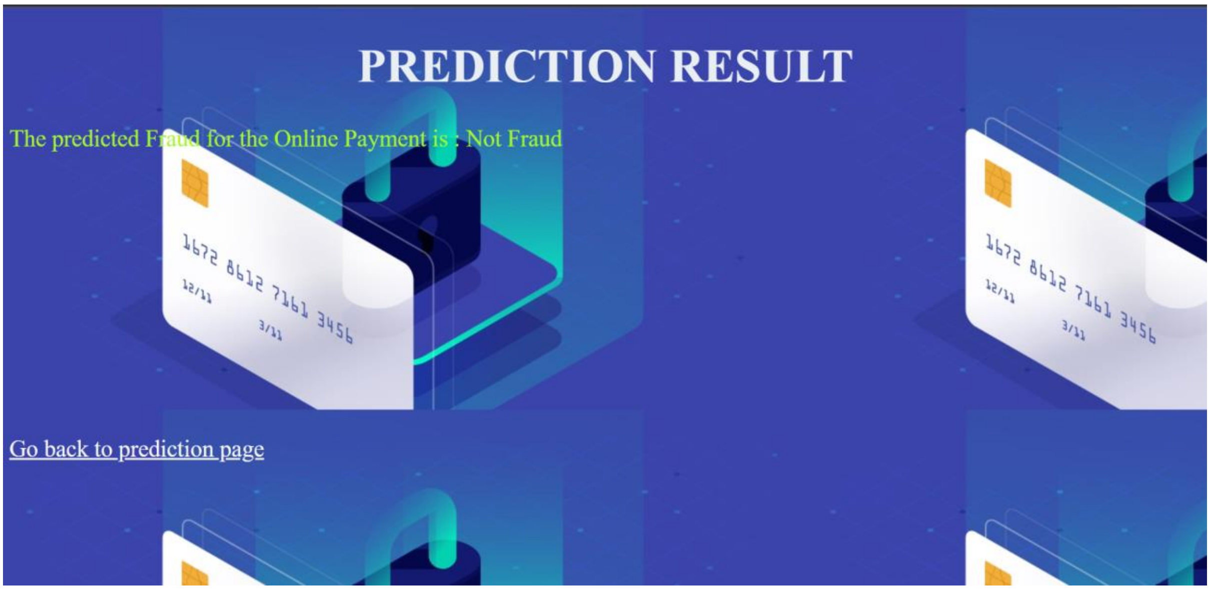
Now when you click on predict button from top right corner you will get redirected to predict.html

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



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

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



Build Python code:

Import the libraries

```
flask > app.py > ...
from flask import Flask, render_template, request
import numpy as np
import pickle
import warnings


# Load your machine learning model
model_path = r"C:\Users\Rohith\Downloads\Online Payment Fraud Detection\flask\onlinefraudDetection.pkl"

```

Load the saved model.

Flask constructor takes the name of the current module (__name__) as argument.

```
#Load the saved model
app = Flask(__name__)
app.config['STATIC_FOLDER'] = 'static'
```

Render HTML page:

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

```python
# Ignore warnings during model loading
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=UserWarning)
    model = pickle.load(open(model_path, 'rb'))

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

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

@app.route("/pred", methods=['POST', 'GET'])
def pred():
    if request.method == 'POST':
        # Your prediction logic here
        x = [[float(x) for x in request.form.values()]]
        print("Input Data:", x)
        x = np.array(x)
        print("Input Shape:", x.shape)

        print("Input Data Array:", x)
        pred = model.predict(x)
        print("Raw Prediction:", pred[0])

        # Map prediction to labels
        label_mapping = {0: "Not Fraud", 1: "Fraud"}
        prediction_text = label_mapping.get(pred[0], "Unknown")

        print("Final Prediction:", prediction_text)
        return render_template('submit.html', prediction_text=prediction_text)
    else:
        return render_template('submit.html', prediction_text=None)
```

Here we are routing our app to predict() function.

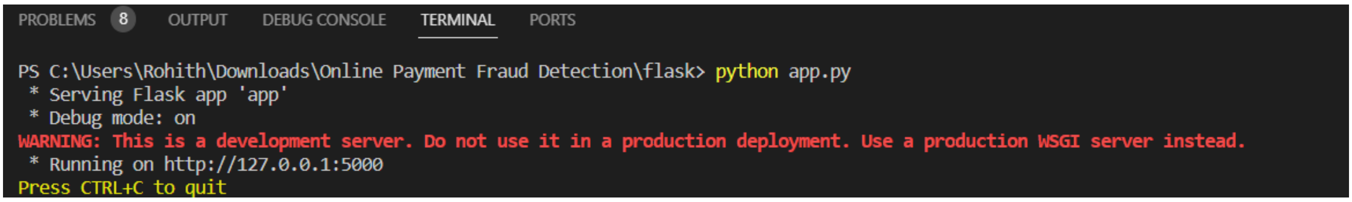
- This function retrieves all the values from the HTML page using Post request.
- That is stored in an array.
- This array is passed to the model.predict() function.
- This function returns the prediction.
- This prediction value will be rendered to the text that we have mentioned in the submit.html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

Main Function:

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

Run the application

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



Output screenshots:


PREDICTION RESULT
The predicted Fraud for the Online Payment is : Not Fraud
Go back to prediction page