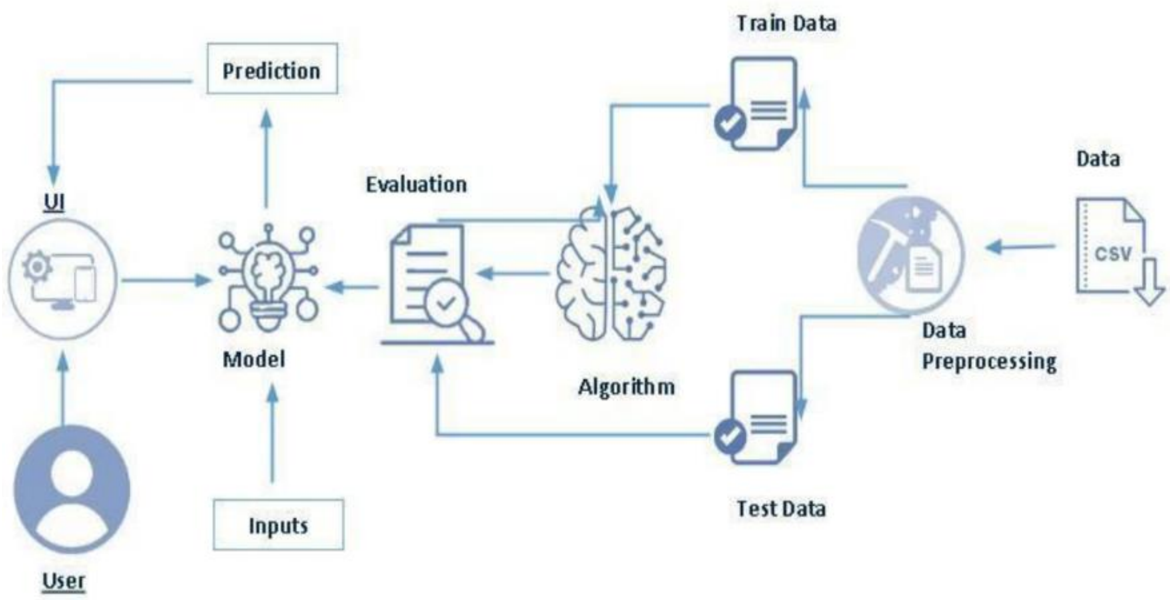
ONLINE PAYMENT FRAUD DETECTION USING ML

In the rapidly evolving landscape of digital transactions, the prevalence of online payment fraud has become a significant concern for individuals, businesses, and financial institutions alike. As the volume of online transactions continues to surge, traditional methods of fraud detection are proving inadequate in identifying and preventing sophisticated fraudulent activities. In response to this growing challenge, the integration of machine learning (ML) algorithms has emerged as a powerful tool in bolstering online payment security. It allows for the identification of complex patterns and anomalies in large datasets, enabling organizations to detect and prevent fraudulent transactions in real-time while adapting to evolving tactics employed by fraudsters.

Classification techniques like Decision Tree, Random Forest, and Extra Tree Classifier will be employed. We will use these methods to train and test the data. The optimal model is chosen from this and saved in PKL format.

Technical Architecture:

Let us look at the Technical Architecture of the project



Project flow:

- Customer is shown the Home page. The customer will browse through Home page and click on the Predict button.
- After clicking the Predict button the customer will be directed to the Predict page where the customer will input the details they have and click on the Predict button.
- Customer will be redirected to the Submit page. The model will analyse the inputs given by the customer and showcase the prediction of the payment.

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

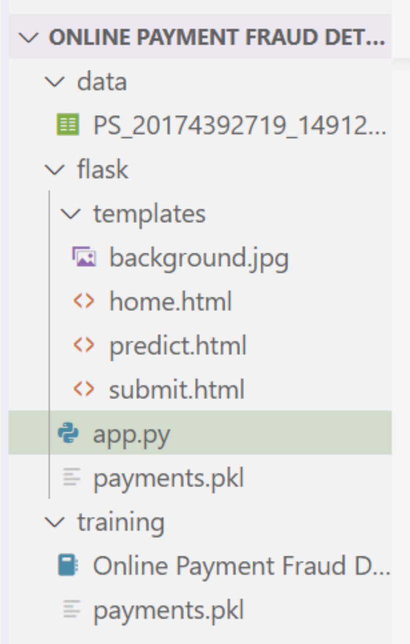
- Import the model building libraries
- Initialising the model
- Training and testing the model
- Evaluating performance of model
- Save the model

Application Building

- Create an HTML file
- Build python code

Project Structure:

Project folder which contains files as shown below:



- The data obtained is in two csv files, one for training and another for testing.
- App.py file is used for routing purposes using scripting.
- Packets.pkl is the saved model.

Milestone 1: Data Collection

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

Milestone 2: Visualising and analysing data

Activity 1: Importing the 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

```
df = pd.read_csv(r"C:\Users\vudda\OneDrive\Desktop\PS_20174392719_1491204439457_log.csv")
```

```
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0	0
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

Here, the input features in the dataset are known using the df.columns function and the dataset's superfluous columns are being removed using the drop method.

```
df.columns
```

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

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

```
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0
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:

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

```
df.head()
```

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

```
df.tail()
```

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

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

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

```
df.corr()
```

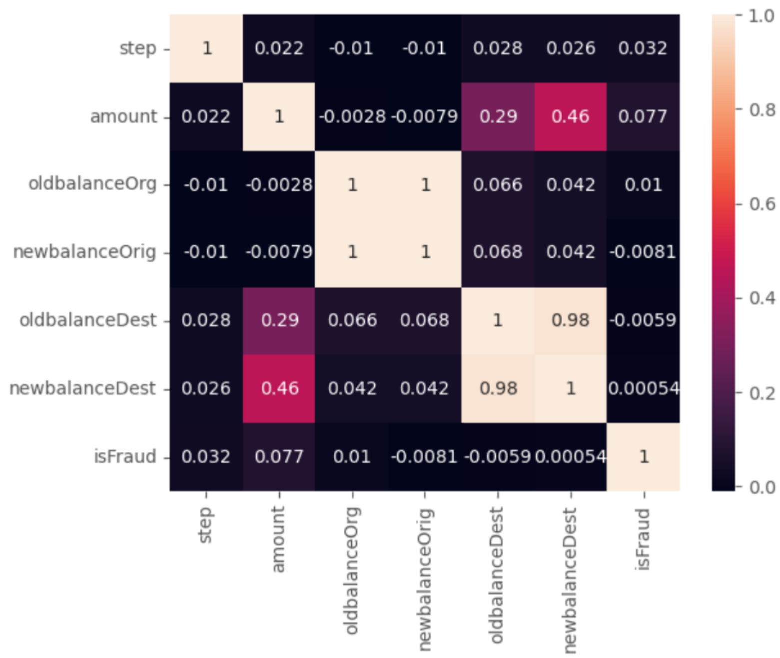
	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
step	1.000000	0.022373	-0.010058	-0.010299	0.027665	0.025888	0.031578
amount	0.022373	1.000000	-0.002762	-0.007861	0.294137	0.459304	0.076688
oldbalanceOrg	-0.010058	-0.002762	1.000000	0.998803	0.066243	0.042029	0.010154
newbalanceOrig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

Heatmap:

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

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

<Axes: >



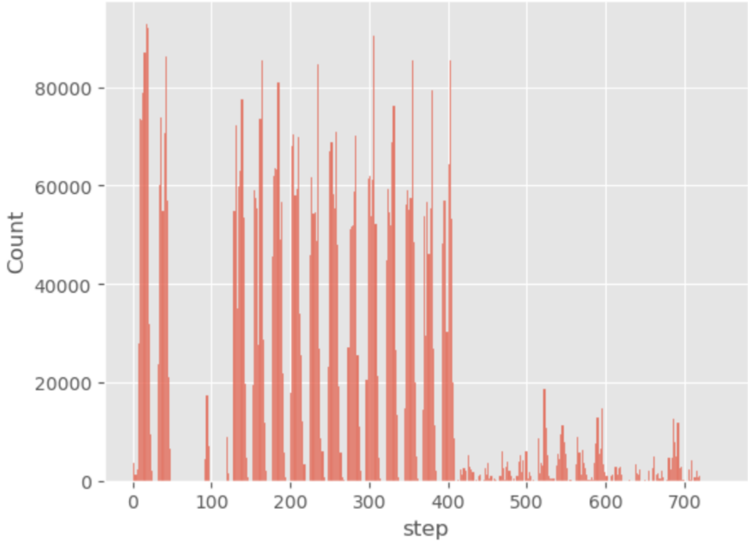
Activity 3: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature.

Here we have displayed the graph such as histplot .

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

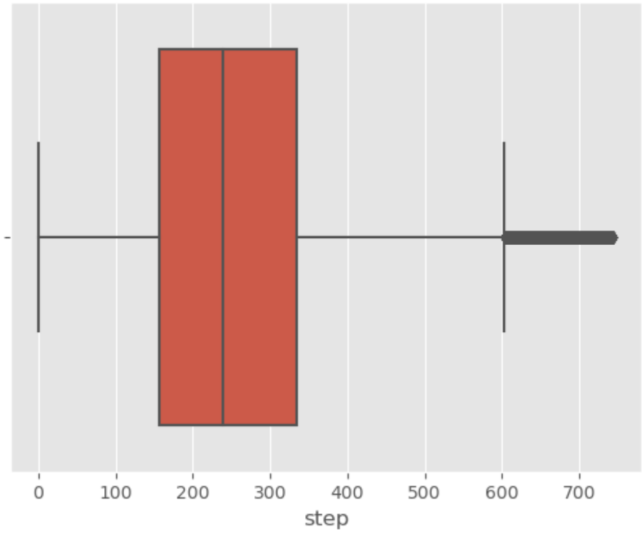
```
<Axes: xlabel='step', ylabel='Count'>
```



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

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

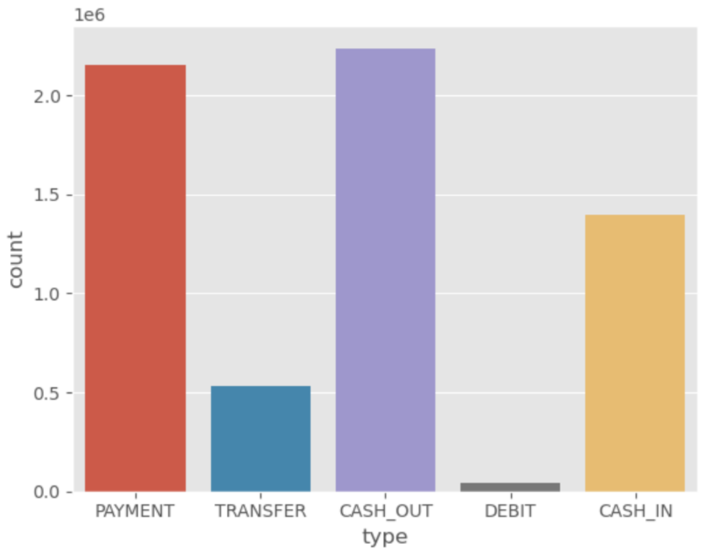
```
<Axes: xlabel='step'>
```



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

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

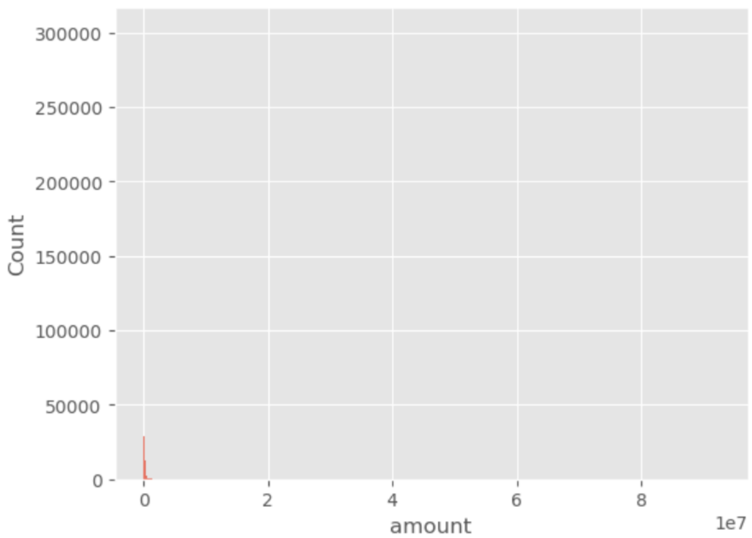
```
<Axes: xlabel='type', ylabel='count'>
```



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

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

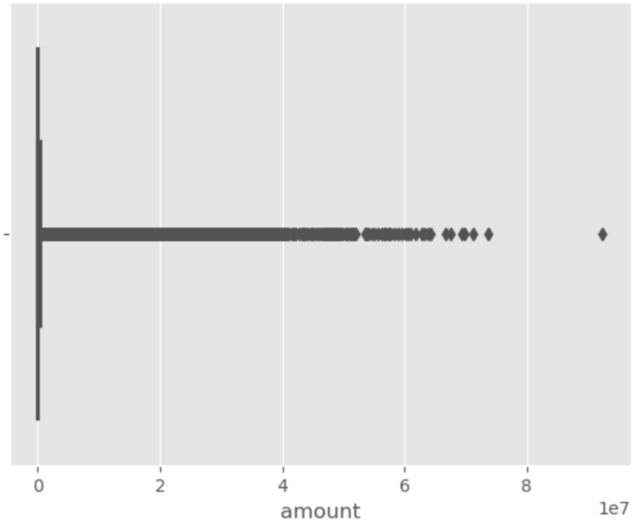
```
<Axes: xlabel='amount', ylabel='Count'>
```



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

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

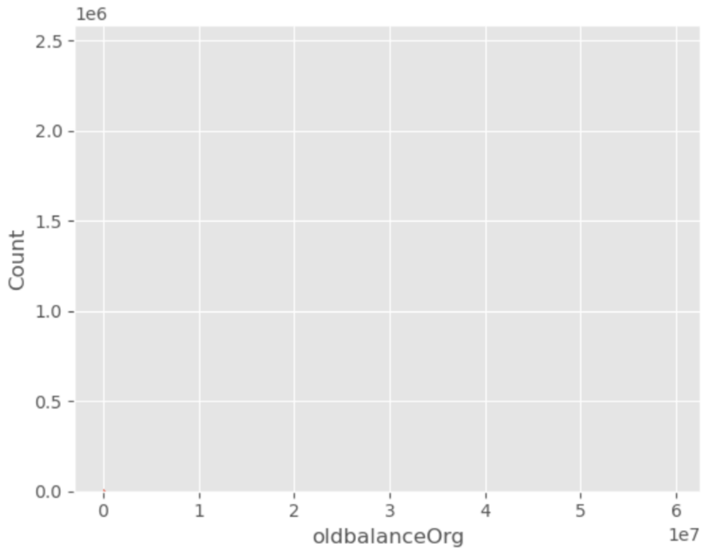
<Axes: xlabel='amount'>



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

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

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



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

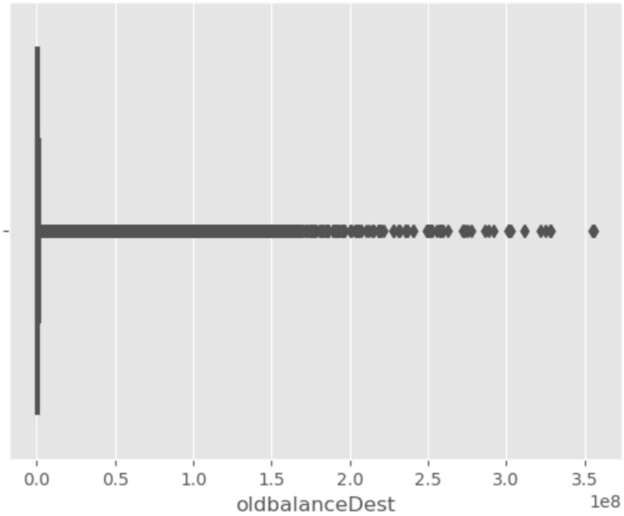
```
df['nameDest'].value_counts()
```

```
C1286084959    113
C985934102     109
C665576141     105
C2083562754    102
C248609774     101
              ...
M1470027725      1
M1330329251      1
M1784358659      1
M2081431099      1
C2080388513      1
Name: nameDest, 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')
```

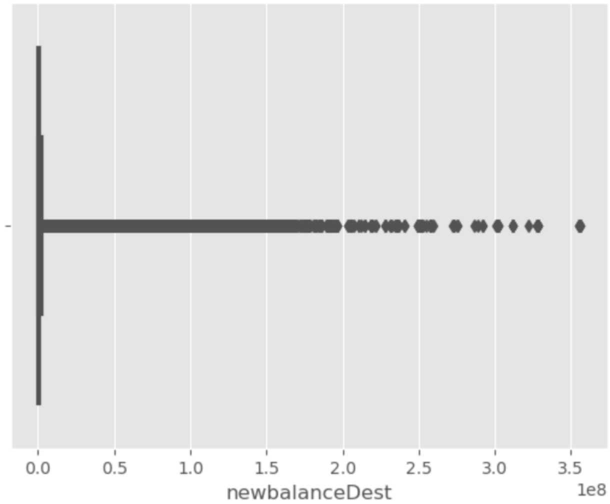
```
<Axes: xlabel='oldbalanceDest'>
```



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

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

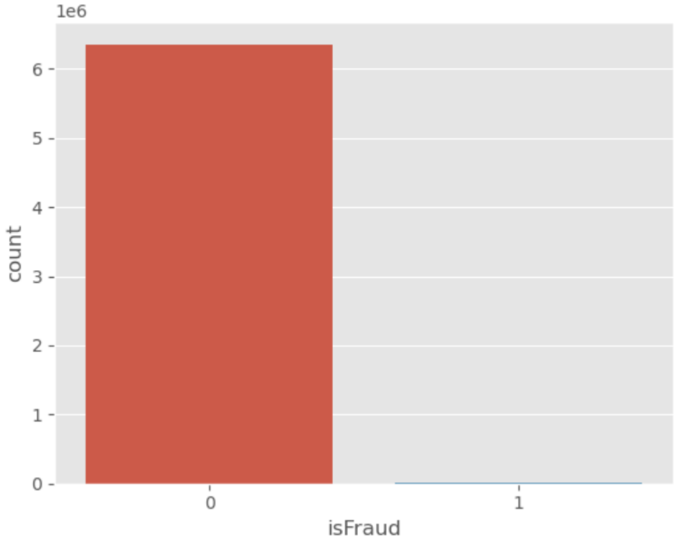
```
<Axes: xlabel='newbalanceDest'>
```



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

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

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



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

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

```
0    6354407
1       8213
Name: isFraud, dtype: int64
```

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

```
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	is not Fraud
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	is not Fraud
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	is Fraud
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	is Fraud
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	is not Fraud
...	...	...	...	...	...	...	...	...	...	...
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C776919290	0.00	339682.13	is Fraud
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C1881841831	0.00	0.00	is Fraud
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C1365125890	68488.84	6379898.11	is Fraud
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C2080388513	0.00	0.00	is Fraud
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C873221189	6510099.11	7360101.63	is Fraud

6362620 rows × 10 columns

Here, we're using the value counts method to figure out how many classes there are in the dataset's target isFraud column.

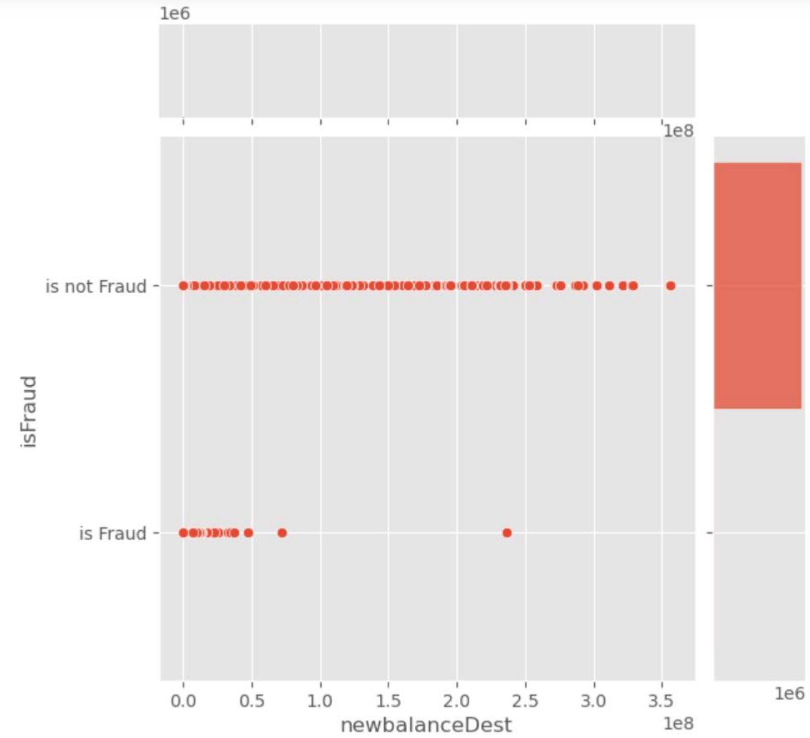
Converting 0-means: is not fraud and 1-means: is fraud using the loc technique here

Activity 4: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we are visualising the relationship between newbalanceDest and isFraud. Jointplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.jointplot(data=df,x='newbalanceDest',y='isFraud')
```

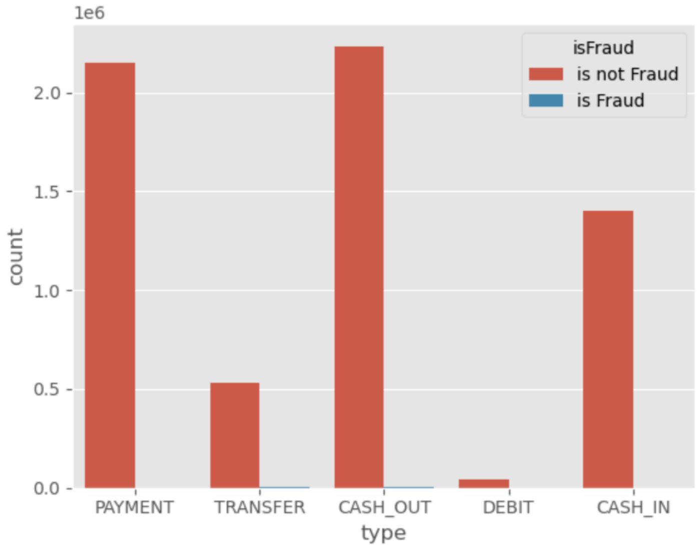
```
<seaborn.axisgrid.JointGrid at 0x1be1546d450>
```



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')
```

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



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

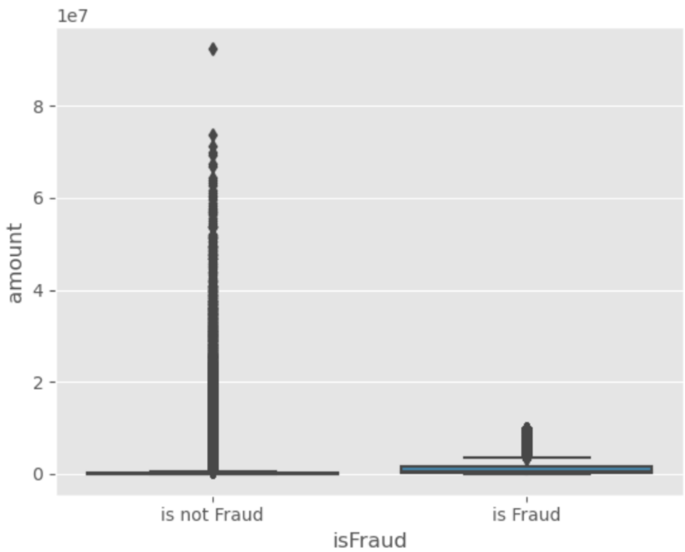
```
sns.boxplot(data=df,x='isFraud',y='step')
```

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

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

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

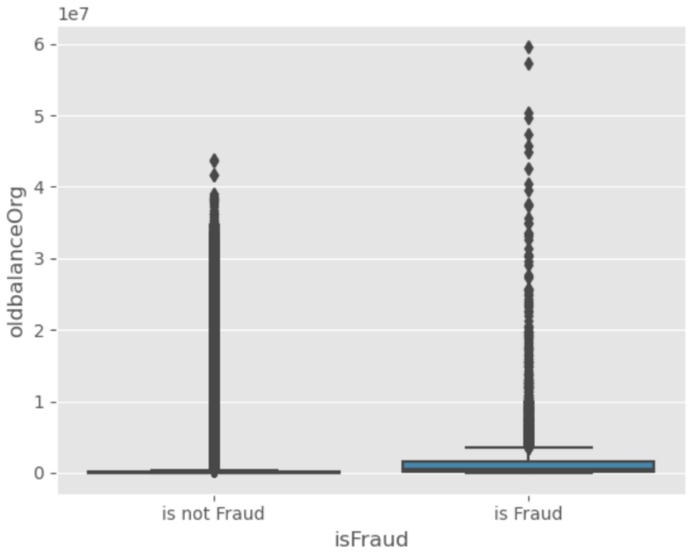
```
<Axes: xlabel='isFraud', ylabel='amount'>
```



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

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

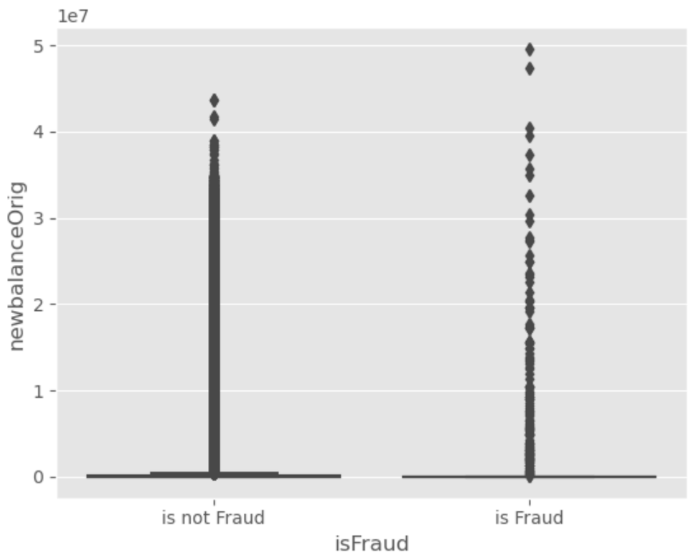
```
<Axes: xlabel='isFraud', ylabel='oldbalanceOrg'>
```



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

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

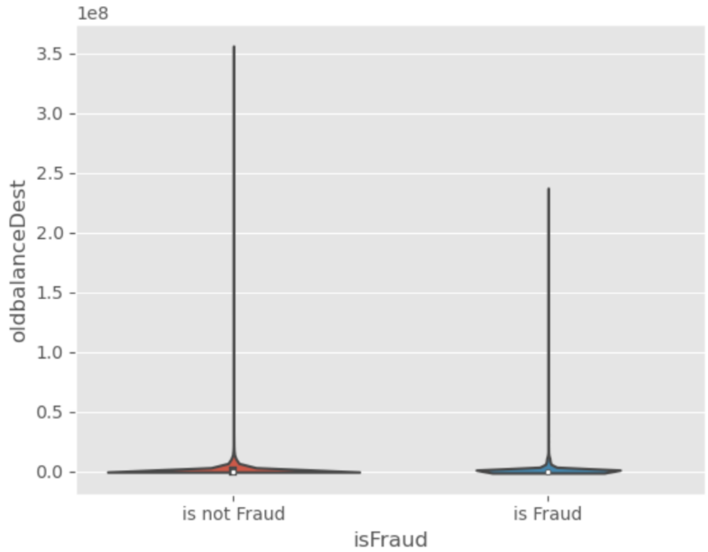
```
<Axes: xlabel='isFraud', ylabel='newbalanceOrig'>
```



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

```
sns.violinplot(data=df,x='isFraud',y='oldbalanceDest')
```

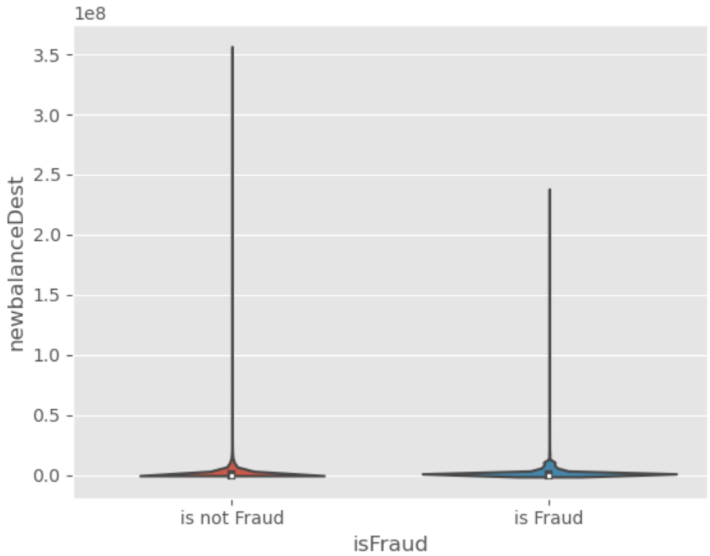
```
<Axes: xlabel='isFraud', ylabel='oldbalanceDest'>
```



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

```
sns.violinplot(data=df,x='isFraud',y='newbalanceDest')
```

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



Activity 5: Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

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

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
count	6.362620e+06	6362620	6.362620e+06	6362620	6.362620e+06	6.362620e+06	6362620	6.362620e+06	6.362620e+06	6362620
unique	NaN	5	NaN	6353307	NaN	NaN	2722362	NaN	NaN	2
top	NaN	CASH_OUT	NaN	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

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

```
df.shape
```

```
(6362620, 10)
```

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

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

```
df.head
```

```
<bound method NDFrame.head of          step      type      amount  oldbalanceOrg  newbalanceOrig  \
0           1   PAYMENT     9839.64      170136.00       160296.36
1           1   PAYMENT     1864.28       21249.00        19384.72
2           1  TRANSFER      181.00         181.00            0.00
3           1  CASH_OUT      181.00         181.00            0.00
4           1   PAYMENT    11668.14       41554.00        29885.86
...       ...       ...         ...            ...             ...
6362615   743  CASH_OUT   339682.13      339682.13            0.00
6362616   743  TRANSFER  6311409.28     6311409.28            0.00
6362617   743  CASH_OUT  6311409.28     6311409.28            0.00
6362618   743  TRANSFER   850002.52      850002.52            0.00
6362619   743  CASH_OUT   850002.52      850002.52            0.00

         oldbalanceDest  newbalanceDest       isFraud
0                  0.00            0.00  is not Fraud
1                  0.00            0.00  is not Fraud
2                  0.00            0.00      is Fraud
3              21182.00            0.00      is Fraud
4                  0.00            0.00  is not Fraud
...                 ...             ...           ...
6362615            0.00       339682.13      is Fraud
6362616            0.00            0.00      is Fraud
6362617        68488.84      6379898.11      is Fraud
6362618            0.00            0.00      is Fraud
6362619      6510099.11      7360101.63      is Fraud

[6362620 rows x 8 columns]>
```

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.

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

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

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

```
df.info()
```

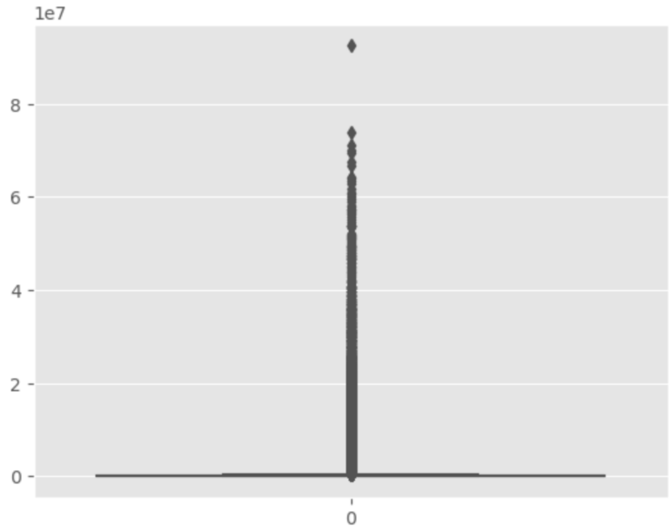
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 8 columns):
 #   Column          Dtype  
---  ------          -----  
 0   step            int64  
 1   type            object 
 2   amount          float64
 3   oldbalanceOrg   float64
 4   newbalanceOrig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

Activity 2: Handling outliers

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

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

```
<Axes: >
```



Remove the Outliers

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

```
ModeResult(mode=array([10000000.]), count=array([3207]))
179861.90354913071
```

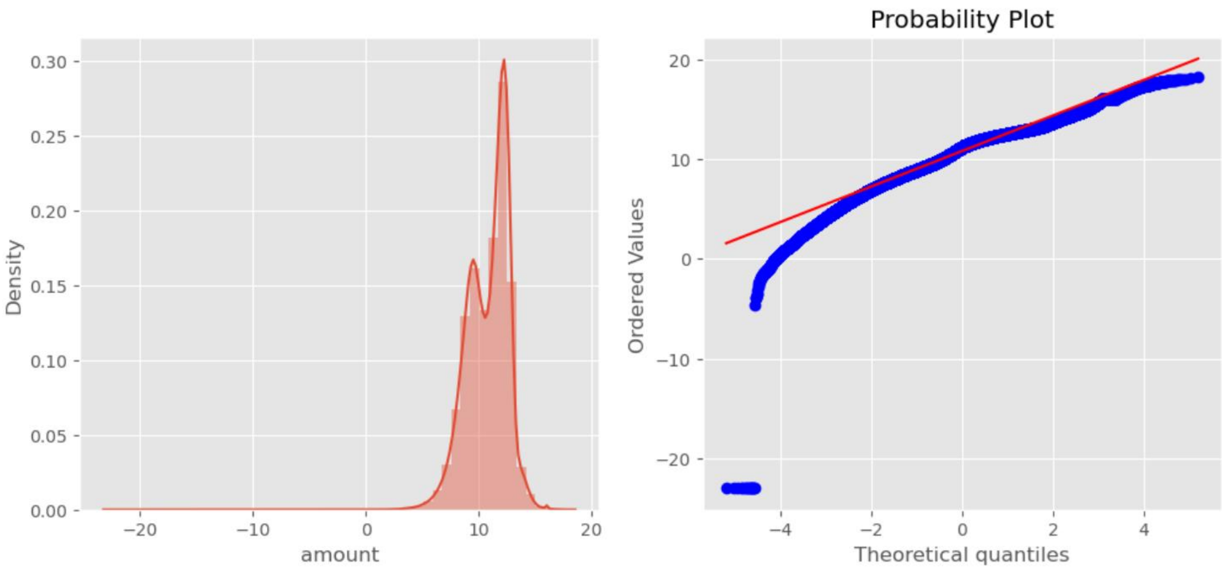
```
q1 = np.quantile(df['amount'],0.25)
q3 = np.quantile(df['amount'],0.75)

IQR = q3-q1

upper_bound = q3+(1.5*IQR)
lower_bound = q1-(1.5*IQR)

print('q1 :',q1)
print('q3 :',q3)
print('IQR :',IQR)
print('Upper Bound :', upper_bound)
print('Lower Bound :', lower_bound)
print('Skewed data :',len(df[df['amount']>upper_bound]))
print('Skewed data :',len(df[df['amount']>lower_bound]))
```

```
q1 : 13389.57
q3 : 208721.4775
IQR : 195331.9075
Upper Bound : 501719.33875
Lower Bound : -279608.29125
Skewed data : 338078
Skewed data : 6362620
```

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

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

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

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

```
transformationPlot(np.log(df['amount'] + 1e-10))
```

```
transformationPlot(np.log(df['amount'] + 1e-10))
```




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

Activity 3: Object data labelencoding

```
from sklearn.preprocessing import LabelEncoder

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

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

```
1    2237500
3    2151495
0    1399284
4     532909
2      41432
Name: type, dtype: int64
```

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

```
mask_infinite = np.isinf(x).any(axis=1)

x = x[~mask_infinite]
y = y[~mask_infinite]
```

```
x
```

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

6362604 rows × 7 columns

```
y
```

```
0          is not Fraud
1          is not Fraud
2              is Fraud
3              is Fraud
4          is not Fraud
               ...
6362615        is Fraud
6362616        is Fraud
6362617        is Fraud
6362618        is Fraud
6362619        is Fraud
Name: isFraud, Length: 6362604, dtype: object
```

Activity 4: Splitting data into train and test

Now let's split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set. Here x and y variables are created. On x variable, df is passed with dropping the target variable. And my target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x,y, test_size, random_state.

```
from sklearn.model_selection import train_test_split

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

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

```
(5090083, 7)
(1272521, 7)
(1272521,)
(5090083,)
```

Milestone 4: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on 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.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rfc=RandomForestClassifier()
```

```
rfc.fit(x_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
y_test_predict1=rfc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict1)
test_accuracy
```

```
0.9997218120565398
```

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

```
0.9999996070790987
```

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

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	1342	326
is not Fraud	28	1270825

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

```
              precision    recall  f1-score   support

    is Fraud       0.98      0.80      0.88      1668
is not Fraud       1.00      1.00      1.00   1270853

    accuracy                           1.00   1272521
   macro avg       0.99      0.90      0.94   1272521
weighted avg       1.00      1.00      1.00   1272521
```

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.

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

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_test_predict2=rfc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict2)
test_accuracy
```

```
0.9997218120565398
```

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

```
0.9999996070790987
```

```
pd.crosstab(y_test,y_test_predict2)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	1342	326
is not Fraud	28	1270825

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

```
              precision    recall  f1-score   support

    is Fraud       0.98      0.80      0.88      1668
is not Fraud       1.00      1.00      1.00   1270853

    accuracy                           1.00   1272521
   macro avg       0.99      0.90      0.94   1272521
weighted avg       1.00      1.00      1.00   1272521
```

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.

```
from sklearn.ensemble import ExtraTreesClassifier
etc=ExtraTreesClassifier()
etc.fit(x_train,y_train)
```

```
▾ ExtraTreesClassifier
ExtraTreesClassifier()
```

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

```
0.9997218120565398
```

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

```
0.9999996070790987
```

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

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	1342	326
is not Fraud	28	1270825

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

```
              precision    recall  f1-score   support

    is Fraud       0.98      0.80      0.88      1668
is not Fraud       1.00      1.00      1.00   1270853

    accuracy                           1.00   1272521
   macro avg       0.99      0.90      0.94   1272521
weighted avg       1.00      1.00      1.00   1272521
```

Activity 4: Compare the model and Saving the model

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

```
def compareModel():
    print("train accuracy for rfc",accuracy_score(y_train_predict1,y_train))
    print("train accuracy for rfc",accuracy_score(y_test_predict1,y_test))
    print("train accuracy for dtc",accuracy_score(y_train_predict2,y_train))
    print("train accuracy for dtc",accuracy_score(y_test_predict2,y_test))
    print("train accuracy for etc",accuracy_score(y_train_predict3,y_train))
    print("train accuracy for etc",accuracy_score(y_test_predict3,y_test))
```

```
compareModel()
```

```
train accuracy for rfc 0.9999996070790987
train accuracy for rfc 0.9997218120565398
train accuracy for dtc 0.9999996070790987
train accuracy for dtc 0.9997218120565398
train accuracy for etc 0.9999996070790987
train accuracy for etc 0.9997218120565398
```

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

Milestone 5: Application Building

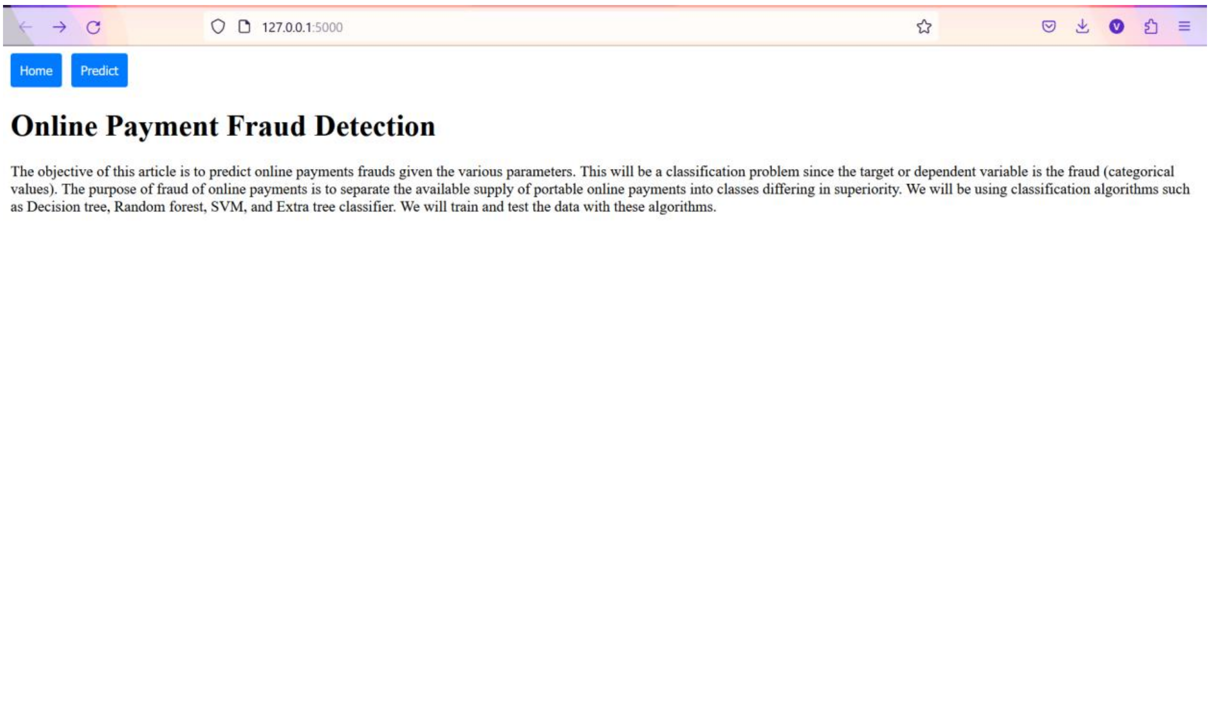
In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

Activity1: Building Html Pages:

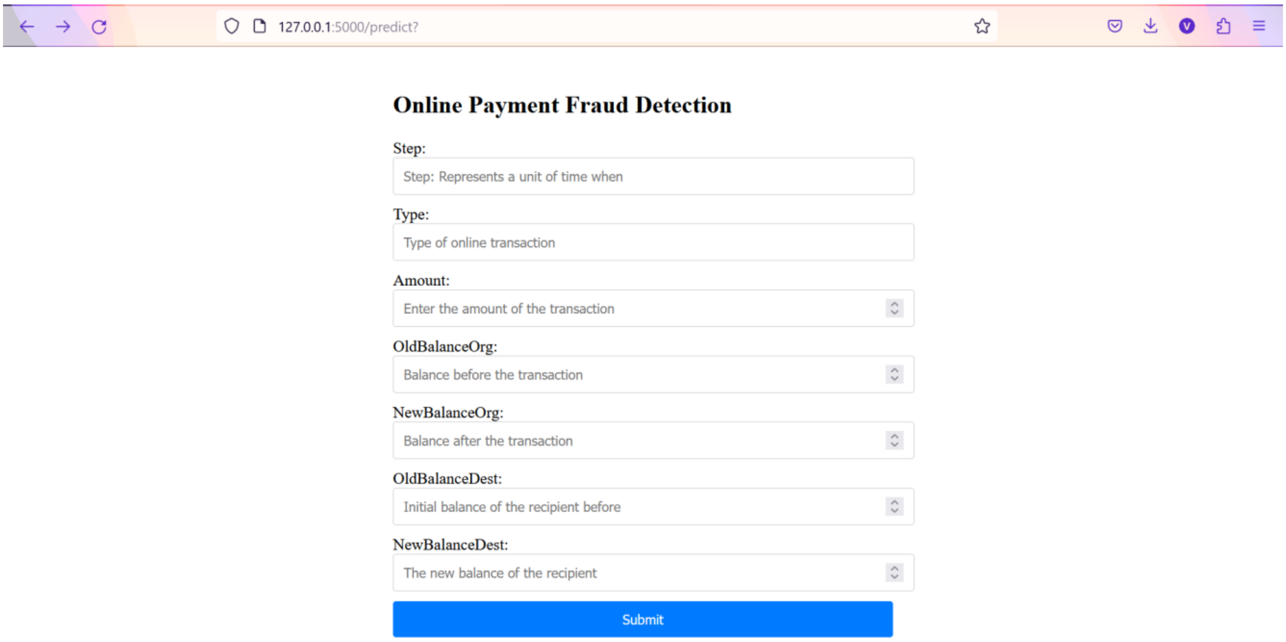
For this project create three HTML files namely

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

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



predict.html



Activity 2: Build Python code

Create a new app.py file which will be store in the Flask folder. Import the necessary Libraries.

```
#pip install flask
from flask import Flask, render_template, request
import pickle
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
```

This code first loads the saved Random Forest Classifier from the "packets.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened in binary mode to read data from it. After loading the model, the code creates a new Flask web application object named "app" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.

```python
# Loading the mlr model
model = pickle.load(open('../training/payments.pkl', 'rb'))
app = Flask(__name__)  # your application
```

This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed. The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called. The "render_template()" method is used to render an HTML template named "home.html". The “home.html” is the home page.

```python
@app.route('/')  # default route
def home():
    return render_template('home.html')  # rendering your home page.
```

The route in this case is "/predict". When a user accesses the "/predict" route of the website, this function is “home()” called. The "render_template()" method is used to render an HTML template named "predict.html".

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

This code sets up another route for the Flask web application using the "@app.route()" decorator. The route in this case is "/predict", and the method is set to GET and POST.

Main function:

```python
@app.route('/submit-form', methods=['POST'])  # prediction route
def predict1():
    '''
    For rendering results on HTML
    '''
    try:
        # Extracting data from the form
        Step = int(request.form.get('step'))
        Type = int(request.form.get('type'))
        Amount1 = int(request.form.get('amount'))
        Amount2 = int(request.form.get('oldBalanceOrg'))
        Amount3 = int(request.form.get('newBalanceOrg'))
        CardNumber1 = int(request.form.get('oldBalanceDest'))
        CardNumber2 = int(request.form.get('newBalanceDest'))

        # Create a DataFrame from the form data
        input_df = pd.DataFrame({
        'step': [Step],
        'type': [Type],
        'amount': [Amount1],
        'oldbalanceOrg': [Amount2],
        'newbalanceOrig': [Amount3],
        'oldbalanceDest': [CardNumber1],
        'newbalanceDest': [CardNumber2]
    })

        # Make prediction using the pre-trained model
        prediction = model.predict(input_df)
        result_str = str(prediction[0])
```

```python
        return render_template("submit.html", result="The prediction is " + result_str + "!")
    except Exception as e:
        return render_template("submit.html", result="Error: " + str(e))

# running your application
if __name__ == "__main__":
    app.run()
```

Activity 3:

Run the Web Application When you run the “app.py” file this window will open in the console or output terminal. Copy the URL given in the form http://127.0.0.1:5000 and paste it in the browser.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                    python - flask
PS C:\Users\vudda\OneDrive\Desktop\Online Payment Fraud Detection> cd flask
PS C:\Users\vudda\OneDrive\Desktop\Online Payment Fraud Detection\flask> python app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [22/Nov/2023 23:16:33] "GET / HTTP/1.1" 200 -
```

When we paste the URL in a web browser, our home.html page will open and after entering the details in predict.html we will get our submit.html page.