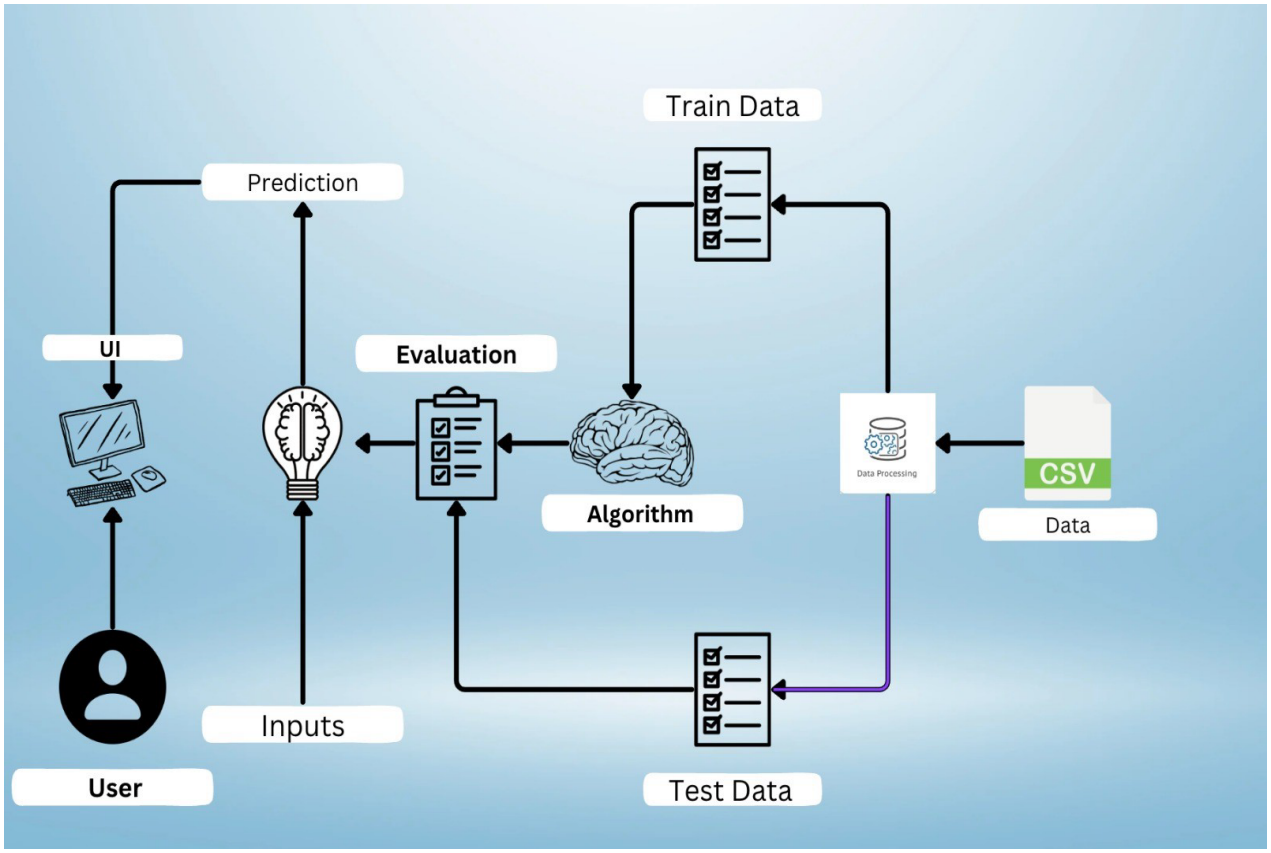
Online Payments Fraud Detection using ML

Project Description:

The growth in internet and e-commerce appears to involve the use of online credit/debit card transactions. The increase in the use of credit / debit cards is causing an increase in fraud. The frauds can be detected through various approaches, yet they lag in their accuracy and its own specific drawbacks. If there are any changes in the conduct of the transaction, the frauds are predicted and taken for further process. Due to large amount of data credit / debit card fraud detection problem is rectified by the proposed method

We will be using classification algorithms such as Decision tree, Random forest, SVM, and Extra tree classifier, Xgboost Classifier.We will train and test the data with these algorithms. From this the best model is selected and saved in pkl format. We will be doing flask integration and IBM deployment.

Technical Architecture:



Pre requisites:

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

- **Anaconda navigator and pycharm:**
  - Refer the link below to download anaconda navigator ○ Link : https://youtu.be/1ra4zH2G4o0
- **Python packages:**
  - Open anaconda prompt as administrator ○ Type"pip install numpy"and click enter. ○ Type"pip install pandas"andclickenter.
  - Type"pip install scikit-learn"andclickenter. ○ Type"pip install matplotlib"andclickenter.
  - Type"pip install scipy"andclickenter.
  - Type"pip install pickle-mixin"andclickenter. ○ Type"pip install seaborn"andclickenter.
  - Type"pipinstallFlask"and click enter.

Prior Knowledge:

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

- **ML Concepts** ○ Supervisedlearning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervisedlearning: https://www.javatpoint.com/unsupervised-machine-learning
  - Regression and classification ○ Decisiontree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
  - Randomforest: https://www.javatpoint.com/machine-learning-random-forest-algorithm **o**

    xgboost Classifier https://www.javatpoint.com/xgboost-classifier-algorithm-for-machine-le arning
  - Svm:

https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to understand-the-math-behind-Svm/

- Extra tree classifier:

https://www.javatpoint.com/Extratreeclassifier-algorithm-for-machine-learning

- Evaluationmetrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-eva luation-error-metrics/
- **Flask Basics** : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and some visualisation concepts.

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 o

  Univariate analysis ○

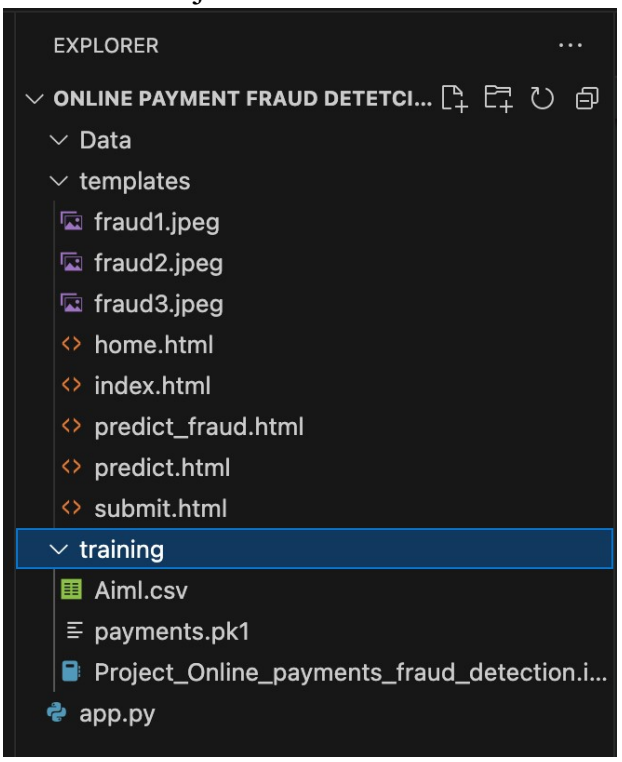
  Bivariate analysis

  - 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.

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 Aiml.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/ruksana27/online-payments-fraud-detection

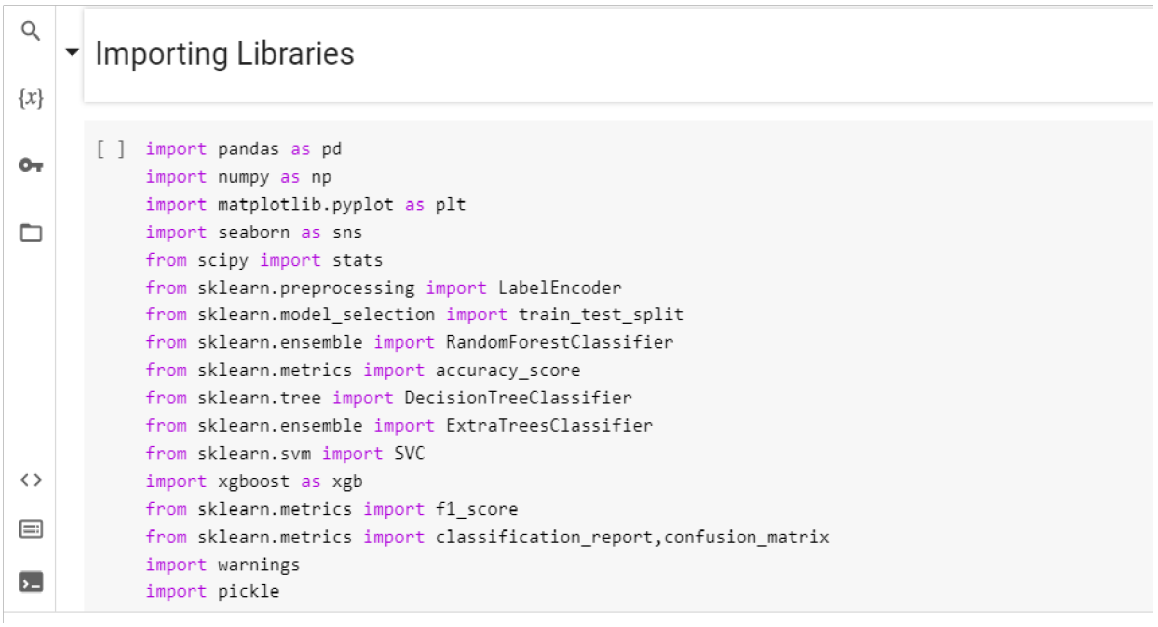
Milestone 2: Visualising and analysing data

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.



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.

```
df=pd.read_csv("Aiml.csv")
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFra
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	
...	...	...	...	...	...	...	...	...	...	...	
4994	5	CASH_IN	185680.98	C1186328673	3756863.21	3942544.19	C985934102	1774746.94	1589065.96	0	
4995	5	CASH_IN	67017.13	C1000600589	3942544.19	4009561.32	C1163619825	118844.67	51827.53	0	
4996	5	CASH_IN	122744.28	C277549599	4009561.32	4132305.60	C1850042097	207106.34	84362.06	0	
4997	5	CASH_IN	414729.24	C1185631996	4132305.60	4547034.84	C991505714	2109808.94	1695079.69	0	
4998	5	CASH_IN	328776.10	C804559024	4547034.84	4875810.94	C977993101	1019467.84	962737.60	0	

```
df.columns
```

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

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

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

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

```
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
...	...	...	...	...	...	...	...	...	...	...
4994	5	CASH_IN	185680.98	C1186328673	3756863.21	3942544.19	C985934102	1774746.94	1589065.96	0
4995	5	CASH_IN	67017.13	C1000600589	3942544.19	4009561.32	C1163619825	118844.67	51827.53	0
4996	5	CASH_IN	122744.28	C277549599	4009561.32	4132305.60	C1850042097	207106.34	84362.06	0
4997	5	CASH_IN	414729.24	C1185631996	4132305.60	4547034.84	C991505714	2109808.94	1695079.69	0
4998	5	CASH_IN	328776.10	C804559024	4547034.84	4875810.94	C977993101	1019467.84	962737.60	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
8. oldbalanceDest: initial balance of recipient before the transaction
9. newbalanceDest: the new balance of recipient after the transaction
10. isFraud: fraud transaction

```
[ ] 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

above, the dataset's first five values are loaded using the head method.

```
[ ] df.tail()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
4994	5	CASH_IN	185680.98	C1186328673	3756863.21	3942544.19	C985934102	1774746.94	1589065.96	0
4995	5	CASH_IN	67017.13	C1000600589	3942544.19	4009561.32	C1163619825	118844.67	51827.53	0
4996	5	CASH_IN	122744.28	C277549599	4009561.32	4132305.60	C1850042097	207106.34	84362.06	0
4997	5	CASH_IN	414729.24	C1185631996	4132305.60	4547034.84	C991505714	2109808.94	1695079.69	0

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

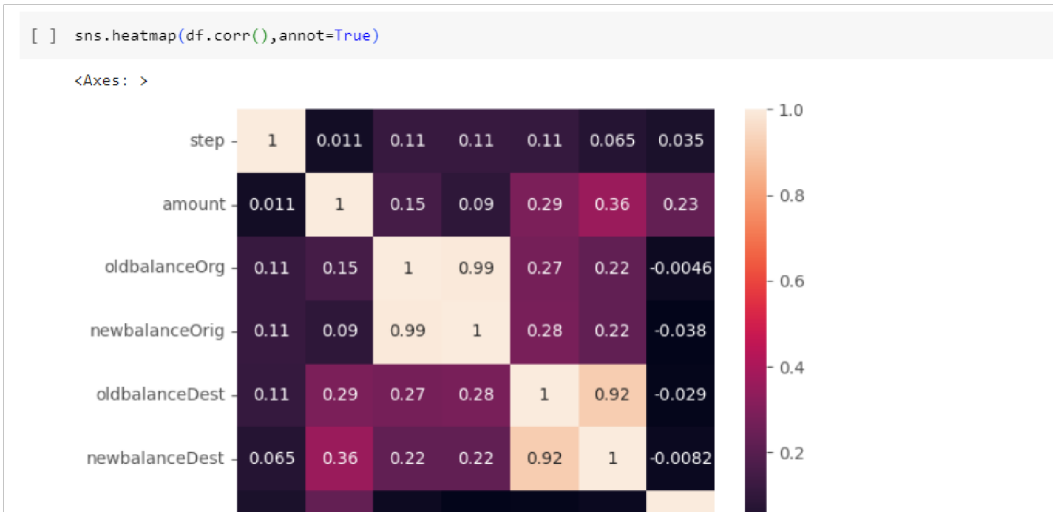
```
plt.style.use('ggplot')
warnings.filterwarnings('ignore')
```

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.

```
df.corr()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
step	1.000000	0.010893	0.113231	0.112264	0.108365	0.065309	0.035104
amount	0.010893	1.000000	0.148887	0.090259	0.289956	0.360693	0.228112
oldbalanceOrg	0.113231	0.148887	1.000000	0.994942	0.270620	0.222078	-0.004570
newbalanceOrig	0.112264	0.090259	0.994942	1.000000	0.279013	0.224265	-0.037963
oldbalanceDest	0.108365	0.289956	0.270620	0.279013	1.000000	0.915803	-0.028900
newbalanceDest	0.065309	0.360693	0.222078	0.224265	0.915803	1.000000	-0.008196
isFraud	0.035104	0.228112	-0.004570	-0.037963	-0.028900	-0.008196	1.000000

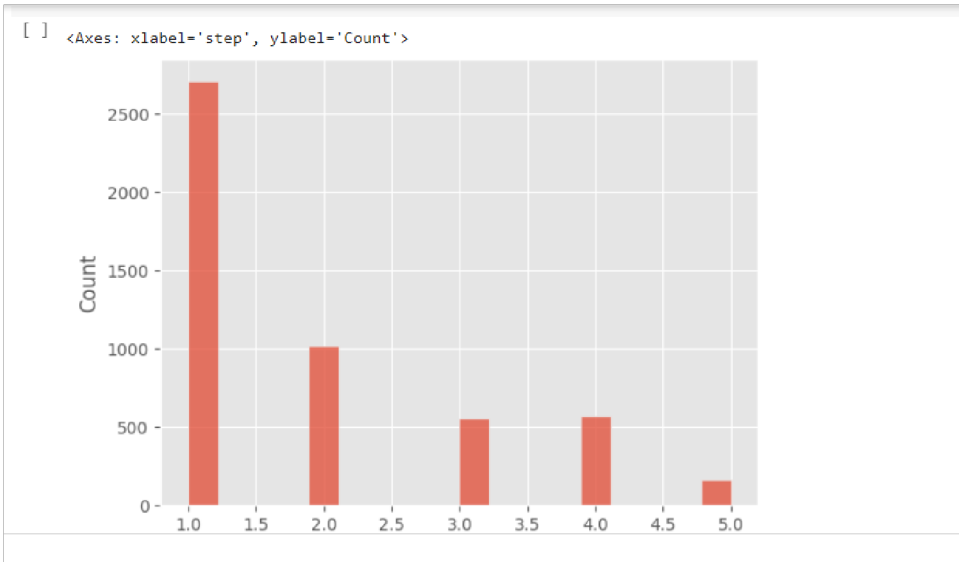
utilising the corr function to examine the dataset's correlation



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

Activity 3: Univariate analysis

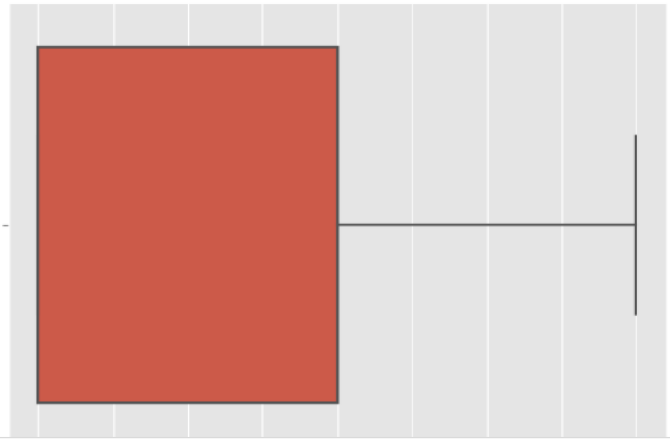
In simple words, univariate analysis is understanding the data with a single feature. Here I have displayed the graph such as histplot .



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.

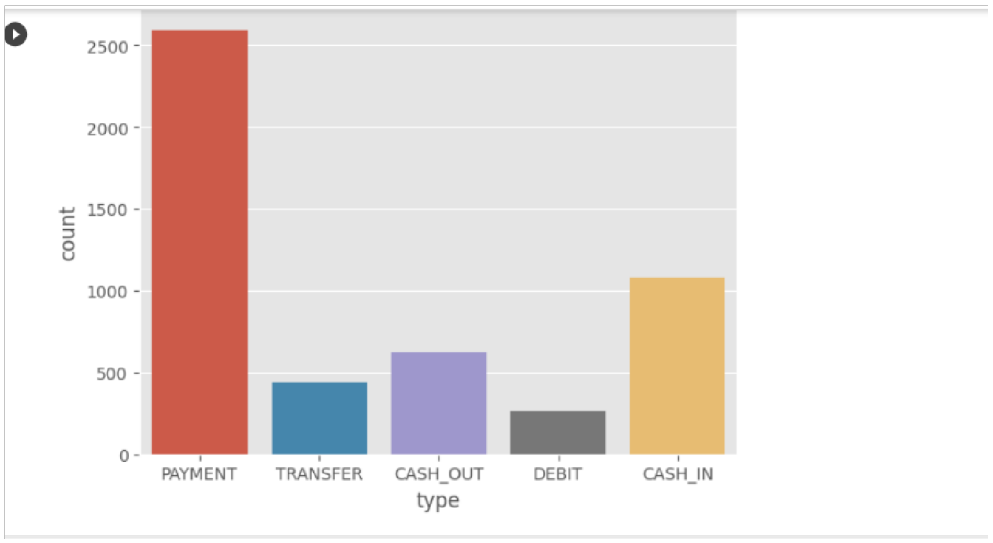
```
[ ] sns.boxplot(data=df,x='step')
```

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



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

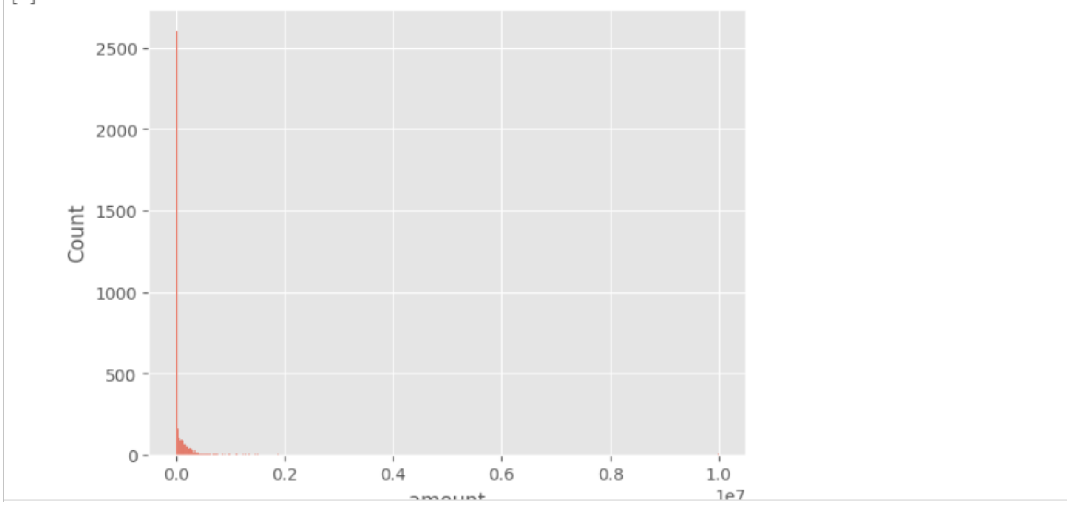
```
sns.countplot(data=df,x='type')
```



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

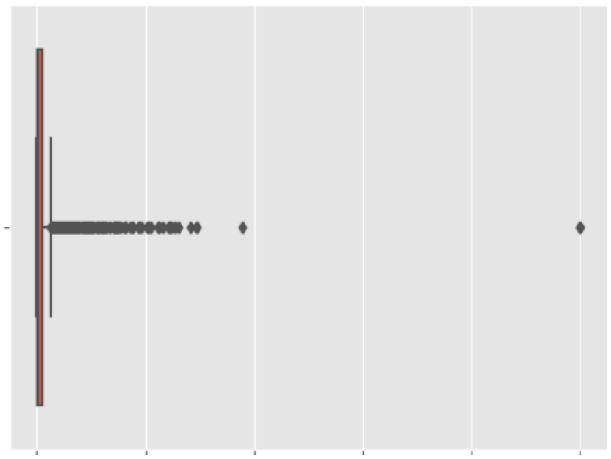
```
[ ] sns.histplot(data=df,x='amount')
```

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



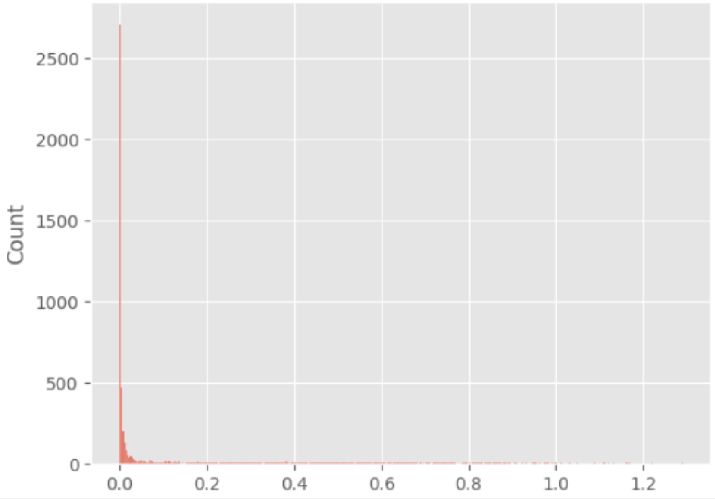
By creating bins along the data's range and then drawing bars to reflect the number of observations that fall within the amount attribute in the dataset.

```
[ ] 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'>
```



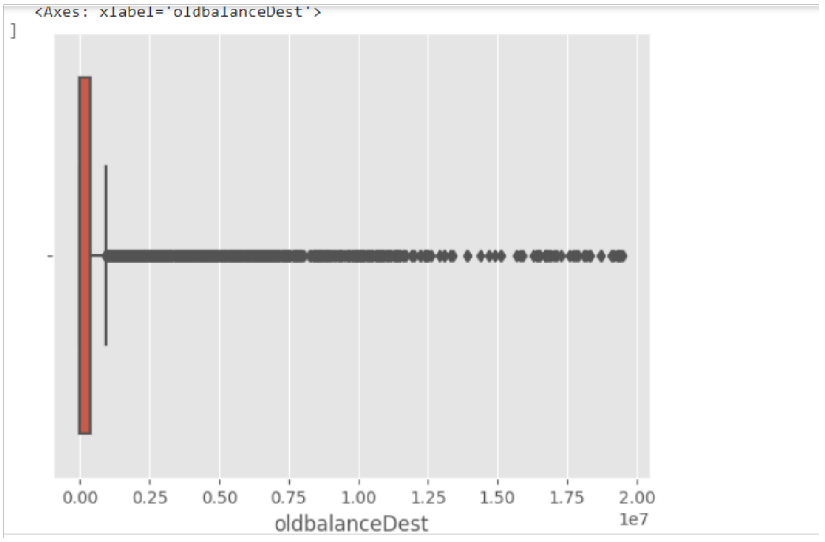
By creating bins along the data's range and then drawing bars to reflect the number of observations that fall within the oldbalanceOrg attribute in the dataset.

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

C985934102     46
C1590550415    40
C564160838     38
C998351292     33
C977993101     33
              ..
M1447685190     1
M1345293143     1
C243745864      1
M1427247001     1
C539058198      1
Name: nameDest, Length: 3012, dtype: int64
```

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

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

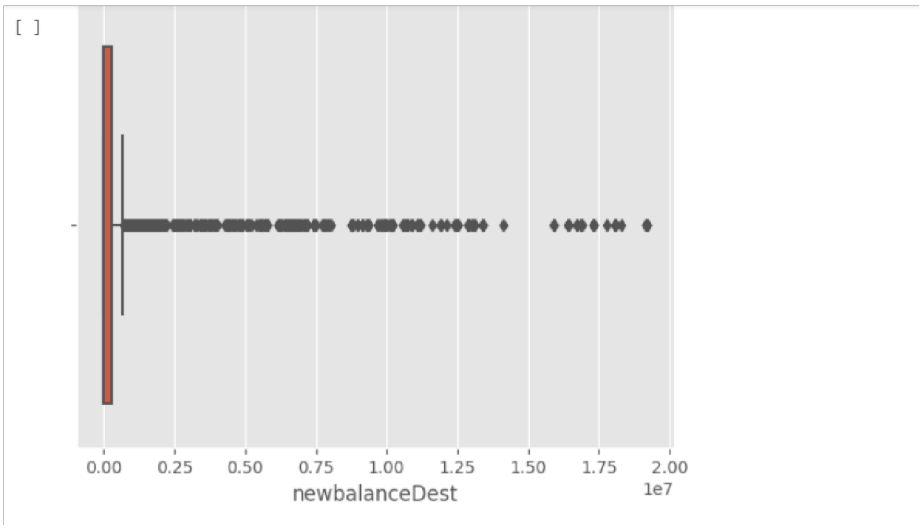
```
<Axes: xlabel='oldbalanceDest'>
```



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

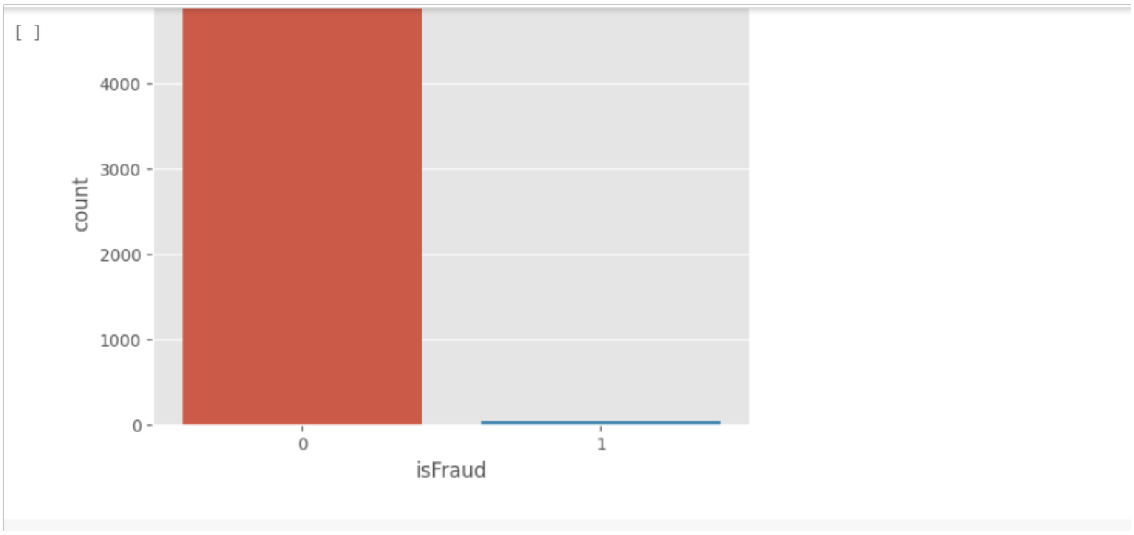
```
[ ]  sns.boxplot(data=df,x='newbalanceDest')
```

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



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

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



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

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

    0    4959
    1      40
    Name: isFraud, dtype: int64
```

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

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

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	is not Fraud
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	is not Fraud
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	is Fraud
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	is Fraud
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	is not Fraud
...	...	...	...	...	...	...	...	...	...	...
4994	5	CASH_IN	185680.98	C1186328673	3756863.21	3942544.19	C985934102	1774746.94	1589065.96	is not Fraud
4995	5	CASH_IN	67017.13	C1000600589	3942544.19	4009561.32	C1163619825	118844.67	51827.53	is not Fraud
4996	5	CASH_IN	122744.28	C277549599	4009561.32	4132305.60	C1850042097	207106.34	84362.06	is not Fraud
4997	5	CASH_IN	414729.24	C1185631996	4132305.60	4547034.84	C991505714	2109808.94	1695079.69	is not Fraud

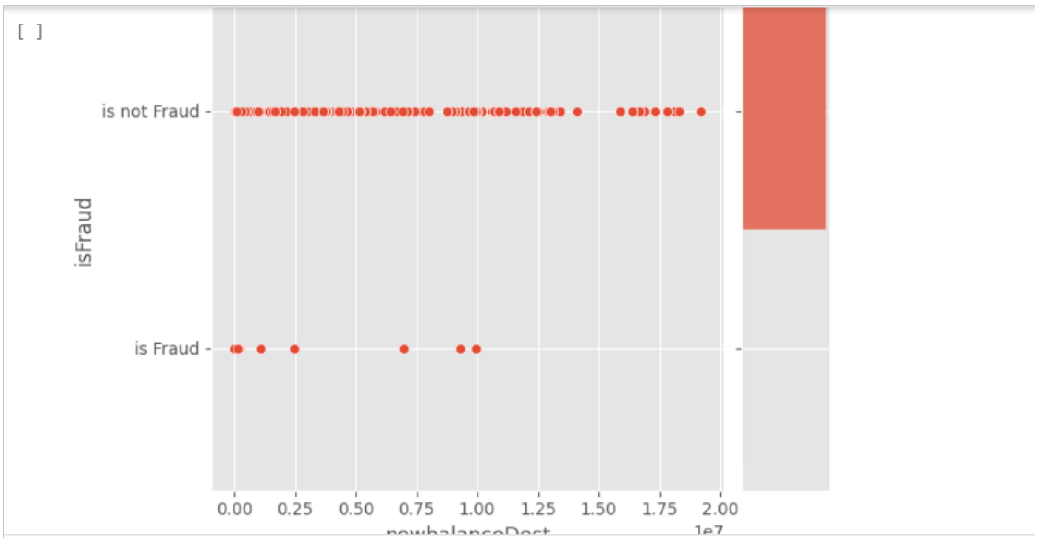
converting 0-means: is not fraud and 1-means: is fraud using the loc technique here

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.

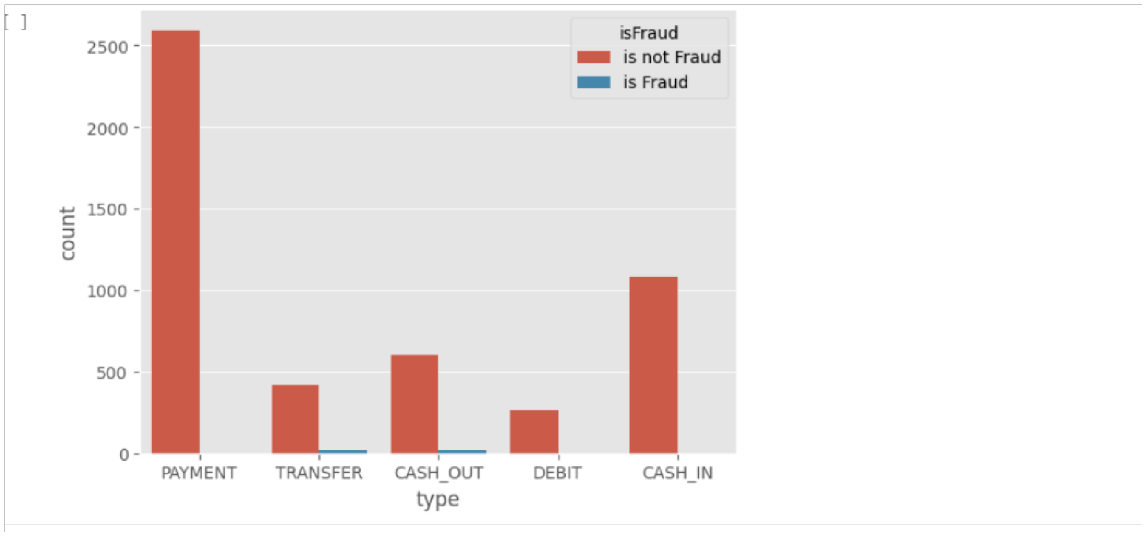
jointplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

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



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')
```



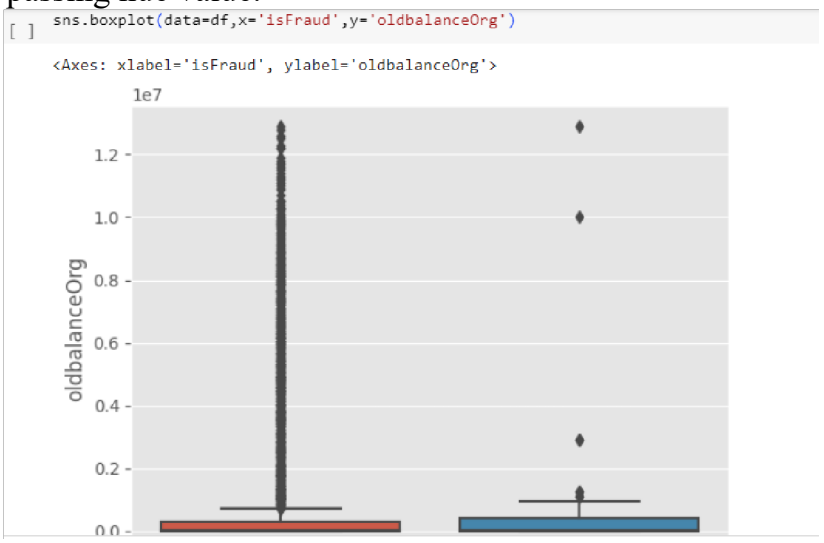
Here we are visualising the relationship between isFraud and step.boxtplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
[ ] <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.

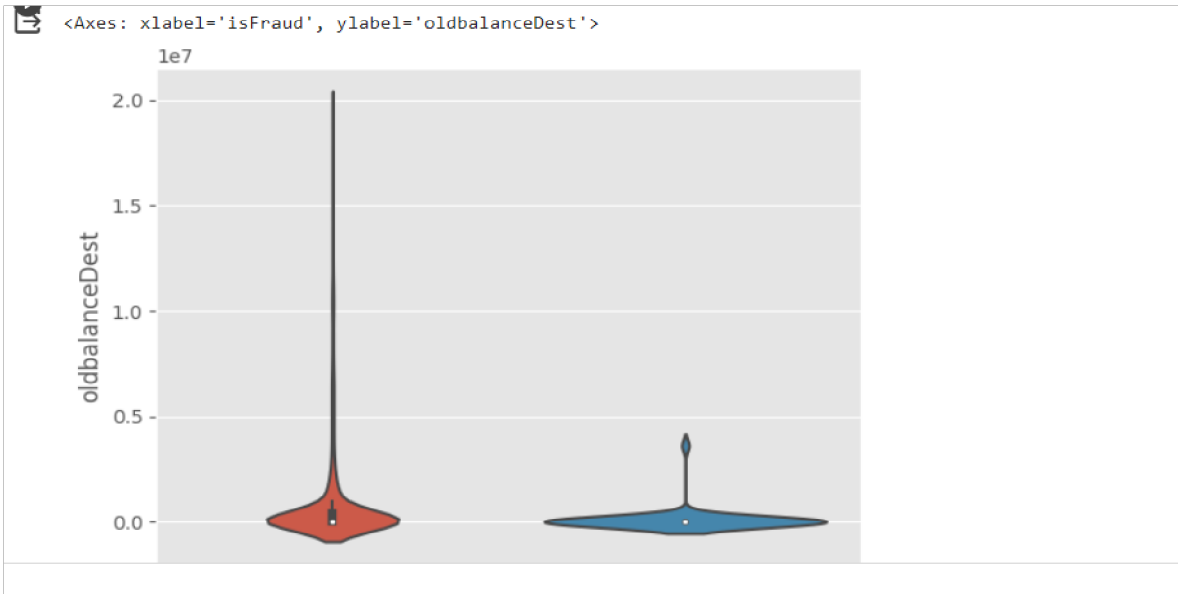
```
[ ] sns.boxplot(data=df,x='isFraud',y='amount')

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

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

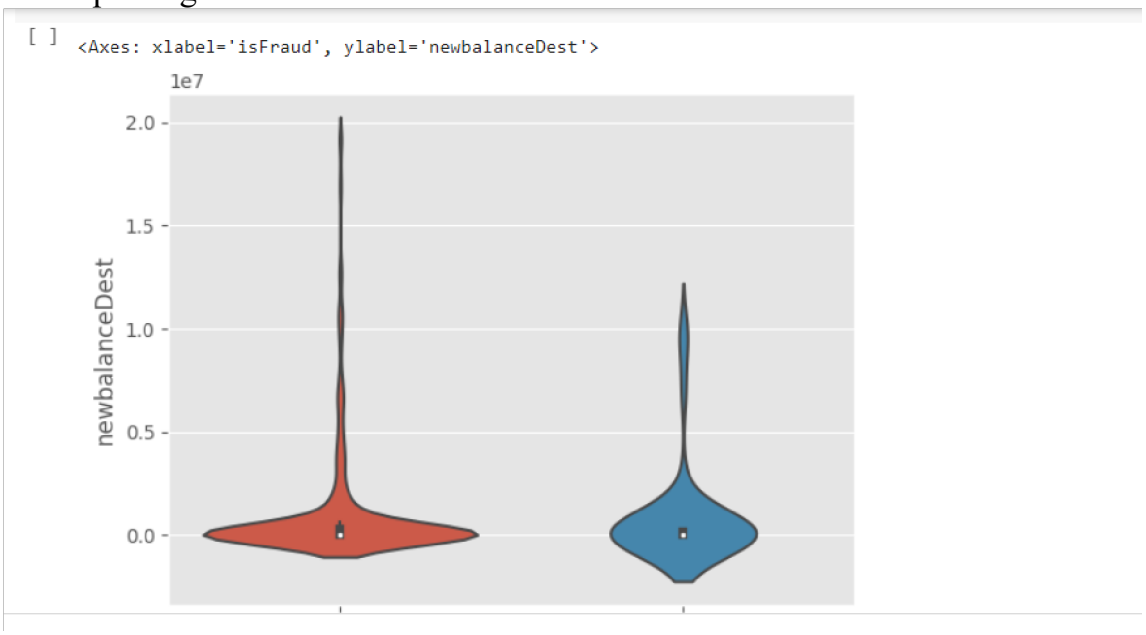


Here we are visualising the relationship between isFraud and newbalanceOrig. boxtplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

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.



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.



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	isFrau
count	4999.000000	4999	4.999000e+03	4999	4.999000e+03	4.999000e+03	4999	4.999000e+03	4.999000e+03	499
unique	NaN	5	NaN	4991	NaN	NaN	3012	NaN	NaN	
top	NaN	PAYMENT	NaN	C361380654	NaN	NaN	C985934102	NaN	NaN	is not Frau
freq	NaN	2593	NaN	2	NaN	NaN	46	NaN	NaN	495
mean	1.890778	NaN	1.014858e+05	NaN	1.041848e+06	1.063625e+06	NaN	9.178844e+05	1.062722e+06	Na
std	1.175000	NaN	3.009751e+05	NaN	2.301421e+06	2.343236e+06	NaN	2.541364e+06	2.912463e+06	Na
min	1.000000	NaN	6.420000e+00	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	Na
25%	1.000000	NaN	3.734685e+03	NaN	5.940000e+02	0.000000e+00	NaN	0.000000e+00	0.000000e+00	Na
50%	1.000000	NaN	1.095121e+04	NaN	2.684541e+04	1.837144e+04	NaN	0.000000e+00	0.000000e+00	Na
75%	3.000000	NaN	1.086811e+05	NaN	2.931860e+05	2.990004e+05	NaN	3.683888e+05	2.701108e+05	Na
max	5.000000	NaN	1.000000e+07	NaN	1.290000e+07	1.300000e+07	NaN	1.950000e+07	1.920000e+07	Na

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

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

```
[ ]  df.shape

     (4999, 10)
```

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

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

```
[ ] df.head()
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	170136.0	160296.36	0.0	0.0	is not Fraud
1	1	PAYMENT	1864.28	21249.0	19384.72	0.0	0.0	is not Fraud
2	1	TRANSFER	181.00	181.0	0.00	0.0	0.0	is Fraud
3	1	CASH_OUT	181.00	181.0	0.00	21182.0	0.0	is Fraud
4	1	PAYMENT	11668.14	41554.0	29885.86	0.0	0.0	is not Fraud

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

Activity 1: Checking for null values

Isnull is used (). sum() to check your database for null values. Using the df.info() function, the data type can be determined.

checking null values

```
[ ] df.isnull().any()
```

```
step            False
type            False
amount          False
oldbalanceOrg   False
newbalanceOrig  False
oldbalanceDest  False
newbalanceDest  False
isFraud         False
dtype: bool
```

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.info()
```

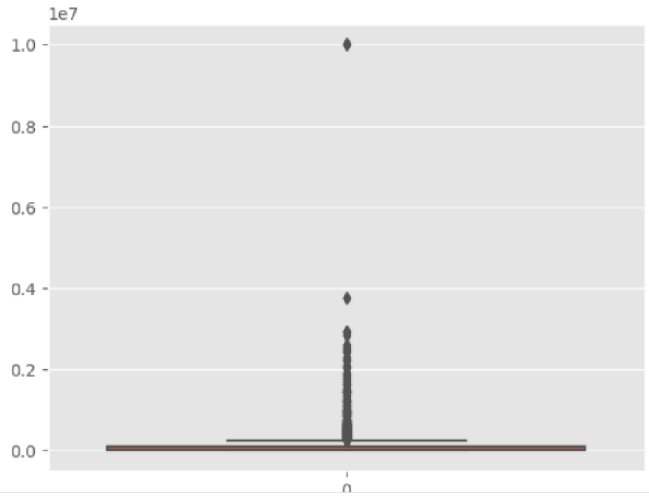
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4999 entries, 0 to 4998
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   step            4999 non-null   int64
 1   type            4999 non-null   object
 2   amount          4999 non-null   float64
 3   oldbalanceOrg   4999 non-null   float64
 4   newbalanceOrig  4999 non-null   float64
 5   oldbalanceDest  4999 non-null   float64
 6   newbalanceDest  4999 non-null   float64
 7   isFraud         4999 non-null   object
dtypes: float64(5), int64(1), object(2)
memory usage: 312.6+ KB
```

determining the types of each attribute in the dataset using the info() function **Activity 2: Handling outliers**

Handling outliers

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

```
[ ] <Axes: >
```



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

Remove the Outliers

Remove outliers

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

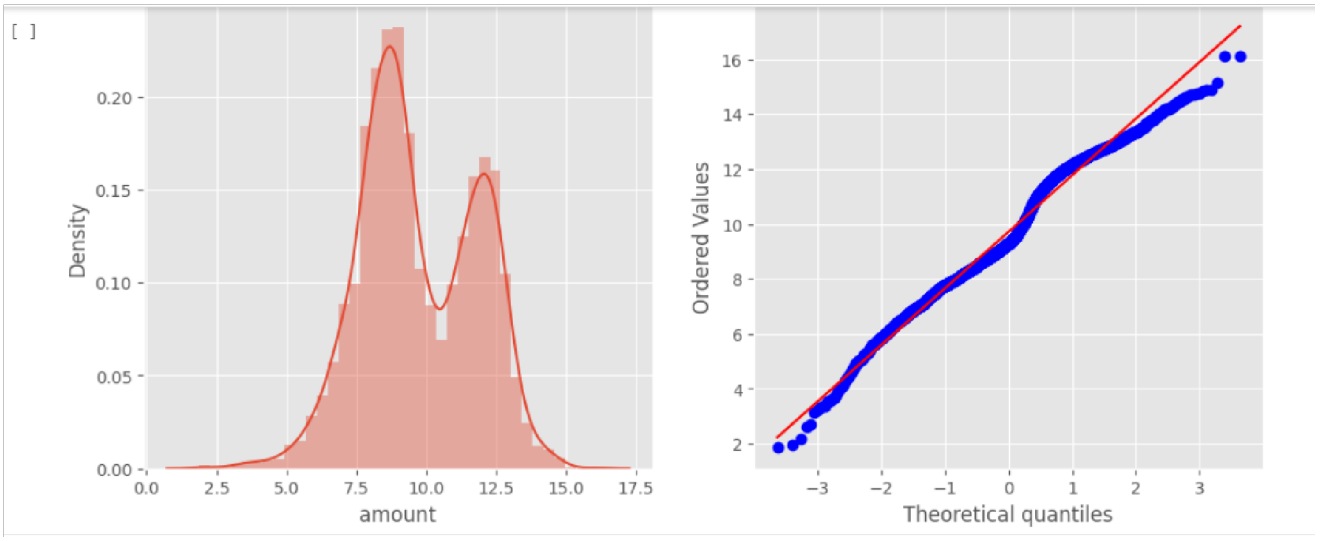
```
ModeResult(mode=181.0, count=2)
101485.80004240847
```

```
[ ] 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('lower_bound: ',lower_bound)
    print('upper_bound: ',upper_bound)
    print('skewed_data: ',len(df[df['amount']>upper_bound]))
    print('skewed_data: ',len(df[df['amount']<lower_bound]))
```

```
q1:  3734.685
q3:  108681.1
lower_bound:  -153684.9375
upper_bound:  266100.72250000003
skewed_data:  519
skewed_data:  0
```

```
[ ] #To handle outliers transformation techniques are used
    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']))
```



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

Activity 3: Object data labelencoding using labelencoder to encode the dataset's object type

object data labelencoding

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

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

```
3    2593
0    1081
1     623
4     438
2     264
Name: type, dtype: int64
```

Separating the data into dependent and independent variables

separating dependent and independent variables

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

```
[ ] 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

```
[ ] 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
...	...	...	...	...	...	...	...
4994	5	0	12.131785	3756863.21	3942544.19	1774746.94	1589065.96
4995	5	0	11.112704	3942544.19	4009561.32	118844.67	51827.53
4996	5	0	11.717858	4009561.32	4132305.60	207106.34	84362.06
4997	5	0	12.935381	4132305.60	4547034.84	2109808.94	1695079.69
4998	5	0	12.703132	4547034.84	4875810.94	1019467.84	962737.60

```
[ ] y
```

```
0       is not Fraud
1       is not Fraud
2           is Fraud
3           is Fraud
4       is not Fraud
           ...
4994    is not Fraud
4995    is not Fraud
4996    is not Fraud
4997    is not Fraud
4998    is not Fraud
Name: isFraud, Length: 4999, 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.

▾ splitting data into training and testing

```
[ ] from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
    x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((3999, 7), (1000, 7), (3999,), (1000,))
```

Milestone 4: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

Activity 1: Random Forest classifier¶

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

▾ 1)Random Forest Classifier

```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score
    rfc=RandomForestClassifier()
    rfc.fit(x_train,y_train)
    y_test_predict1=rfc.predict(x_test)
    test_accuracy=accuracy_score(y_test,y_test_predict1)
    test_accuracy
```

```
0.997
```

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

```
1.0
```

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

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	3	3
is not Fraud	0	994

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

```
                precision    recall  f1-score   support

      is Fraud       1.00      0.50      0.67         6
  is not Fraud       1.00      1.00      1.00       994

      accuracy                           1.00      1000
     macro avg       1.00      0.75      0.83      1000
  weighted avg       1.00      1.00      1.00      1000
```

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.

2)Decision Tree Classifier

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

```
0.998
```

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

```
1.0
```

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

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

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	4	2
is not Fraud	0	994

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

```
                precision    recall  f1-score   support

    is Fraud       1.00      0.67      0.80         6
is not Fraud       1.00      1.00      1.00       994

    accuracy                           1.00      1000
   macro avg       1.00      0.83      0.90      1000
weighted avg       1.00      1.00      1.00      1000
```

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.

▾ 3)ExtraTree Classifier

```
[ ] from sklearn.ensemble import ExtraTreesClassifier
    from sklearn.metrics import accuracy_score
    etc=ExtraTreesClassifier()
    etc.fit(x_train,y_train)
    y_test_predict3=etc.predict(x_test)
    test_accuracy=accuracy_score(y_test,y_test_predict3)
    test_accuracy
```

```
0.996
```

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

```
1.0
```

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

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	2	4
is not Fraud	0	994

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

```
                precision    recall  f1-score   support

      is Fraud       1.00      0.33      0.50         6
  is not Fraud       1.00      1.00      1.00       994

      accuracy                           1.00      1000
     macro avg       1.00      0.67      0.75      1000
  weighted avg       1.00      1.00      1.00      1000
```

Activity 4: SupportVectorMachine Classifier¶

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

▾ Support Vector Machine Classifier

```
[ ] 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
```

```
0.994
```

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

```
0.991747936984246
```

```
[ ] pd.crosstab(y_test,y_test_predict4)
```

col_0	is not Fraud
isFraud	
is Fraud	6
is not Fraud	994

```
[ ] from sklearn.metrics import classification_report,confusion_matrix
    print(classification_report(y_test,y_test_predict4))
```

```
                precision    recall  f1-score   support

      is Fraud       0.00      0.00      0.00         6
  is not Fraud       0.99      1.00      1.00       994

      accuracy                           0.99      1000
     macro avg       0.50      0.50      0.50      1000
  weighted avg       0.99      0.99      0.99      1000
```

preprocessing class of sklearn. LabelEncoder[source] 0 to n classes-1 as the range for the target labels to be encoded. Instead of encoding the input X, the target values, i.e. y, should be encoded using this transformer.

```
[ ] df.columns

    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)
```

```
[ ] y_train1

    array([1, 0, 1, ..., 1, 1, 1])
```

```
[ ] y_test1=la.fit_transform(y_test)
```

```
[ ] y_test1

    array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Activity 5: 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 with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, confusion matrix and classification report is done

xgboost Classifier

```
[ ] import xgboost as xgb
    from sklearn.metrics import accuracy_score
    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
```

```
0.997
```

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

```
1.0
```

```
[ ] pd.crosstab(y_test1,y_test_predict5)
```

col_0	0	1
row_0		
0	3	3
1	0	994

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test1,y_test_predict5))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         6
           1       1.00      1.00      1.00       994

    accuracy                           1.00      1000
   macro avg       1.00      0.75      0.83      1000
weighted avg       1.00      1.00      1.00      1000
```

Activity 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 79% accuracy. .

```
[ ] 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 svc ",accuracy_score(y_test_predict4,y_test))
      print("train accuracy for xgb1 ",accuracy_score(y_train_predict5,y_train1))
      print("test accuracy for xgb1 ",accuracy_score(y_test_predict5,y_test1))
    compareModel()
```

```
train accuracy for rfc  1.0
test accuracy for rfc  0.997
train accuracy for dtc  1.0
test accuracy for dtc  0.998
train accuracy for etc  1.0
test accuracy for etc  0.996
train accuracy for svc  0.991747936984246
test accuracy for svc  0.994
```

```
train accuracy for rfc  1.0
test accuracy for rfc  0.997
train accuracy for dtc  1.0
test accuracy for dtc  0.998
train accuracy for etc  1.0
test accuracy for etc  0.996
train accuracy for svc  0.991747936984246
test accuracy for svc  0.994
train accuracy for xgb1  1.0
test accuracy for xgb1  0.997
```

Activity 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 performance of the model and saving the model

```
[ ] 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
```

```
0.994
```

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

```
0.991747936984246
```

```
[ ] import pickle
    pickle.dump(svc,open('payments.pk1','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

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

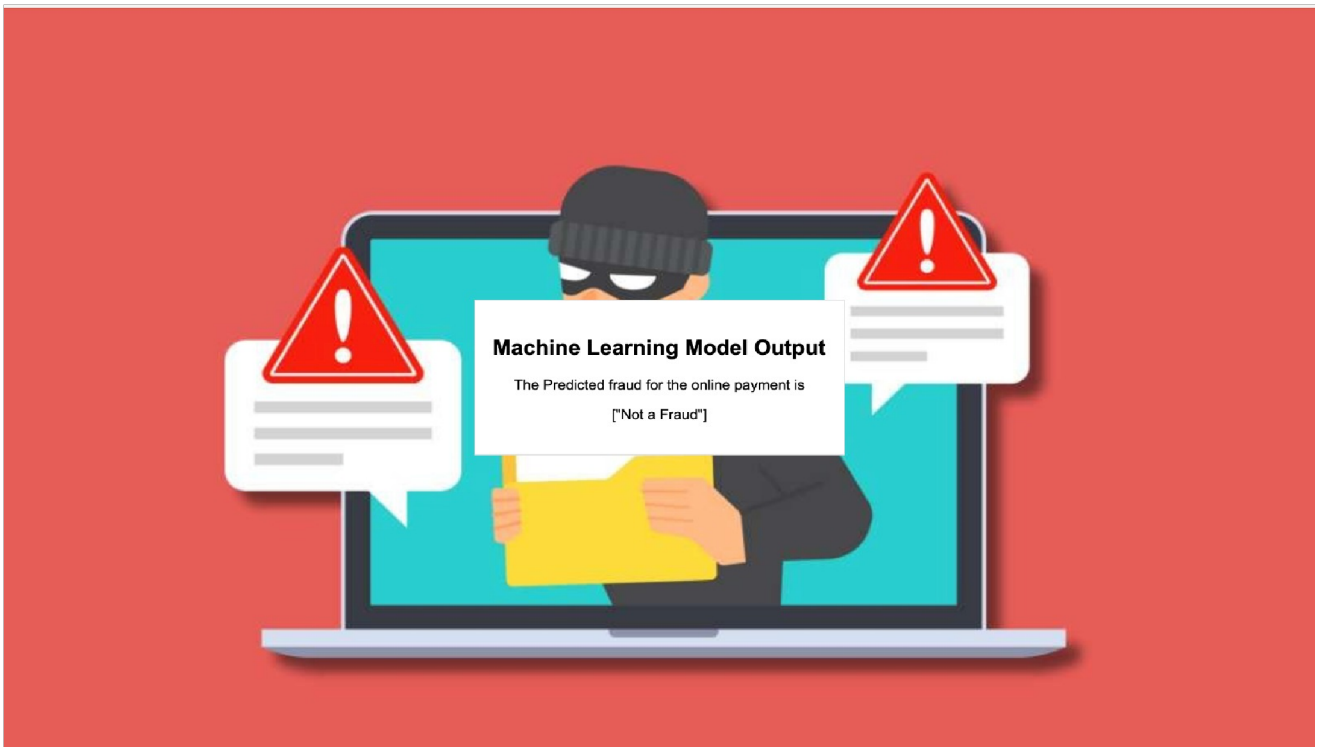
Let’s see how our home.html page looks like:

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:



Activity 2: Build Python code:

Import the libraries

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

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.

```
model=pickle.load(open(r"/Users/omprakashbhukya/Documents/Online Payment Fraud Detetcion/training/payments.pk1",'rb'))

app=Flask(__name__)
```

Render HTML page:

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

@app.route("/home")
def about1():
  return render_template('home.html')
```

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

In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

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

@app.route("/pred",methods=['POST','GET'])
def predict():
  x=[[x for x in request.form.values()]]
  print(x)

  x=np.array(x)
  print(x.shape)

  print(x)
  pred=model.predict(x)
  print(pred[0])
  return render_template('submit.html',prediction_text=str(pred))
```

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.

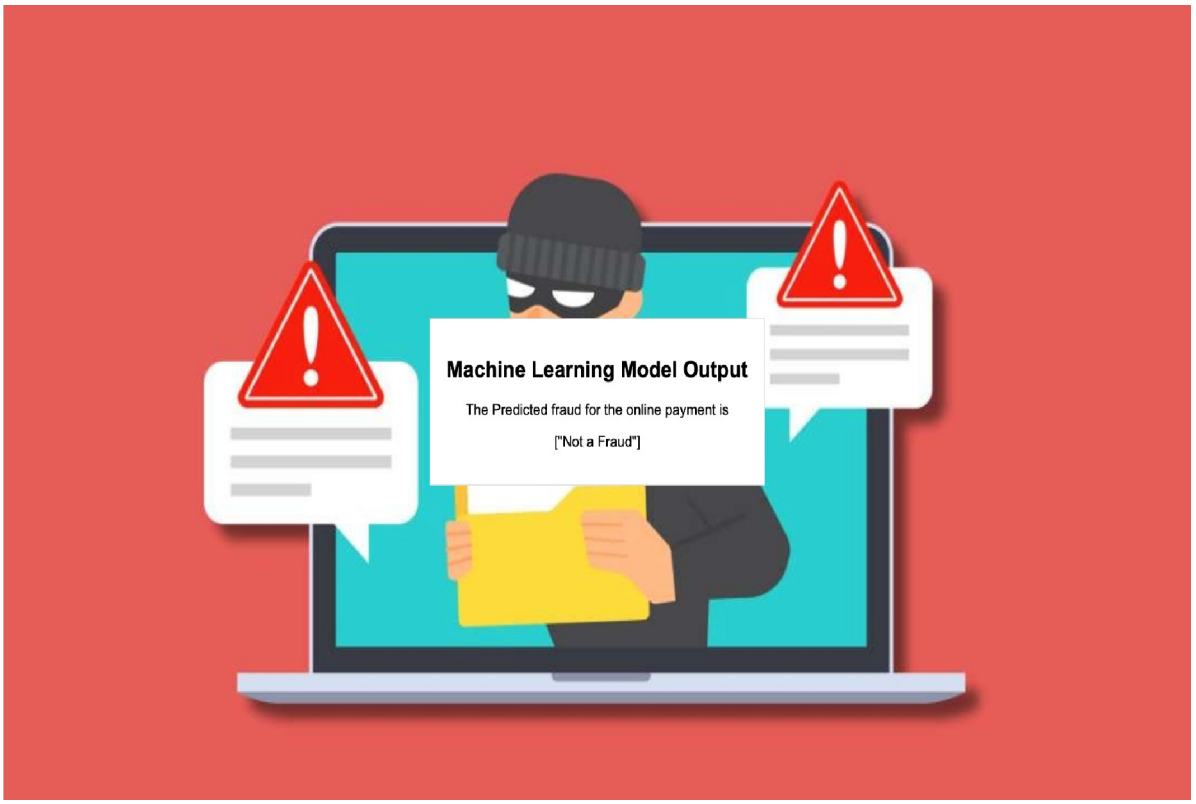
Main Function:

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

Activity 3: Run the application:

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

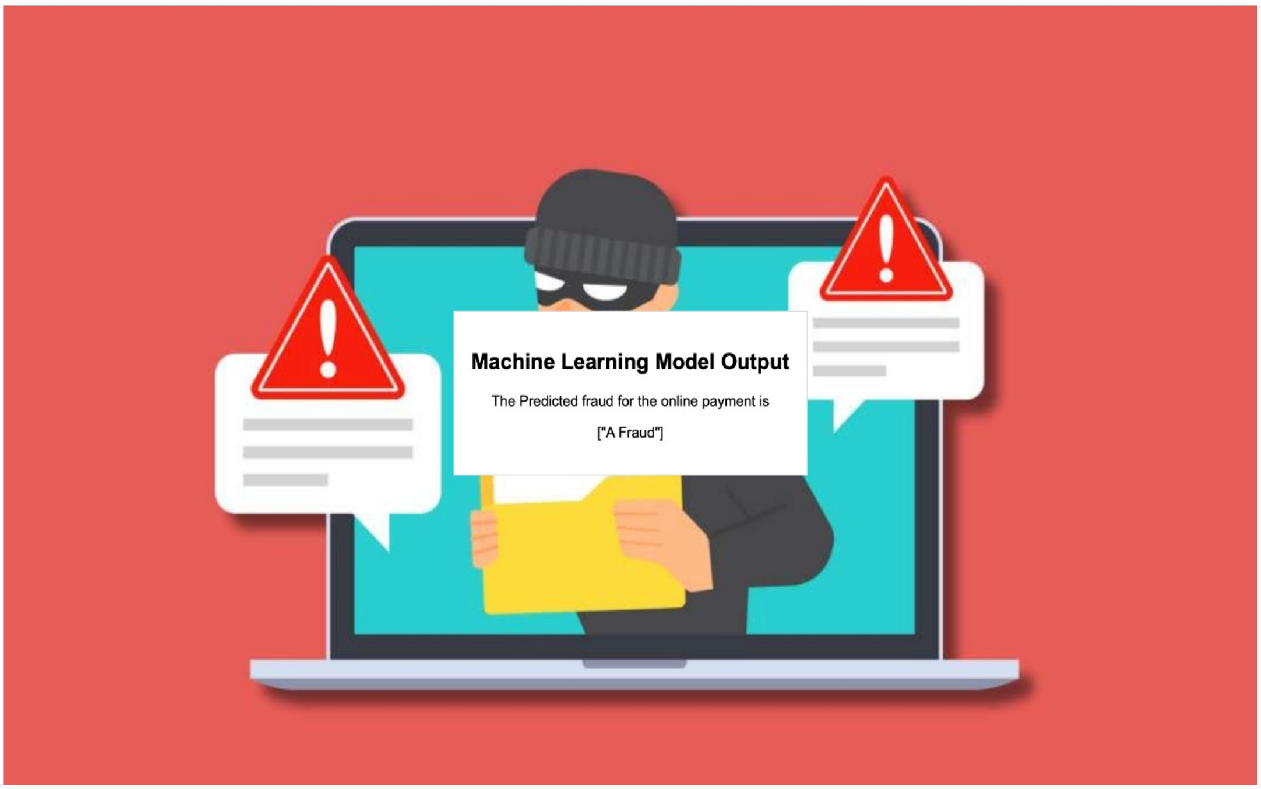
Output screenshots 1:



Output screenshots 2:


Home
Predict
Online Fraud Payment Detection
1234 8765 9876 0987
06/07


Home
Predict
Step:
2
Type:
1
Amount:
9.138070
Old Balance (Origin):
11299.00
New Balance (Origin):
1996.21
Old Balance (Destination):
29832.0
New Balance (Destination):
16896.70
Submit


Machine Learning Model Output
The Predicted fraud for the online payment is
["A Fraud"]