Online Payments Fraud Detection using ML

Team - 592416

Team Members

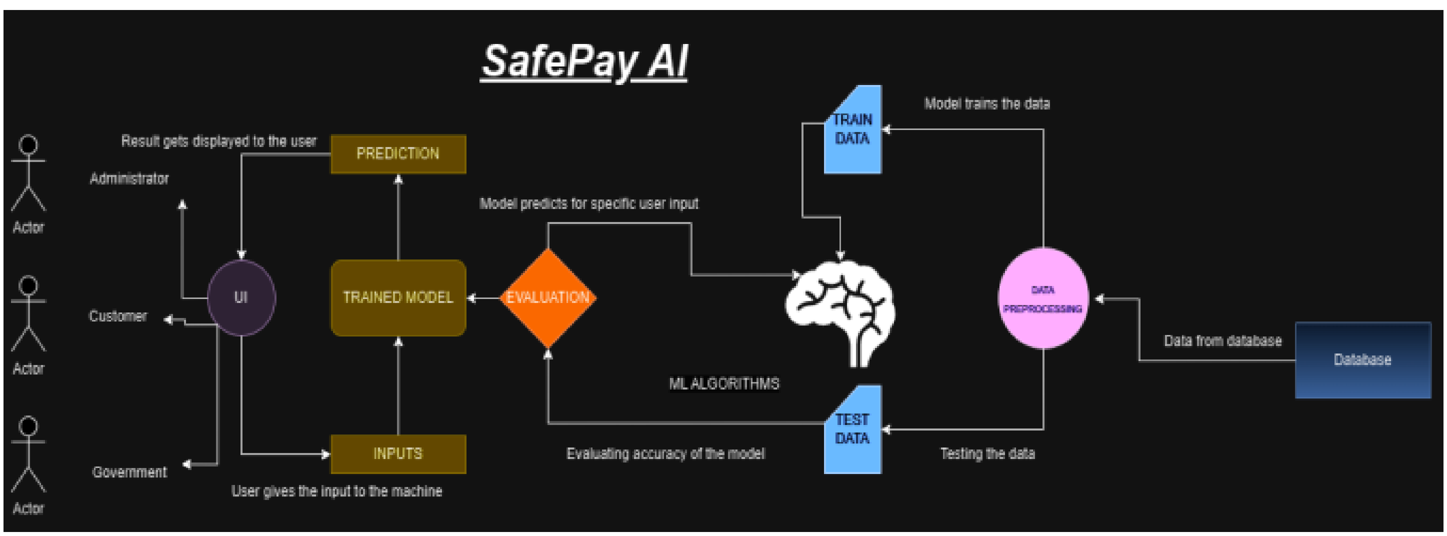
1. Aryan Sharma (21BBS0106)
2. Arjun Malhotra (21BBS0110)
3. Pranav Singh Mahara (21BBS0188)
4. Kanishak Khanna (21BCI0147)

Collab Link - https://colab.research.google.com/drive/1exzxsGaXeiniCsu_rdgzMuGrfszGCIX9?usp=sharing

Project Description

The surge in online credit/debit card transactions has led to an upsurge in fraud. To combat this, we're employing classification algorithms like Decision Trees, Random Forest, SVM and XGBoost. Using a vast dataset, we'll train and evaluate these models, selecting the one with the highest accuracy, precision, and a low false positive rate. The chosen model will be saved in pkl format. We'll integrate it into a Flask web application for real-time fraud detection. This project aims to enhance fraud detection, improve user experience, save costs, and offer real-time protection.

Technical Architecture



Pre requisites:

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

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

Prior Knowledge:

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

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

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

o Evaluationmetrics:
https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

o 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**
    - o Collect the dataset or create the dataset
- **Visualising and analysing data**

Importing the libraries

- o Read the Dataset
- o Univariate analysis
- o Bivariate analysis
- o Descriptive analysis

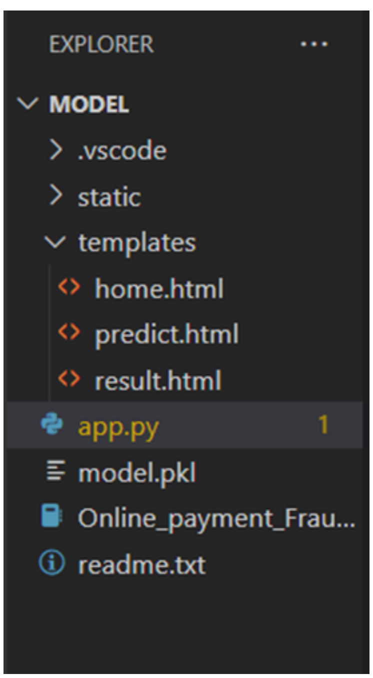
- **Data pre-processing**
    - o Checking for null values
    - o Handling outlier
    - o Handling categorical(object) data
    - o Splitting data into train and test

- **Model building**
    - o Import the model building libraries
    - o Initialising the model
    - o Training and testing the model
    - o Evaluating performance of model
    - o Save the model
- **Application Building**
    - o Create an HTML file
    - o Build python code

Project Structure:

Create the Project folder which contains files as shown below

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Model.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains model training files



Process - 1 : Data Collection

In this project we have used PS_20174392719_1491204439457_logs.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

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

```
[ ] #Importing the libraries
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    #for model building
    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.svm import SVC
    import xgboost as xgb
    #for comparing the models
    from sklearn.metrics import classification_report, confusion_matrix
    import pickle
```

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

```
[ ] df.shape

    (6362620, 10)
```

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()
```

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

Visualizing and analysing Data

'step': Transaction time step.

'type': Transaction type.

'amount': Transaction amount.

'oldbalanceOrg': Original sender's balance.

'newbalanceOrig': Updated sender's balance.

'oldbalanceDest': Original recipient's balance.

'newbalanceDest': Updated recipient's balance.

'isFraud': Indicates if the transaction is fraudulent.

The df.info() function in Python, when applied to a DataFrame 'df', provides a concise summary of the DataFrame's structure. It displays the number of non-null entries, data types of each column, memory usage, and additional information like the count of non-null values, facilitating quick data assessment and identification of missing values or data types. This method is useful for an initial data exploration and understanding the dataset's characteristics.

```
[ ]  df.info()

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

df.isnull().sum(), a result of "no null values" indicates that there are no missing (null) values in the DataFrame. This is a positive outcome, suggesting that the dataset is complete with no missing data, which is typically preferred for analysis and modeling.

```
[ ]  df.isnull().sum()#no null values

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

The df.corr() function in Python, when applied to a DataFrame 'df', computes the pairwise correlation between numeric columns, providing a correlation matrix. This matrix offers insights into the strength and direction of linear relationships between variables, with values ranging from -1 to 1, where -1 represents a strong negative correlation, 1 indicates a strong positive correlation, and 0 signifies no linear correlation. It's a valuable tool for understanding associations between data features.

```
[ ] #correlation
    df.corr()
```

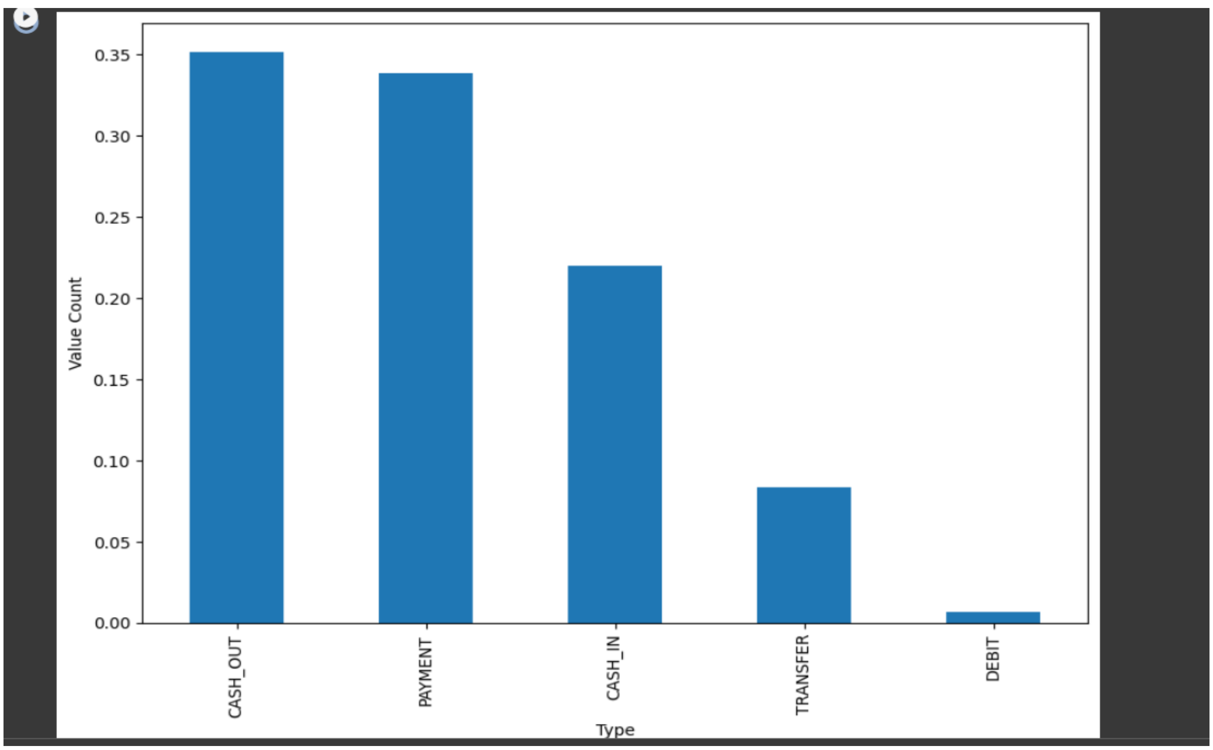
```
<ipython-input-9-a46c601d5826>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will defau
  df.corr()
```

	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

Univariate Analysis

Univariate analysis examines one variable's characteristics and distribution.

```
#univariate Analysis

import matplotlib.pyplot as plt

fig = plt.figure(figsize=(10, 7))
df['type'].value_counts(normalize=True).plot(kind='bar')
plt.xlabel("Type")
plt.ylabel("Value Count")
plt.show()
```

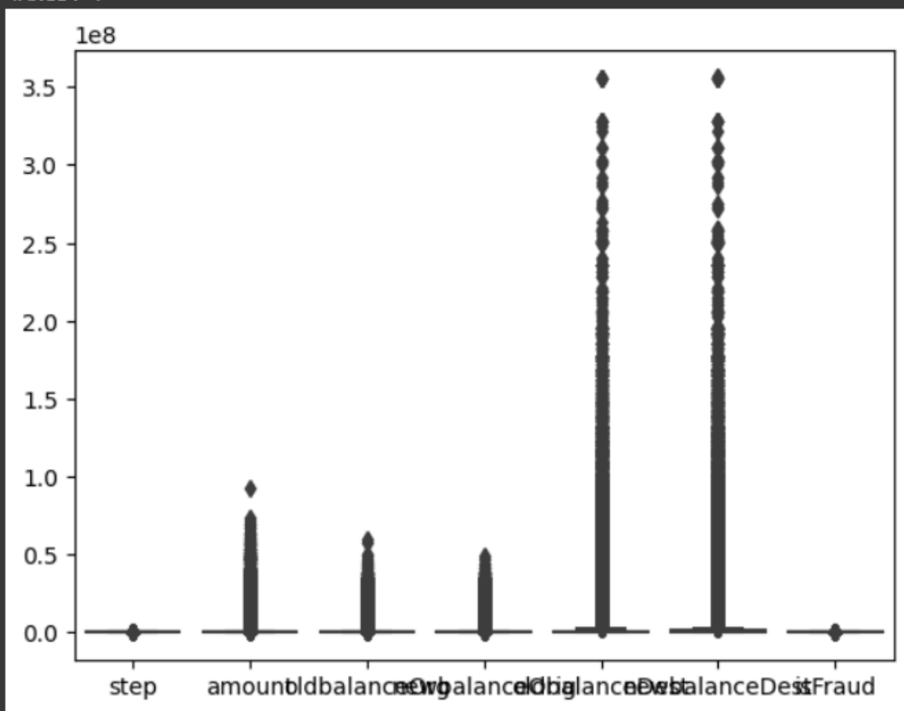


Boxplot

A boxplot is a graphical summary of a variable's distribution, showing its median, quartiles, and potential outliers.

```
sns.boxplot(df) #UNIVARIATE AS WE ARE CONSIDERING ONE PARAMETER AT A TIME
```

<Axes: >

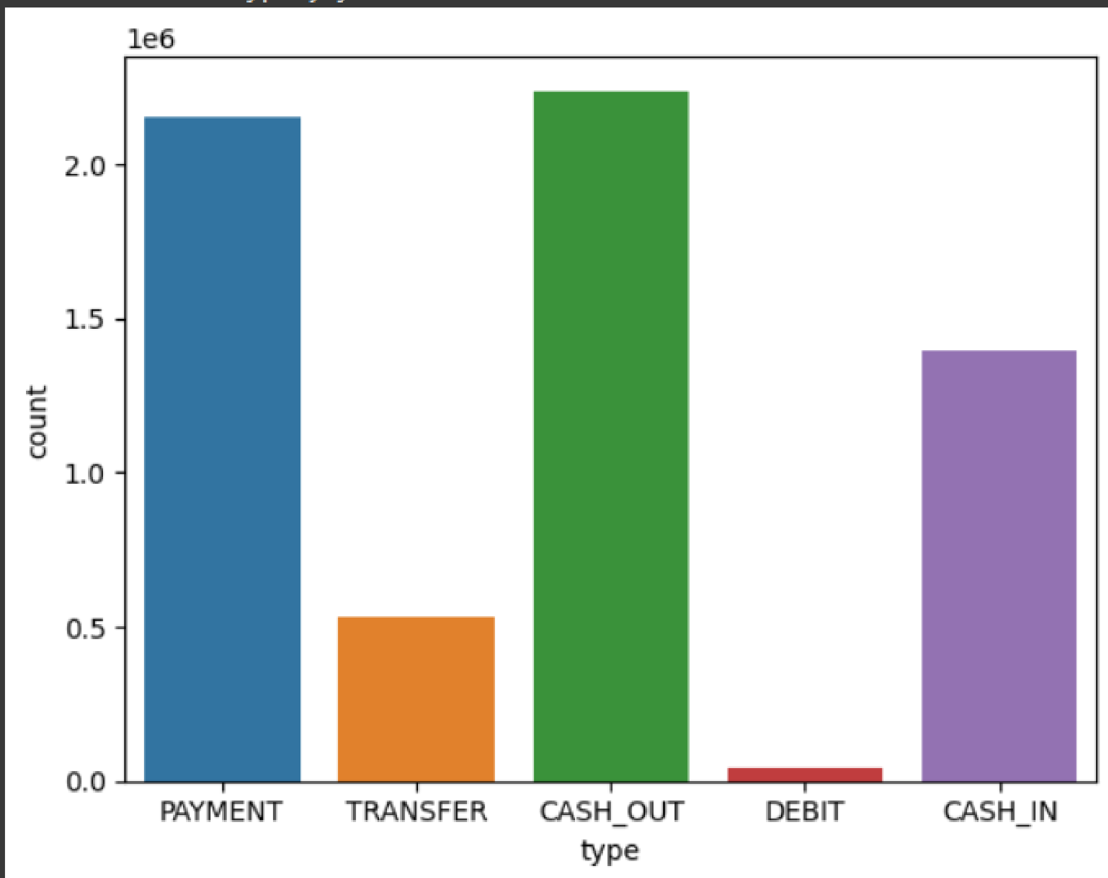


Countplot

A countplot is a type of bar plot that displays the frequency of categorical data in a dataset.

```
[ ] #type
    sns.countplot(data=df,x="type")
```

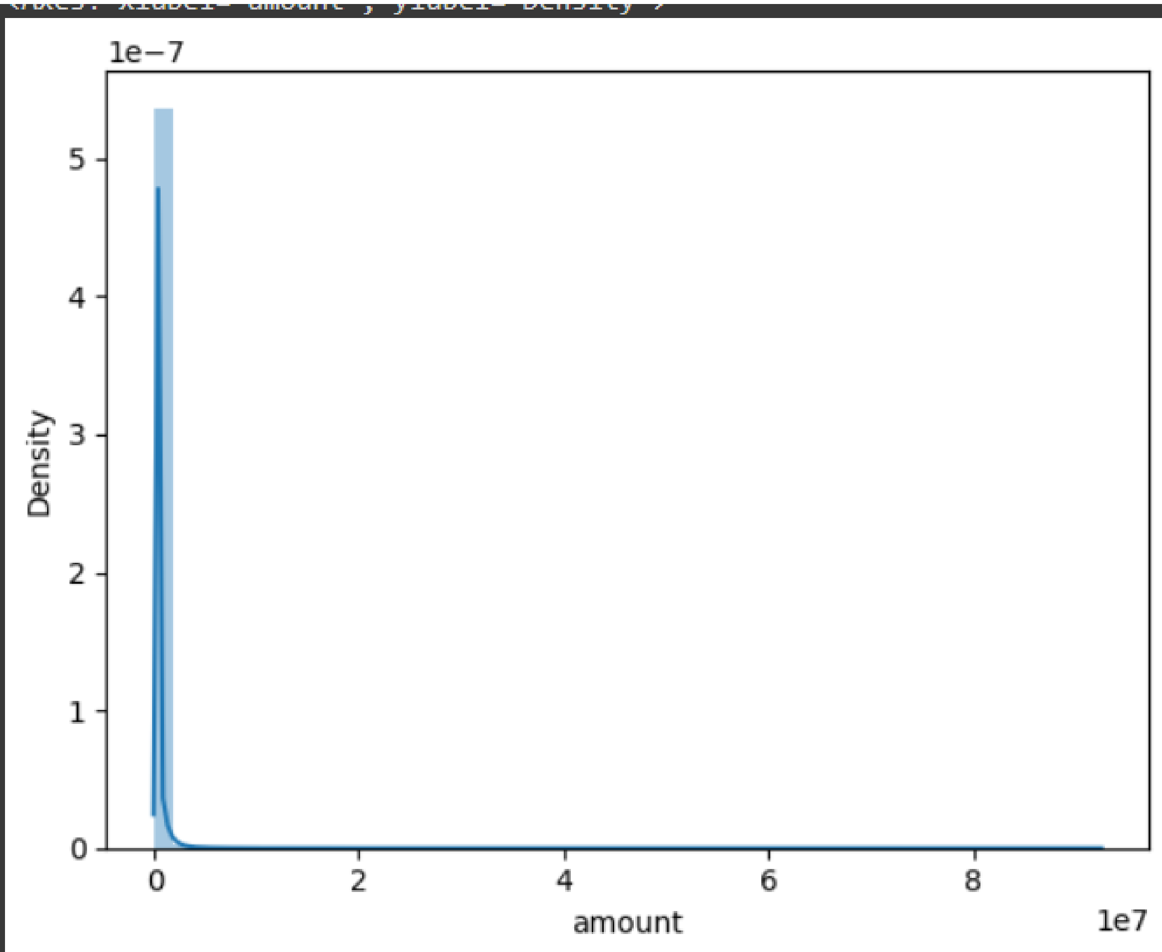
```
<Axes: xlabel='type', ylabel='count'>
```



Displot

A distplot, short for distribution plot, is a data visualization in Python often created using the Seaborn library. It combines a histogram with a kernel density estimate to provide an overview of the data's distribution

```python
sns.distplot(df.amount)
```

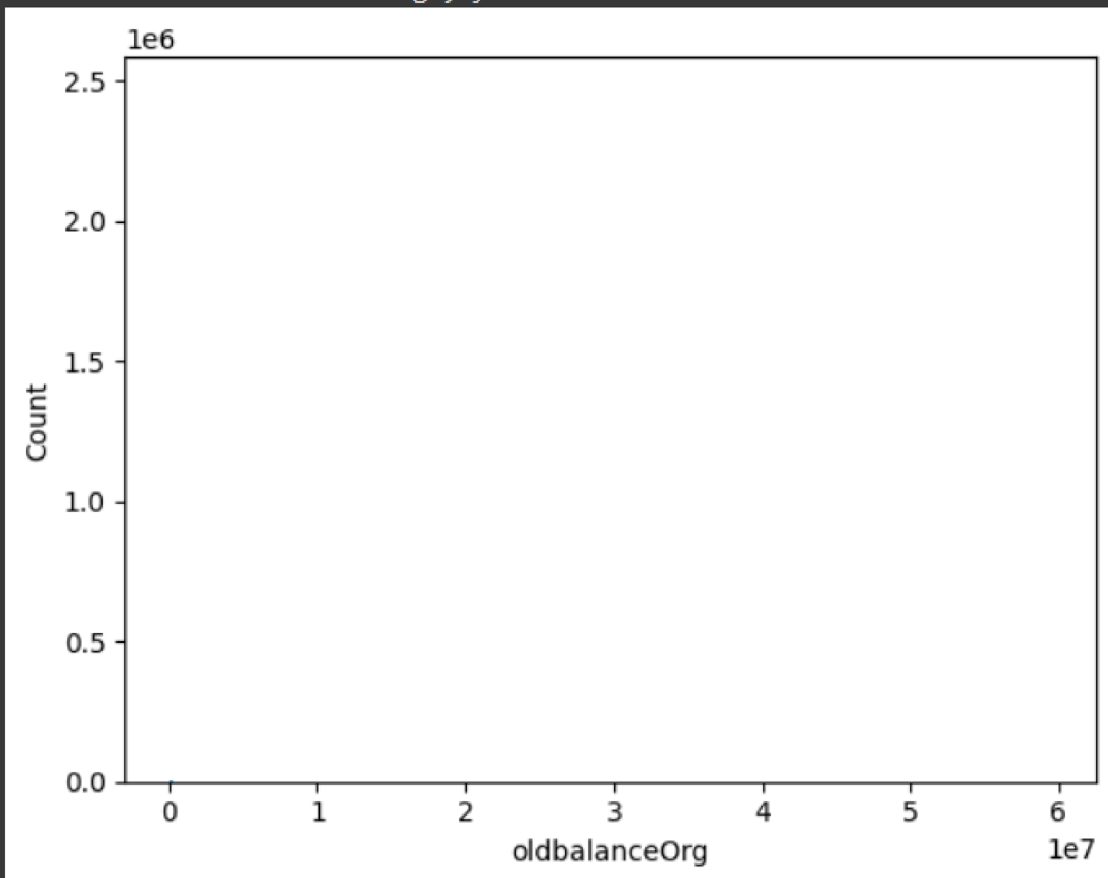


Hisplot

A histplot is a graphical representation that displays the distribution of a single variable through a histogram.

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

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

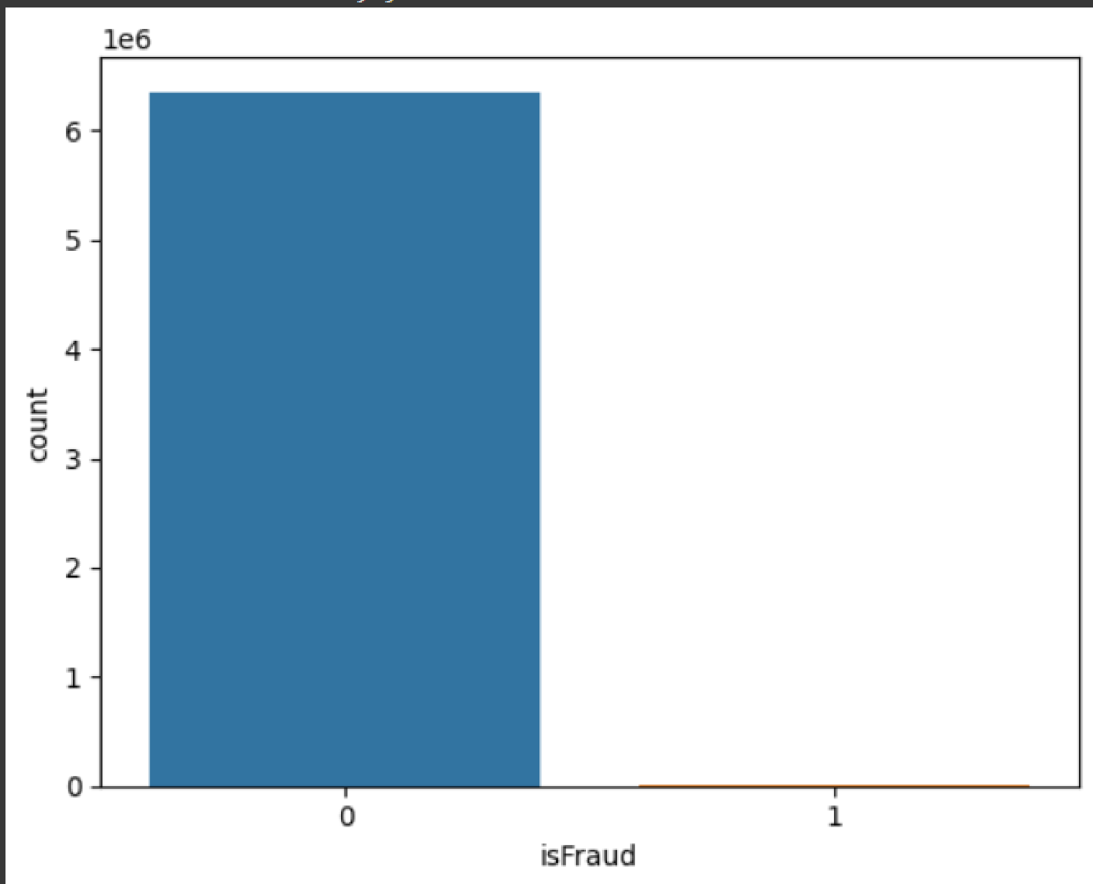


Countplot

A countplot is a type of bar plot that displays the frequency of categorical data in a dataset.

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

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



Counting the number of isFraud

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

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

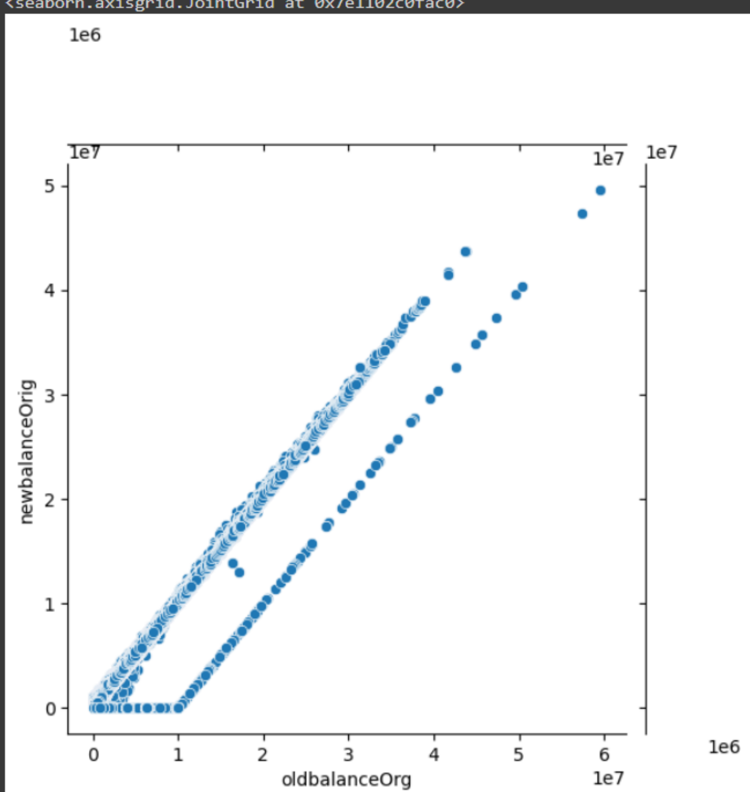
```
[ ] df.loc[df["isFraud"]==0,"isFraud"] = "is not Fraud"
    df.loc[df["isFraud"]==1,"isFraud"] = "is Fraud"
```

Multivariate Analysis

Multivariate analysis is a statistical technique used to analyze and understand relationships among multiple variables simultaneously, helping to identify patterns and correlations within complex data sets. It typically involves methods such as regression analysis, principal component analysis, or factor analysis to explore these relationships.

```
sns.jointplot(x='oldbalanceOrg',y='newbalanceOrig',data=df)
```

<seaborn.axisgrid.JointGrid at 0x7e1102c0fac0>

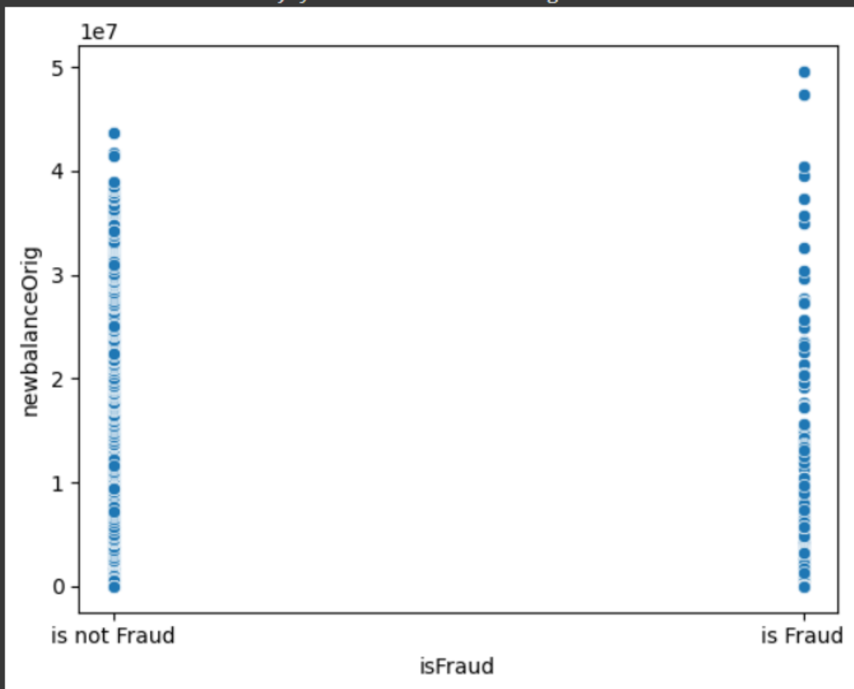


Scatterplot

A scatterplot is a graph that displays individual data points as dots to visualize the relationship between two continuous variables.

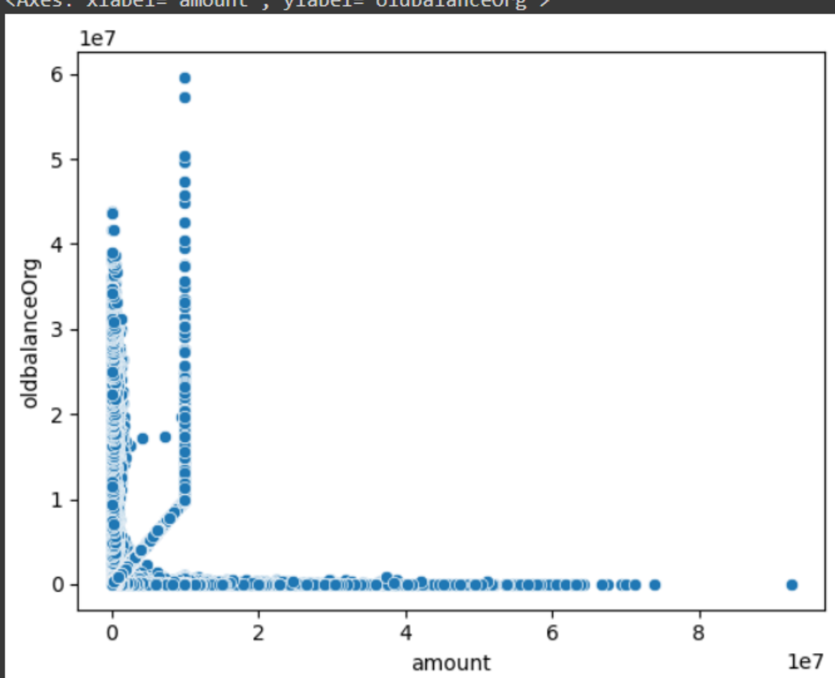
```
sns.scatterplot(x=df.isFraud,y=df.newbalanceOrig)
```

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




```
sns.scatterplot(x=df.amount,y=df.oldbalanceOrg)
```

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

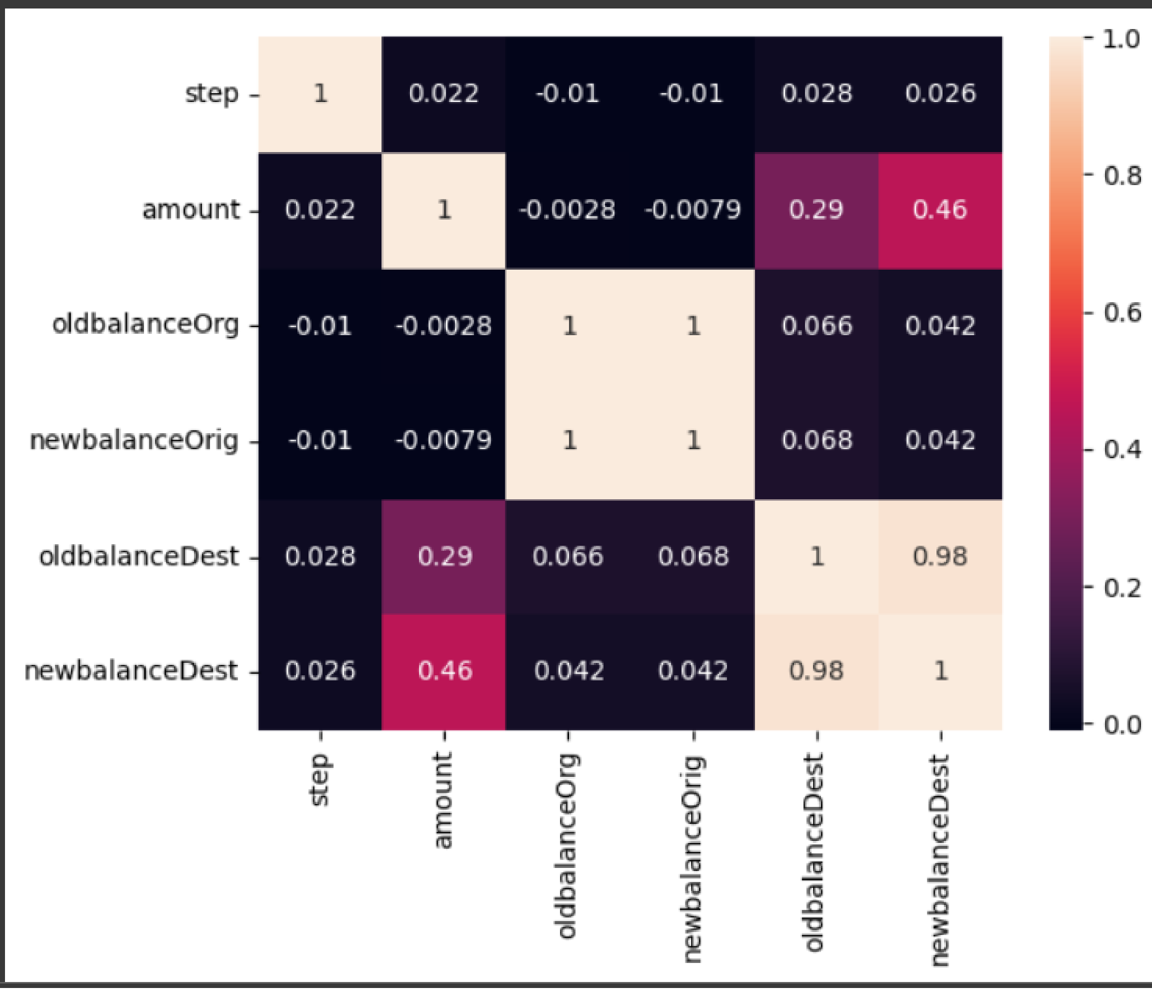


Heatmap

A heatmap is a graphical representation that uses color to depict the relationships and values of a matrix or two-dimensional data set.

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

```
<ipython-input-22-8df7bcac526d>:1: FutureWarning: The default value of numeric_on
  sns.heatmap(df.corr(),annot=True)
<Axes: >
```



Data Preprocessing

The code removes the 'nameOrig' and 'nameDest' columns from the DataFrame 'df' by specifying the column names and the 'axis' parameter set to 1 (indicating columns). The 'inplace=True' argument modifies the DataFrame directly. After this operation, the 'df' DataFrame will have these columns removed from its structure.

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

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

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

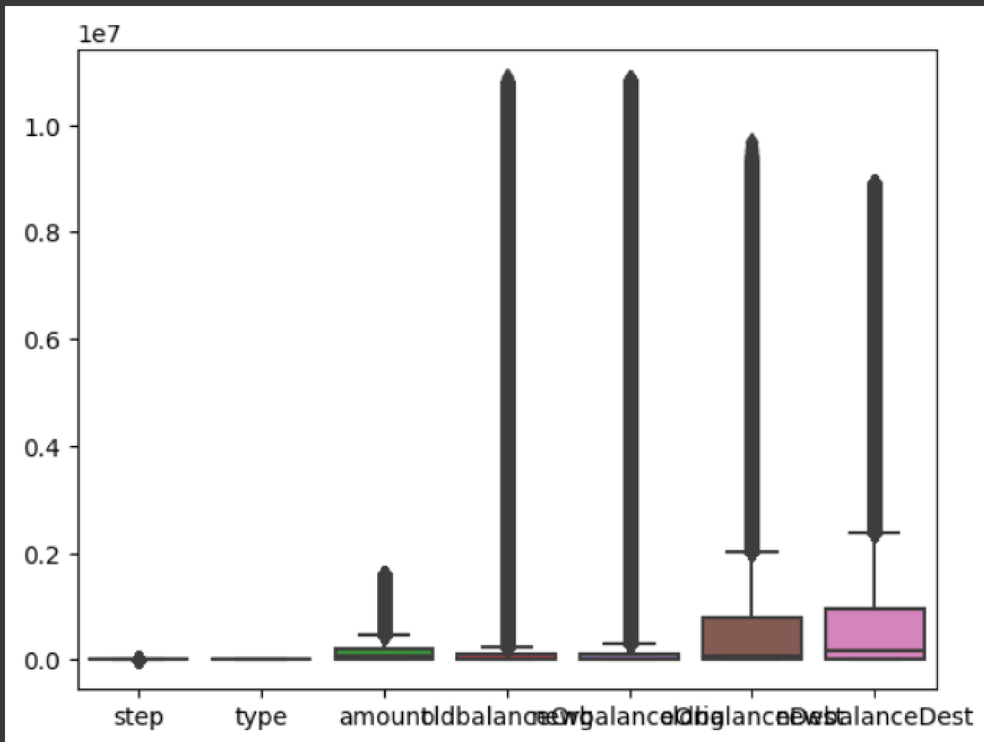
	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
0	1	3	9839.64	170136.0	160296.36	0.0	0.0	is not Fraud
1	1	3	1864.28	21249.0	19384.72	0.0	0.0	is not Fraud
2	1	4	181.00	181.0	0.00	0.0	0.0	is Fraud
3	1	1	181.00	181.0	0.00	21182.0	0.0	is Fraud
4	1	3	11668.14	41554.0	29885.86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

Removal of Outliers (Using Percentile Method)

The removal of outliers by the percentile method is a technique used to eliminate extreme data points in a dataset. To implement this method, first, you calculate the lower and upper percentile boundaries, often using values like the 5th and 95th percentiles. These boundaries help define the range within which the majority of the data points are expected to fall. Next, you identify data points that fall below the lower percentile boundary or exceed the upper percentile boundary, marking them as outliers. Finally, these identified outliers are removed from the dataset. This process results in a more robust dataset for analysis, as it reduces the impact of extreme values on statistical analyses and visualizations.

```
sns.boxplot(df)
```

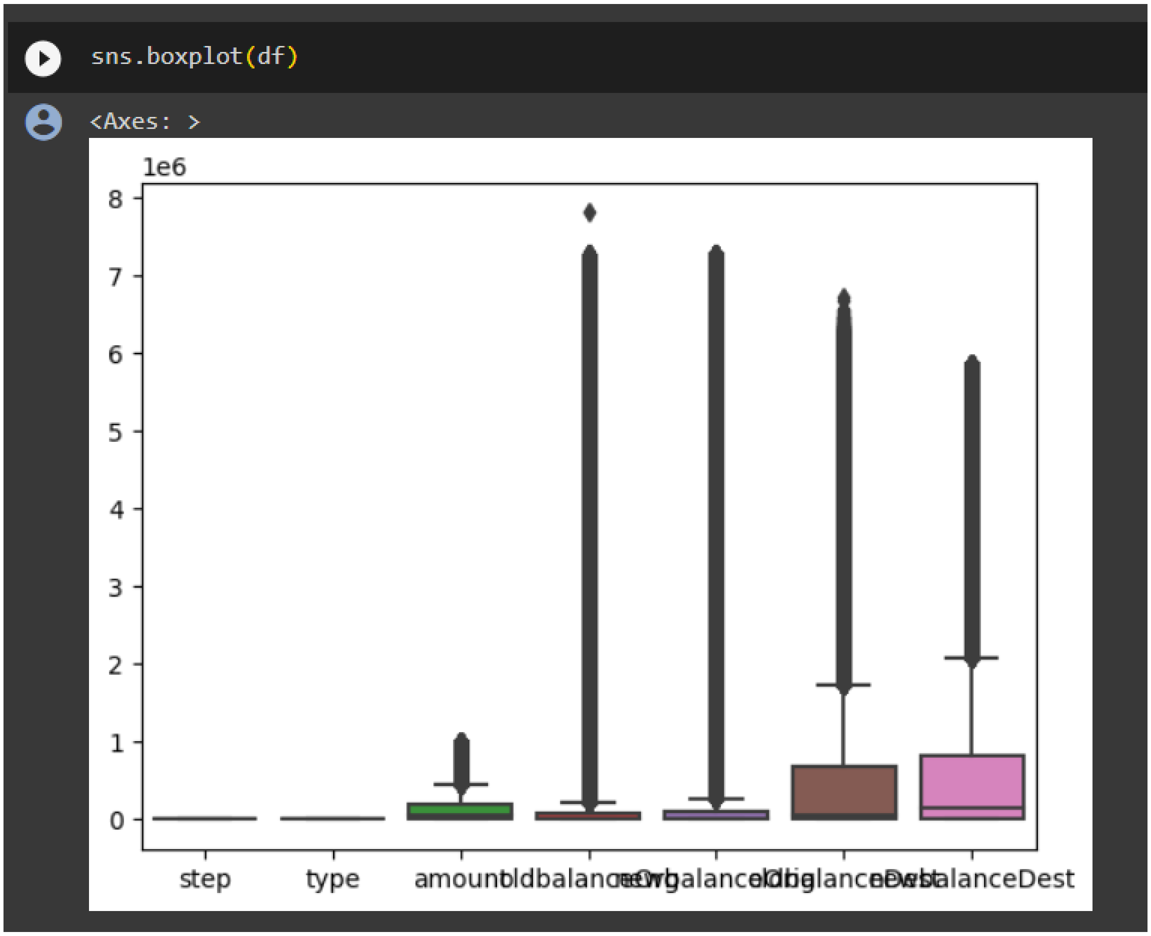
```
<Axes: >
```



This code first identifies numeric columns (excluding 'isFraud') and removes extreme outliers by calculating the 99th percentile for each column. It filters the DataFrame to keep only values below this threshold, making the data more suitable for subsequent analysis or modeling. Additionally, this data preprocessing step helps improve the robustness of the dataset and ensures that extreme values do not unduly influence the analysis. It's a common practice to enhance the quality of numeric data before conducting further statistical or machine learning tasks.

```
[ ]  num=[var for var in df.columns if df[var].dtype!='O' and var!='isFraud']
```

```
for x in num:
  p99=df[x].quantile(0.99)
  df=df[df[x]<=p99]
```



LABEL ENCODING

Label encoding is a fundamental technique in machine learning for converting categorical data into numerical format, a necessary step for many algorithms. In Python, it's commonly implemented using libraries like Scikit-Learn. First, you import the LabelEncoder class from Scikit-Learn. Then, you create an instance of the LabelEncoder class, which is used to transform the categorical data into numeric values. Finally, you apply the label encoder to a specific column in your DataFrame, effectively converting the categorical labels into corresponding numeric values. This numeric representation enables machine learning algorithms to work with the data, making label encoding a crucial preprocessing step in data analysis and modeling tasks.

LABEL ENCODING

```
[ ] le=LabelEncoder()
    df["type"]=le.fit_transform(df["type"])
```

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

    3    2110214
    1    2070100
    0    1066610
    4     361700
    2      38272
    Name: type, dtype: int64
```

These two lines of code segment the dataset into independent variables (X) and the dependent variable (y). "X" includes all columns except "isFraud," while "y" contains only the "isFraud" column. This division prepares the data for supervised machine learning, where "X" represents the features used for prediction, and "y" is the target variable to predict.

Train Test split

```
[ ] # Dividing the dataset into dependent and independent y and x respectively
    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    is not Fraud
    1    is not Fraud
    2        is Fraud
    3        is Fraud
    4    is not Fraud
    Name: isFraud, dtype: object
```

Train test split

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

A train-test split is a common technique used in machine learning to evaluate the performance of a model. It involves dividing your dataset into two separate subsets: one for training the model and another for testing the model's performance.

To perform a train-test split:

- You start with your dataset, which typically includes both the features (input data) and the corresponding target values (output or labels).
- You specify a ratio, often referred to as the "test size," which determines the proportion of your data to be set aside for testing. Common choices include 70/30, 80/20, or 90/10, with the training set being the larger portion.
- The data is then randomly divided into two subsets: the training set and the testing set. The training set is used to train your machine learning model, while the testing set is used to assess the model's performance by making predictions and comparing them to the true values.

The train-test split helps you gauge how well your model generalizes to new, unseen data. It is a fundamental step in model evaluation and validation, allowing you to check for overfitting (when a model performs well on the training data but poorly on new data) and to estimate the model's predictive accuracy on real-world data.

Random Forest Classifier

A Random Forest Classifier is an ensemble machine learning algorithm that combines multiple decision trees to make more accurate predictions. It's used for both classification and regression tasks. It improves prediction accuracy, handles overfitting, and provides feature importance rankings by averaging the predictions of multiple decision trees. Random Forest is a versatile and powerful algorithm widely used in various applications, including image classification, medical diagnosis, and financial forecasting.

```
[ ]  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
```

```
0.9997004661547614
```

```
[ ]  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	807	336
is not Fraud	23	1197363

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

```
              precision    recall  f1-score   support

    is Fraud       0.97      0.71      0.82      1143
is not Fraud       1.00      1.00      1.00   1197386

    accuracy                           1.00   1198529
   macro avg       0.99      0.85      0.91   1198529
weighted avg       1.00      1.00      1.00   1198529
```

Decision Tree Classifier

A Decision Tree Classifier is a machine learning algorithm that creates a tree-like model to make predictions. It's used for classification tasks, where it splits the data into subsets based on feature attributes, ultimately assigning labels to instances. Decision trees are interpretable and easy to visualize, making them useful for understanding the decision-making process in a model. They can handle both categorical and numerical data, and by recursively splitting the data based on the most informative features, decision trees are capable of capturing complex decision boundaries.

```
[ ] 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

    0.9996912882374978
```

```
[ ] 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)
```

col_0	is Fraud	is not Fraud
isFraud		
is Fraud	1193	245
is not Fraud	204	1496519

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

              precision    recall  f1-score   support

    is Fraud       0.85      0.83      0.84      1438
is not Fraud       1.00      1.00      1.00   1496723

    accuracy                           1.00   1498161
   macro avg       0.93      0.91      0.92   1498161
weighted avg       1.00      1.00      1.00   1498161
```

SVM

Support Vector Machine (SVM) is a powerful supervised machine learning algorithm used for both classification and regression tasks. It finds the optimal hyperplane that maximizes the margin between different classes in the data, making it effective for separating data points in high-dimensional spaces. SVM can handle linear and non-linear problems through techniques like kernel functions, and it's known for its ability to handle complex decision boundaries while avoiding overfitting. SVMs have applications in image recognition, text classification, and more.

```
[ ] 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
```

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

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

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

+ Code + Text

```
[ ] df.columns

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

```
[ ] la= LabelEncoder()
    y_train1 = la.fit_transform(y_train)
```

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

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

```
[ ] y_test1

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

```
[ ] y_train1

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

XgBoost Classifier

XGBoost, short for "Extreme Gradient Boosting," is a popular and highly effective ensemble machine learning algorithm primarily used for classification and regression tasks. It enhances predictive accuracy by combining the predictions of multiple decision trees. XGBoost uses gradient boosting, which optimizes model performance by iteratively adding decision trees to correct errors made by the previous trees. It's known for its speed, scalability, and the ability to handle complex relationships in the data, making it a top choice in data science competitions and real-world applications like customer churn prediction and anomaly detection.

```
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
```

```
0.9997904401680998
```

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

```
0.9998602933377643
```

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

col_0	0	1
row_0		
0	642	172
1	32	972623

```
[ ] print(classification_report(y_test1,y_test_predict5))
```

```
              precision    recall  f1-score   support

           0       0.95      0.79      0.86       814
           1       1.00      1.00      1.00    972655

    accuracy                           1.00    973469
   macro avg       0.98      0.89      0.93    973469
weighted avg       1.00      1.00      1.00    973469
```

Result

We successfully implemented multiple different Machine Learning Algorithms on the given dataset to determine which approach to use for our product. We implemented Random Forest, Decision Trees, SVM classifier, XgBoost classifier and obtained accuracies of 99.97, 99.96, 80 and 99.97 respectively.

*Hence, we concluded that the model which is best fit for the given dataset is **99.97** which is given by **XgBoost** as the recall for 0 and 1 is high as compared to other models*

Application Building

In this section of the project, we will create a web application that interfaces with the machine learning model we previously developed. This application will include a user interface (UI) where users can input values for making predictions. These input values will be forwarded to the saved machine learning model, and the predictions generated will be displayed on the UI.

The tasks involved in this section are as follows:

1. Building HTML Pages: We will design and create the web pages that make up the user interface. These pages will include forms or input fields where users can provide the necessary data for predictions.
2. Building Server-Side Script: We will develop the server-side logic, which is responsible for handling user input, passing it to the machine learning model, obtaining predictions, and then presenting the results back to the user on the UI.

This integration of machine learning into a web application allows users to interact with the model and receive predictions in a user-friendly manner, making it practical for various applications, such as recommendation systems, fraud detection, or any scenario where predictive models need to be put into practical use.

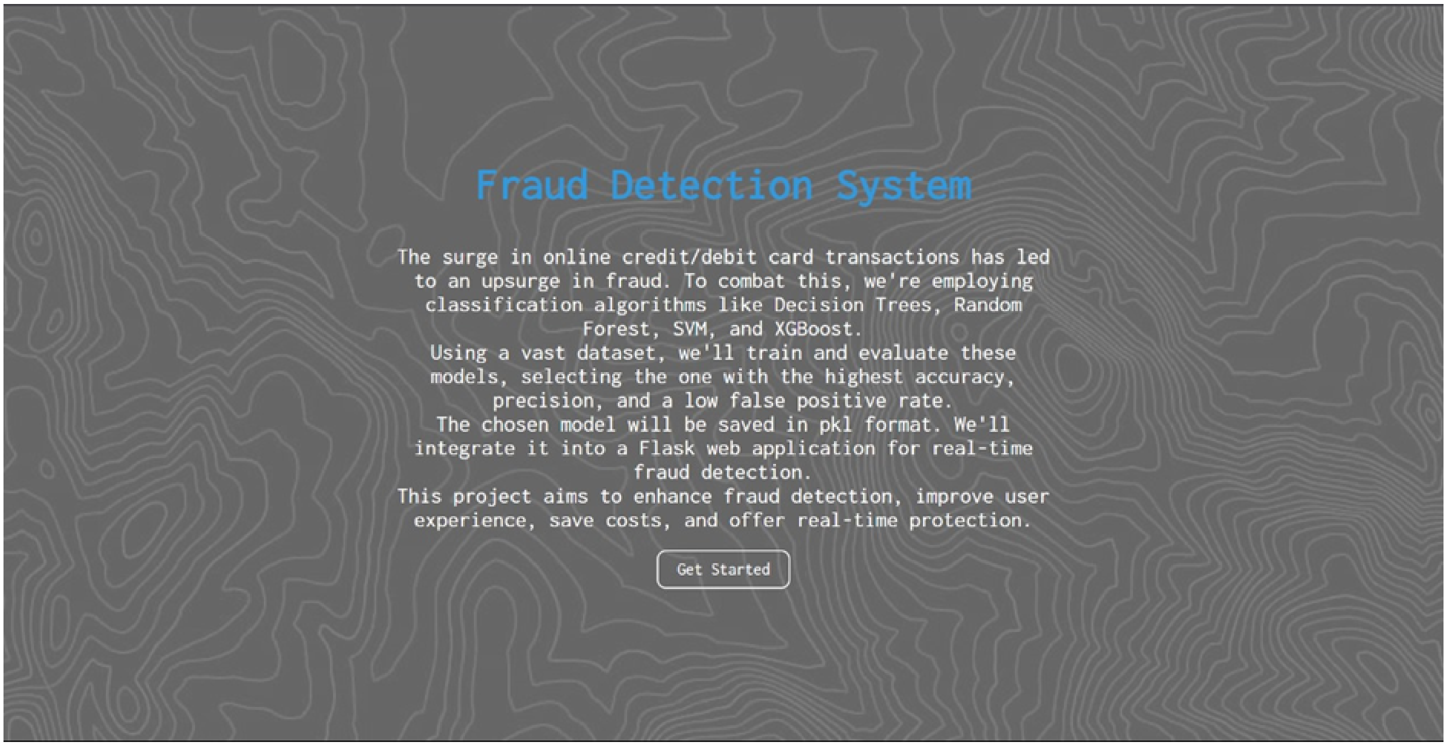
Flask File

This code sets up a Flask web application for a machine learning model. It loads a pre-trained model from a saved pickle file, provides routes for different pages (about, home, predict), and handles form submission. When a user submits data on the 'predict' page, it passes the input to the model and displays the prediction on the 'result' page. The application runs in debug mode when executed as the main program, enabling web development and testing.

app.py

```python
from flask import Flask, render_template, request
import pickle
import numpy as np
import pandas as pd
model=pickle.load(open('model.pkl','rb'))
app = Flask(__name__)

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

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

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

@app.route('/pred',methods=['POST','GET'])
def pred():
    x=[[obj for obj in request.form.values()]]
    x=np.array(x)
    output=model.predict(x)
    return render_template ("result.html",pred =str(output[0]) )

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

Our User Interface

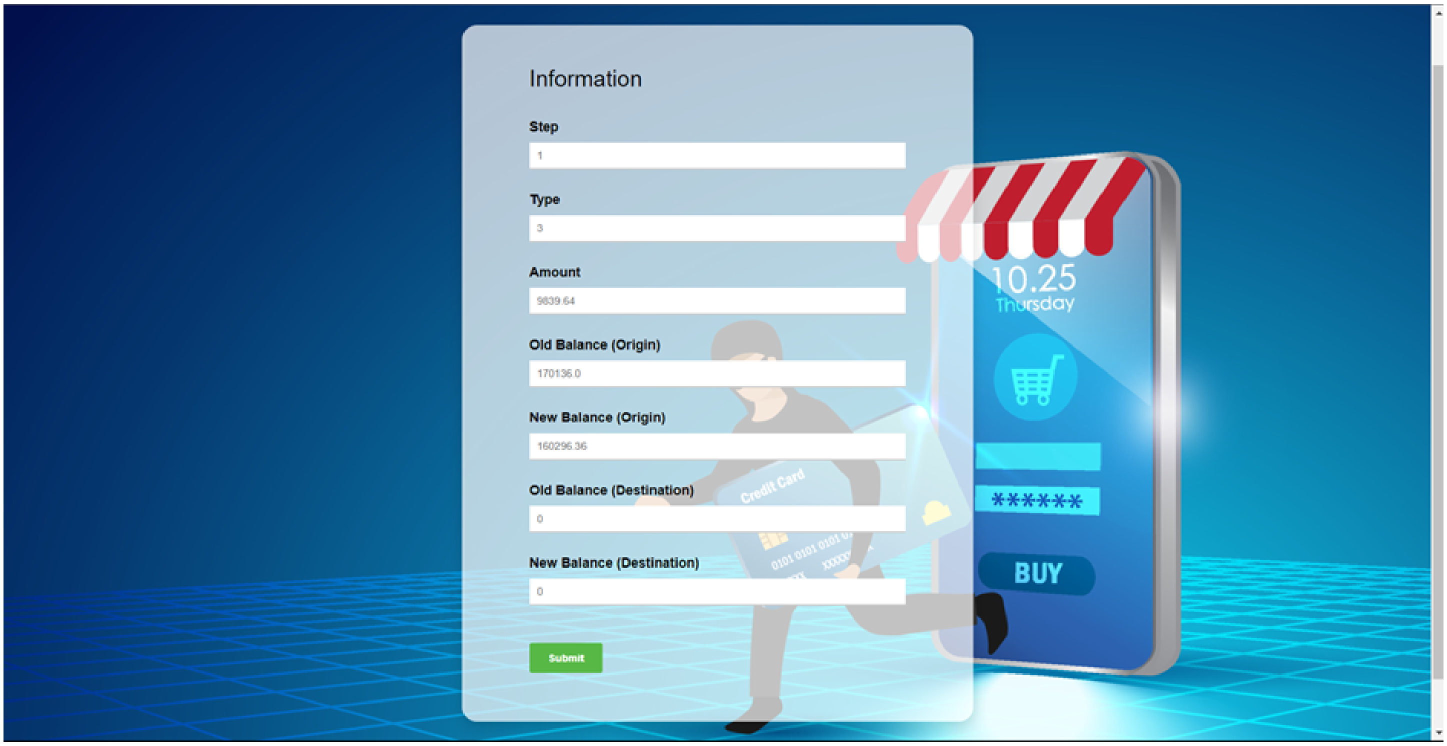


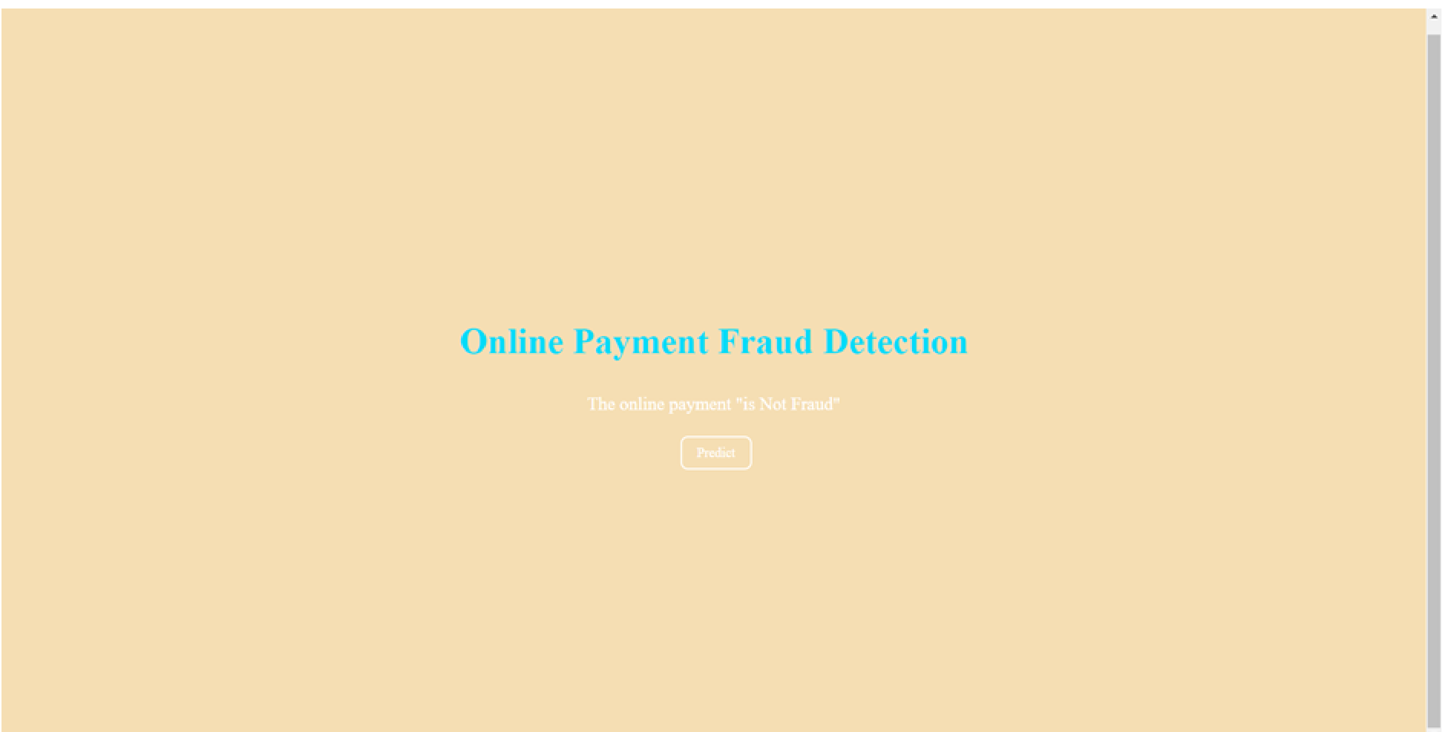
Form Page (User Input):

Form: Display a form with fields for users to input information related to a transaction for fraud detection:

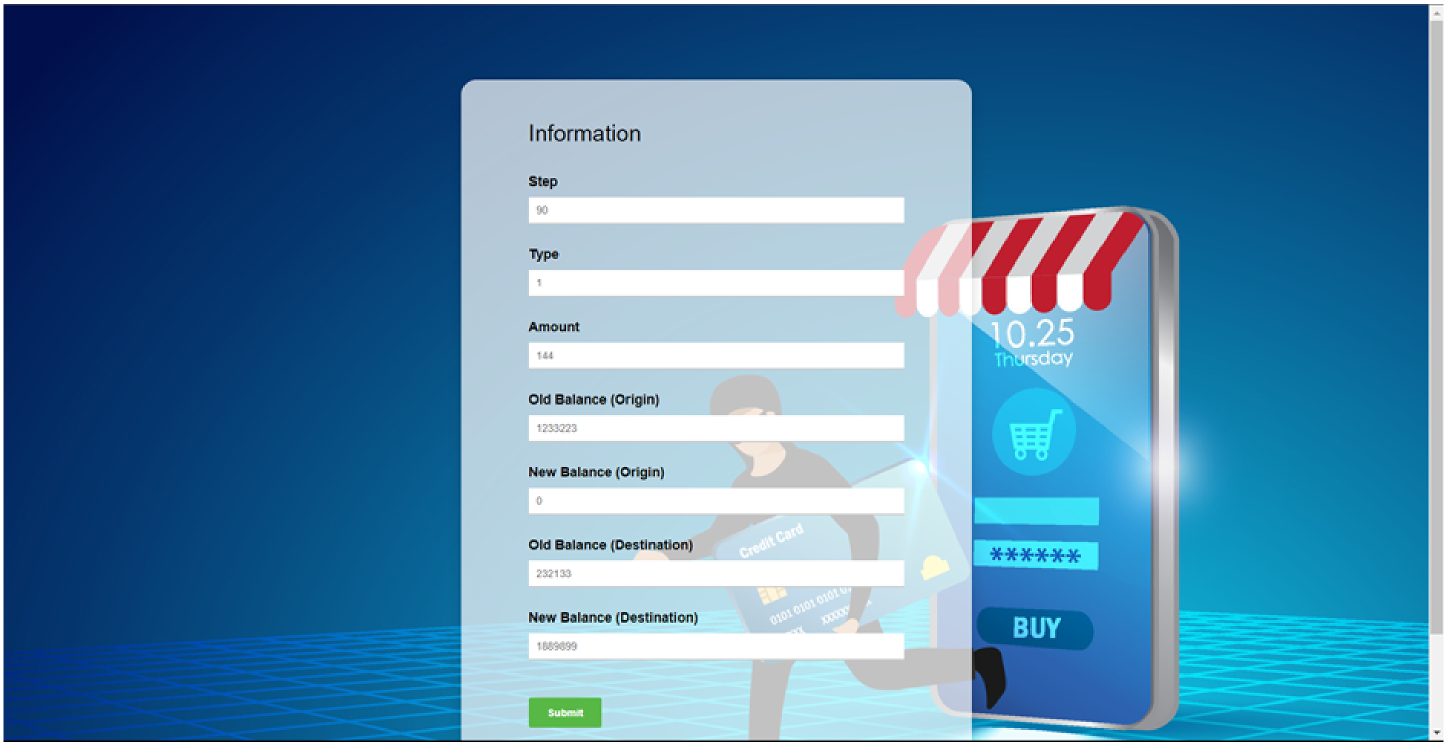
1. 'step': [Input Field]
2. 'type': [Input Field]
3. 'amount': [Input Field]
4. 'oldbalanceOrg': [Input Field]
5. 'newbalanceOrig': [Input Field]
6. 'oldbalanceDest': [Input Field]
7. 'newbalanceDest': [Input Field]
8. Submit Button: Provide a "Submit" button that allows the user to submit the transaction details for fraud detection.

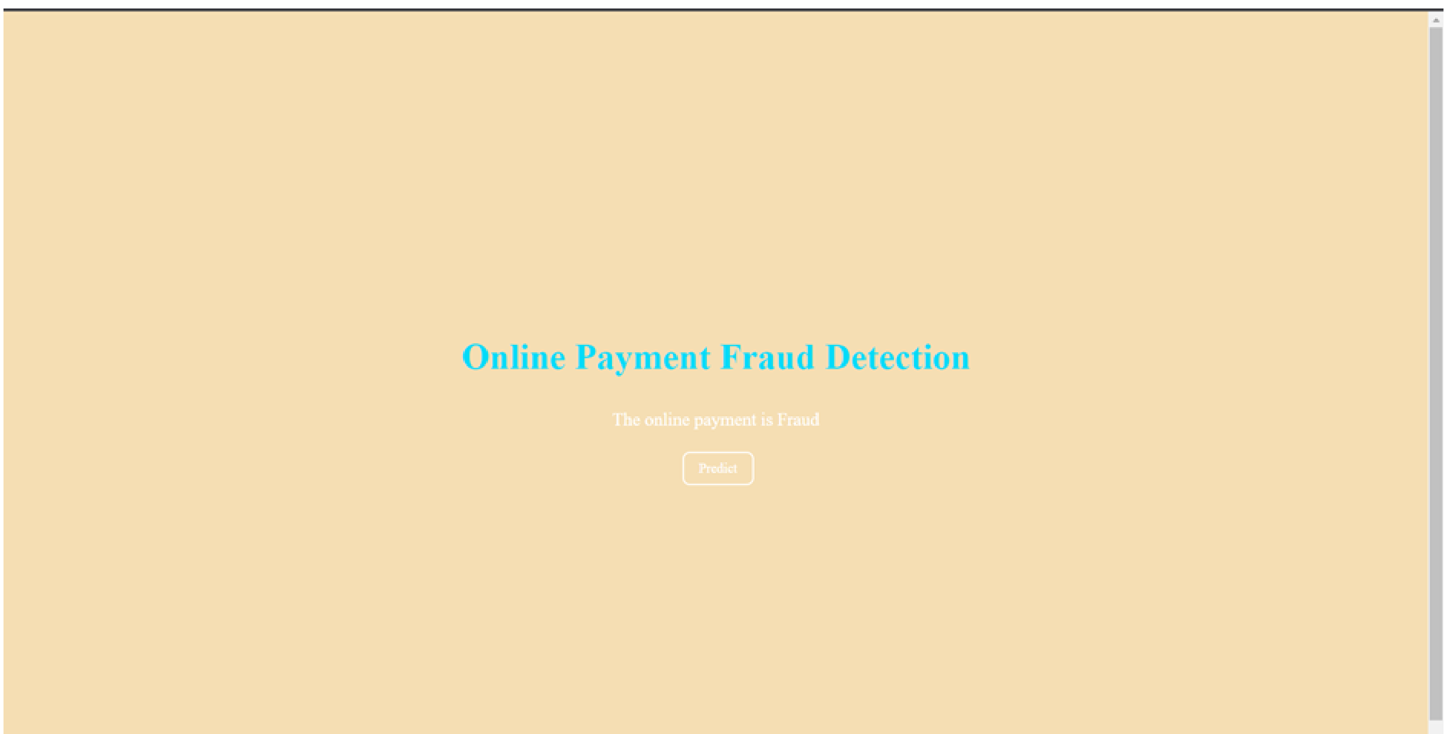
Result Display: After submitting the form, display the result on the same page. It may indicate whether the provided transaction information is classified as potential fraud or not. For example:
"Result: No Fraud Detected" or "Result: Possible Fraud Alert"





When the system classifies a transaction as "Not a Fraud," it means that the provided transaction details do not exhibit suspicious or fraudulent behavior.
Users can be reassured that the transaction appears legitimate, and they can proceed with confidence.





When the system classifies a transaction as "Fraud," it indicates that the provided transaction details raise suspicions of fraudulent activity.
Users should be alerted to the potential risk and advised to take immediate action to secure their accounts and prevent further damage.

Thank You

Team - 592416

Team Members

1. **Aryan Sharma (21BBS0106)**
2. **Arjun Malhotra (21BBS0110)**
3. **Pranav Singh Mahara (21BBS0188)**
4. **Kanishak Khanna (21BCI0147)**