Online Payments Fraud Detection using Machine Learning

1. Project Description:

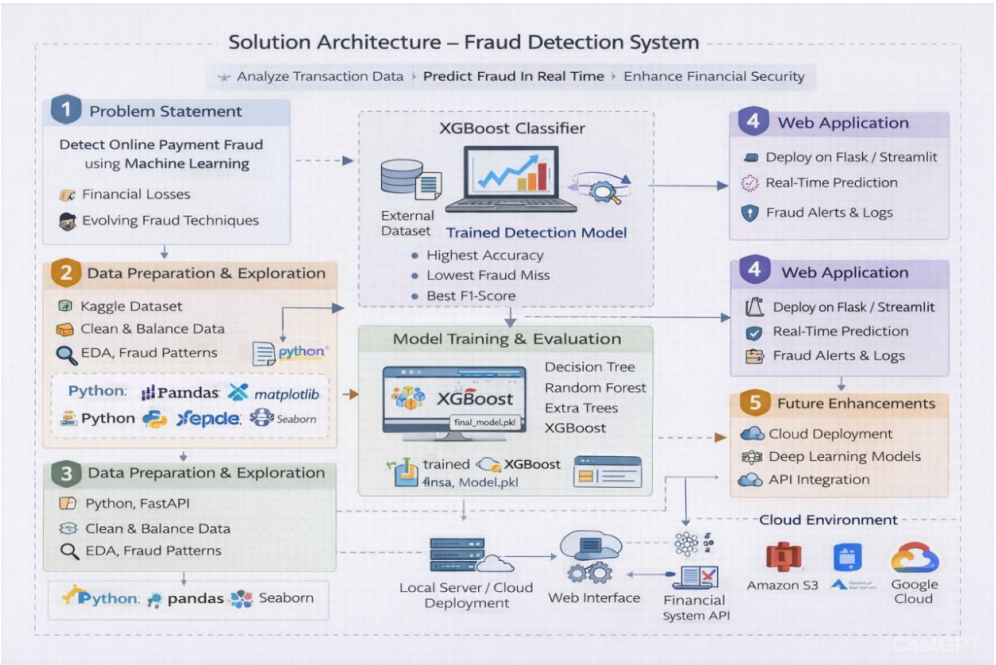
Online Payments Fraud Detection using Machine Learning is a proactive approach to identify and prevent fraudulent activities during online transactions. By leveraging historical transaction data, customer behavior patterns, and machine learning algorithms, this project aims to detect potential fraud in real time, ensuring secure and trustworthy online payment experiences for users and businesses alike.

Scenario 1: Real-time Fraud Monitoring The system continuously monitors online payment transactions in real time. By analyzing transaction features such as transaction amount, location, device information, and user behavior, it can flag suspicious transactions for further investigation, preventing fraudulent activities before they occur.

Scenario 2: Fraudulent Account Detection Machine learning models can detect patterns indicative of fraudulent accounts or activities. By analyzing user behavior over time, such as unusual login times, multiple failed login attempts, or sudden changes in spending patterns, the system can identify and block potentially fraudulent accounts, protecting legitimate users and businesses.

Scenario 3: Adaptive Fraud Prevention The system adapts and improves its fraud detection capabilities over time. By continuously learning from new data and adjusting its algorithms, it can stay ahead of evolving fraud techniques and trends, providing ongoing protection against online payment fraud for businesses and their customers.

Technical Architecture:



2. Pre requisites:

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

- **Python packages:**
  - Open anaconda prompt as administrator o Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter.
  - Type "pip install scikit-learn" and click enter.
  - Type "pip install matplotlib" and click enter.
  - Type "pip install scipy" and click enter.
  - Type "pip install pickle-mixin" and click enter.
  - Type "pip install seaborn" and click enter.
  - Type "pip install Flask" and click enter.

3. 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 visualization concepts.

4. Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed 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
- Visualizing and analyzing data
  - Univariate analysis
  - Bivariate analysis
  - Multivariate analysis
  - Descriptive analysis
- Data pre-processing
  - Checking for null values
  - Handling outlier
  - Handling categorical data
  - Splitting data into train and test

- Model building
  - Import the model building libraries
  - Initializing the model
  - Training and testing the model
  - Evaluating performance of model
  - Save the model
- Application Building
  - Create an HTML file
  - Build python code
  - Run the application

4.1. Data Collection:

The data used for this project was obtained from a publicly available dataset on Kaggle. The dataset file used is: PS_20174392719_1491204439457_log.csv

This dataset contains real-world simulated online payment transaction records designed specifically for fraud detection research.

Dataset Link: https://www.kaggle.com/datasets/ealaxi/paysim1

4.2. Visualizing and Analyzing Data

Before performing data analysis and model development, the required Python libraries were imported. These libraries help in data manipulation, visualization, machine learning modeling, and evaluation.

Importing Libraries

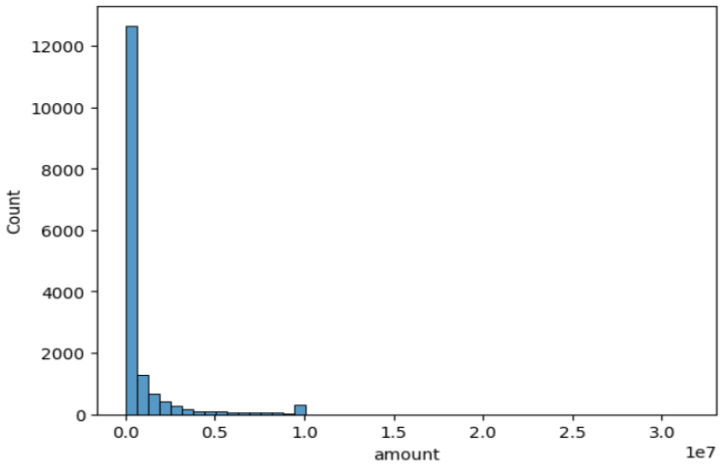
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

```

Univariate Analysis:

Univariate analysis involves analyzing one variable at a time to understand its distribution, pattern, and characteristics. It helps in identifying outliers, skewness, and overall data behavior before applying machine learning models.

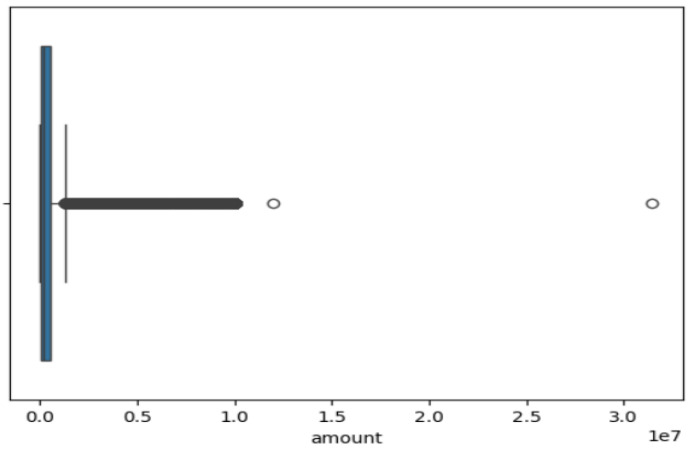
Histogram:

A histogram is a graphical representation used to display the distribution of numerical data by dividing it into bins or intervals. It helps in understanding how data values are spread, whether the distribution is normal or skewed, and if extreme values exist. In this project, histograms were used to analyze transaction amounts and balance-related features to observe variations and data concentration.



Boxplot:

A boxplot is a statistical visualization that represents data distribution using quartiles. It shows the median, interquartile range, and potential outliers. Boxplots are useful for detecting extreme values and understanding the spread of financial transactions. In this project, boxplots helped identify unusual transaction amounts that may indicate fraudulent activity.

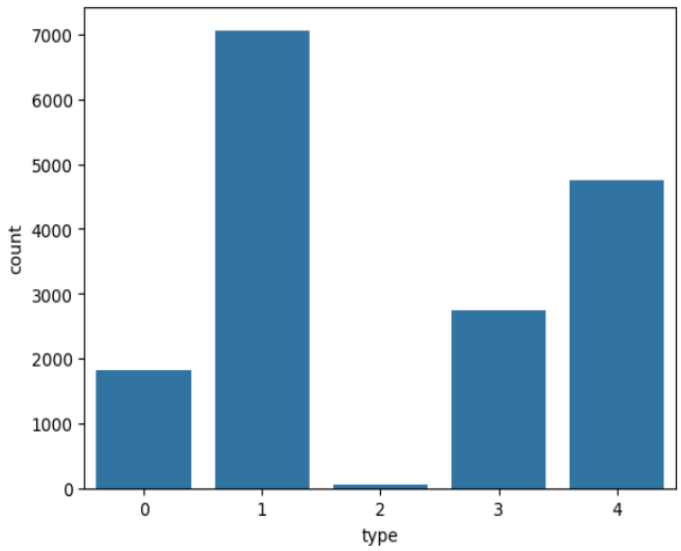


Bivariate Analysis:

Bivariate analysis examines the relationship between two variables to understand how they interact with each other. It is particularly useful in fraud detection to compare features with the target variable (isFraud). In this project, bivariate analysis helped identify patterns between transaction type, amount, balance changes, and fraudulent behavior.

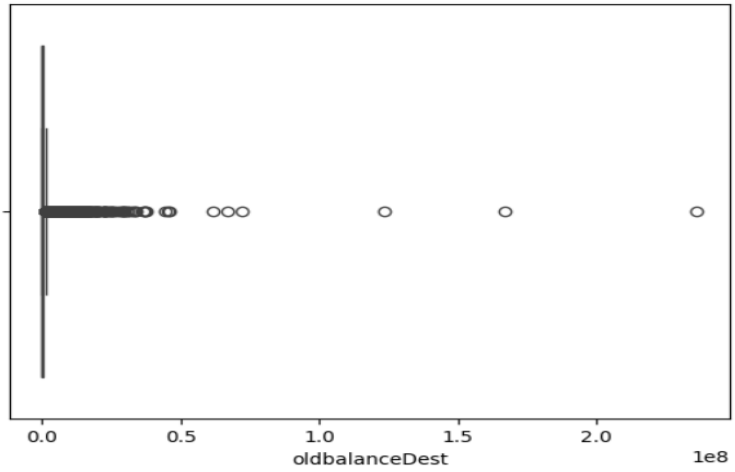
Countplot (Count vs Transaction Type)

The countplot of count vs transaction type was used to visualize the frequency of different types of transactions in the dataset. This graph shows how many transactions belong to each category, such as transfer, cash-out, payment, debit, or cash-in. It helps in understanding which transaction types are most common and identifying categories that may be more prone to fraudulent activity. In this project, the countplot provided insights into transaction distribution patterns and supported further fraud-related analysis based on transaction type.



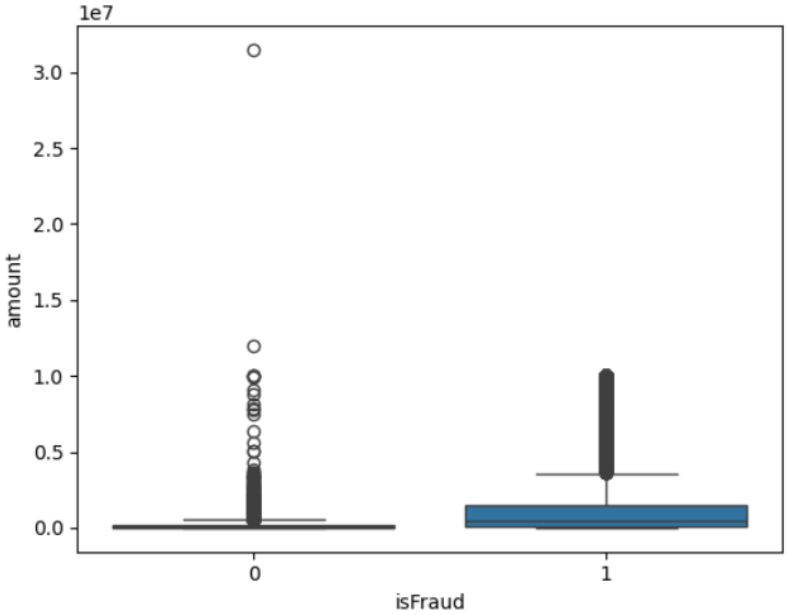
Boxplot (oldbalanceOrg vs isFraud)

The boxplot of oldbalanceOrg vs isFraud was used to analyze the sender's account balance before the transaction in relation to fraud occurrence. This visualization helped compare balance distributions between fraudulent and legitimate transactions. The analysis indicated that fraud transactions often show abnormal or inconsistent balance patterns. The boxplot also highlighted variations and extreme values in account balances, which are strong indicators in identifying suspicious activities.



Boxplot (Amount vs isFraud)

The boxplot of transaction amount vs isFraud was used to compare the distribution of transaction amounts between fraudulent and non-fraudulent transactions. This visualization helps in identifying differences in median values, spread, and outliers for both classes. From the analysis, it was observed that fraudulent transactions often involve unusual or extreme transaction amounts compared to legitimate transactions. The boxplot also helped in detecting outliers and understanding how transaction size influences fraud detection.



Descriptive Analysis:

Descriptive analysis summarizes the basic characteristics of the dataset using statistical measures such as mean, median, minimum, maximum, and standard deviation. It also includes checking data types and missing values. In this project, descriptive analysis provided an overview of the dataset structure and helped in understanding its overall properties before preprocessing and modeling.

```
1 df.describe()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
count	6.362620e+06	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	2.514687e-06
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06	3.590480e-02	1.585775e-03
min	1.000000e+00	0.000000e+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	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	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	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	1.000000e+00

4.3. Data Pre-processing

Data preprocessing is an essential step performed before model training to clean and prepare the dataset. It ensures that the data is accurate, consistent, and suitable for machine learning algorithms. In this project, several preprocessing techniques were applied to improve model performance and reliability.

Checking for Null Values

The dataset was checked for missing or null values using data inspection methods. It was observed that the dataset did not contain any missing values. This ensured data completeness and eliminated the need for imputation techniques.

Object Label Encoding:

```
1 from sklearn.preprocessing import LabelEncoder
2
3 le = LabelEncoder()
4 df["type"] = le.fit_transform(df["type"])
5
6 df.head()
7
```

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

Handling Outliers

Outliers were identified using boxplots and distribution analysis. Since financial transaction data naturally contains extreme values, careful observation was made to ensure that important fraud-related data points were not removed unnecessarily. Outlier handling helped improve model stability.

Handling Categorical Data

The categorical feature transaction type was converted into numerical form using Label Encoding. This step was necessary because machine learning algorithms require numerical input features for model training.

Splitting Data into Train and Test

The dataset was divided into training and testing sets using an 80:20 split ratio. The training data was used to train the machine learning models, while the testing data was used to evaluate model performance and ensure generalization.

```
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=42
)

X_train.shape, X_test.shape

```

```
((13140, 8), (3286, 8))
```

4.4. Model Building

1. Decision Tree Classifier

Decision Tree is a supervised machine learning algorithm that splits data into branches based on feature values. It creates a tree-like structure to make classification decisions. In this project, it was used to classify transactions as fraud or legitimate based on balance and transaction-related features.

2. Random Forest Classifier

Random Forest is an ensemble learning algorithm that combines multiple decision trees to improve prediction accuracy and reduce overfitting. It provides more stable and reliable results compared to a single decision tree. In this project, Random Forest achieved high accuracy and low fraud miss rate.

3. Extra Trees Classifier

Extra Trees (Extremely Randomized Trees) is similar to Random Forest but introduces more randomness while splitting nodes. This improves model robustness and reduces variance. It was used to compare performance with other tree-based models

4. Support Vector Machine (SVM)

Support Vector Machine is a classification algorithm that finds the optimal boundary (hyperplane) that separates classes. It performs well on smaller datasets but may struggle with

large and highly imbalanced datasets. In this project, SVM showed lower performance compared to tree-based models.

5. XGBoost Classifier (Final Model)

XGBoost is an advanced gradient boosting algorithm that builds models sequentially to correct previous errors. It is highly efficient and performs exceptionally well on structured data. In this project, XGBoost achieved the highest recall and lowest fraud miss rate, making it the final selected model.

Import the Model Libraries (Main Model):

The required machine learning libraries were imported to implement classification algorithms and evaluation metrics.

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

```
from xgboost import XGBClassifier
```

Initiliazing the Model (Main Model):

Each machine learning model was initialized with default or tuned parameters before training.

Training and Testing the Model (Main Model):

The models were trained using the training dataset and tested on the unseen test dataset to evaluate their predictive performance.

Evaluating Performance of Model (Main Model):

Model performance was evaluated using metrics such as Accuracy, Precision, Recall, F1-score, and Confusion Matrix. Special focus was given to recall for the fraud class to minimize missed fraud cases.

```
from xgboost import XGBClassifier

xgb_model = XGBClassifier()
xgb_model.fit(X_train, y_train)

xgb_pred = xgb_model.predict(X_test)

print(confusion_matrix(y_test, xgb_pred))
print(classification_report(y_test, xgb_pred))

```

4.5. Application Building

After selecting the XGBoost model as the final model, a simple web application was developed to allow users to check whether a transaction is fraudulent or legitimate. The application collects transaction details from the user, sends them to the trained model, and displays the prediction result in real time.

Create an HTML File

An HTML file was created to design the user interface of the web application. The webpage contains input fields where users can enter transaction details such as transaction type, amount, sender balance, and receiver balance. A submit button was added to send the data to the backend for prediction. The interface was designed to be simple, user-friendly, and easy to understand.

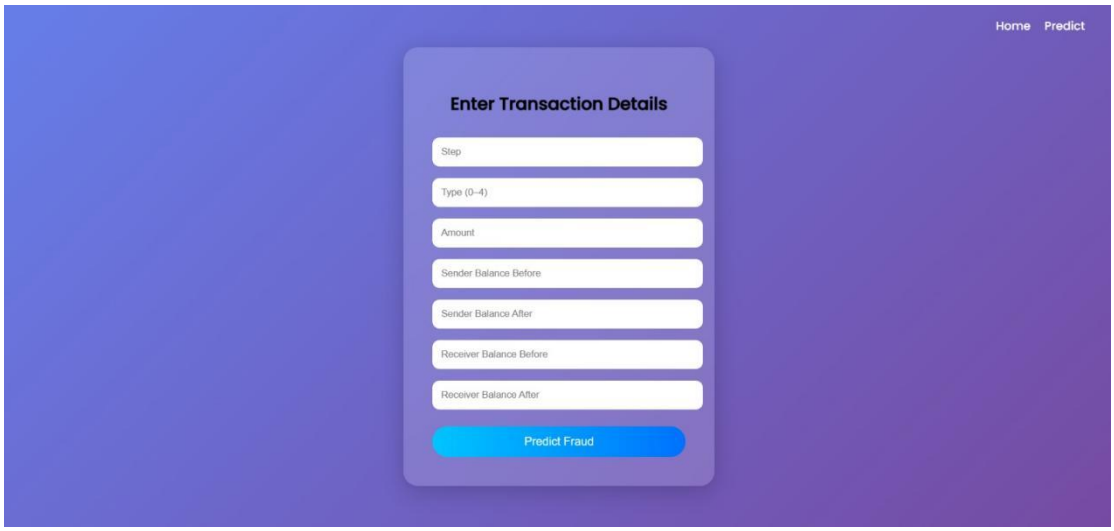
Build Python Code

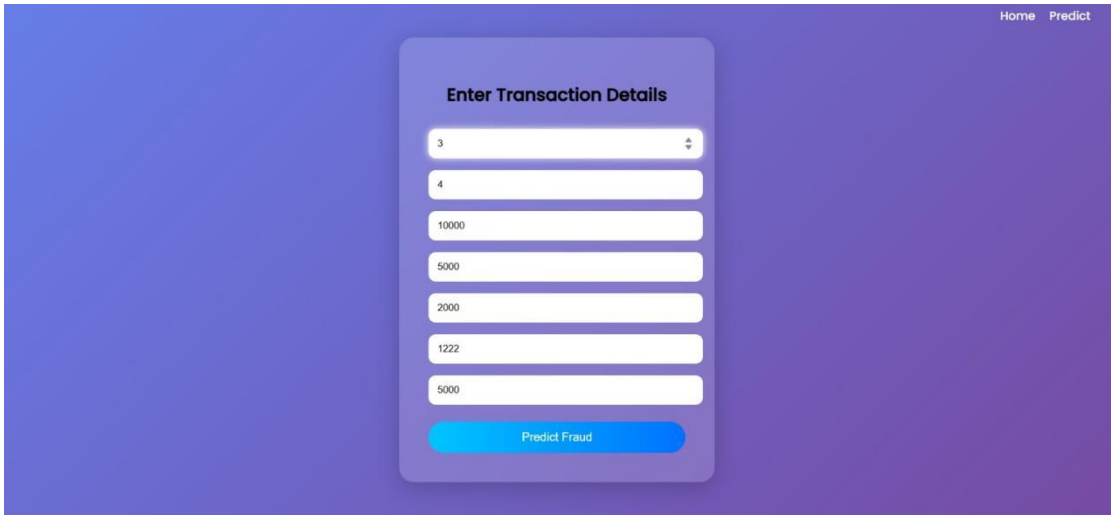
The backend of the application was developed using Python. The saved model file (final_model.pkl) was loaded into the application. When the user enters transaction details and clicks submit, the Python code processes the input data, converts it into the required format, and passes it to the trained XGBoost model. The model then predicts whether the transaction is fraudulent or legitimate and returns the result to the webpage.

Run the Application

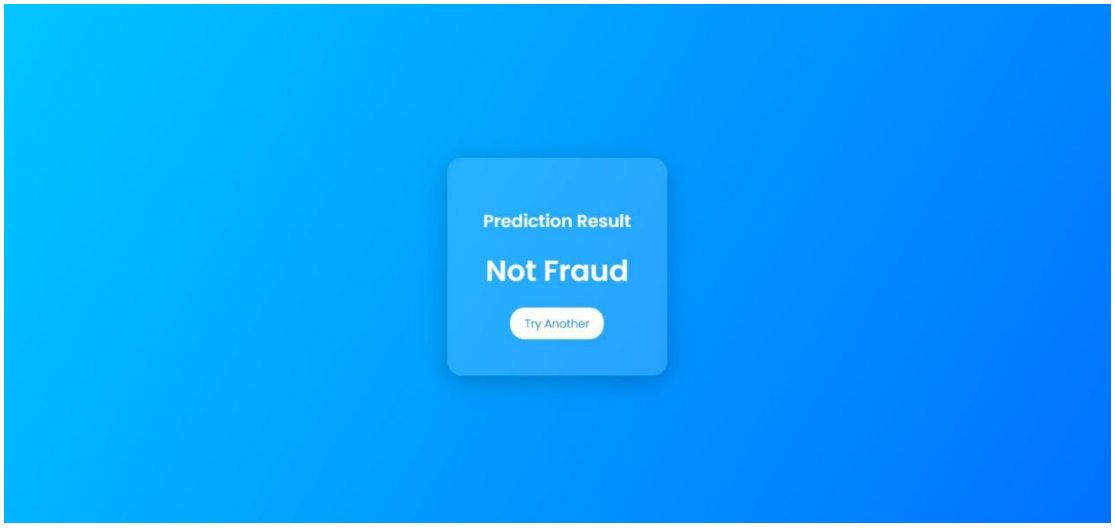
The application was executed using a local server environment. After running the Python script, the web application was opened in a browser.

The user first enters the transaction details on the webpage,





and after submission, the system instantly displays whether the transaction is fraud or not fraud.



This demonstrates real-time fraud detection capability of the developed system.

Website Link: https://fraud-detection-app-48dy.onrender.com

Source Code Link:

https://colab.research.google.com/drive/1gc0AHQgcensb8XXZGwJJzF00OOd6LSE0?usp=sharing

Demo Video Link:

https://drive.google.com/file/d/1E9teX2zck-xy6dNzayUsJNHNj1Vi3Z0x/view?usp=sharing