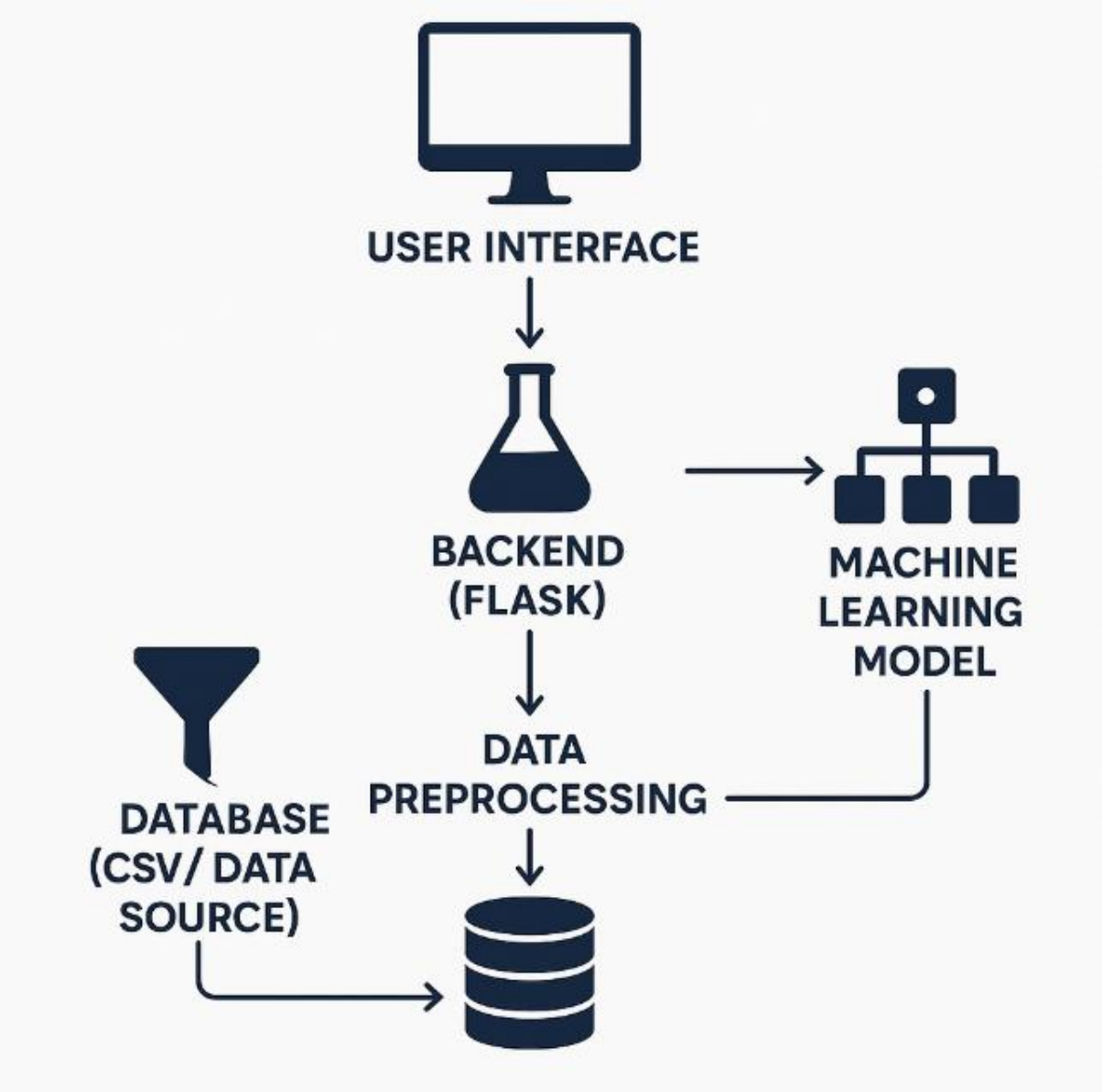
Online Payment Fraud Detection Using Machine Learning

Online payment fraud is a significant concern in today's digital economy. As online transactions proliferate, so do attempts by malicious actors to commit financial fraud. This project aims to develop a robust system to detect fraudulent online payment transactions using machine learning techniques. The system analyzes various transaction parameters to classify whether a payment is legitimate or fraudulent, thereby mitigating financial losses for individuals and businesses.

Technical Architecture:

The technical architecture for the Online Payment Fraud Detection system follows a standard machine learning pipeline integrated into a web application.



- **User Interface (UI):** The front-end where users interact, inputting transaction details or viewing prediction results.
- **Backend (Flask):** A lightweight web framework that handles requests from the UI, processes data, and interacts with the machine learning model.

- **Machine Learning Model:** A trained machine learning model (XGBoost, chosen after experimentation with Random Forest) responsible for making predictions on transaction data.
- **Data Preprocessing:** Steps involved in cleaning, transforming, and preparing raw transaction data for model consumption.
- **Database (CSV/Data Source):** Storage for historical transaction data used for training and testing the model.

Project Flow:

The project development and deployment follow a structured flow to ensure effective fraud detection.

1. **User Interaction:** The user interacts with the User Interface (UI) to input payment transaction data.
2. **Data Analysis by Model:** The entered input is sent to the integrated machine learning model for analysis.
3. **Prediction Display:** Once the model analyzes the input, the prediction (Fraudulent or Legitimate) is displayed back on the UI.

To accomplish this, the following activities are completed:

- **Define Problem / Problem Understanding**
    - Specify the business problem
    - Business requirements
    - Literature Survey
    - Social or Business Impact.
- **Data Collection & Preparation**
    - Collect the dataset
    - Data Preparation (Handling Missing Values, Outliers, Imbalance, Scaling)
    - Exploratory Data Analysis (Descriptive statistical, Visual Analysis)
- **Model Building**
    - Training the model using XGBoost and Random Forest algorithms
    - Testing the model
- **Performance Testing & Hyperparameter Tuning**
    - Testing model with multiple evaluation metrics
    - Comparing model accuracy before & after applying hyperparameter tuning (Optional)
- **Model Deployment**

- Save the best model
- Integrate with Web Framework (Flask)

- **Project Demonstration & Documentation**
  - Record explanation Video for project end-to-end solution
  - Project Documentation - Step-by-step project development procedure

Project Structure:

The project is organized into a clear directory structure for maintainability and deployment.

(Insert an image of the project folder)

As indicated in the project structure, the main components include:

- **project/**: The root directory for the application.
- **templates/**: Contains HTML files for the web interface (index.html, result.html).
- **static/**: Stores static assets like CSS files (styles.css).
- **app.py**: The main Flask backend script.
- **fraud_model.pkl**: The saved trained machine learning model.
- **data_preprocessing.ipynb**: Jupyter Notebook for data cleaning and preprocessing.
- **model_training.ipynb**: Jupyter Notebook for model training and evaluation.
- **requirements.txt**: Lists all necessary Python dependencies for the project.

1: Define Problem / Problem Understanding

1: Specify the Business Problem

The business problem addresses the growing challenge of financial losses due to fraudulent online payment transactions. Detecting these anomalies quickly and accurately is crucial for financial institutions, e-commerce platforms, and consumers to prevent monetary damage and maintain trust in online payment systems.

2: Business Requirements

An online payment fraud detection project needs to fulfill several key business requirements:

- **High Accuracy & Low False Positives:** The system must accurately identify fraudulent transactions while minimizing false positives (legitimate transactions incorrectly flagged as fraud) to avoid inconveniencing users.
- **Real-time Prediction Capability:** The model should be capable of providing near real-time predictions for incoming transactions to prevent fraud before it completes.
- **Scalability:** The system must be scalable to handle a high volume of transactions as the business grows.
- **Interpretability (where possible):** Understanding the reasons behind a fraud prediction can be valuable for investigators and for improving the model.

- **Security & Data Privacy:** Compliance with data protection regulations (e.g., GDPR, PCI DSS) is paramount, ensuring sensitive transaction data is handled securely.
- **Maintainability & Updates:** The system should be designed to allow for easy updates and retraining of the model as fraud patterns evolve.

3: Literature Survey

Fraud detection in digital payments has become increasingly critical with the exponential growth of online transactions. Financial institutions and fintech platforms face immense pressure to safeguard users against unauthorized or suspicious activities. The literature reveals a progression from traditional rule-based systems to sophisticated machine learning and deep learning models, each aiming to improve the detection of fraudulent behavior with minimal false alarms.

4: Social or Business Impact.

- **Social Impact:**
    - **Enhanced Consumer Trust:** A reliable fraud detection system builds consumer confidence in online payment methods, encouraging more digital transactions.
    - **Reduced Financial Stress:** Protects individuals from unexpected financial losses due to fraud.
    - **Fairness:** A well-tuned system can reduce biased outcomes for certain user groups.
- **Business Impact:**
    - **Financial Loss Prevention:** Directly reduces monetary losses for businesses and financial institutions caused by fraudulent transactions.
    - **Improved Operational Efficiency:** Automates fraud detection, reducing the need for manual review and allowing resources to be allocated elsewhere.
    - **Reputation Protection:** Safeguards the brand image and credibility of companies handling online payments.
    - **Regulatory Compliance:** Helps businesses adhere to financial regulations concerning fraud prevention.

2: Data Collection & Preparation

Machine learning models heavily rely on high-quality data. This section details the process of collecting and preparing the dataset for training the fraud detection model.

1: Collect the Dataset

The dataset used for this project is "Online Payment Fraud Detection" from Kaggle, available at the following link:

Link: https://www.kaggle.com/datasets/jainilcoder/online-payment-fraud-detection

This dataset contains transactional data for online payments, including features that can help identify fraudulent activities. It typically includes features like step, type, amount, nameOrig, oldbalanceOrg, newbalanceOrig, nameDest, oldbalanceDest, newbalanceDest, and isFraud. The isFraud column is the target variable, indicating whether a transaction is fraudulent.

1.1: Importing the Libraries

Import the necessary Python libraries for data manipulation, analysis, and machine learning:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb # Import XGBoost
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from imblearn.over_sampling import SMOTE # For handling imbalanced data
from sklearn.preprocessing import StandardScaler, LabelEncoder # LabelEncoder for categorical features
import warnings
import pickle

warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight') # Optional: for plotting style
```

1.2: Read the Dataset

The dataset is in .csv format and can be read using pandas' read_csv() function.

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.0	0.00	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.0	0.00	0	0
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.0	0.00	1	0
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.0	0.00	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.0	0.00	0	0
5	1	PAYMENT	7817.71	C90045638	53860.00	46042.29	M573487274	0.0	0.00	0	0

2: Data Preparation

The raw dataset often requires preprocessing to be suitable for machine learning models. This activity includes handling missing values, outliers, data imbalance, and scaling.

2.1: Handling Missing Values

First, check for any missing values in the dataset.

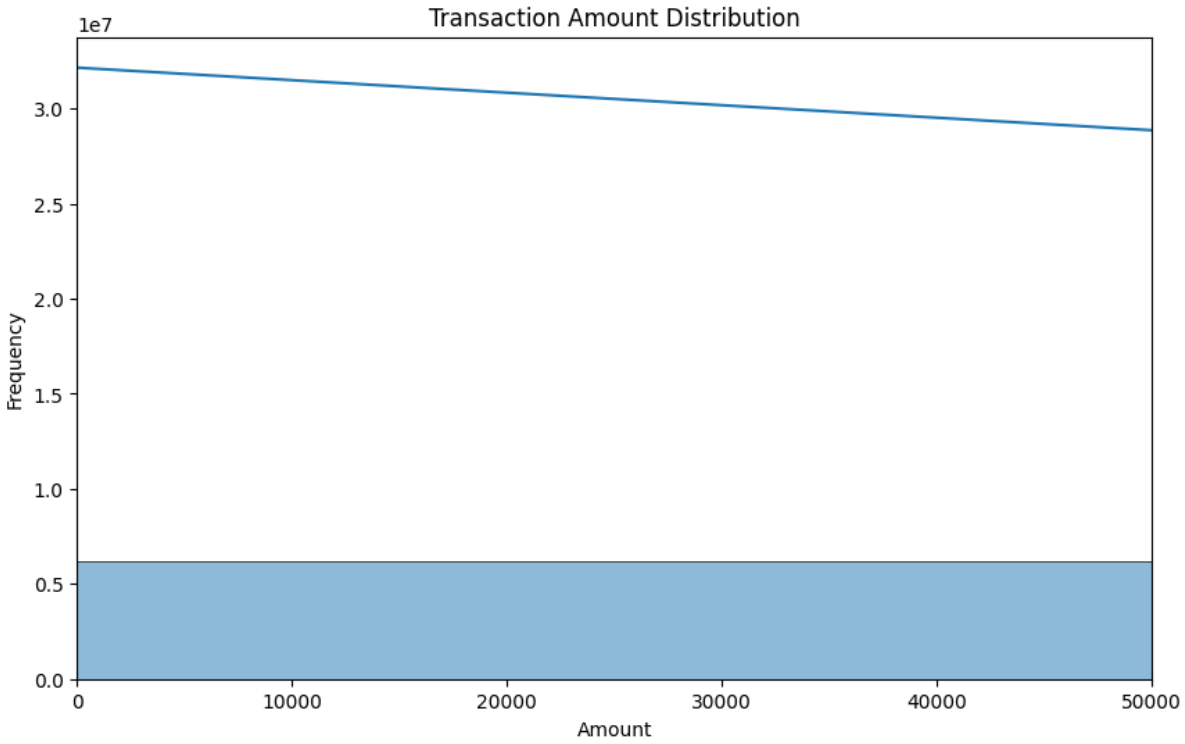
df.isnull().sum()

For this particular dataset, it is crucial to verify and handle any missing values found.

2.2: Handling Outliers

Outliers can significantly impact model performance. Boxplots are useful for visualizing outliers. For numerical features like 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', and 'newbalanceDest', statistical methods (e.g., IQR method) or visualization can help identify them.

```
plt.figure(figsize=(10, 6))
sns.histplot(dataset['amount'], bins=100, kde=True)
plt.title("Transaction Amount Distribution")
plt.xlabel("Amount")
plt.ylabel("Frequency")
plt.xlim(0, 50000)  # Optional to focus on smaller transactions
plt.show()
```



2.3: Handling Imbalanced Dataset

Online payment fraud datasets are typically highly imbalanced, meaning the number of fraudulent transactions (minority class) is significantly smaller than legitimate ones (majority class). This is a critical challenge in fraud detection. Techniques like SMOTE (Synthetic Minority Over-sampling Technique) are essential to address this imbalance.

2.4: Scaling and Encoding

Scaling transforms numerical features to a similar scale, which is important for many machine learning algorithms. Features like 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', and 'newbalanceDest' might need scaling. Standard Scaler is a common choice.

Additionally, this dataset contains categorical features like 'type' (e.g., 'CASH_OUT', 'PAYMENT', 'TRANSFER') that need to be encoded into numerical representations using techniques like Label Encoding or One-Hot Encoding before being fed into a machine learning model.

```
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Encode categorical features ('type')

le = LabelEncoder()

df['type_encoded'] = le.fit_transform(df['type'])

df.drop('type', axis=1, inplace=True) # Drop original 'type' column

# Scale numerical features (excluding 'step', as it's time-related and usually not scaled directly)

scaler = StandardScaler()

numerical_cols = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']
```

df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

```
# Build XGBoost model
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
```

3: Exploratory Data Analysis (EDA)

EDA involves analyzing and visualizing data to understand its characteristics, identify patterns, and uncover insights.

1: Descriptive Statistics

Descriptive statistics provide a summary of the central tendency, dispersion, and shape of the dataset's distribution.

df.describe()

2: Visual Analysis

Visualizations help in understanding data distributions and relationships.

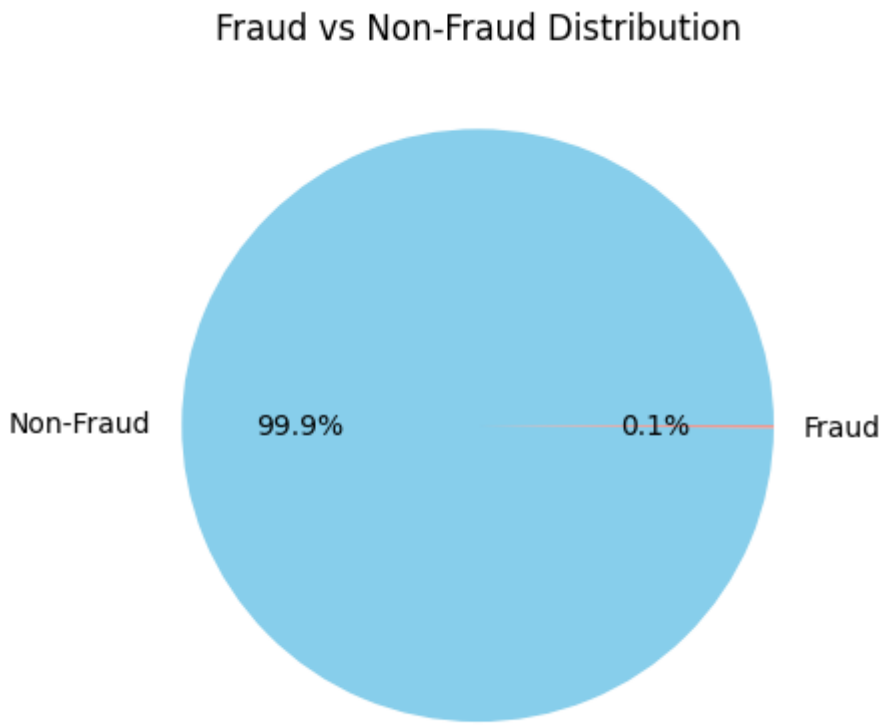
2.1: Univariate Analysis

Understanding individual features. For the 'isFraud' feature, a countplot and pie chart can show the severe imbalance.

Distribution of Fraud v/s Non-Fraud Transactions


```
sns.countplot(x='isFraud', data=dataset)
plt.title("Fraud vs Non-Fraud Transactions")
plt.xlabel("Is Fraud")
plt.ylabel("Count")
plt.show()
```

```
dataset['isFraud'].value_counts().plot.pie(autopct='%1.1f%%', labels=['Non-Fraud', 'Fraud'], colors=['skyblue', 'salmon'])
plt.title('Fraud vs Non-Fraud Distribution')
plt.ylabel('')
plt.show()
```



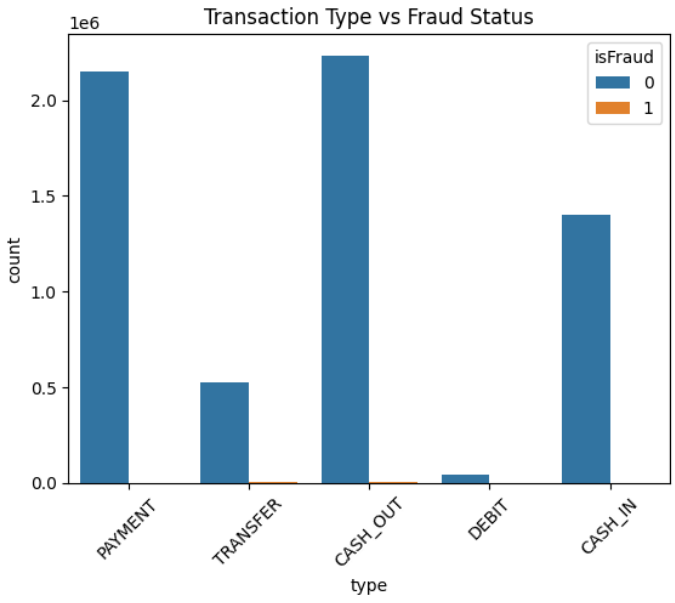
2.2: Bivariate Analysis

Examining relationships between two features.

```
sns.boxplot(x='isFraud', y='amount', data=dataset)
plt.title("Transaction Amount vs Fraud Status")
plt.show()
```

Countplot with hue-Categorical v/s Target

```
sns.countplot(x='type', hue='isFraud', data=dataset)
plt.title('Transaction Type vs Fraud Status')
plt.xticks(rotation=45)
plt.show()
```



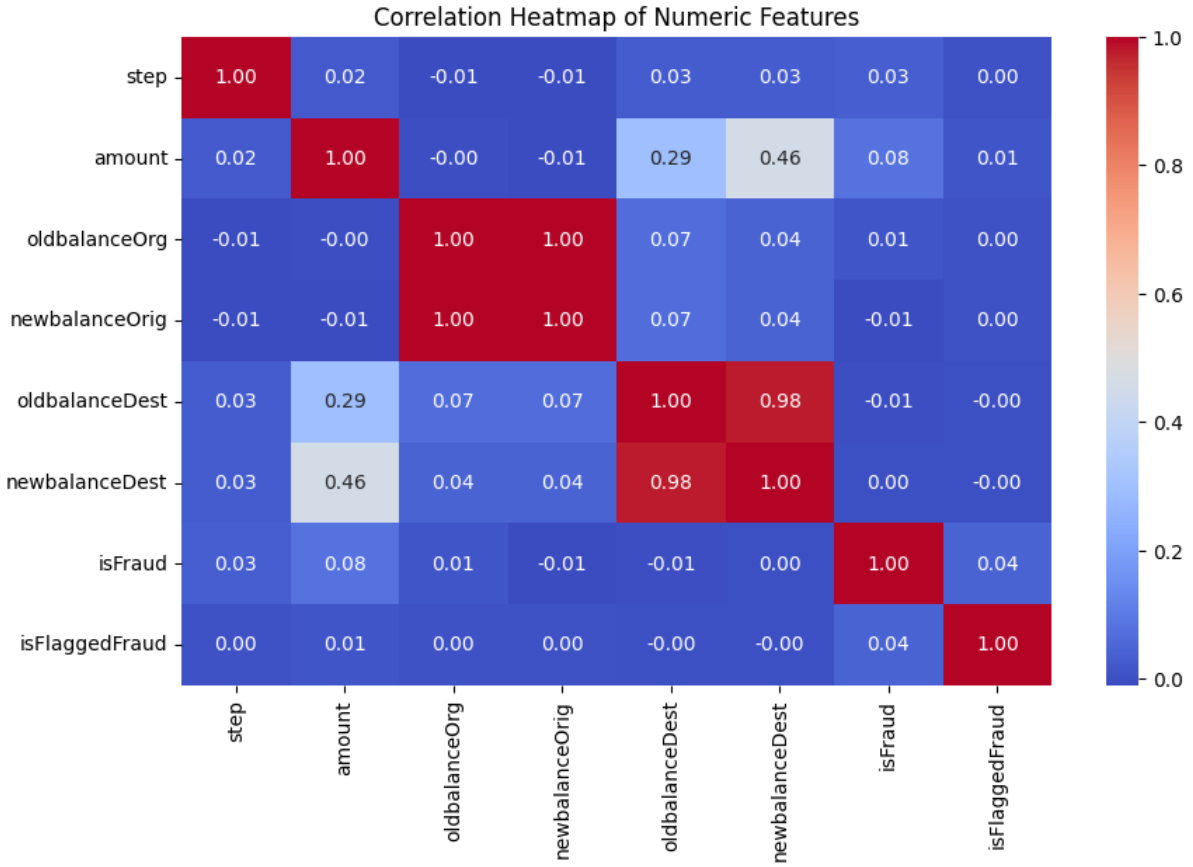
2.3: Multivariate Analysis

Exploring relationships among multiple features, often through correlation matrices or heatmaps.

```
plt.figure(figsize=(12, 8))

# Select only numeric columns
numeric_data = dataset.select_dtypes(include='number')

plt.figure(figsize=(10, 6))
sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```



2.4:Descriptive Analysis

```
print("\nDescriptive Statistics:")
print(dataset.describe())
```

```
Descriptive Statistics:
               step        amount  oldbalanceOrg  newbalanceOrig  \
count  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
std    1.423320e+02  6.038582e+05   2.888243e+06    2.924049e+06
min    1.000000e+00  0.000000e+00   0.000000e+00    0.000000e+00
25%    1.560000e+02  1.338957e+04   0.000000e+00    0.000000e+00
50%    2.390000e+02  7.487194e+04   1.420800e+04    0.000000e+00
75%    3.350000e+02  2.087215e+05   1.073152e+05    1.442584e+05
max    7.430000e+02  9.244552e+07   5.958504e+07    4.958504e+07

       oldbalanceDest  newbalanceDest       isFraud  isFlaggedFraud
count    6.362620e+06    6.362620e+06  6.362620e+06    6.362620e+06
mean     1.100702e+06    1.224996e+06  1.290820e-03    2.514687e-06
std      3.399180e+06    3.674129e+06  3.590480e-02    1.585775e-03
min      0.000000e+00    0.000000e+00  0.000000e+00    0.000000e+00
25%      0.000000e+00    0.000000e+00  0.000000e+00    0.000000e+00
50%      1.327057e+05    2.146614e+05  0.000000e+00    0.000000e+00
75%      9.430367e+05    1.111909e+06  0.000000e+00    0.000000e+00
max      3.560159e+08    3.561793e+08  1.000000e+00    1.000000e+00
```

Splitting Data into Train and Test: The dataset is split into training and testing sets to evaluate the model's performance on unseen data.

4: Model Building

With the data prepared, the next step is to build and train the machine learning models. We experimented with Random Forest and XGBoost, with XGBoost yielding better results.

1: Training the Models

We will train both Random Forest and XGBoost classifiers.

1.1: Random Forest Model

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, accuracy_score

import time


# Train Random Forest

start = time.time()

rf_model = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)

rf_model.fit(X_train, y_train)

end = time.time()


# Predict

rf_pred = rf_model.predict(X_test)
```

1.2: XGBoost Model

```
from xgboost import XGBClassifier

# Train XGBoost
start = time.time()
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb_model.fit(X_train, y_train)
end = time.time()

# Predict
xgb_pred = xgb_model.predict(X_test)
```

2: Testing the Models

```
models = {
    "Random Forest": rf_model,
    "XGBoost": xgb_model
}

for name, model in models.items():
    pred = model.predict(X_test)
    print(f"\n 🔹 {name} Evaluation:")
    print(" ✅ Accuracy:", accuracy_score(y_test, pred))
    print(" ✅ ROC AUC:", roc_auc_score(y_test, pred))
    print(" ✅ F1 Score:", classification_report(y_test, pred, zero_division=0))
print(" ✅ Confusion Matrix:\n", confusion_matrix(y_test, pred))
```

5: Performance Testing & Hyperparameter Tuning

Evaluating the model's performance goes beyond simple accuracy, especially for imbalanced datasets like fraud detection.

1: Testing Model with Multiple Evaluation Metrics

For fraud detection, precision, recall, and F1-score are crucial.

- **Precision:** Of all predicted frauds, how many were actually fraudulent? (Minimizes false positives)
- **Recall:** Of all actual frauds, how many did the model correctly identify? (Minimizes false negatives)
- **F1-Score:** The harmonic mean of precision and recall, providing a single metric that balances both.
- **Accuracy:** Overall correctness, but can be misleading for imbalanced datasets.

Evaluate

print(" 🔹 XGBoost Model")

print(" ✅ Accuracy:", accuracy_score(y_test, xgb_pred))

print(" ✅ ROC AUC:", roc_auc_score(y_test, xgb_pred))

print(" ✅ Classification Report:\n", classification_report(y_test, xgb_pred))

print(" ✅ Confusion Matrix:\n", confusion_matrix(y_test, xgb_pred))

print(f" ⏱️ Time taken: {round(end - start, 2)} seconds")

2: Comparing Model Accuracy Before & After Applying Hyperparameter Tuning (Optional)

Hyperparameter tuning involves optimizing the model's hyperparameters (e.g., n_estimators, max_depth for Random Forest; learning_rate, n_estimators, max_depth for XGBoost) to improve performance. Techniques like Grid Search or Random Search with cross-validation are commonly used. While optional for this project, it's a vital step for production-grade models.

from sklearn.model_selection import cross_val_score

MODEL REPORT OF LOGISTIC REGRESSION

Logistic Regression

Metric	Class 0	Class 1	Macro Avg	Weighted Avg	Overall
Precision	1.00	0.35	0.68	1.00	
Recall	1.00	0.44	0.72	1.00	
F1-Score	1.00	0.39	0.70	1.00	
Accuracy					0.9982
ROC AUC					0.7180
Support	1,270,881	1,643			1,272,524
Confusion Matrix	TP: 1,269,575	FN: 925	FP: 1,306	TN: 718	

MODEL REPORT OF LIGHTBGM

(instead of random forest, due to very large dataset of 12 lakh rows)

LightGBM

Metric	Class 0	Class 1	Macro Avg	Weighted Avg	Overall
Precision	1.00	0.02	0.51	1.00	
Recall	0.99	0.09	0.54	0.99	
F1-Score	1.00	0.03	0.51	0.99	
Accuracy					0.9916
ROC AUC					0.5396
Support	1,270,881	1,643			1,272,524
Confusion Matrix	TP: 1,261,703	FN: 1,501	FP: 9,178	TN: 142	

MODEL REPORT OF XG BOOST (BEST MODEL TO BE SAVED)

XGBoost

Metric	Class 0	Class 1	Macro Avg	Weighted Avg	Overall
Precision	1.00	0.88	0.94	1.00	
Recall	1.00	0.77	0.89	1.00	
F1-Score	1.00	0.82	0.91	1.00	
Accuracy					0.9996
ROC AUC					0.8855
Support	1,270,881	1,643			1,272,524
Confusion Matrix	TP: 1,270,715	FN: 373	FP: 166	TN: 1,267	

6: Model Deployment

The final phase involves saving the trained model (XGBoost, as it performed better) and integrating it into a web application for user interaction.

1: Save the Best Model

After evaluating both Random Forest and XGBoost, the XGBoost model is identified as the best-performing one and saved to avoid retraining it every time it's needed. The pickle module is commonly used for this.

```
import pickle

# Define filename for the saved model (XGBoost)

filename = 'fraud_model_xgb.pkl' # Using a distinct name for the XGBoost model

# Save the trained XGBoost model

with open(filename, 'wb') as file:

    pickle.dump(xgb_model, file) # Saving the xgb_model

print(f"Best model (XGBoost) saved as '{filename}'")

# To load the model later:

# with open(filename, 'rb') as file:
```

loaded_model = pickle.load(file)

print("Model loaded successfully!")

```python
from flask import Flask, render_template, request
import joblib
import numpy as np

app = Flask(__name__)

# Load the saved XGBoost model
model = joblib.load("payments.pkl")

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

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

@app.route('/submit', methods=['POST'])
def submit():
    # Get data from form
```

2: Integrate with Web Framework (Flask)

A web application is built using Flask to provide a user interface for entering transaction details and receiving fraud predictions.

2.1: Building HTML Pages

For this project, the following HTML files are created in the templates/ folder:

- **index.html**: The main page where users input transaction features.
- **result.html**: Displays the prediction outcome (Fraudulent or Legal).

(Please insert screenshots or example code snippets of index.html and result.html here, showing the input fields and prediction display area.)

2.2: Build Python Code (app.py)

The app.py script serves as the backend logic using Flask. It imports necessary libraries, loads the saved model, defines routes, handles user input, makes predictions, and renders HTML templates.

(in terminal)

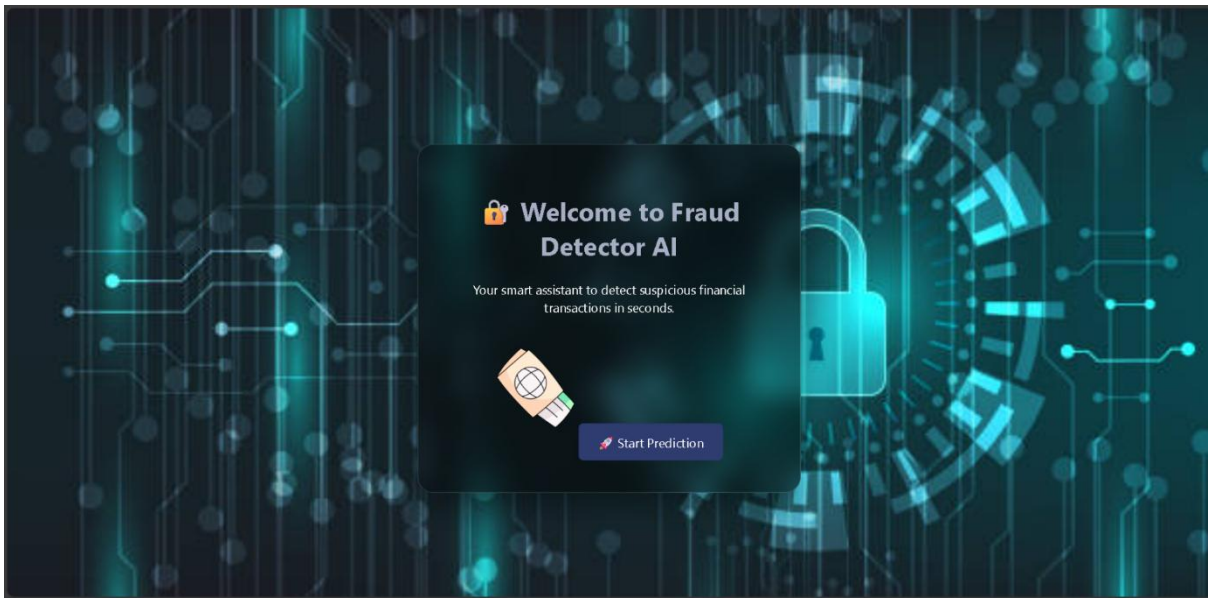
cd flask

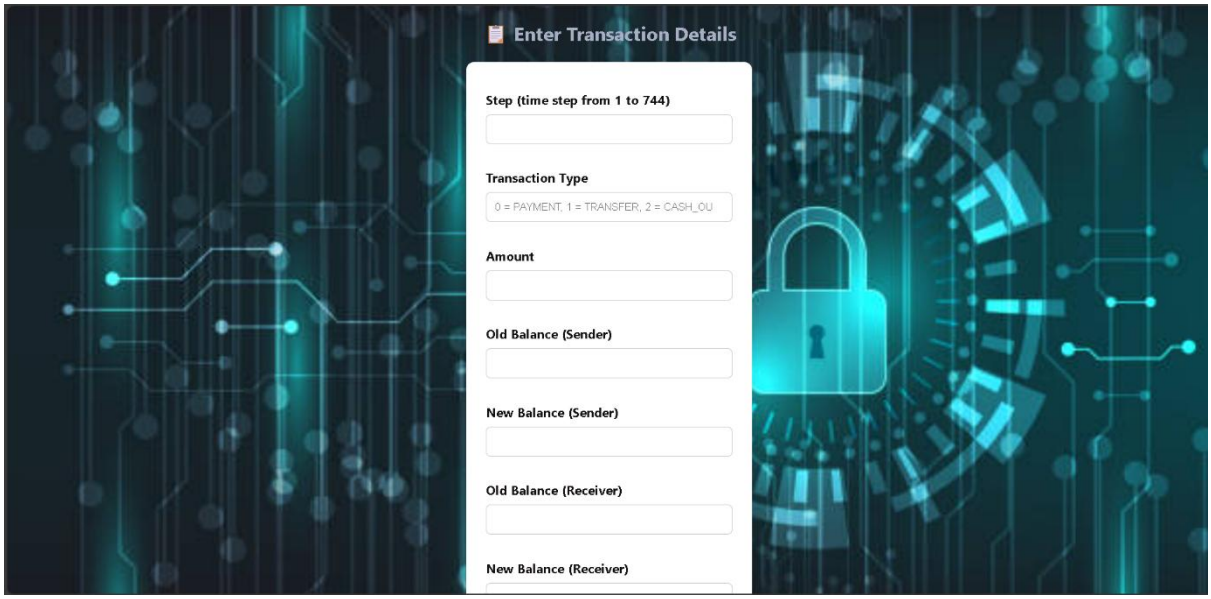
python app.py

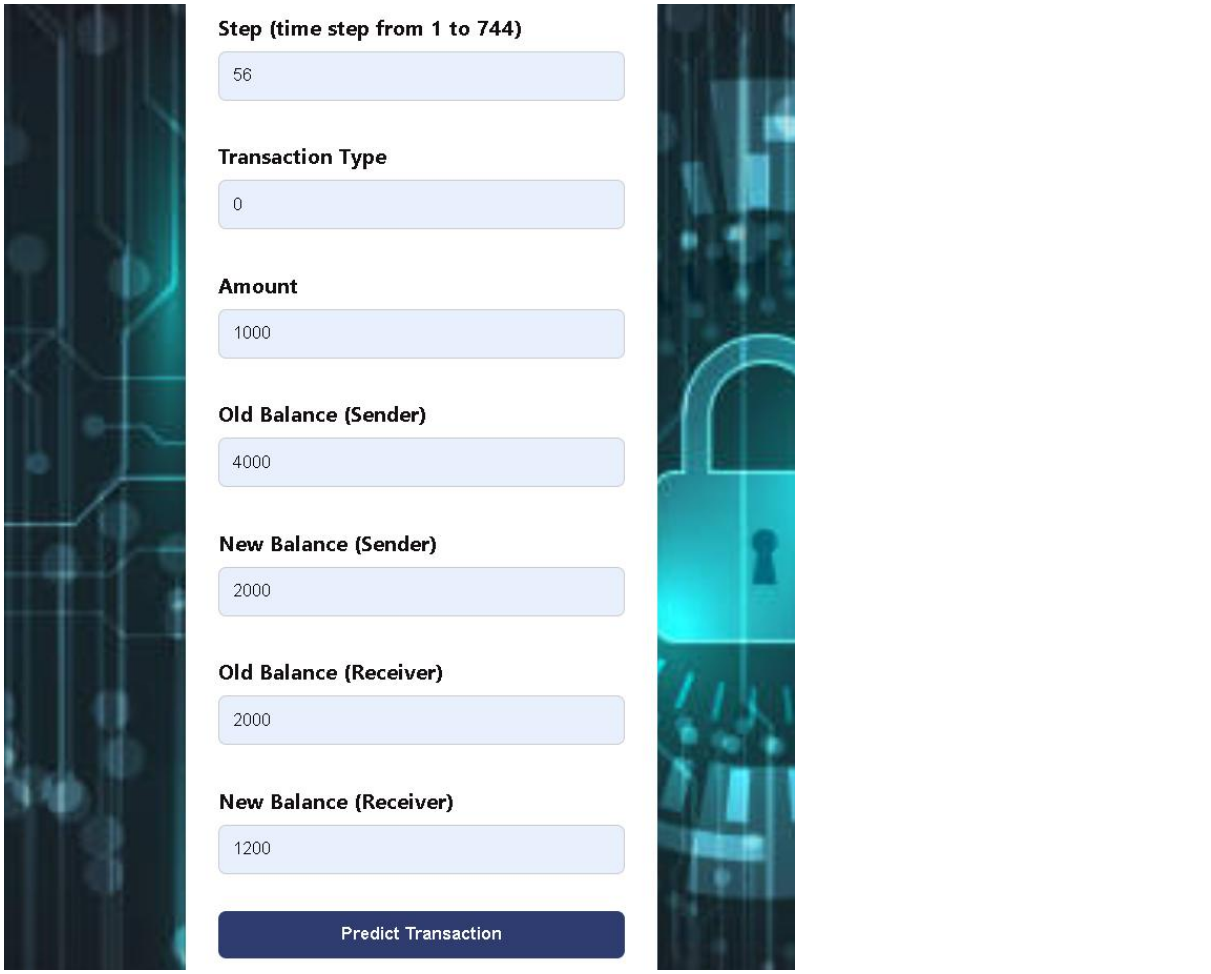
2.3: Run the Web Application.

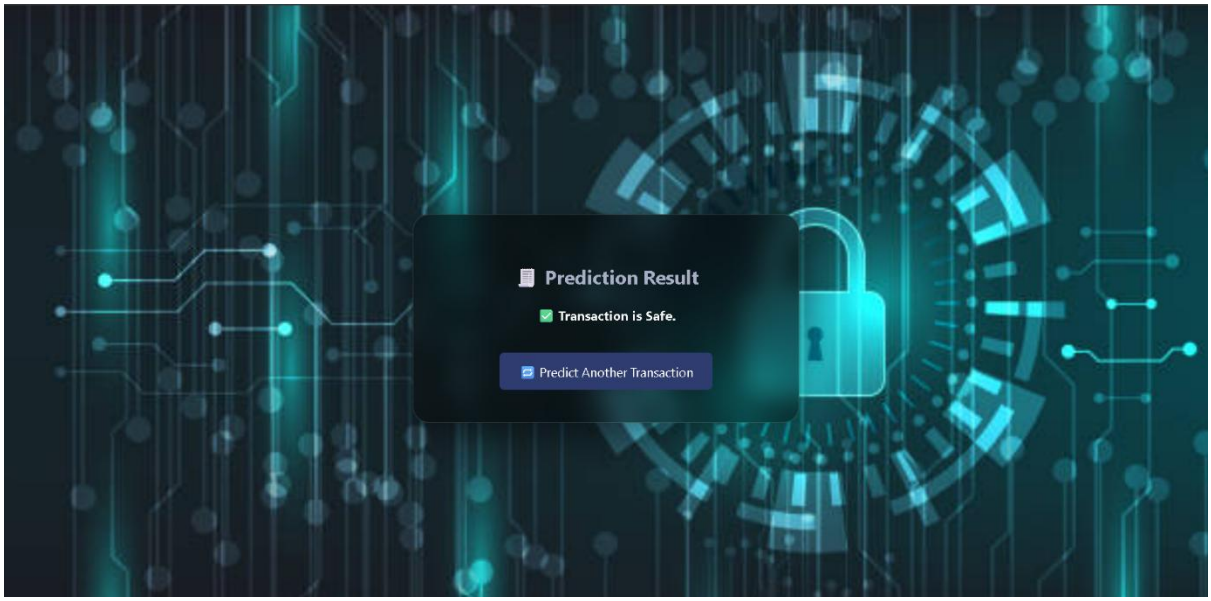
Cloud deployment Link on Render

https://dashboard.render.com/




Enter Transaction Details
Step (time step from 1 to 744)
Transaction Type
0 = PAYMENT, 1 = TRANSFER, 2 = CASH_OU
Amount
Old Balance (Sender)
New Balance (Sender)
Old Balance (Receiver)
New Balance (Receiver)


Step (time step from 1 to 744)
56
Transaction Type
0
Amount
1000
Old Balance (Sender)
4000
New Balance (Sender)
2000
Old Balance (Receiver)
2000
New Balance (Receiver)
1200
Predict Transaction



Step (time step from 1 to 744)

45

Transaction Type

1

Amount

1000000

Old Balance (Sender)

1000000

New Balance (Sender)

0

Old Balance (Receiver)

0

New Balance (Receiver)

0

Predict Transaction

7: Project Demonstration (Vedio)

Explaining the working of project, about saved model etc.

https://drive.google.com/file/d/1XmKvl7NU7eS2rRpfXFji051qaLwMm-Wl/view?usp=sharing

8: Team Contributions / Credits

This section acknowledges the contributions of each team member to the "Online Payment Fraud Detection using Machine Learning" project.

- **Muskan Sharma : EDA, Model training, deployment.**
- **Felix Samuel.C : Frontend, backend, documentation.**
- **Runeet:video demonstration, Frontend.**

9: Cloud deployment Link on Render

https://dashboard.render.com/