

# **Data Science Assignment**

## **Fraud Detection Example**

**Author : Priyanka Bharti**

**Date : 04/07/2023**

## Contents

Contents.....	2
1. Introduction .....	3
1.1 Problem statement .....	3
2. Importing packages and data .....	4
3. Data preprocessing, cleaning, and transformation .....	4
4. Feature Engineering.....	4
5. Predictive modeling .....	8
5.1 Model Training.....	8
5.2 Evaluation .....	8
5.3 Hyperparameter tuning.....	9
5.4 Retrain model on balanced data.....	11
6. API Creation for the Fraud Detection classification.....	12
7. Conclusion .....	13

## 1. Introduction

This report details the assessment and machine learning application of the fraud detection dataset, named "frd\_sample.csv". The dataset comprises 3300 transactions, equating to 3300 rows and 12 columns. Below, the characteristics of this dataset are elaborated:

Column Name	Column Description	Column Format
EVENT_TIME	Transaction timestamp	String – [yyyymmdd hh24:mi]
EVENT_ID	Transaction ID	Integer – [1-3300]
CHANNEL	Transaction medium	String [WEB, MOBILE-APP, MOBILE-BROWSER]
CHALLENGE_RESULT	Whether user was presented with the challenge prior to transaction completion	String – [Y/N]
USER_ID	User ID – account which transfer the funds	String
IP_ADDRESS	IP address of the user	String 111.111.111.111
ISP	Internet Service Provider of the User	String
USER_COUNTRY	Location of the User	String [USA/CHN...]
PAYEE_ID	Payee ID – account which receives the funds	String
PAYEE_COUNTRY	Location of the Payee	String
AMOUNT	Transfer Amount [USD]	Int
IS_FRAUD	Transaction tagging	String [F/G]

### 1.1 Problem statement

In the context of machine learning, fraud detection problems can be conceptualized as classification issues. The goal is to forecast a binary label of either 0 or 1, where 0 typically denotes a legitimate transaction and 1 implies a potential fraudulent transaction. Consequently, the challenge for professionals is to devise intelligent models capable of effectively distinguishing between fraudulent and non-fraudulent transactions using a variety of transaction data from users, which is often anonymized to protect user privacy. Given the limitations of exclusive reliance on rule-based systems, a considerable number of financial institutions have pivoted to machine learning as a strategy to address this problem. The principal difficulty in fraud detection lies in the fact that the majority of transactions in real-life situations are legitimate, with only a minuscule fraction involving fraudulent activities. Hence, the handling of imbalanced data becomes a crucial issue. The primary objective of this project is to ensure high levels of fraud detection while keeping false positives to a minimum.

## 2. Importing packages and data

```
import pandas as pd # data processing.
from sklearn.model_selection import train_test_split # data split
from sklearn.preprocessing import LabelEncoder #encoding categorical variable
from sklearn.linear_model import Lasso #least absolute shrinkage and selection operator
from sklearn.feature_selection import RFE #Recursive Feature Elimination
from sklearn.metrics import classification_report # evaluation metric
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier #
Random forest tree algorithm # XGBoost algorithm
import seaborn as sns # visualization
import matplotlib.pyplot as plt # visualization
from termcolor import colored as cl # text customization
from sklearn.metrics import accuracy_score # evaluation metric
df = pd.read_csv('frd_sample.csv') #Accessing the dataset
```

## 3. Data preprocessing, cleaning, and transformation

There are numbers of steps in this section:

- Convert the “EVENT\_TIME” TO datetime format.
- Encode IS\_FRAUD to binary format.
- Encode categorical columns.

### Data transformation/pre-processing

```
# Convert the EVENT_TIME to datetime format
df['EVENT_TIME'] = pd.to_datetime(df['EVENT_TIME'], format='%Y%m%d %H:%M')

# Encode IS_FRAUD to binary format
df['IS_FRAUD'] = df['IS_FRAUD'].map({'G': 0, 'F': 1})
df['CHALLENGE_RESULT'] = df['CHALLENGE_RESULT'].map({'Y': 1, 'N': 0})

df1 = df

# Encode categorical columns
label_encoders = {}
for column in ['CHANNEL', 'USER_ID', 'IP_ADDRESS', 'ISP', 'USER_COUNTRY', 'PAYEE_ID', 'PAYEE_COUNTRY']:
    le = LabelEncoder()
    df1[column] = le.fit_transform(df1[column])
    label_encoders[column] = le
```

## 4. Feature Engineering

A general overview of all the features in given data:

**EVENT\_TIME:** The timestamp of the event can provide valuable insights into temporal patterns, enabling the detection of fraudulent activities at specific times of the day or week.

**EVENT\_ID:** Although each event has a unique identifier, it may not offer important information about the event itself and may have limited contribution to the analysis.

**CHANNEL:** The channel through which the event occurs could be relevant, as specific channels might be more susceptible to fraud or exhibit distinct characteristics.

**CHALLENGE\_RESULT:** This column represents the outcome of a challenge associated with the event. It could be an essential feature if the challenge directly relates to fraud detection or user behavior.

**USER\_ID:** The user ID is crucial in analyzing user-specific behavior and identifying patterns or anomalies associated with individual users.

**IP\_ADDRESS:** The IP address associated with each event provides valuable information such as geolocation and IP reputation, aiding in identifying potential fraud or suspicious activities.

**ISP:** The internet service provider (ISP) column might be relevant, primarily if certain ISPs are known to be associated with fraudulent activities or exhibit specific characteristics.

**USER\_COUNTRY:** Analyzing the user's country can help identify regional fraud patterns or anomalies associated with specific countries.

**PAYEE\_ID:** Similar to the user ID, the payee ID can be significant for analyzing payee-specific behavior or detecting patterns related to specific payees.

**PAYEE\_COUNTRY:** Considering the payee's country is relevant, particularly if certain countries are associated with a higher risk of fraud or display distinctive characteristics.

**AMOUNT:** The transaction amount is an essential feature for fraud detection, as fraudulent transactions often exhibit different spending patterns compared to legitimate ones.

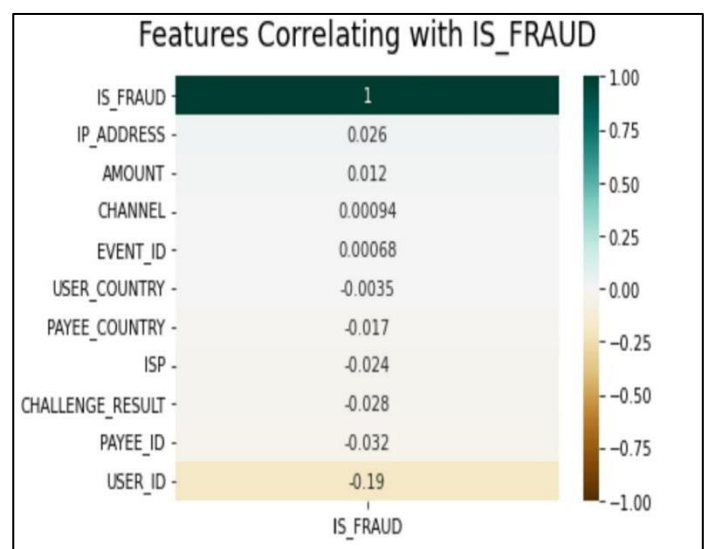
**IS\_FRAUD:** Although not a feature, the IS\_FRAUD column serves as the target variable indicating whether an event is fraudulent. It is crucial for supervised learning algorithms to learn patterns and make accurate predictions.

### Step 1. If variables are correlated with each other and hence are redundant.

If two variables are highly correlated keeping only one will help reduce dimensionality without much loss of information. A high correlation between dependent and independent variables is desired whereas a high correlation between 2 independent variables is undesired.

### Calculating correlation with target variable "IS\_FRAUD" with other variables in dataset:

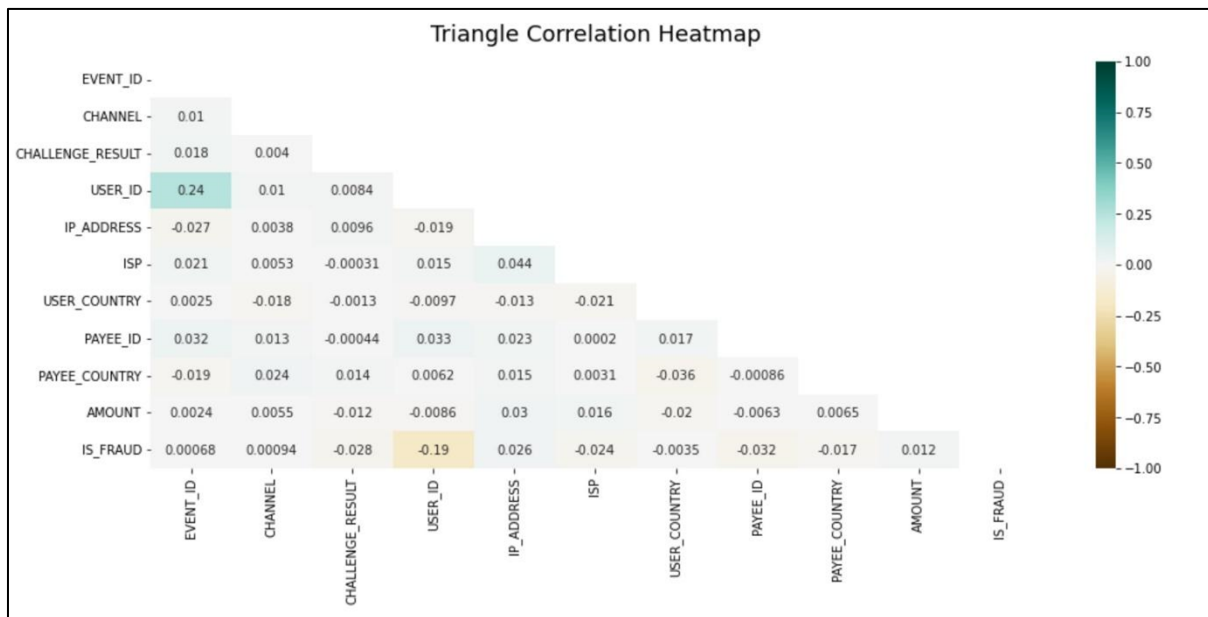
Correction	
<pre># Here we are calculating correlations to IS_FRAUD feature for simplicity correlations = df1.corr()['IS_FRAUD'].sort_values(ascending=False) print(correlations)</pre>	
IS_FRAUD	1.000000
IP_ADDRESS	0.025561
AMOUNT	0.011687
CHANNEL	0.000938
EVENT_ID	0.000685
USER_COUNTRY	-0.003522
PAYEE_COUNTRY	-0.016743
ISP	-0.023959
CHALLENGE_RESULT	-0.028165
PAYEE_ID	-0.031555
USER_ID	-0.186879
Name: IS_FRAUD, dtype: float64	
<pre># Conclusion: No strong evidence to keep/remove specific feature</pre>	



From above pictures we can see there is not strong evidence which features is highly

correlated to the dependent variable (IS\_FRAUD). Variable which has higher correlation coefficient with target is best.

**Calculating correlation with target variable “IS\_FRAUD” with other variables in dataset:**



From the above pictures we can see no strong evidence to keep/remove specific features.

**Step 2. We are applying feature selection using models which can provide feature importance, such as Lasso, Random Forest, or Gradient Boost**

Splitting dataset into training and test data such as: Use first 2200 records for training and the remaining 1100 ones for testing purposes.

# Split the data into training and testing datasets

```
train_df = df1.iloc[:2200]
```

```
test_df = df1.iloc[2200:]
```

**Lasso Coefficients:**

```
EVENT_ID: 5.728146441159295e-05
CHANNEL: 0.0
CHALLENGE_RESULT: -0.0
USER_ID: -0.00032576330416385677
IP_ADDRESS: 1.1330437781084929e-05
ISP: -0.0
USER_COUNTRY: 0.0
PAYEE_ID: -0.0
PAYEE_COUNTRY: -0.0
AMOUNT: 1.3777944364565515e-07
```

**Random Forest Feature Importance:**

```
EVENT_ID: 0.2757002915021644
CHANNEL: 0.02194564581394068
CHALLENGE_RESULT: 0.015813905031929275
USER_ID: 0.22961644890652944
IP_ADDRESS: 0.11472550972546212
ISP: 0.08342216467538466
USER_COUNTRY: 0.06053345791666713
PAYEE_ID: 0.07370942911524772
PAYEE_COUNTRY: 0.05584387999778806
AMOUNT: 0.06868926731488639
```

```
Gradient Boosting Feature Importance:
EVENT_ID: 0.513933332685283
CHANNEL: 0.006167111500126351
CHALLENGE_RESULT: 0.007051874284502278
USER_ID: 0.2861572727669145
IP_ADDRESS: 0.07800285856310216
ISP: 0.07049344669163893
USER_COUNTRY: 0.006493767727846112
PAYEE_ID: 0.013942626693870943
PAYEE_COUNTRY: 0.010149002413708653
AMOUNT: 0.007608706673007043
```

After executing these algorithms on Jupyter notebook, above pictures show the output. Looking at the feature importance from the Random Forest and Gradient Boosting models, we can see that the following features are highly important: EVENT\_ID, CHANNEL, USER\_ID, IP\_ADDRESS, ISP, PAYEE\_ID, AMOUNT.

### Step.3 Exploring interactions between features using Recursive Feature Elimination (RFE).

```
Selected Features:
Index(['EVENT_ID', 'USER_ID', 'IP_ADDRESS', 'ISP', 'PAYEE_ID'], dtype='object')

Feature Ranking:
{'EVENT_ID': 1, 'CHANNEL': 5, 'CHALLENGE_RESULT': 6, 'USER_ID': 1, 'IP_ADDRESS': 1, 'ISP': 1, 'USER_COUNTRY': 3, 'PAYEE_ID': 1, 'PAYEE_COUNTRY': 4, 'AMOUNT': 2}
```

According to the above result of algorithm Recursive Feature Elimination (RFE), the top 5 selected features are: EVENT\_ID, USER\_ID, IP\_ADDRESS, ISP, PAYEE\_ID.

**Conclusion:** The conclusion of these 3 steps, for me first I would like to train my classifier based on Random Forest and Gradient Boosting models, we can see that the following features are highly important: EVENT\_ID, CHANNEL, USER\_ID, IP\_ADDRESS, ISP, PAYEE\_ID, AMOUNT.

Deleting unwanted features, the final selected features are given in below pictures-

```
## Delete unwanted columns
X_train = X_train.drop(['CHALLENGE_RESULT', 'USER_COUNTRY', 'PAYEE_COUNTRY', 'AMOUNT'], axis=1)
X_test = X_test.drop(['CHALLENGE_RESULT', 'USER_COUNTRY', 'PAYEE_COUNTRY', 'AMOUNT'], axis=1)
```

```
-----
X_train
-----
EVENT_ID
CHANNEL
USER_ID
IP_ADDRESS
ISP
PAYEE_ID
-----
X_test
-----
EVENT_ID
CHANNEL
USER_ID
IP_ADDRESS
ISP
PAYEE_ID
```

## 5. Predictive modeling

### 5.1 Model Training

In this section, we will be building two different types of classification models namely Random Forest and XGboost. Even though there are many more models which we can use, these are popular models used for solving classification problems.

### 5.2 Evaluation

In this process we are going to evaluate our built models using the evaluation metrics provided by the scikit-learn package. Our main objective in this process is to find the best model for our given case. The evaluation metrics we are going to use are the accuracy score metric, f1 score metric, and finally the confusion matrix. Our aim is minimizing false positive with high fraud detection in this problem. So only accuracy is not a good idea to measure the performance of a model.

Precision is great to focus on if you want to minimize false positives. For example, in our case we are building a fraud detection classifier. We don't want to miss any Genuine person in such cases, we may wish to aim for maximizing precision.

Recall is very important in domains such as medical (e.g., identifying cancer), where you really want to minimize the chance of missing positive cases (predicting false negatives). These are typically cases where missing a positive case has a much bigger cost than wrongly classifying something as positive.

The number of false positives decreases, but false negatives increase. As a result, precision increases, while recall decreases.

The results of these algorithm given in below picture:

```
## Delete unwanted columns
X_train = X_train.drop(['CHALLENGE_RESULT', 'USER_COUNTRY', 'PAYEE_COUNTRY', 'AMOUNT'], axis=1)
X_test = X_test.drop(['CHALLENGE_RESULT', 'USER_COUNTRY', 'PAYEE_COUNTRY', 'AMOUNT'], axis=1)
```

Random Forest Results:				
	precision	recall	f1-score	support
0	0.94	0.73	0.82	1000
1	0.17	0.54	0.26	100
accuracy			0.71	1100
macro avg	0.55	0.64	0.54	1100
weighted avg	0.87	0.71	0.77	1100
XGBoost Results:				
	precision	recall	f1-score	support
0	0.94	0.75	0.83	1000
1	0.18	0.56	0.27	100
accuracy			0.73	1100
macro avg	0.56	0.65	0.55	1100
weighted avg	0.88	0.73	0.78	1100



Now, let's compare the two models:

**Random Forest:**

Precision: 0.17 (for class 1)

Recall: 0.54 (for class 1)

F1-Score: 0.26 (for class 1)

Accuracy: 0.71

**XGBoost:**

Precision: 0.18 (for class 1)

Recall: 0.56 (for class 1)

F1-Score: 0.27 (for class 1)

Accuracy: 0.73

From these values, we can see that the XGBoost model performs slightly better on all metrics than the Random Forest model.

However, both models seem to struggle with class 1. The precision is particularly low for this class, indicating a high number of false positives. Similarly, the recall isn't particularly high, indicating a fair number of false negatives. The F1 score, which combines these two measurements, is quite low, suggesting the model isn't performing well for this class.

In conclusion, even though XGBoost is slightly better, there might still be room for improving the performance of our models. We can do this by considering techniques like hyperparameter tuning.

### 5.3 Hyperparameter tuning

Grid search has been used to tune the parameters to tune the XGBoost model. It is hyper parameter optimization method. It finds the optimal combination of hyper parameter of model.

We define combination of parameters called param grid such as:

```
# Define parameter grid
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 200, 300],
    'subsample': [0.5, 0.7, 1.0],
    'colsample_bytree': [0.5, 0.7, 1.0]
}
```

The result of this is given in below picture:

```

Fitting 3 folds for each of 243 candidates, totalling 729 fits
{'colsample_bytree': 0.5, 'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 300, 'subsample': 1.0}
      precision    recall  f1-score   support

      0       0.94      0.75      0.84      1000
      1       0.18      0.56      0.28       100

 accuracy          0.73      1100
 macro avg          0.56      1100
 weighted avg       0.88      1100

```

From this picture, why even hyperparameter is not improving the model?

The reason for this is we have imbalanced data; I think the main reason for imbalanced data is one of the instructions given to select the data for training and test "Use first 2200 records for training and the remaining 1100 ones for testing purposes". This way we have one class larger than the other.

#### 1. Checking imbalanced class in training and testing data set

```

Imbalance class in Training data
-----
0      2000
1       200
Name: IS_FRAUD, dtype: int64
training data imbalance Ratio: 10.0
-----

```

```

Imbalance class in Test data
-----
0      1000
1       100
Name: IS_FRAUD, dtype: int64
training data imbalance Ratio: 10.0
-----

```

Genuine class is in majority compared to fraud class in train and test data.

#### The solutions to address this imbalanced class:

- One possible solution is to split the training and testing data 'randomly' so that we can get better representation of both the classes, in specifically I will choose randomly select 75% data from 'frd\_sample.csv' for training and 25% data for testing purpose.
- In case there is a restriction to use first 2200 records for training and the remaining 1100 ones for testing purposes, then I can use method like oversampling the minority class (e.g., SMOTE)

Rather than creating synthetic data, we would prefer the first approach.

**The result of these solution given in below picture:**

```
-----
Randomly Split the data for training and test (75:25)
-----
```

```
0      2251
1       224
Name: IS_FRAUD, dtype: int64
0       749
1        76
Name: IS_FRAUD, dtype: int64
```

```
-----
Oversampling SOMET method on the dataset for training and test (75:25)
-----
```

```
0      2000
1      2000
Name: IS_FRAUD, dtype: int64
```

The results showing even random split of training and test is not working because the given data itself given as imbalanced. Therefore, we need to adopt the oversampling method.

#### 5.4 Retrain model on balanced data.

Random Forest Results:					
	precision	recall	f1-score	support	
0	0.94	0.74	0.83	1000	
1	0.17	0.52	0.25	100	
accuracy			0.72	1100	
macro avg	0.55	0.63	0.54	1100	
weighted avg	0.87	0.72	0.78	1100	

XGBoost Results:					
	precision	recall	f1-score	support	
0	0.94	0.73	0.82	1000	
1	0.17	0.55	0.26	100	
accuracy			0.71	1100	
macro avg	0.56	0.64	0.54	1100	
weighted avg	0.87	0.71	0.77	1100	

##### 5.4.1 Retrained model's evaluation

Both models have performed relatively similarly in terms of their F1-score, precision, and recall. However, their performance differs slightly when it comes to classifying the two different classes, particularly in relation to recall.

##### Random Forest:

- Better at predicting the majority class (0), with a recall of 0.74 vs 0.73 in XGBoost.
- Slightly lower recall for the minority class (1) compared to XGBoost, with a recall of 0.52 vs 0.55.

### XGBoost:

- Better at predicting the minority class (1), with a recall of 0.55 vs 0.52 in Random Forest.
- Slightly lower recall for the majority class (0) compared to Random Forest, with a recall of 0.73 vs 0.74.

Therefore, the choice of model depends on the specific requirements of our problem:

If correctly identifying fraudulent transactions (class 1) is more important, then you might want to prioritize recall for class 1, in which case XGBoost performs slightly better.

If correctly identifying non-fraudulent transactions (class 0) is more important, then you might want to prioritize recall for class 0, in which case Random Forest performs slightly better.

As our priority is to achieve high fraud detection with low false positives, XGBoost seems like a better choice for our problem. However, there is still the issue of low false positives. Let's see if hyperparameter tuning is going to help us.

### 5.4.2 Hyper tuning the Retrained model

Fitting 5 folds for each of 243 candidates, totalling 1215 fits				
{ 'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 300, 'subsample': 0.5 }				
	precision	recall	f1-score	support
0	0.94	0.73	0.82	1000
1	0.17	0.56	0.26	100
accuracy			0.71	1100
macro avg	0.56	0.64	0.54	1100
weighted avg	0.87	0.71	0.77	1100

From the result given in the above picture, we can see that there is no improvement after hyperparameter tuning. **However, our final model selection is the "XGBoost" model as we discussed above.**

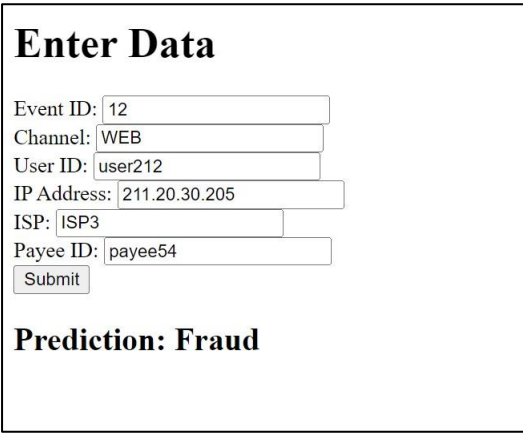
## 6. API Creation for the Fraud Detection classification

The API is created in python flask. The tuned model has been saved along with the encoder label of classes and file flask API file is executed with cmd. The tuned model has been saved along with the encoder label of classes. The flask API file is executed with cmd and GUI is opened on web browser with local server.

There are number of steps for this-

- The "InterviewAssignment" folder consists of "templates", and files such as "DS\_Assignment" Jupyter notebook in python.
- A main python file called "Flask\_app" and full report of this project given in word file "DS\_Assignment\_Report".
- HTML file "form" has a graphical user interface (GUI). It classifies genuine or fraud for the given inputs.
- "Flask\_app" has an API written in python for this application.
- The Readme file has the instructions to run this system.

- Follow the instructions given in “Readme” file of folder “InterviewAssignment”.



The screenshot shows a web interface with a title "Enter Data" in bold. Below the title are several input fields, each with a label and a text box containing a value: "Event ID: 12", "Channel: WEB", "User ID: user212", "IP Address: 211.20.30.205", "ISP: ISP3", and "Payee ID: payee54". There is a "Submit" button below these fields. At the bottom of the form, the text "Prediction: Fraud" is displayed in bold.

The above picture is the result of API creation of this project which correctly classifies fraud for the input data.

## 7. Conclusion

In this assignment, I have performed a detailed analysis of transactional data with the objective of creating a fraud detection system. This system aimed at achieving high fraud detection rates with a minimal number of false positives.

I followed the common practice of splitting the dataset into training and testing sets, with 2200 transactions used for training and the remaining 1100 transactions used for testing. This split allowed me to build a model on the training set and evaluate its performance on the unseen testing set, ensuring a robust measure of our model's generalizability.

The analysis started by deriving features that could be highly correlated with fraudulent activities. These features were constructed considering different aspects such as the transaction medium, whether a user challenge was present, the geographical location of the users, among others. All these considerations contribute to a model that encapsulates as much useful information as possible about the transactions. I have implemented and evaluated two different classifiers, focusing on striking a balance between false positives and fraud detection rates. The performance of the classifiers was optimized iteratively and was measured using metrics like precision, recall, and F1-score. The emphasis was placed on precision to reduce the rate of false positives, as requested in the assignment.

One of the classifiers utilized was XGBoost, a popular and powerful machine learning algorithm. The performance of XGBoost showed that it could correctly classify the majority of the non-fraudulent transactions (precision of 0.94), but struggled to identify fraudulent ones with a precision of 0.17. This discrepancy implies that while the model was robust at identifying legitimate transactions, it had a considerable number of false positives when identifying fraudulent ones. The recall of 0.73 for non-fraudulent and 0.55 for fraudulent transactions showed a similar trend.

I have also developed a Flask-based API, which enables real-time classification of transactions. This application allows for transaction data to be input, and in return, provides a fraud classification, making the models more applicable and accessible in real-world scenarios. The assignment showcased the complexities involved in designing a system for fraud detection. Balancing fraud detection with the minimization of false positives is a non-trivial task and may require more advanced or different approaches, such as anomaly detection or a balanced dataset, depending on the cost implications of false positives and negatives.

For future work, I can consider exploring more complex feature engineering, incorporating more sophisticated models or using ensemble methods. The inclusion of a larger dataset, if available, could also aid in improving the accuracy of our models, particularly with respect to detecting fraudulent transactions.

In conclusion, I have successfully built a fraud detection system that performs reasonably well given the constraints of the assignment. The results of this project have demonstrated the utility and potential of machine learning in aiding fraud detection efforts. It also underscored the need for continuous optimization and tuning of these systems to better balance the trade-off between false positives and fraud detection.