Vellore Institute of Technology Chennai

Internet of Things Domain Analyst

Project Report for the Mini Project titled:

Financial Fraud Detection using IoT and Machine Learning

Submitted by:

Pritish Gupta

Reg. No.: 21BAI1609

Introduction:

Financial fraud is a prevalent issue that plagues various industries, resulting in substantial financial losses and undermining consumer trust. The advent of Internet of Things (IoT) technology and the proliferation of interconnected devices along with growth in digital transactions have opened new avenues for fraudulent activities, necessitating innovative solutions to combat this emerging threat. This project aims to address this critical challenge by leveraging the power of IoT data and advanced machine learning techniques.

Research Background:

Financial fraud has been a longstanding concern, with traditional detection methods relying heavily on manual processes and rule-based systems. However, the exponential growth of digital transactions and the increasing sophistication of fraudsters have rendered these traditional approaches inadequate. The integration of IoT devices into financial ecosystems has further exacerbated the problem, as these devices generate vast amounts of data that can potentially harbor indicators of fraudulent activities. Combining IoT (Internet of Things) with machine learning offers several advantages for financial fraud detection. IoT devices, such as sensors and cameras, can gather real-time transaction data. This data provides a comprehensive view of financial activities, enabling quicker detection of anomalies. By leveraging IoT, banks can monitor user behavior patterns, for instance, analyzing how a user interacts with their account, their spending habits, and the devices they use can help identify suspicious activities. Machine learning algorithms can process vast amounts of data efficiently. They recognize complex patterns that might be impossible for humans to identify. These algorithms can pinpoint anomalous behavior indicative of fraud. Combining IoT-generated data with big data analytics allows for more accurate fraud detection. The system can correlate various data points to identify irregularities. With real-time insights, banks can take immediate preventive measures. For example, if an unusual transaction occurs, the system can trigger alerts or block the transaction before any financial loss occurs.

Motivation behind this Project:

The primary motivation behind this project is to develop a robust and intelligent system capable of detecting financial fraud in real-time, leveraging the wealth of data generated by IoT devices. By harnessing the power of machine learning algorithms, the project aims to identify complex patterns and anomalies that may be indicative of fraudulent activities. This proactive approach can significantly reduce financial losses, enhance consumer confidence, and maintain the integrity of financial systems.

Problem Statement:

Financial fraud can manifest in various forms, such as identity theft, credit card fraud, money laundering, and cybercrime. The challenge lies in effectively analyzing the massive volumes of data generated by IoT devices to uncover potentially fraudulent activities amidst legitimate transactions. Traditional rule-based systems struggle to keep pace with the ever-evolving tactics employed by fraudsters, necessitating the development of intelligent, adaptive, and scalable solutions.

Technical Gaps in Existing Solutions:

While there have been efforts to address financial fraud using machine learning techniques, existing solutions often fail to effectively integrate IoT data into the fraud detection process. Many traditional approaches rely solely on structured data from financial transactions, overlooking the valuable insights that can be gleaned from the rich contextual information provided by IoT devices. Additionally, existing solutions may struggle to handle the velocity, variety, and volume of data generated by IoT ecosystems, limiting their effectiveness in real-time fraud detection scenarios.

Methodology:

Dataset:

The dataset used for this project is a synthetic dataset generated using the simulator called PaySim. PaySim uses aggregated data from the private dataset to generate a synthetic dataset that resembles the normal operation of transactions and injects malicious behavior to later evaluate the performance of fraud detection methods.

PaySim simulates mobile money transactions based on a sample of real transactions extracted from one month of financial logs from a mobile money service implemented in an African country. The original logs were provided by a multinational company, who is a provider of mobile financial services which is currently running in more than 14 countries all around the world.

This synthetic dataset is scaled down 1/4 of the original dataset. This dataset has more than 6 million records and 11 features per record.

This is a sample of 1 row with headers explanation:

1,PAYMENT,1060.31,C429214117,1089.0, 28.69, M1591654462, 0.0, 0.0, 0, 0

step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).

type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.

amount - amount of the transaction in local currency.

nameOrig - customer who started the transaction

oldbalanceOrg - initial balance before the transaction

newbalanceOrig - new balance after the transaction.

nameDest - customer who is the recipient of the transaction

oldbalanceDest - initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).

newbalanceDest - new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).

isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control of customers' accounts and trying to empty the funds by transferring them to another account and then cashing out of the system.

isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.

Transactions which are detected as fraud are cancelled, so for fraud detection these columns (oldbalanceOrg, newbalanceOrig, oldbalanceDest, newbalanceDest) must not be used.

The dataset can be accessed from the link given below:

Source: https://www.kaggle.com/datasets/ealaxi/paysim1

Implementation of the analytical model:

This project has been implemented using Python programming language. There are several libraries used for implementation of this project:

NumPy:

NumPy (Numerical Python) is a powerful Python library designed for numerical computations and data manipulation. It provides a high-performance multidimensional array object called ndarray. These arrays can hold elements of the same type (usually numbers) and are indexed by positive integers. The shape of an array specifies its dimensions (number of elements along each axis). NumPy allows efficient manipulation of large datasets. It includes functions for linear algebra, Fourier transforms, and random number generation. NumPy enables element-wise operations on arrays of different shapes.

Pandas:

Pandas is a powerful Python library used for data manipulation and analysis. It is built on top of the NumPy library. It leverages NumPy's efficient array operations for data processing. It provides a convenient way to work with structured data, such as tabular data in the form of tables and time series data. Pandas offers functions for analyzing, cleaning, exploring, and manipulating data. It helps handle missing values, incorrect formats, and duplicates. We can manipulate and explore data by summarizing, filtering, and aggregating it. Using pandas, we can perform operations like merging, joining, and reshaping data.

Matplotlib:

Matplotlib is a powerful data visualization library for the Python programming language. It allows you to create various types of plots and charts. It is widely used for data visualization, scientific plotting, and creating publication-quality graphics.

Seaborn:

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn offers beautiful default styles and color palettes, making statistical plots more visually appealing. It works seamlessly with Pandas dataframes, allowing easy visualization and exploration of data. Using Seaborn library, we can quickly visualize data distributions, relationships, and patterns. We can create statistical plots such as bar plots, scatter plots, line plots, and more. It can visualize correlations and patterns in matrices.

Scikit-learn:

Scikit-learn (sklearn) is a powerful open-source machine learning library for Python. It provides a wide array of tools and algorithms for various tasks related to predictive data analysis. Scikit-learn is widely used in research, industry, and academia for machine learning tasks. It is a go-to library for both supervised and unsupervised learning.

It has these key features:

Classification: Identifying which category an object belongs to (e.g., spam detection, image recognition).

Regression: Predicting continuous-valued attributes (e.g., drug response, stock prices).

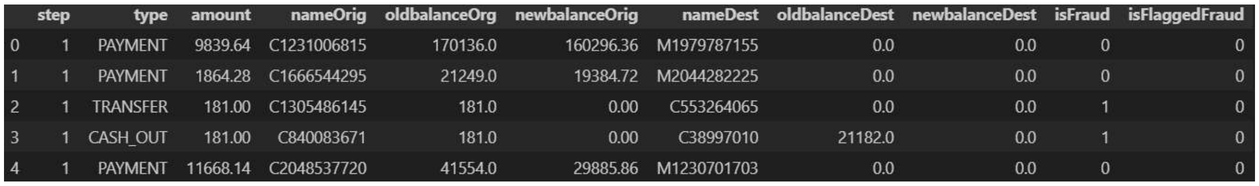
Clustering: Grouping similar objects (e.g., customer segmentation).

Dimensionality Reduction: Reducing the number of variables (e.g., visualization).

Model Selection: Comparing, validating, and choosing models.

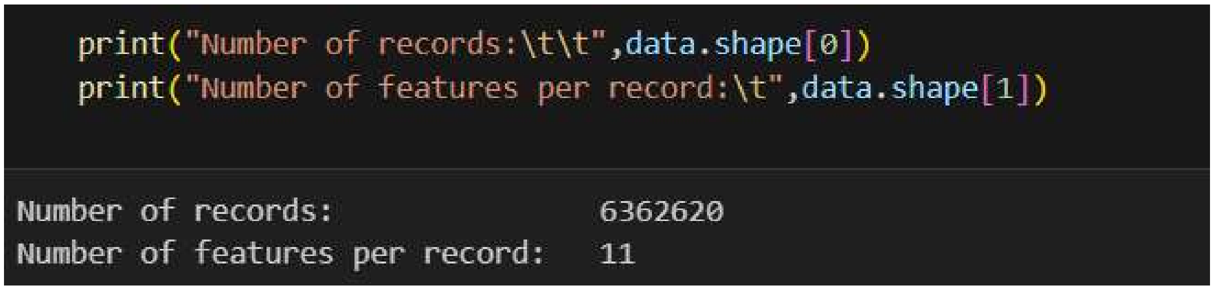
Preprocessing: Feature extraction and normalization.

The model takes input of transactions as comma separated values(csv) using Pandas. The model consists of data analysis and pre-processing modules and machine learning model.



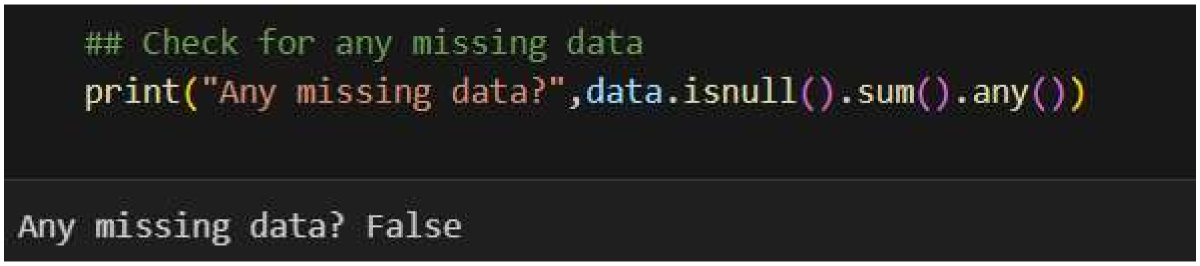
	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	0

After obtaining the data, the data is analyzed for missing values and features of the dataset is extracted.



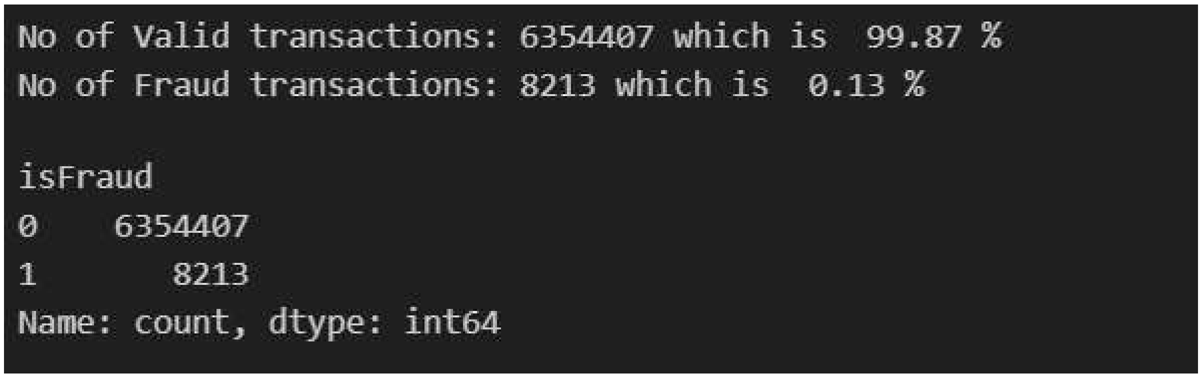
```python
print("Number of records:\t\t",data.shape[0])
print("Number of features per record:\t",data.shape[1])
```

```
Number of records:              6362620
Number of features per record:  11
```

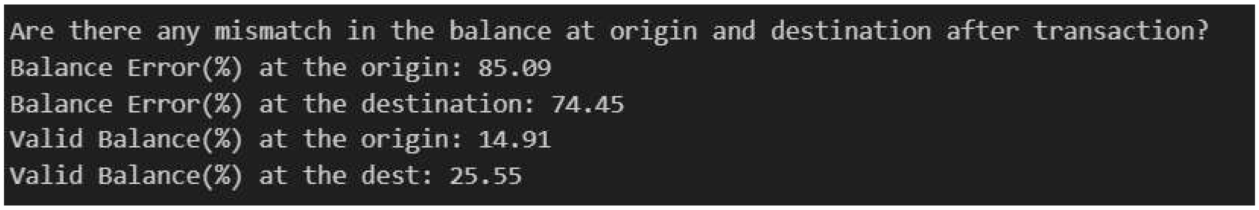


```python
## Check for any missing data
print("Any missing data?",data.isnull().sum().any())
```

```
Any missing data? False
```

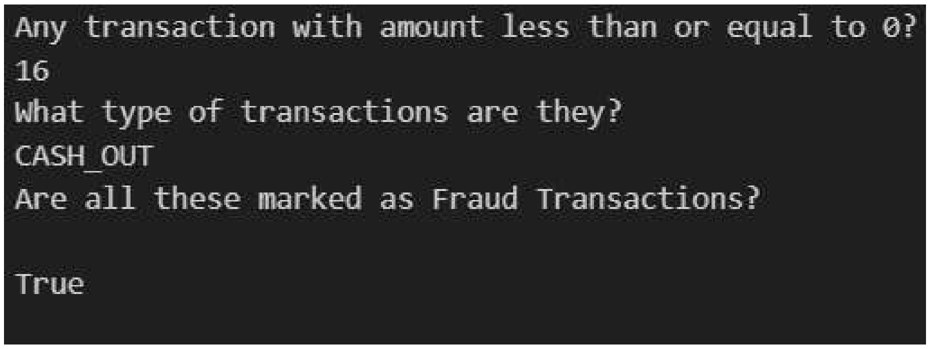


```
No of Valid transactions: 6354407 which is  99.87 %
No of Fraud transactions: 8213 which is  0.13 %

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

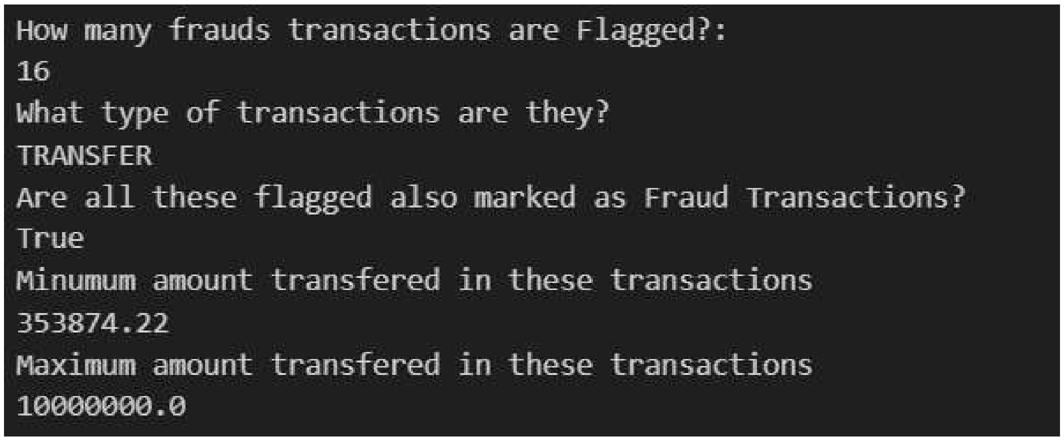


```
Are there any mismatch in the balance at origin and destination after transaction?
Balance Error(%) at the origin: 85.09
Balance Error(%) at the destination: 74.45
Valid Balance(%) at the origin: 14.91
Valid Balance(%) at the dest: 25.55
```

For this implementation we are focusing on Cashout and Transfer transactions as these are the transactions with the most intensity of fraud.



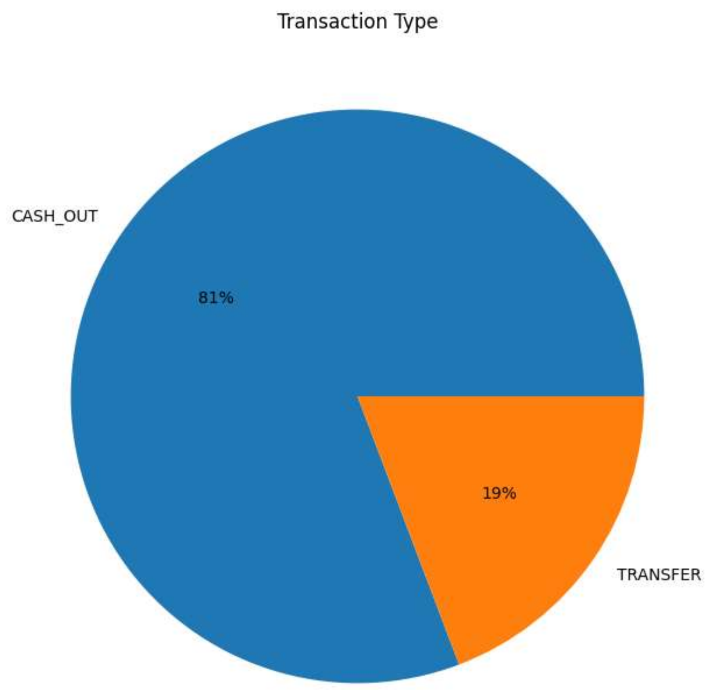
```
Any transaction with amount less than or equal to 0?
16
What type of transactions are they?
CASH_OUT
Are all these marked as Fraud Transactions?

True
```



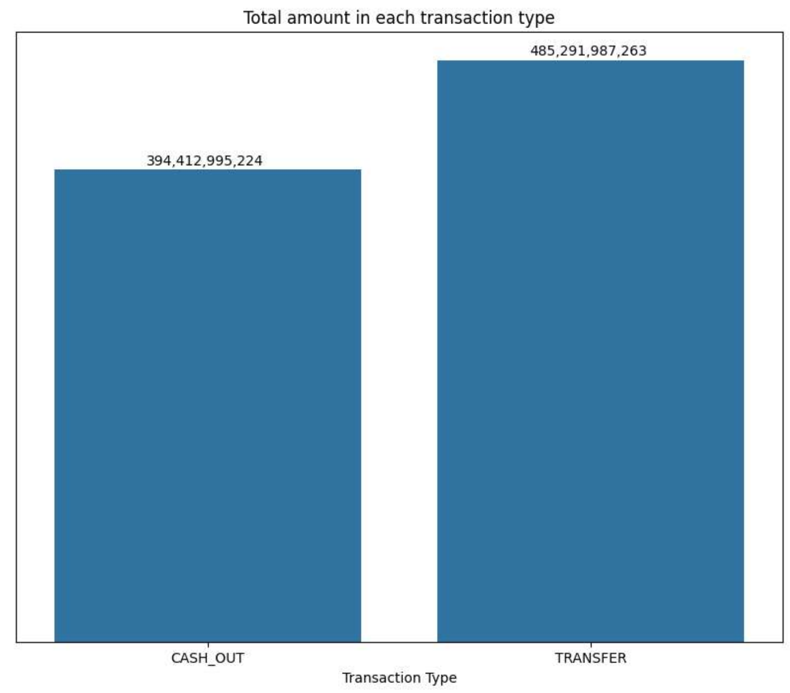
```
How many frauds transactions are Flagged?:
16
What type of transactions are they?
TRANSFER
Are all these flagged also marked as Fraud Transactions?
True
Minumum amount transfered in these transactions
353874.22
Maximum amount transfered in these transactions
10000000.0
```

Using Matplotlib and seaborn libraries we can do data visualization of this analysis.

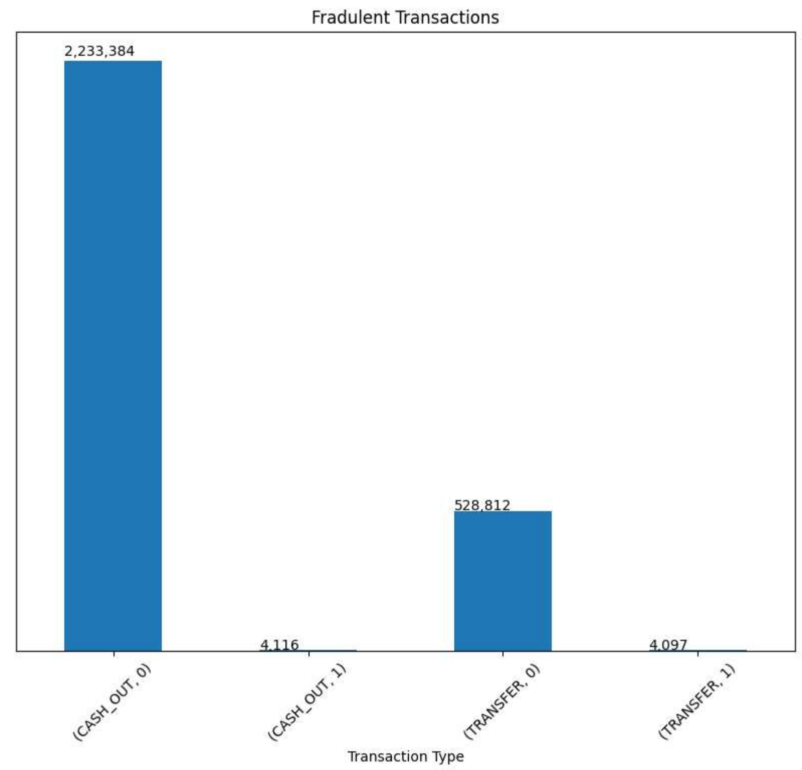
Pie Chart for the percentage of transaction type:



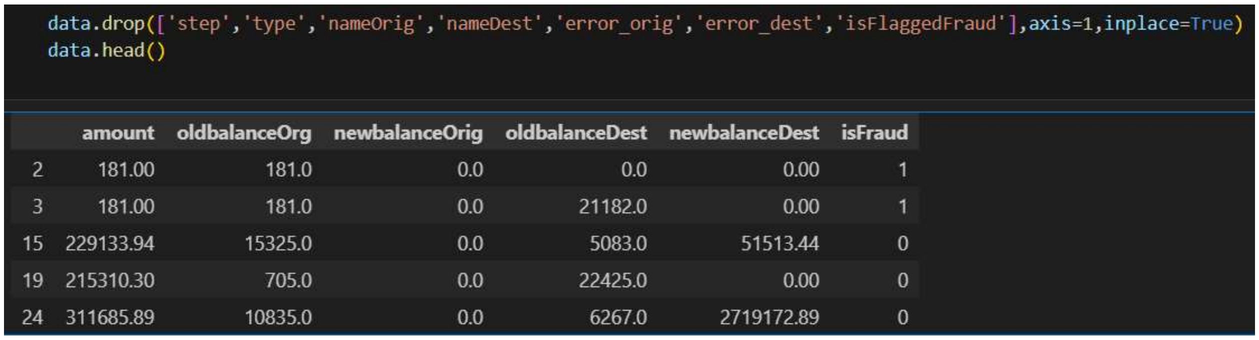
Bar chart for total amount in each transaction type:



Bar graph for Number of fraudulent transactions (denoted by 1) and legal transactions (denoted by 0) for each transaction type:



The dataset contains 11 features in which some of them may not be useful for model training. So, we can remove certain unwanted features to reduce the dimension of dataset.



```
data.drop(['step','type','nameOrig','nameDest','error_orig','error_dest','isFlaggedFraud'],axis=1,inplace=True)
data.head()
```

	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
2	181.00	181.0	0.0	0.0	0.00	1
3	181.00	181.0	0.0	21182.0	0.00	1
15	229133.94	15325.0	0.0	5083.0	51513.44	0
19	215310.30	705.0	0.0	22425.0	0.00	0
24	311685.89	10835.0	0.0	6267.0	2719172.89	0

Data Balancing:

In some datasets (like the one chosen here), one class may be significantly underrepresented compared to others. A fraud detection model has instances of fraud occur in only 0.5% of transactions.

With so few positive examples relative to negatives, the training model may not learn enough from the minority class. This imbalance can lead to poor prediction performance, especially for minority classes that are of particular interest such as detecting fraud. There are various techniques to overcome this:

Up-Sampling Techniques:

Naive Oversampling: This method involves duplicating examples from the minority class in the training dataset. By doing so, it balances the class distribution but doesn't provide additional information to the model.

SMOTE (Synthetic Minority Over-sampling Technique): SMOTE generates synthetic examples by interpolating between existing minority class samples. It creates new data points by considering the feature space between neighboring instances.

Down-Sampling Techniques:

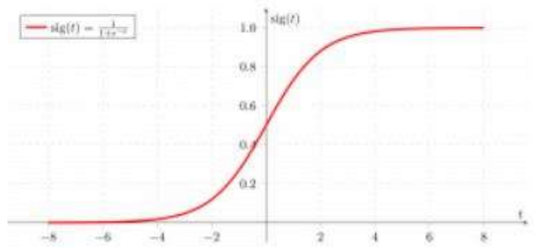
Down sampling involves reducing the number of samples in the majority class to match the number of samples in the minority class.

There are various machine learning algorithms used to understand the performance of different algorithms on the dataset:

Logistic Regression:

Logistic regression is a supervised machine learning algorithm used for classification tasks. Its primary goal is to predict the probability that an instance belongs to a given class (usually binary: 0 or 1). Logistic regression predicts the output of a categorical dependent variable. The outcome can be binary (e.g., Yes/No, 0/1, True/False).

$$\sigma(z) = \frac{1}{1-e^{-z}}$$



Sigmoid function

Logistic regression uses the sigmoid function (also called the logistic function) to map real-valued inputs to probabilities. The sigmoid function produces values between 0 and 1, forming an "S"-shaped curve. Given input features (independent variables), logistic regression computes a linear combination of these features. The linear combination is then transformed using the sigmoid function to produce a probability value. If the probability exceeds a threshold (usually 0.5), the instance is classified as Class 1; otherwise, it's classified as Class 0.

$$p(X; b, w) = \frac{e^{w \cdot X + b}}{1+e^{w \cdot X + b}} = \frac{1}{1+e^{-w \cdot X + b}}$$

Random Forest Classifier:

Random Forest algorithm is a powerful tree learning technique in Machine Learning. It works by creating several Decision Trees during the training phase. Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition. This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance. In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks) This collaborative decision-making process, supported by multiple trees with their insights, provides an example stable and precise results. Random forests are widely used for classification and regression functions, which are known for their ability to handle complex data, reduce overfitting, and provide reliable forecasts in different environments.

Random Forest leverages the power of ensemble learning by constructing an army of Decision Trees. These trees are like individual experts, each specializing in a particular aspect of the data. Importantly, they operate independently, minimizing the risk of the model being overly influenced by the nuances of a single tree. To ensure that each decision tree in the ensemble brings a unique perspective, Random Forest employs random feature selection. During the training of each tree, a random subset of features is chosen. This randomness ensures that each tree focuses on different aspects of the data, fostering a diverse set of predictors within the ensemble. The technique of bagging is a cornerstone of Random Forest's training strategy which involves creating multiple bootstrap samples from the original dataset, allowing instances to be sampled with replacement. This results in different subsets of data for each decision tree, introducing variability in the training process and making the model more robust. When it comes to making predictions, each decision tree in the Random Forest casts its vote. For classification tasks, the final prediction is determined by the mode (most frequent prediction) across all the trees. In regression tasks, the average of the individual tree predictions is taken. This internal voting mechanism ensures a balanced and collective decision-making process.

The following evaluation metrices are used for the evaluation of the implemented model:

Confusion Matrix:

A confusion matrix is a matrix that summarizes the performance of a machine learning model on a set of test data. It is a means of displaying the number of accurate and inaccurate instances based on the model's predictions.

F1 Score:

The F1 score is an essential evaluation metric commonly used in classification tasks within machine learning. It combines precision and recall into a single value.

Precision represents the accuracy of positive predictions. It calculates how often the model predicts correctly for positive values.

Precision = (True Positives) / (True Positives + False Positives)

Recall measures how well a model identifies actual positive cases. It is the ratio of true positive predictions to the total number of actual positive instances.

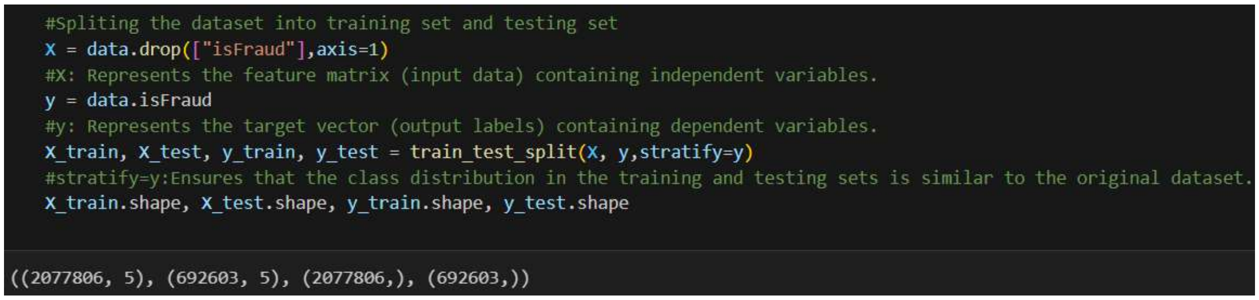
Recall = (True Positives) / (True Positives + False Negatives)

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

Accuracy:

Accuracy represents the overall correctness of the model's predictions. It measures the ratio of correctly predicted instances (true positives and true negatives) to the total number of instances.

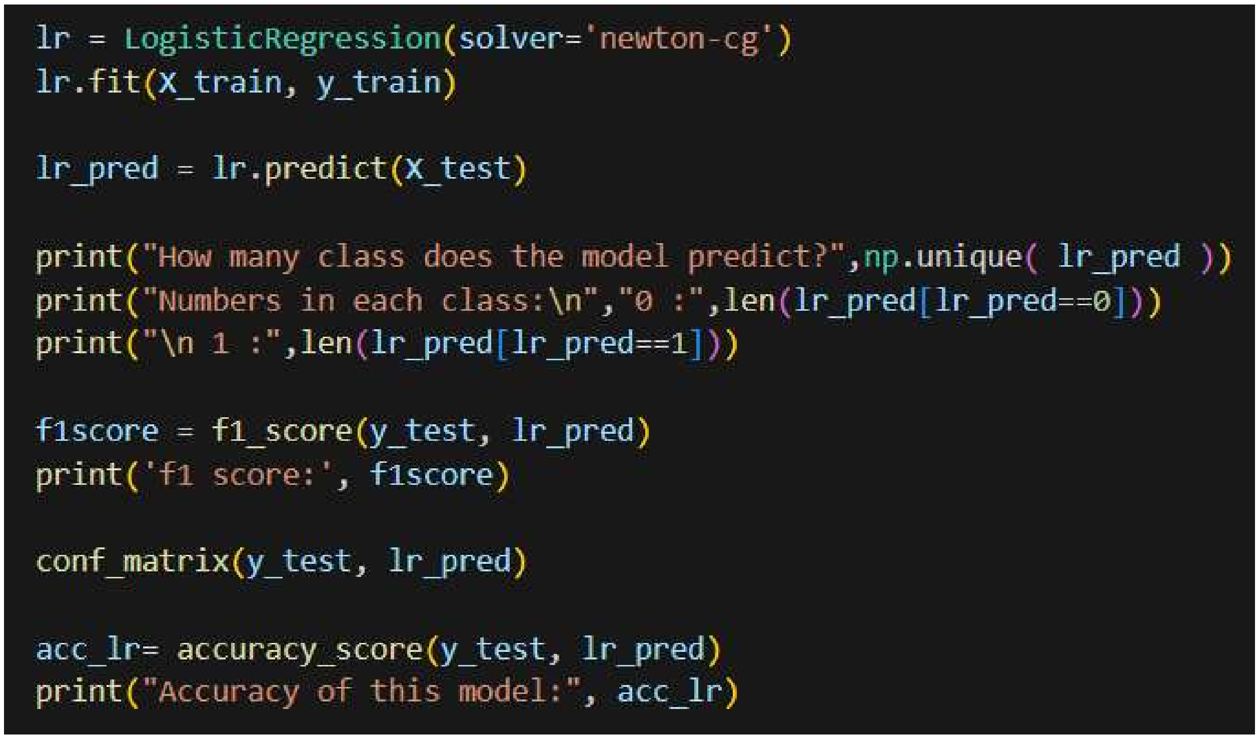
To train the model, the dataset is split into training set and testing set.



```
#Spliting the dataset into training set and testing set
X = data.drop(["isFraud"],axis=1)
#X: Represents the feature matrix (input data) containing independent variables.
y = data.isFraud
#y: Represents the target vector (output labels) containing dependent variables.
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify=y)
#stratify=y:Ensures that the class distribution in the training and testing sets is similar to the original dataset.
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((2077806, 5), (692603, 5), (2077806,), (692603,))
```

Then, we train the model on imbalanced dataset using LogisticRegression function from scikit-learn library and evaluate the model.

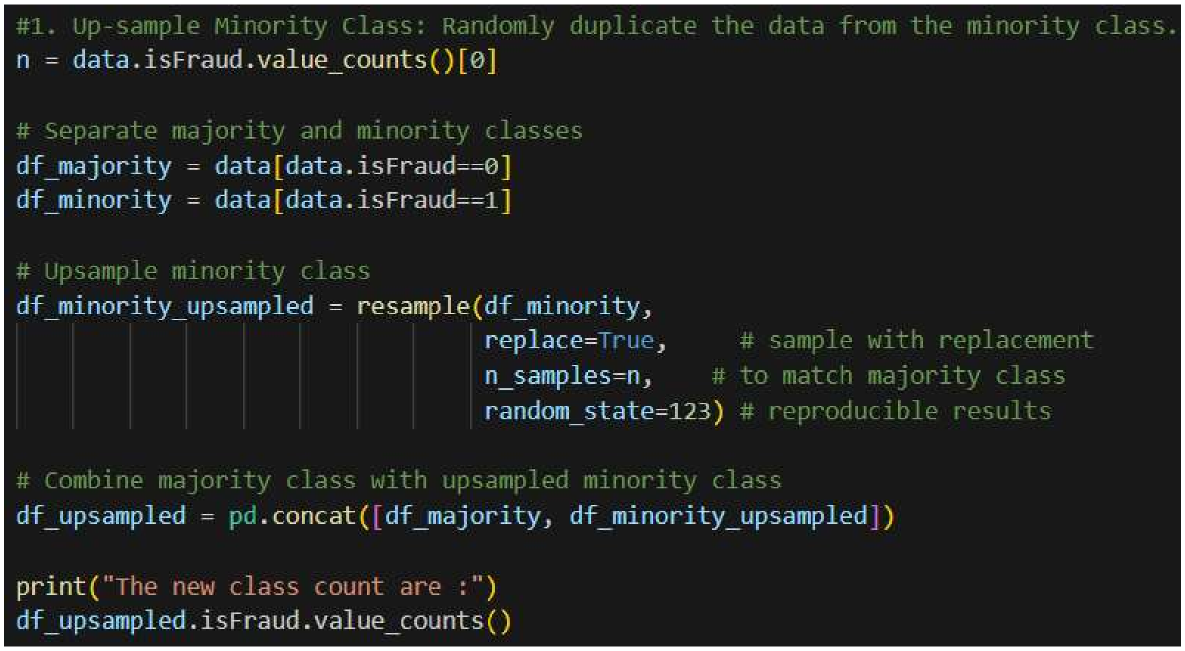


```
lr = LogisticRegression(solver='newton-cg')
lr.fit(X_train, y_train)

lr_pred = lr.predict(X_test)

print("How many class does the model predict?",np.unique( lr_pred ))
print("Numbers in each class:\n","0 :",len(lr_pred[lr_pred==0]))
print("\n 1 :",len(lr_pred[lr_pred==1]))

f1score = f1_score(y_test, lr_pred)
print('f1 score:', f1score)

conf_matrix(y_test, lr_pred)

acc_lr= accuracy_score(y_test, lr_pred)
print("Accuracy of this model:", acc_lr)
```

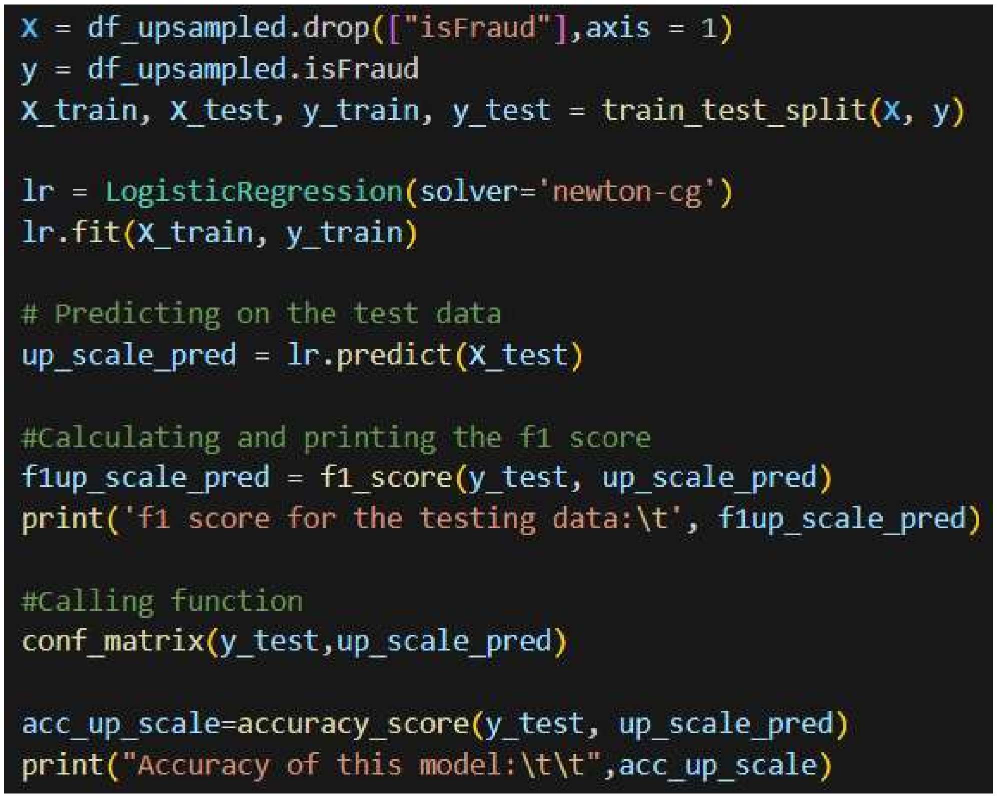
Now we train the model after balancing the dataset by

1) Upscaling minority class and

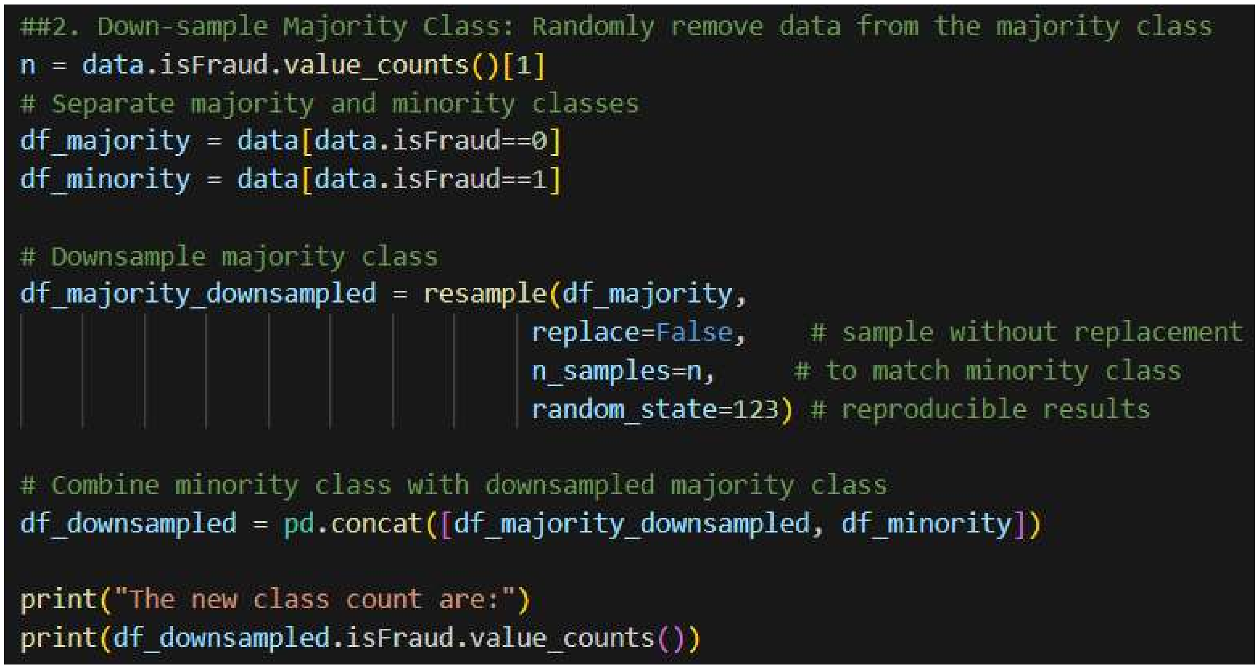
2) Downscaling majority class.



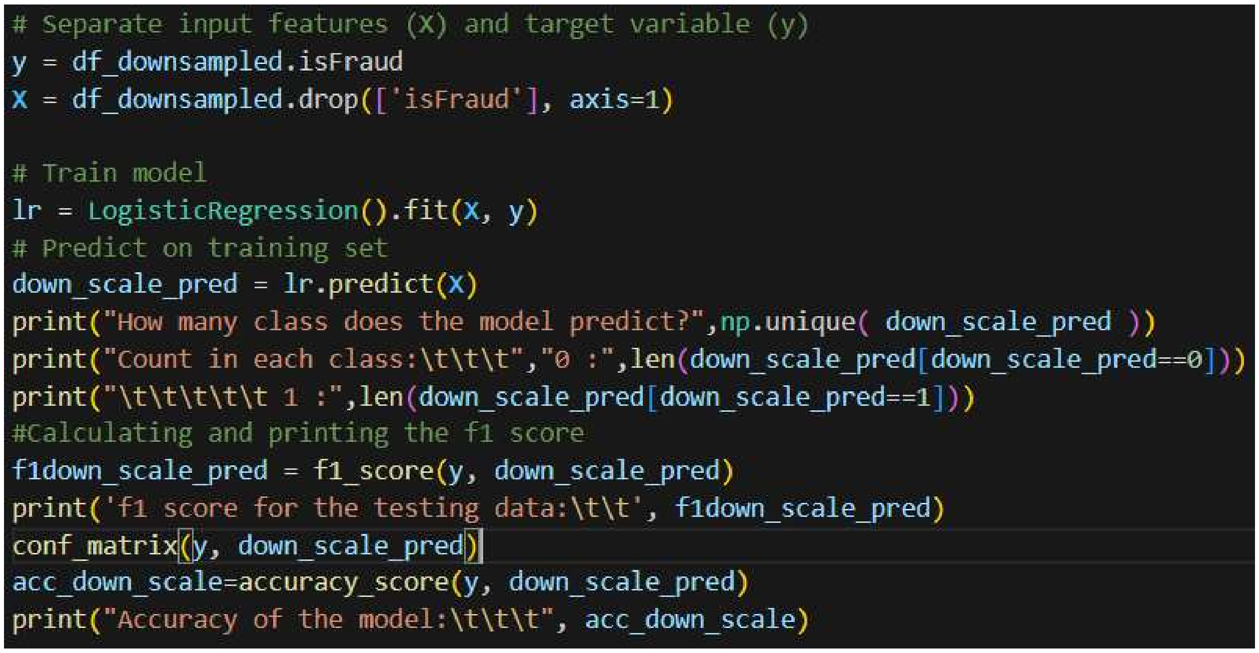
```
#1. Up-sample Minority Class: Randomly duplicate the data from the minority class.
n = data.isFraud.value_counts()[0]

# Separate majority and minority classes
df_majority = data[data.isFraud==0]
df_minority = data[data.isFraud==1]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=n,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

print("The new class count are :")
df_upsampled.isFraud.value_counts()
```



```
X = df_upsampled.drop(["isFraud"],axis = 1)
y = df_upsampled.isFraud
X_train, X_test, y_train, y_test = train_test_split(X, y)

lr = LogisticRegression(solver='newton-cg')
lr.fit(X_train, y_train)

# Predicting on the test data
up_scale_pred = lr.predict(X_test)

#Calculating and printing the f1 score
f1up_scale_pred = f1_score(y_test, up_scale_pred)
print('f1 score for the testing data:\t', f1up_scale_pred)

#Calling function
conf_matrix(y_test,up_scale_pred)

acc_up_scale=accuracy_score(y_test, up_scale_pred)
print("Accuracy of this model:\t\t",acc_up_scale)
```

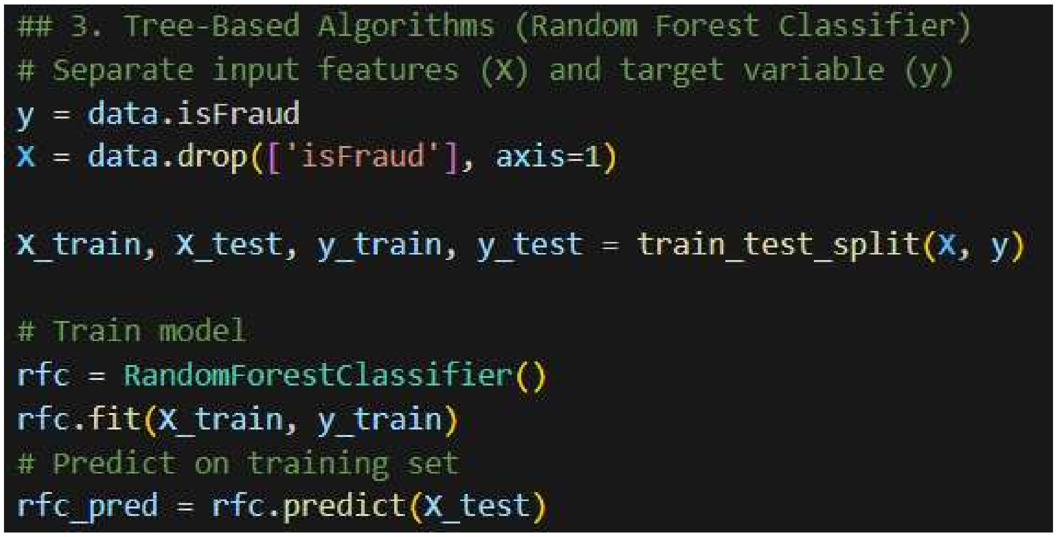


```
##2. Down-sample Majority Class: Randomly remove data from the majority class
n = data.isFraud.value_counts()[1]
# Separate majority and minority classes
df_majority = data[data.isFraud==0]
df_minority = data[data.isFraud==1]

# Downsample majority class
df_majority_downsampled = resample(df_majority,
                                   replace=False,    # sample without replacement
                                   n_samples=n,     # to match minority class
                                   random_state=123) # reproducible results

# Combine minority class with downsampled majority class
df_downsampled = pd.concat([df_majority_downsampled, df_minority])

print("The new class count are:")
print(df_downsampled.isFraud.value_counts())
```

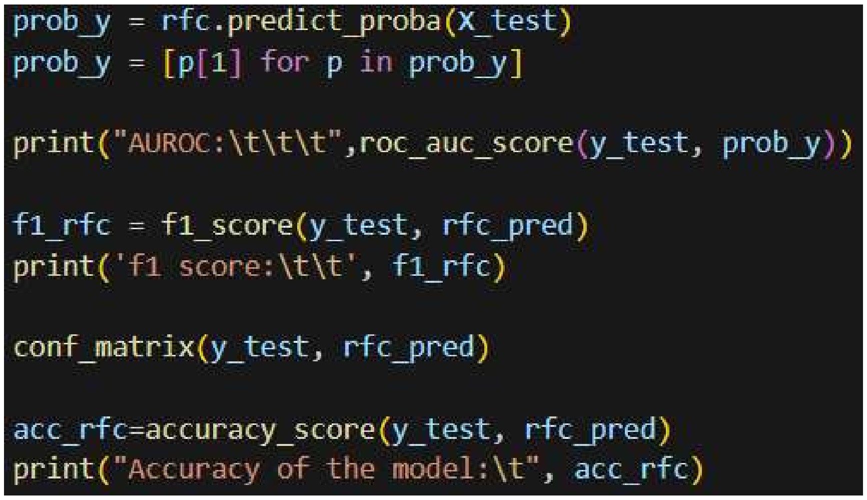


```
# Separate input features (X) and target variable (y)
y = df_downsampled.isFraud
X = df_downsampled.drop(['isFraud'], axis=1)

# Train model
lr = LogisticRegression().fit(X, y)
# Predict on training set
down_scale_pred = lr.predict(X)
print("How many class does the model predict?",np.unique( down_scale_pred ))
print("Count in each class:\t\t\t","0 :",len(down_scale_pred[down_scale_pred==0]))
print("\t\t\t\t\t 1 :",len(down_scale_pred[down_scale_pred==1]))
#Calculating and printing the f1 score
f1down_scale_pred = f1_score(y, down_scale_pred)
print('f1 score for the testing data:\t\t', f1down_scale_pred)
conf_matrix(y, down_scale_pred)
acc_down_scale=accuracy_score(y, down_scale_pred)
print("Accuracy of the model:\t\t\t", acc_down_scale)
```

Implementation of model using Random Forest classifier on imbalanced dataset:



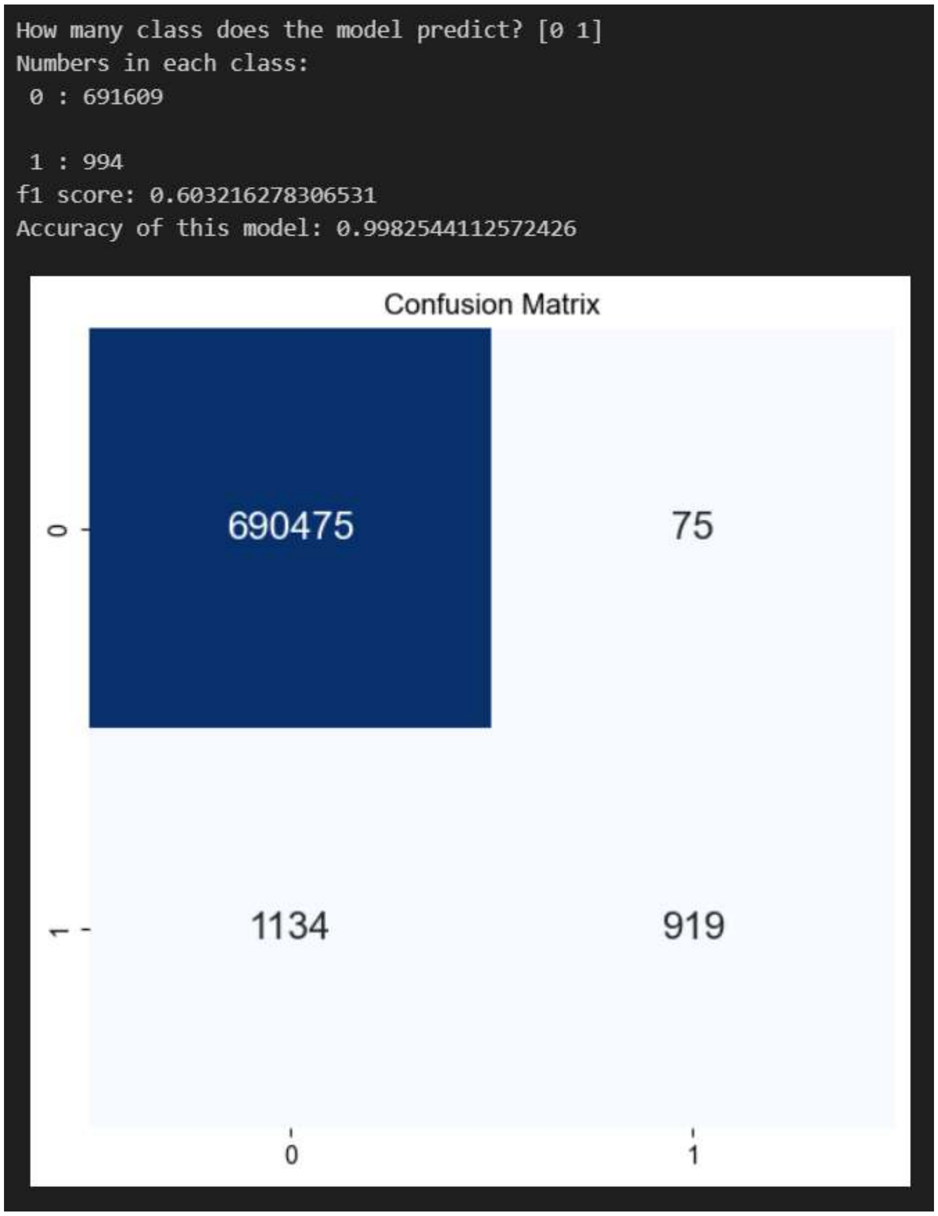
```
## 3. Tree-Based Algorithms (Random Forest Classifier)
# Separate input features (X) and target variable (y)
y = data.isFraud
X = data.drop(['isFraud'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y)

# Train model
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
# Predict on training set
rfc_pred = rfc.predict(X_test)
```



```
prob_y = rfc.predict_proba(X_test)
prob_y = [p[1] for p in prob_y]

print("AUROC:\t\t\t",roc_auc_score(y_test, prob_y))

f1_rfc = f1_score(y_test, rfc_pred)
print('f1 score:\t\t', f1_rfc)

conf_matrix(y_test, rfc_pred)

acc_rfc=accuracy_score(y_test, rfc_pred)
print("Accuracy of the model:\t", acc_rfc)
```

Results and Discussions:

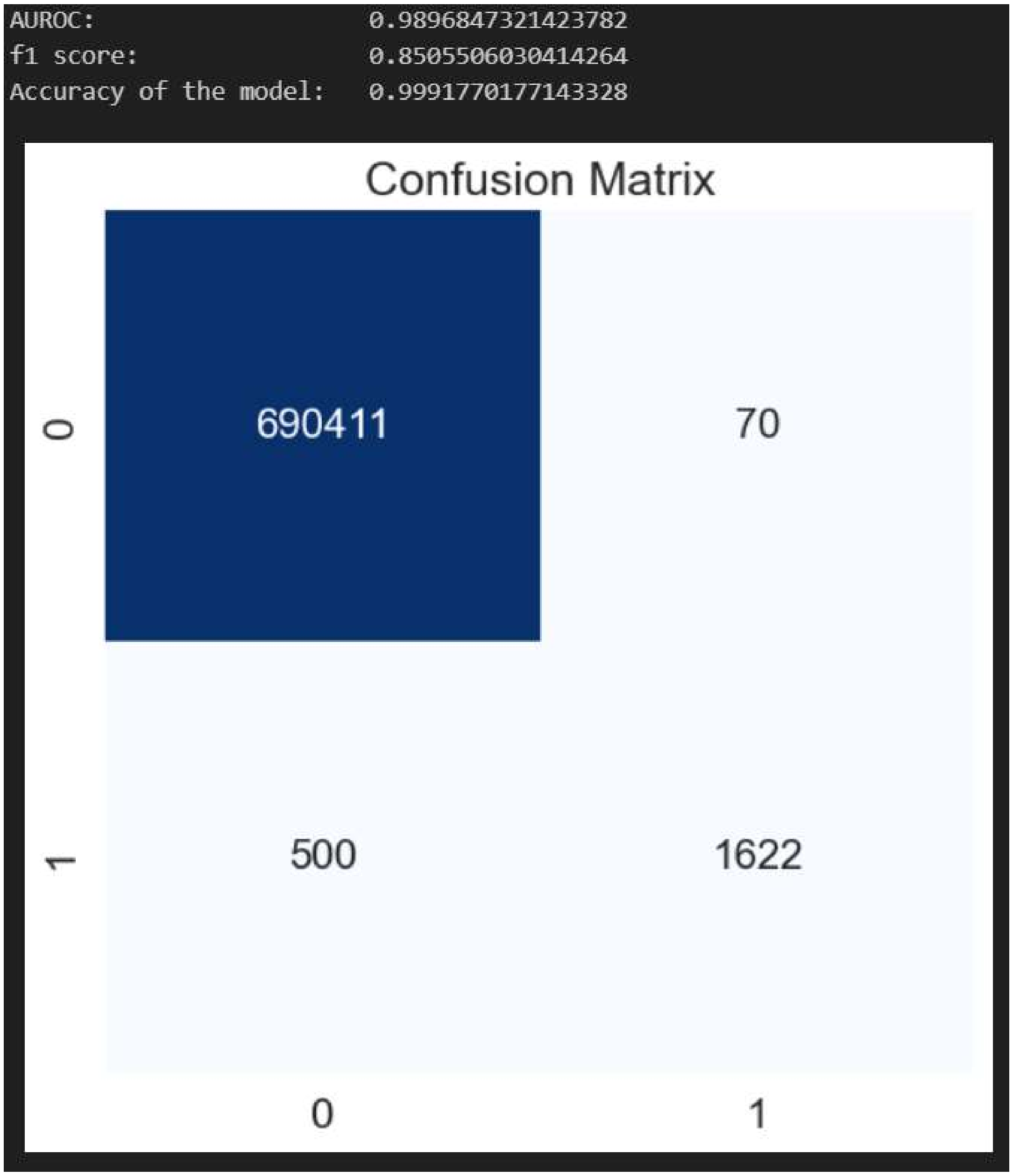
When training the model using logistic regression on the Imbalanced dataset, we get the following results:



```
How many class does the model predict? [0 1]
Numbers in each class:
 0 : 691609

 1 : 994
f1 score: 0.603216278306531
Accuracy of this model: 0.9982544112572426
```

As we can see from the figure above, the F1 score of the model is around 0.6 which is not satisfactory.

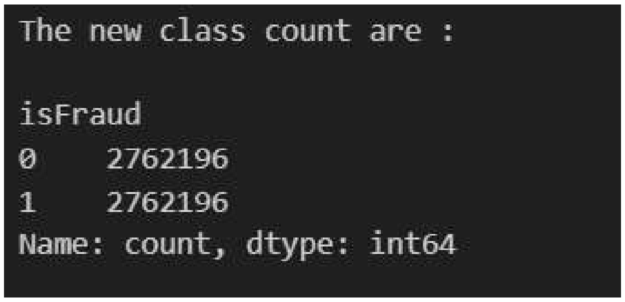
When training the model using random forest on the Imbalanced dataset, we get the following results:



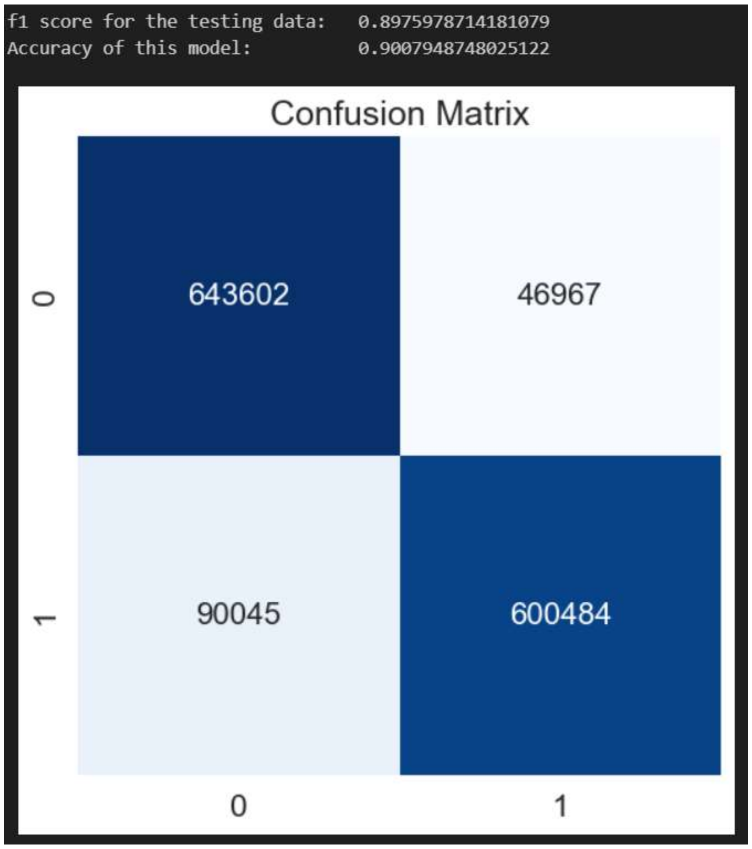
```
AUROC:                      0.9896847321423782
f1 score:                   0.8505506030414264
Accuracy of the model:      0.9991770177143328
```

We get a better F1 score (around 0.85) for random forest classifier compared to logistic regression, but it can be improved further by using balancing techniques on the dataset.

After Upscaling the imbalanced data and training the model using logistic regression,

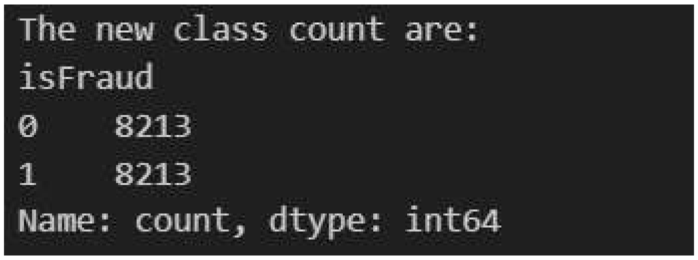


```
The new class count are :

isFraud
0    2762196
1    2762196
Name: count, dtype: int64
```

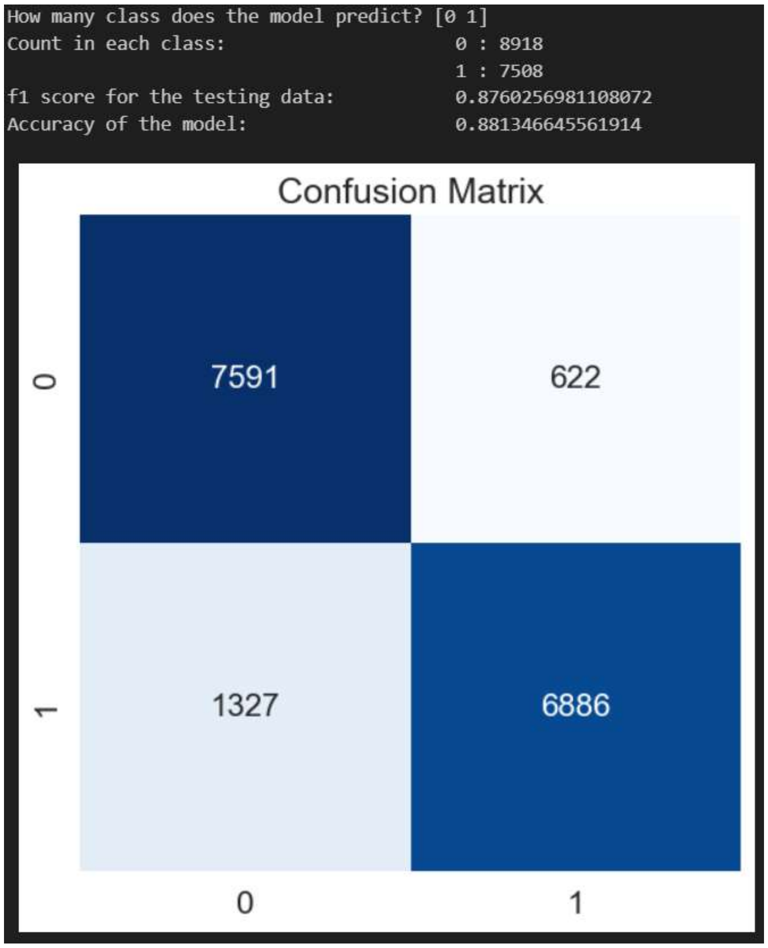


```
f1 score for the testing data:    0.8975978714181079
Accuracy of this model:           0.9007948748025122
```

After Downscaling the imbalanced data and training the model using logistic regression,

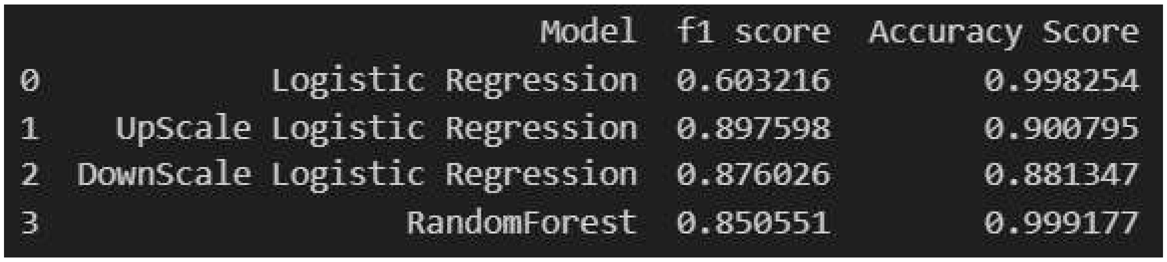


```
The new class count are:
isFraud
0    8213
1    8213
Name: count, dtype: int64
```


```
How many class does the model predict? [0 1]
Count in each class:                    0 : 8918
                                        1 : 7508
f1 score for the testing data:          0.8760256981108072
Accuracy of the model:                  0.881346645561914
```

We can observe that the F1 score of the model improves significantly (around after balancing the dataset and training the model using logistic regression which performs even better than random forest classifier on imbalanced dataset.

Comparing all the model performance:


```
                          Model  f1 score  Accuracy Score
0           Logistic Regression  0.603216        0.998254
1    UpScale Logistic Regression  0.897598        0.900795
2  DownScale Logistic Regression  0.876026        0.881347
3                  RandomForest  0.850551        0.999177
```

So, we can conclude that the performance of the fraud detection model which gets data using IoT can improve significantly when the transaction data is analyzed and preprocessed appropriately and using machine learning to facilitate fraud detection on huge number of transactions occurring daily.