CREDIT CARD FRAUD DETECTION

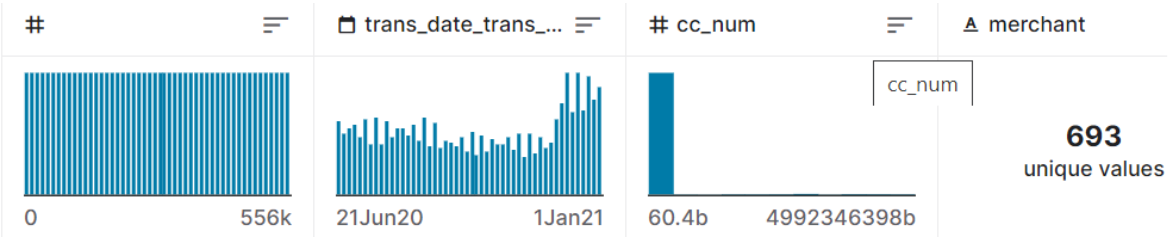
PHASE 4

The primary goal of this project is to develop a machine learning model that can effectively detect fraudulent credit card transactions.

Dataset: The project typically involves working with a dataset containing legitimate and fraud transactions from the duration 1st Jan 2019 - 31st Dec 2020. It covers credit cards of 1000 customers doing transactions with a pool of 800 merchants from Kaggle. This dataset contains the sequence of transaction information like trans_date_trans_time, cc_num, merchant, category, amt, firstname, lastname, genderstreet, city, state, zip, latitude, longitude, city_pop, job, trans_num, unix_time, merch_latitude,merch_longitude is_fraud.

STEP 1:Data Loading: Load the dataset into your preferred programming environment .We can use libraries like Pandas to read the dataset. It involves loading the raw transaction data from data source into data analysis tool or programming environment.

STEP 2 : Data Processing: - For our application we have trained the best model with the column contains the sequence of transaction informations .And the output or independent variable is Class which contains 0 or 1. Class 0 represents that the transaction is genuine and 1 represents that it is a fraudulent transaction. The model studies the specific behavior and predicts an accurate result.



STEP 3: Data Sampling: - Here in we have Over-Sampling and Under-Sampling. We have used Over-Sampling in our project. We could have used Under-Sampling but the data we have is highly imbalanced data giving us only 492 fraud transactions out of 2 lakh transactions. And Under-Sampling uses imitation to decrease the no. of high-class samples equal to the number of low-class samples and that would create a problem for prediction. Thus, we used OverSampling which increases the number of low-class samples equal to high-class samples for balancing with low-class. Lastly, our data is balanced using Over-Sampling.

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.3, random_state=42)
```

STEP 4: Model Training: - Model validation is very important to ensure the verification of the model predictions. That's the reason we went for Training after balancing this data, we went for dividing the data into 2 parts namely, training data and testing data. The training data was used for training the machine learning models which were around 75% of the total data available.

RandomForestClassifier

```
RandomForestClassifier(random_state=42)
```

STEP 5: Model Testing: - The remaining 25% of data was used to test the trained models i.e., for prediction purposes.

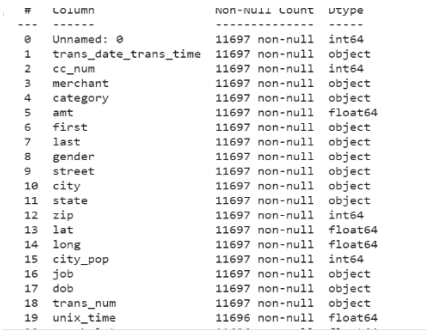
STEP 6: Evaluation: - Our main aim was to compare the best model i.e., Random Forest Classifier which comes under ensemble learning methods and is a bagging method with other three boosting methods that included AdaBoost, CatBoost, and XGBoost Classifiers. The result was clear that Catboost gave us high accuracy in very less time.

```
Accuracy: 0.9998095238095238
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      3509
         1.0       1.00      1.00      1.00      1741

    accuracy                           1.00      5250
   macro avg       1.00      1.00      1.00      5250
weighted avg       1.00      1.00      1.00      5250
```

STEP 7: Fraud/ Genuine: - The model after getting the value predicts the transactions and gives an accurate result to the user whether it is Fraud or Genuine.

1. **Import Libraries:**
   Import the required python libraries ,especially Pandas for data manipulation and analysis.
2. **Load the Dataset:**
   Load the raw transaction data from your data source into your chosen data analysis tool or programming environment.
3. **Explore the Dataset:**

Includes initial data inspection ,check for missing values and get basic statistics.



#	Column	Non-Null Count	Dtype
0	Unnamed: 0	11697 non-null	int64
1	trans_date_trans_time	11697 non-null	object
2	cc_num	11697 non-null	int64
3	merchant	11697 non-null	object
4	category	11697 non-null	object
5	amt	11697 non-null	float64
6	first	11697 non-null	object
7	last	11697 non-null	object
8	gender	11697 non-null	object
9	street	11697 non-null	object
10	city	11697 non-null	object
11	state	11697 non-null	object
12	zip	11697 non-null	int64
13	lat	11697 non-null	float64
14	long	11697 non-null	float64
15	city_pop	11697 non-null	int64
16	job	11697 non-null	object
17	dob	11697 non-null	object
18	trans_num	11697 non-null	object
19	unix_time	11696 non-null	float64

4.**Split Data into Features and Target**:

Split the dataset into training and testing sets to evaluate your model's performance. A common split is 70-30 or 80-20.

```
df = df.dropna(subset=['is_fraud'])
X = df.drop(['is_fraud', 'trans_date_trans_time'], axis=1)
y = df['is_fraud']
```

```
non_numeric_columns = ['first', 'last', 'gender', 'street', 'city',
'state', 'job', 'dob', 'trans_num']
X = X.drop(non_numeric_columns, axis=1)
```

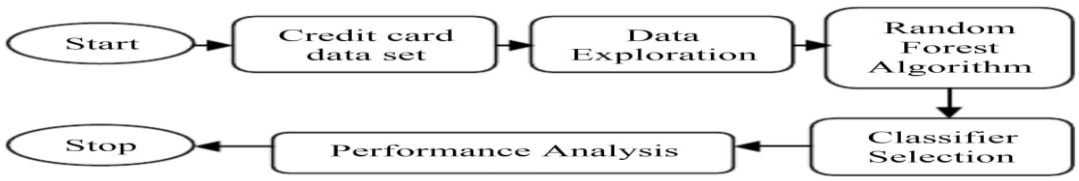
5.Perform one-hot encoding for categorical columns merchant and bold:

```
X = pd.get_dummies(X, columns=['merchant', 'category'],
drop_first=True)
```

6.Make predictions on the test set:

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

7.Data cleaning:



Handling missing values, Data cleaning, Feature selection and Data transformation.

1. data['amt'].fillna(data['amt'].mean(), inplace=True): This line is filling missing values in the 'amt' column with the mean value of the 'amt' column. It's a common strategy for handling missing numerical data. Text

2. data.drop_duplicates(inplace=True): This line removes duplicate rows from the dataset. Duplicates can be undesirable in a dataset as they can introduce bias and redundancy.

3. selected_columns = ['trans_date_trans_time', 'cc_num', 'merchant', 'category', 'amt', 'is_fraud']: Here, you're selecting a subset of columns from the dataset that are relevant for your analysis, including transaction date/time, credit card number, merchant, transaction category, transaction amount, and the target variable 'is_fraud.'

4. data = data[selected_columns]: This line creates a new DataFrame containing only the selected columns, effectively reducing the dataset to the chosen features.

5. data['trans_date_trans_time'] = pd.to_datetime(data['trans_date_trans_time']): You're converting the 'trans_date_trans_time' column to a datetime format, which can be useful for time-based analysis.

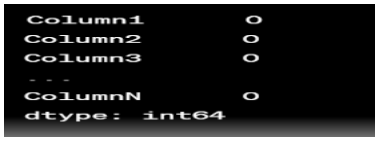
6. from sklearn.preprocessing import StandardScaler: You're importing the StandardScaler from Scikit-Learn, which is used for standardizing or scaling numerical features.

7. scaler = StandardScaler(): This line initializes the StandardScaler.

8. data['amt'] = scaler.fit_transform(data[['amt']]): You're using the StandardScaler to scale the 'amt' column. This step is important to ensure that all numerical features are on the same scale, which is often required for many machine learning algorithms to work effectively.

8. Handling Imbalanced Data:

Credit card fraud datasets are often highly imbalanced, with a small number of fraudulent transactions. You can oversample the minority class using techniques like SMOTE (Synthetic Minority Over-sampling Technique)



1. smote = SMOTE(sampling_strategy=0.5): This line initializes the Synthetic Minority Over-sampling Technique (SMOTE) from the imbalanced-learn (imblearn) library. SMOTE is used to address class imbalance by oversampling the minority class (fraudulent transactions) to achieve a specified sampling strategy. In this case, you've set the sampling strategy to 0.5, which means you want to increase the number of fraudulent transactions to be 50% of the number of non-fraudulent transactions.

2. X_resampled, y_resampled = smote.fit_resample(X, y): This line applies SMOTE to the features (X) and target (y) data. It generates synthetic samples of the minority class (fraudulent transactions) to balance the class distribution. The resulting X_resampled and y_resampled are the oversampled datasets with a balanced class distribution. These datasets can be used for model training to mitigate the impact of class imbalance on your fraud detection model.

After applying SMOTE, you have balanced datasets, which can improve the performance of your machine learning model, especially when dealing with imbalanced classes. You can proceed with training and evaluating your fraud detection model using these resampled datasets.

9. Splitting Data:

Divide the dataset into training and testing sets for model evaluation

1. from sklearn.model_selection import train_test_split: This line imports the train_test_split function from the Scikit-Learn library, which is used to split your dataset into training and testing subsets.

2. X = data.drop('is_fraud', axis=1): Assuming data is your dataset, this line creates a DataFrame X containing all the features (attributes) except the 'is_fraud' column. This separation is essential as 'is_fraud' is typically the target variable you want to predict.

3. y = data['is_fraud']: This line creates a Series y containing the 'is_fraud' column, which is your target variable, i.e., the labels indicating whether a transaction is fraudulent or not.

4. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42): This line splits your dataset into training and testing subsets. Here's what the arguments mean:

- X and y are the data and target variables to be split.

- test_size is set to 0.3, which means that 30% of the data will be reserved for testing, while 70% will be used for training, random_state is set to 42 to ensure reproducibility. The random state is used for shuffling the data before the split, so setting it to a specific value guarantees that the same split can be reproduced later.

The project output consists of a trained machine learning model capable of detecting credit card fraud, along with comprehensive reports, visualizations, and insights into fraud patterns. Regular model updates and improvements are necessary to adapt to evolving fraud techniques and maintain effective fraud detection.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
df = pd.read_csv("fraudTest.csv")
print(df.head())
print(df.info())
print(df.isnull().sum())
print(df.describe())
print(df['is_fraud'].value_counts())
df = df.dropna(subset=['is_fraud'])
X = df.drop(['is_fraud', 'trans_date_trans_time'], axis=1)
y = df['is_fraud']
non_numeric_columns = ['first', 'last', 'gender', 'street', 'city',
'state', 'job', 'dob', 'trans_num']
X = X.drop(non_numeric_columns, axis=1)
X = pd.get_dummies(X, columns=['merchant', 'category'],
drop_first=True)
X['amt'].fillna(X['amt'].mean(), inplace=True)
X.drop_duplicates(inplace=True)
selected_columns = ['cc_num', 'merchant', 'category', 'amt']
X = X[selected_columns]
scaler = StandardScaler()
X[['amt']] = scaler.fit_transform(X[['amt']])
smote = SMOTE(sampling_strategy=0.5)  # Adjust the sampling_strategy as
needed
X_resampled, y_resampled = smote.fit_resample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.3, random_state=42)
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
classification_report = classification_report(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report)
```