Université Mohammed Premier
Faculté Pluridisciplinaire
Département d'Informatique
◇ Nador ◇

Project Report

Credit Card Fraud Detection For Classifying Legitimate and Fraudulent Transactions

---

Realized by :
Aurag Fouad

Supervised by :
Pr. Anas EL ANSARI

Academic year 2023 - 2024

Contents

List of Figures

General Introduction

Context

In today's digital age, the use of credit cards for financial transactions has become commonplace, offering convenience and flexibility to consumers. Credit cards have revolutionized the way we conduct financial transactions, providing individuals and businesses with unprecedented convenience and purchasing power. For that they become an essential tool in our modern economy, empowering consumers to make purchases beyond their immediate financial means. They offer numerous benefits, such as enabling online shopping, providing a secure alternative to carrying cash, and offering rewards and cashback programs. Moreover, credit cards facilitate the growth of businesses by ensuring seamless transactions, reducing reliance on cash, and enabling global commerce.

Challenges

Alongside the many advantages of credit cards, there exists a pervasive threat: credit card fraud. Fraudulent activities can have severe consequences, both for individuals and financial institutions. Unauthorized transactions can result in financial losses, damaging all the involved parts, and prolonged legal battles for victims. Moreover, credit card fraud has far-reaching implications for banks and businesses, leading to significant financial losses, increased operational costs, and compromised customer trust.

The need for Effective Fraud Detection:

Traditional methods of manual verification and rule-based systems have proven insufficient in combating the ever-evolving techniques employed by fraudsters. As a result, financial institutions are turning to advanced technologies, particularly machine learning and data analytics, to develop robust systems to detect and prevent credit card fraud Transactions.

Objectives

The objective of this project is to address the pressing need for effective credit card fraud detection by developing a machine learning model capable of accurately identifying fraudulent transactions, then deploy this model in a web application, to enable Financial institution the ability to detect fraudulant transaction before it's too late. Following the **CRISP-DM** Process we can address the main objectives of this project as follow:

1. Get use of the different Techniques of Exploratory Data Analysis and Data-PreProcessing in order to get the most of data, and prepare it for further Analysis.
2. Explore the different Techniques of handling an imbalanced Dataset.
3. Develop and Evaluate a set of machine learning algorithms, and choose the best preforming one.

4. Develop a strong knowledge in web development by deploy the chosen model in a web application, for public use.

Report structure

This report is Organized into 2 chapters :

Chapter 1, "Methodology," presents the systematic approach we followed to address the challenge of fraud detection. In this chapter, we dive into the CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology, which forms the backbone of our data-driven project.

Chapter 2, "Implementation," takes us through the practical application of the knowledge acquired in the methodology phase. We showcase the diverse modeling techniques utilized to build and evaluate our fraud detection model. Tune the model and finalize it, then deploy it in a web application.

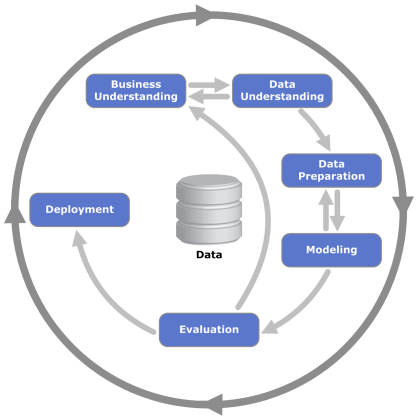
this report encompasses a comprehensive journey through the world of fraud detection. We employ a structured methodology to ensure every step aligns with the project's goals, leading to robust and reliable results. By bridging the gap between theory and practice, our implementation chapter offers valuable insights into the real-world application of our fraud detection model.

Chapter 1

Methodology

1.1 Introduction

The success of any data-driven project hinges on a well-defined and structured methodology. In this chapter, we present the systematic approach we followed to develop an accurate and efficient fraud detection model. Our methodology is a reflection of the **CRISP-DM** Process.



in this chapter we will talk deeply about the **CRISP-DM** Process, and how can we use this process to solve Real-World Data Mining Problems.

1.2 Business Understanding

Business Understanding is the foundational step in the CRISP-DM Process. It involves comprehending the business problem at hand and establishing a clear understanding of the organization's objectives and requirements.

During this phase, we closely collaborate with the client to gather insights into their specific needs, challenges, and opportunities. By gaining a deep understanding of the business context, we can align our data mining efforts with the goals of the organization

and ensure that our analytical solutions are not only technically accurate but also highly relevant and actionable.

In this stage we can also contact experts of a certain domain, to help us get the necessary knowledge needed to attack any problem.

1.3 Data Understanding

Data understanding is a fundamental step in the data mining process that lays the groundwork for extracting meaningful insights from the data set. During this phase, we thoroughly examine and explore the data to gain familiarity with its structure, contents, and quality.

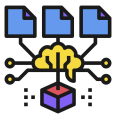
It involves both quantitative and qualitative analyses to comprehend the underlying patterns, relationships, and potential challenges present in the data. Exploring summary statistics, visualizations, and descriptive measures help us grasp the distribution of variables, identify missing values, and detect outliers. By comprehending the data, we can formulate relevant questions, validate assumptions, and determine the suitability of the data set for our specific analysis or modeling objectives.

A comprehensive understanding of the data empowers us to make informed decisions throughout the data mining process, leading to more accurate and meaningful outcomes.

1.4 Data Preparation

Data preparation is a crucial phase in the data mining process, where we transform and clean the raw data into a format suitable for analysis and modeling. This step is essential as it directly impacts the quality and effectiveness of the final results. During data preparation, we address various data challenges, such as missing values, outliers, and inconsistencies. Techniques like imputation, removal, or interpolation are employed to handle missing data, while outlier detection methods help identify and handle extreme values that could skew the analysis. Additionally, data encoding is utilized to convert categorical variables into a numerical format, ensuring compatibility with machine learning algorithms. Feature scaling is applied to normalize the data and bring all variables to a common scale, preventing certain attributes from dominating the analysis due to their larger magnitudes. Furthermore, data normalization is performed to eliminate redundancies and ensure the data conforms to specified constraints. By undertaking thorough data preparation, we can enhance the reliability and accuracy of our analysis, making the data more amenable to modeling and interpretation.

1.5 Modeling and Interpretation



Data modeling is stage of the data mining process, where we apply various machine learning algorithms and statistical techniques to build predictive models from the prepared data. The goal of data modeling is to identify patterns, relationships, and trends within the data set that can be used to make informed predictions or decisions. This involves training the models on a subset of the data (training set) and then evaluating their performance on another subset (testing set) to ensure they can generalize well to new, unseen data. During this phase, we explore a range of different algorithms, such as decision trees, logistic regression, support vector machines, and ensemble methods, to find the most suitable one for our specific problem. Model tuning is also performed to optimize the hyper parameters of the selected algorithms and improve their performance further. Once the models are trained and fine-tuned, we evaluate their accuracy, precision, recall, and other relevant metrics to assess their effectiveness in predicting the target variable. By the end of the data modeling phase, we aim to have a robust and reliable predictive model that can be deployed to make predictions on new data and provide valuable insights for decision-making in various applications.

1.6 Evaluation

Evaluation is a critical phase in the data mining process, where the performance of the developed models is assessed to determine their effectiveness in solving the specific problem at hand. This phase involves comparing the model's predictions against the actual outcomes, using various evaluation metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC). These metrics provide valuable insights into the model's ability to correctly classify instances and its overall performance. Additionally, techniques like cross-validation are used to assess the model's robustness and generalization capabilities by dividing the data set into multiple subsets for training and testing. The evaluation phase helps in identifying potential issues like over fitting or under fitting, guiding us to fine-tune the model and achieve better results. It is crucial to select appropriate evaluation metrics based on the specific requirements of the project to make well-informed decisions and choose the most suitable model for deployment. By rigorously evaluating the models, we can gain confidence in their performance and ensure that they meet the desired objectives.

1.7 Deployment

Deployment is the final phase of a data-driven project, where the developed models and insights are put into practical use to drive real-world decision-making and solutions. In this stage, the focus shifts from experimentation and analysis to implementing the models in a live environment. The deployment process involves integrating the trained machine learning models into production systems, creating user interfaces for easy interaction, and setting up the infrastructure required to handle user requests. Depending on the application, deployment can take various forms, such as integrating models into web applications, mobile apps, or embedded systems. Successful deployment not only validates the value of the data-driven project but also drives tangible benefits to the organization, enabling data-driven decision-making to become an integral part of business operations.

1.8 Conclusion

In conclusion, the CRISP-DM methodology serves as a powerful and structured framework for guiding any data-driven project.

Through the various stages, we can start working on our fraud detection problem, with a systematic and goal-oriented approach. The methodology allowed us to gain a comprehensive understanding of the problem at hand, explore and preprocess the data effectively, and develop and evaluate multiple machine learning models to identify the most suitable one for fraud detection.

By following the CRISP-DM process, we are able to make informed decisions at each step of the project, ensuring that our efforts are aligned with the business objectives and constraints. Additionally, the methodology facilitated collaboration between different stakeholders, including domain experts, data scientists, and IT professionals, leading to a cohesive and integrated solution.

Chapter 2

Implementation

In this section, we will explore the practical aspects of our credit card fraud detection project, encompassing various phases of the CRISP-DM process. We will begin with Data Understanding phase where we will create and analyze our data set. Subsequently, we will proceed to the Data Preparation stage, involving exploratory data analysis and preprocessing. Moving forward, we will focus on model selection, creating and evaluating models to identify the most suitable one. We will interpret the results of our chosen model, followed by the deployment of the model within a Web Application. Finally, we will discuss the outcomes and insights obtained throughout the project, emphasizing the key steps undertaken.

2.1 Data Understanding

In this section, we will talk about the data set we will work with in this project. the Data set used in this project, is downloaded from Kaggle.

Kaggle is a popular online platform that hosts data science and machine learning competitions. It provides a collaborative environment for data scientists, researchers, and enthusiasts to access and work on diverse datasets, solve complex problems, and showcase their skills. you can check the dataset using this link.

The Dataset provided consists of a Training set and a Testing set, the The dataset consists of both a Training set and a Testing set. Our objective is to construct a machine learning model using the Training set and evaluate its performance using the provided Testing set. This approach will enable us to train the model on known instances of fraudulent and legitimate transactions and assess its effectiveness in accurately classifying new, unseen transactions. By leveraging this dataset and following this approach, we aim to develop a robust credit card fraud detection model that can be applied in real-world scenarios.

Both the Training and Testing sets consist of the same column names, types, and possible values. Throughout the rest of this report, our focus will primarily be on the training set, as the same preprocessing steps are applied to the testing set as well.

2.1.1 Data Description

Column Name	Description
Unnamed: 0	Unique Identifier for each observation.
trans_date_trans_time	The date and time of each transaction
cc_num	The credit card numbers used in each transaction.
merchant	Refers to the name or identifier of the merchant or establishment where the credit card transaction took place.
category	Represents the category of the business where the transaction took place.
amt	Represents the amount of money in dollars in each transaction
first	The first name of the credit card holder
last	the last name of the credit card holder
gender	The gender is the user who made the transaction
street	Street address
city	The city where the transaction is made
state	The state where the transaction is made
zip	The zip code of where the transaction was made
lat	The Latitude
long	The Longitude
city_pop	The city population
job	The job of the card owner
dob	The date of birth of the card owner
trans_num	Identifier for each transaction
unix_time	Unix time, represent the date and time as a single numerical value.
merch_lat	The Longitude of the merchant
merch_long	The Latitude of the merchant
is_fraud	The class Target

2.1.2 Read the dataset

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
sns.set_style("darkgrid")
palette = "mako"

import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# read the csv file
df = pd.read_csv('fraudTrain.csv')
test = pd.read_csv('fraudTest.csv')
```

2.1.3 show an example from the dataset

Lets show a random observation from the dataset

[9]:
```
df.sample().T
```

[9]:

	784648
Unnamed: 0	784648
trans_date_trans_time	2019-12-01 04:11:19
cc_num	584673555952
merchant	fraud_Bernhard, Grant and Langworth
category	shopping_pos
amt	17.35
first	Jessica
last	Jordan
gender	F
street	071 Wise Trace
city	Coleharbor
state	ND
zip	58531
lat	47.5196
long	-101.2332
city_pop	307
job	Make
dob	1930-08-13
trans_num	4b378f375c7c3676d4e482b42d5018d4
unix_time	1354335079
merch_lat	46.627106
merch_long	-100.917204
is_fraud	0

2.1.4 Data Description

Lets check the shape of the dataset:

[10]:
```
df.shape
```

[10]:
```
(1296675, 23)
```

Lets The types of our columns using the function **info()**

[13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype
```

```
---   ------                  --------------    -----
 0    Unnamed: 0              1296675 non-null  int64
 1    trans_date_trans_time   1296675 non-null  object
 2    cc_num                  1296675 non-null  int64
 3    merchant                1296675 non-null  object
 4    category                1296675 non-null  object
 5    amt                     1296675 non-null  float64
 6    first                   1296675 non-null  object
 7    last                    1296675 non-null  object
 8    gender                  1296675 non-null  object
 9    street                  1296675 non-null  object
 10   city                    1296675 non-null  object
 11   state                   1296675 non-null  object
 12   zip                     1296675 non-null  int64
 13   lat                     1296675 non-null  float64
 14   long                    1296675 non-null  float64
 15   city_pop                1296675 non-null  int64
 16   job                     1296675 non-null  object
 17   dob                     1296675 non-null  object
 18   trans_num               1296675 non-null  object
 19   unix_time               1296675 non-null  int64
 20   merch_lat               1296675 non-null  float64
 21   merch_long              1296675 non-null  float64
 22   is_fraud                1296675 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

The dataset contains numerical and categorical columns.

Let's check some statistical information on our numerical columns with the help of the function **describe()**

```
In [17]: df.describe().T
```

Out[17]:

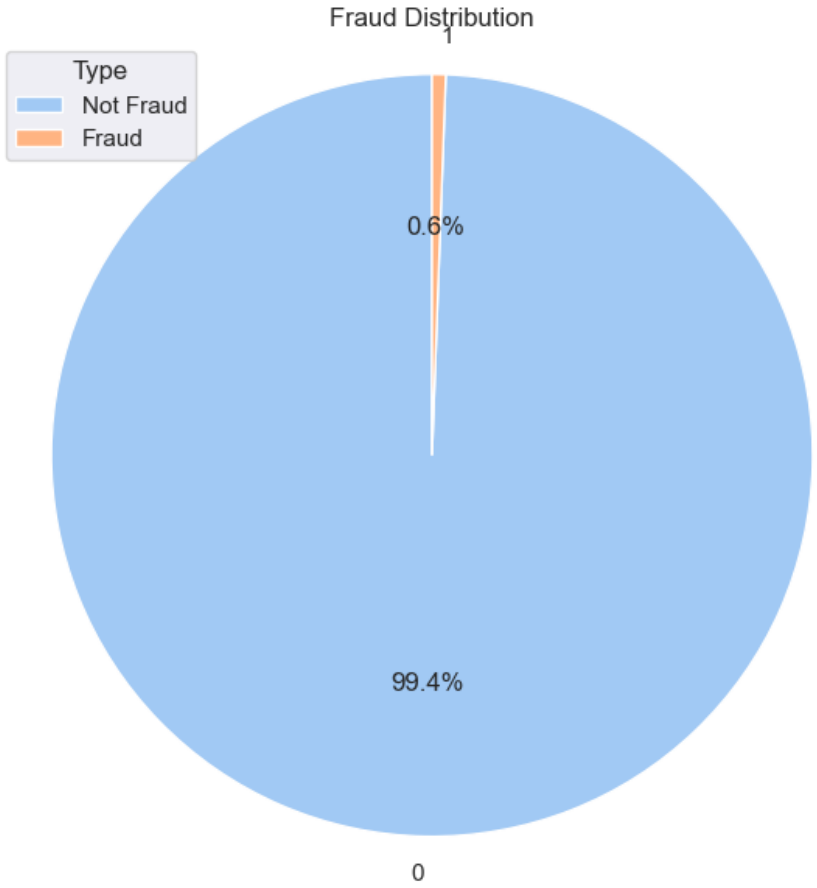
	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	1296675.0	6.483370e+05	3.743180e+05	0.000000e+00	3.241685e+05	6.483370e+05	9.725055e+05	1.296674e+06
cc_num	1296675.0	4.171920e+17	1.308806e+18	6.041621e+10	1.800429e+14	3.521417e+15	4.642255e+15	4.992346e+18
amt	1296675.0	7.035104e+01	1.603160e+02	1.000000e+00	9.650000e+00	4.752000e+01	8.314000e+01	2.894890e+04
zip	1296675.0	4.880067e+04	2.689322e+04	1.257000e+03	2.623700e+04	4.817400e+04	7.204200e+04	9.978300e+04
lat	1296675.0	3.853762e+01	5.075808e+00	2.002710e+01	3.462050e+01	3.935430e+01	4.194040e+01	6.669330e+01
long	1296675.0	-9.022634e+01	1.375908e+01	-1.656723e+02	-9.679800e+01	-8.747690e+01	-8.015800e+01	-6.795030e+01
city_pop	1296675.0	8.882444e+04	3.019564e+05	2.300000e+01	7.430000e+02	2.456000e+03	2.032800e+04	2.906700e+06
unix_time	1296675.0	1.349244e+09	1.284128e+07	1.325376e+09	1.338751e+09	1.349250e+09	1.359385e+09	1.371817e+09
merch_lat	1296675.0	3.853734e+01	5.109788e+00	1.902779e+01	3.473357e+01	3.936568e+01	4.195716e+01	6.751027e+01
merch_long	1296675.0	-9.022646e+01	1.377109e+01	-1.666712e+02	-9.689728e+01	-8.743839e+01	-8.023680e+01	-6.695090e+01
is_fraud	1296675.0	5.788652e-03	7.586269e-02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00

We can notice that these columns vary over a wide range of data; therefore, the use of scaling, specifically "normalization min-max," would be beneficial. This process would make the data range between $[0, 1]$, making it easier for the machine learning algorithms to train.

2.1.5 Target Variable

The target variable in our dataset is the column 'is_fraud', which contains either 0 (for non-fraud transactions) or 1 (for fraud transactions).

Now, let's examine the distribution of the target variable.



it's clear that we are working with a highly imbalanced dataset. we need to explore multiple solutions in order to get the best model possible.

2.2 Data Preparation

The data preparation phase is a crucial step within the CRISP-DM methodology. It revolves around comprehending, cleaning, transforming, and integrating raw data to form a well-prepared dataset for analysis. This phase ensures that the data is in a suitable format and free from any inconsistencies or errors, enabling more effective modeling and analysis in the subsequent steps of the project. Proper data preparation lays the foundation for generating accurate and reliable insights from the data.

2.2.1 Data Pre-Processing

Missing values

let's check for the missing values using pandas function 'isna()' and calculating the sum for each column:

[33]:
```
df.isna().sum()
```

[33]:
```
Unnamed: 0                0
trans_date_trans_time     0
cc_num                    0
merchant                  0
category                  0
amt                       0
first                     0
last                      0
gender                    0
street                    0
city                      0
state                     0
zip                       0
lat                       0
long                      0
city_pop                  0
job                       0
dob                       0
trans_num                 0
unix_time                 0
merch_lat                 0
merch_long                0
is_fraud                  0
dtype: int64
```

Duplicates

Let's check for duplicates using pandas 'duplicated':

[37]:
```
nbr_duplicated = df.duplicated().sum()
print(f'The number of duplicates is: {nbr_duplicated}')
```

```
The number of duplicates is: 0
```

It looks like our dataset contains no missing or duplicated values, we can proceed to the next step

Remove Useless Columns

From our first analysis we know:

1. **Unnamed: 0**: an id
2. **first**: first name
3. **last**: last name
4. **trans_num**: id of a transition
5. **unix_time**: the time in seconds of that transaction

We will remove these columns:

[38]:
```
df = df.drop(['Unnamed: 0', 'first', 'last', 'trans_num', 'unix_time'],␣
↪axis=1)
```

Data Transformation

Data transformation is the process of converting and modifying raw data to make it more suitable and valuable for analysis. It involves fixing data issues, such as inconsistent formats, converting data to the right types, creating new informative features, and removing irrelevant or redundant columns. Data transformation ensures that the data is in a consistent, accurate, and structured format, enabling better insights and more effective decision-making in various data-driven tasks.

The column **"trans_date_trans_time"** is an object, let's change it to pandas datetime, then create two new columns. **"hour"** that will contain the hour when the transaction is made, and **"trans_date"** that will contain the date in format **"yyyy-mm-dd"**. after that drop the column **"trans_date_trans_time"** as we no longer need it.

[39]:
```
# convert the type to datetime

df["trans_date_trans_time"] = pd.to_datetime(df.trans_date_trans_time)
```

[40]:
```
# extract a new column that contains the hour when each transaction␣
↪is made.

import datetime

df['hour'] = df.trans_date_trans_time.dt.hour
```

[41]:
```
# change format to 'yyyy-mm-dd'

df['trans_date'] = df['trans_date_trans_time'].dt.strftime('%Y/%m/%d')
df['trans_date'] = pd.to_datetime(df.trans_date)
```

[42]:
```
df = df.drop('trans_date_trans_time', axis=1)
```

The column **"cc_num"** is an int, the cc number is a unique identifier for a user, the same user ca use to do multiple transactions, if we keep it as an int, some machine learning models will treat them as values not as identifiers.

[43]:
```
# change the type of cc_num.
df['cc_num']= df.cc_num.astype('category')
test['cc_num']= test.cc_num.astype('category')
```

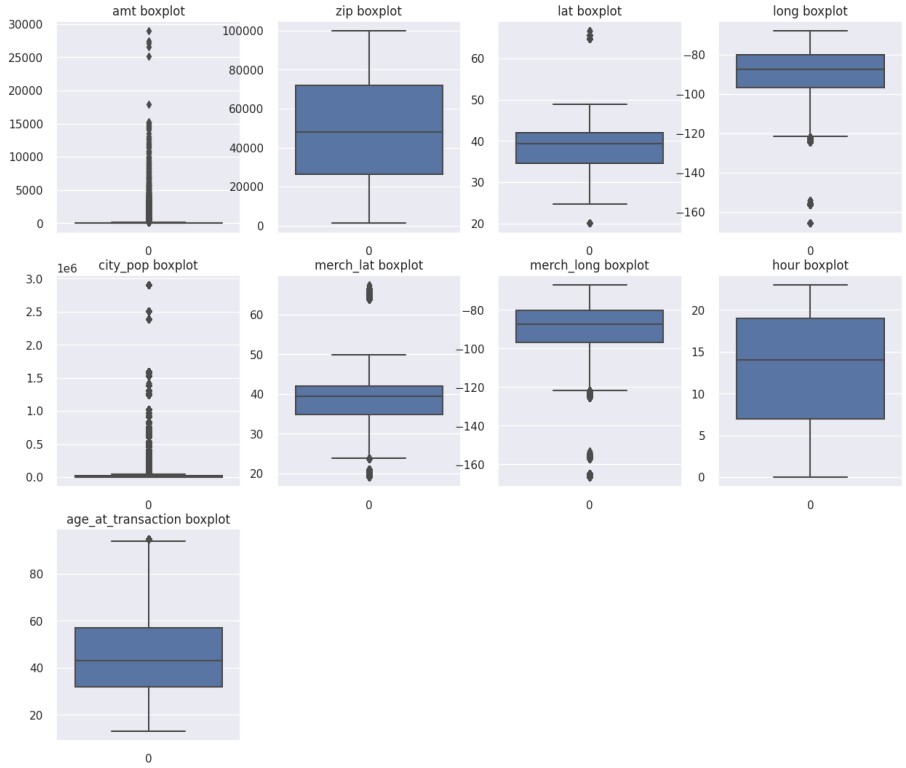
The column **"cc_num"** is in wrong type, we can change it to pandas datetime, and extract the user age at transaction, which mean we will create a new column **"age_at_transaction"**. after that we will no longer need the "dob" column.

[44]:
```
# change the type of dob
df['dob'] = pd.to_datetime(df.dob)
```

[ ]:
```
# extract new column that contains the user age when the transaction␣
↪was made from dob.
df['age_at_transaction'] = ((df['trans_date'] - df['dob']).dt.days /␣
↪365.25).astype(int)
```

```
df = df.drop('dob', axis=1)
```

2.2.2 Outliers



we can also count the number of outliers and the percentage of it in each column.

```
def outlier_count(col, data):
    print(15*'-' + col + 15*'-')
    q75, q25 = np.percentile(data[col], [75, 25])
    iqr = q75 - q25
    min_val = q25 - (iqr*1.5)
    max_val = q75 + (iqr*1.5)
    outlier_count = len(np.where((data[col] > max_val) | (data[col] < min_val))[0])
    outlier_percent = round(outlier_count/len(data[col])*100, 2)
    print('Number of outliers: {}'.format(outlier_count))
    print('Percent of data that are outliers: {}%'.format(outlier_percent))

for col in numerical_columns:
    outlier_count(col, df)
```

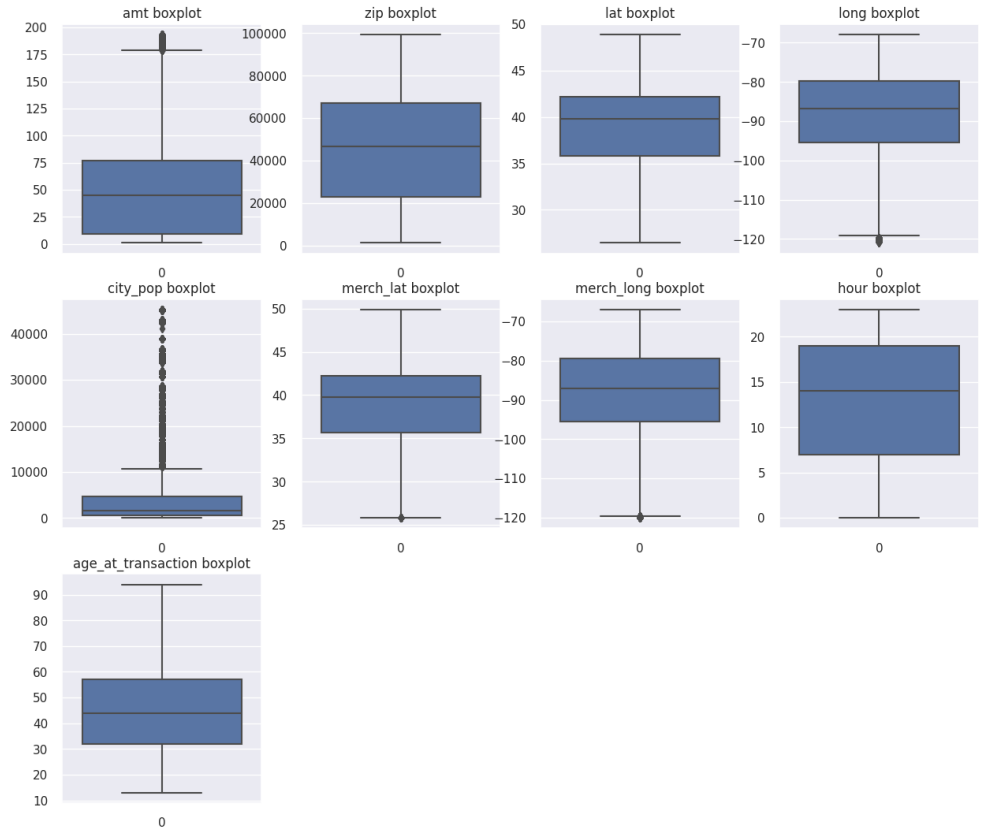
We can notice a really high number of outliers in the columns 'amt' and 'city_pop'. We will use the IQR method to remove this outliers,

```
# remove outliers
def remove_outliers(df, columns):
    cleaned_df = df.copy()

    for column in columns:
        if column in cleaned_df.columns:
            Q1 = cleaned_df[column].quantile(0.25)
            Q3 = cleaned_df[column].quantile(0.75)
            IQR = Q3 - Q1

            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
```

```
---------------amt---------------
Number of outliers: 67290
Percent of data that are outliers: 5.19%
---------------zip---------------
Number of outliers: 0
Percent of data that are outliers: 0.0%
---------------lat---------------
Number of outliers: 4679
Percent of data that are outliers: 0.36%
---------------long---------------
Number of outliers: 49922
Percent of data that are outliers: 3.85%
---------------city_pop---------------
Number of outliers: 242674
Percent of data that are outliers: 18.72%
---------------merch_lat---------------
Number of outliers: 4967
Percent of data that are outliers: 0.38%
---------------merch_long---------------
Number of outliers: 41994
Percent of data that are outliers: 3.24%
---------------hour---------------
Number of outliers: 0
Percent of data that are outliers: 0.0%
---------------age_at_transaction---------------
Number of outliers: 236
Percent of data that are outliers: 0.02%
```

```
        cleaned_df = cleaned_df[(cleaned_df[column] >= lower_bound) & (cleaned_df[column] <= upper_
↪bound)]

    return cleaned_df
```

We visualise the numerical columns after we removed the outliers.



if we check the target variable

[28]:
```
clean_df['is_fraud'].value_counts()
```

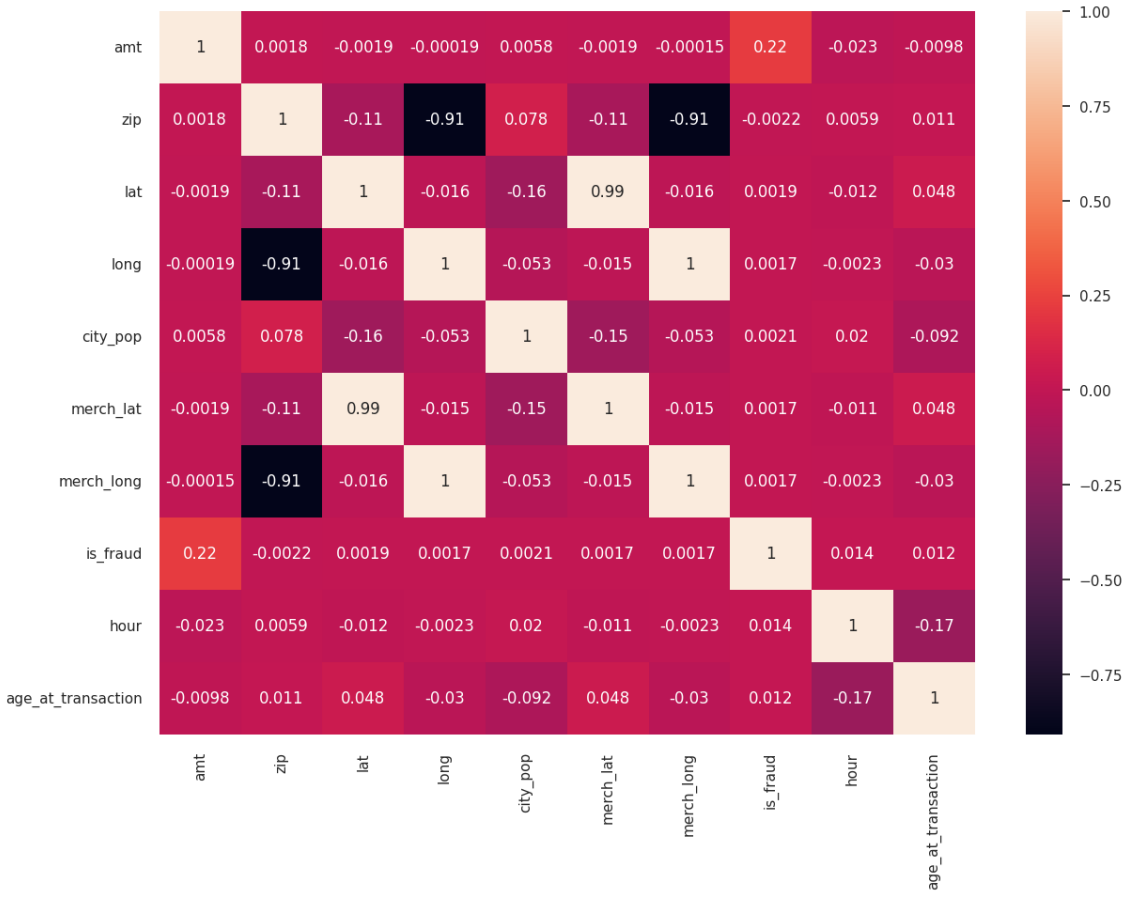
[28]:
```
is_fraud
0    947837
1      1390
Name: count, dtype: int64
```

Removing outliers made us lose a large number of observations of the minority class, since we already have an imbalanced dataset.

In this case, it's better to not remove the outliers.

Correlation Matrix

The correlation matrix is used to understand the relationships between different variables in a dataset. It provides a concise and clear summary of how variables are related to each other, specifically in terms of their linear relationship. The matrix contains correlation coefficients, which are values between -1 and 1.



We can see that there is a really high correlation between the columns 'long', 'merch_long' and 'zip', and also between 'lat' and 'merch_lat'. Which means we have a redundancy in information.

two highly correlated columns, means that the same information carried in this column is carried in the other one as well, it's better to remove one of the two.

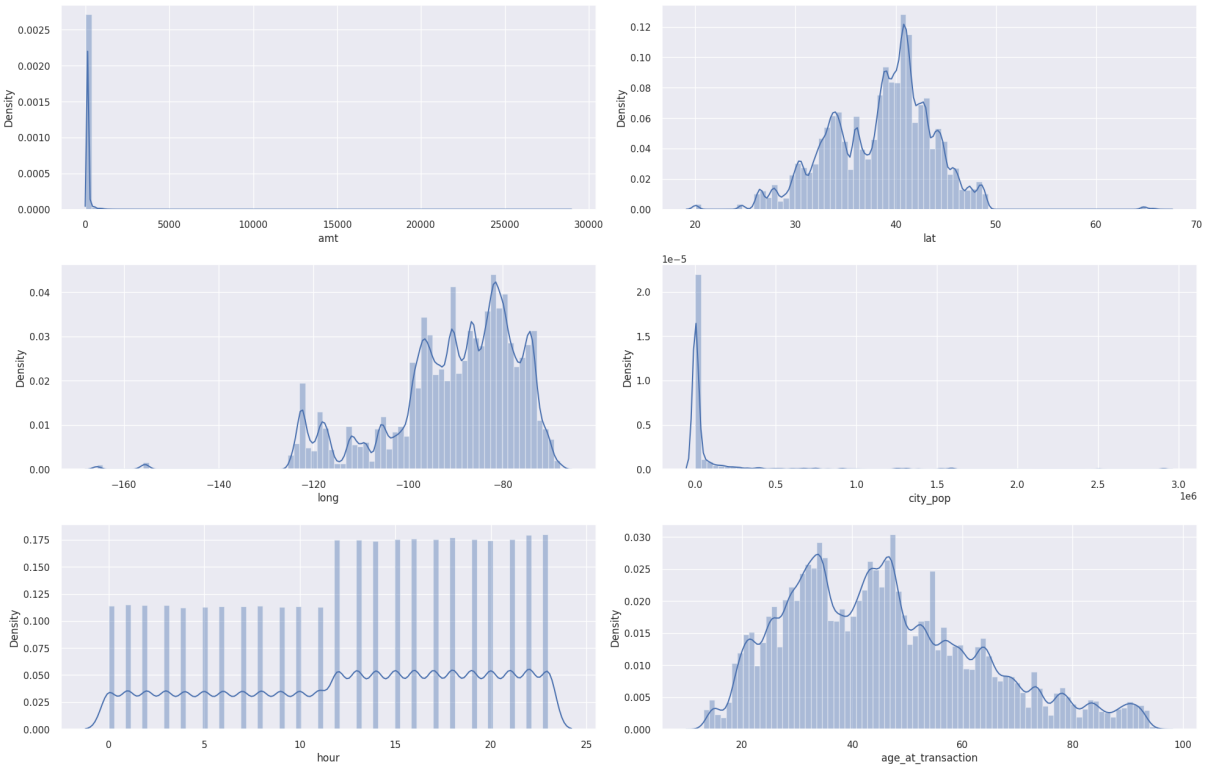
We will remove 'merch_long', 'merch_lat' and 'zip'.

```
df = df.drop(['merch_long', 'merch_lat', 'zip'], axis=1)
```

2.2.3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a critical step in any data Mining project. It involves the use of various techniques and tools to get a good understanding of the dataset without making any assumptions. The primary goal of EDA is to uncover patterns, relationships, anomalies, and insights that can guide further analysis and build our modeling strategies.

Numerical Columns Distribution

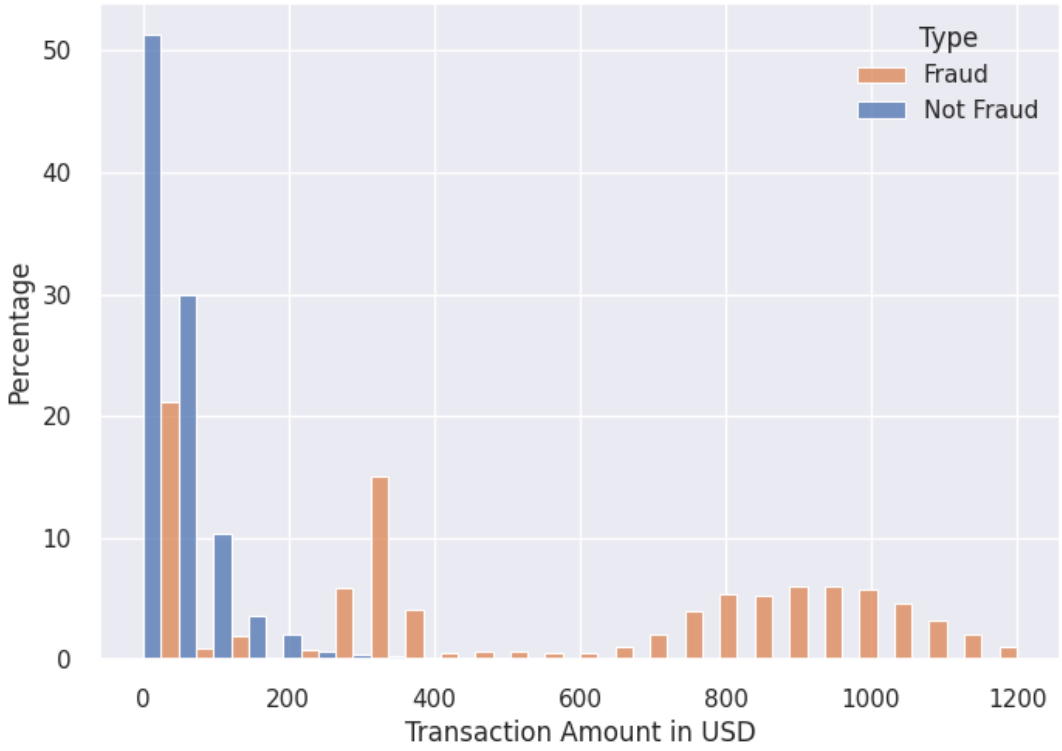


We can see that most of the numerical columns dont follow a Normal D Ditributon. Wich can negatively affect some machine learning algorithms, because they are sensitive to the scale of input features. For instance, distance-based algorithms like k-nearest neighbors can be affected by feature scales.

For that we ca use Transformation(Scaling).

AMT

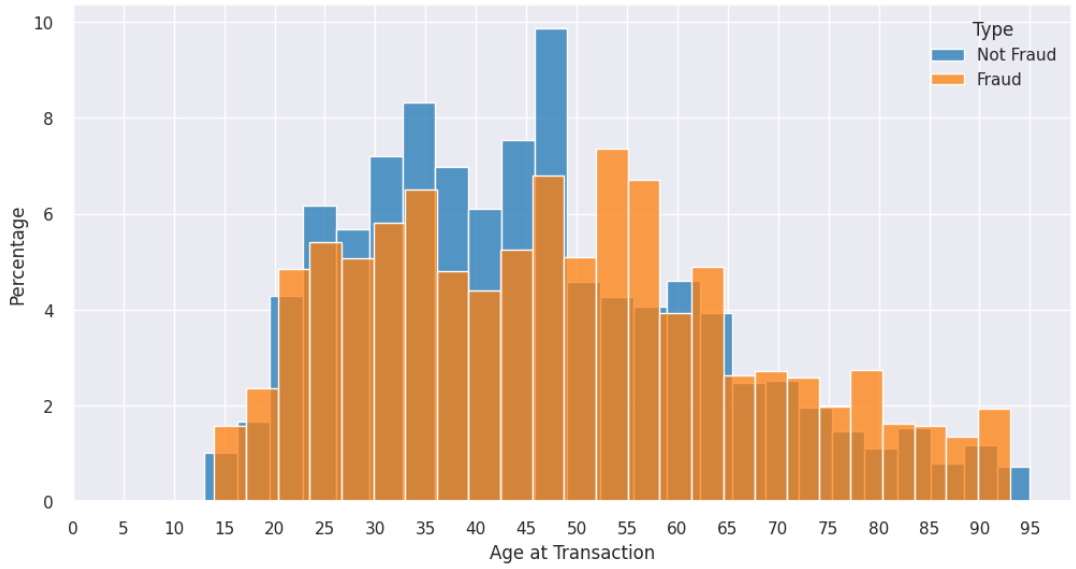
Let's examine the correlation between the transaction amounts and the percentage of fraudulent transactions. For better visualization, I plotted the transaction amounts up to the maximum possible value in the fraud transactions.



We can observe a clear pattern in the data. For non-fraudulent transactions, the majority fall within the range of 1 to 200 dollars, with a few going over 300 dollars. On the other hand, fraudulent transactions cover a wider range, spanning from 1 to 1200 dollars, with a significant portion falling between 600 and 1200 dollars.

This indicates a correlation between the transaction amount and whether it is a fraudulent or non-fraudulent transaction.

Age at Transaction

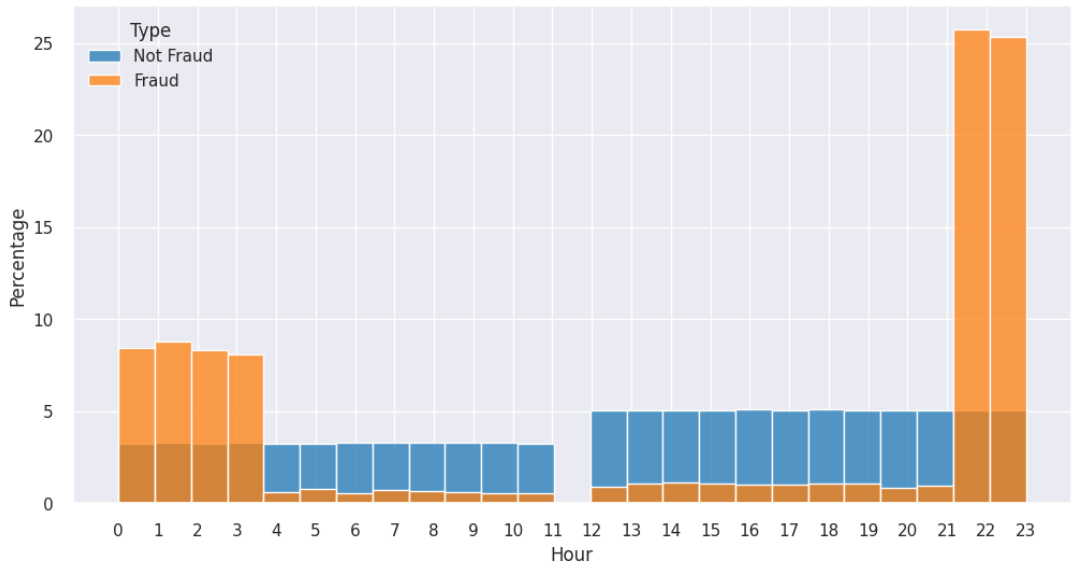


In the range of 25 to 50, we can observe a higher number of non-fraudulent transactions compared to fraudulent ones. The highest concentration occurs between 32 and 38, as well

as between 46 and 50. However, beyond 50, we start to see more fraudulent transactions, particularly between 50 and 60.

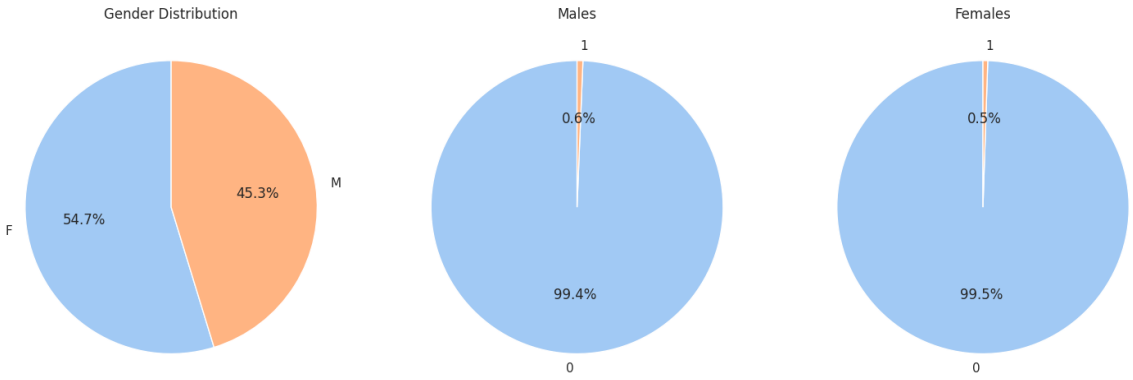
This column may hold valuable information that could be significant for our analysis.

Hour



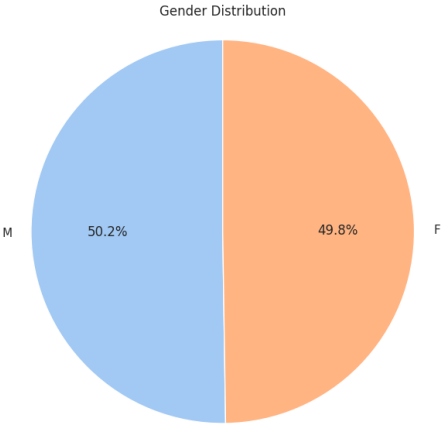
it's clear that there are more fraudulent transactions between 10pm and 3am, after that there are more non fraudulent transactions.

Categorical Columns



We can see from the first plot that females made 9.4% more transactions than males. And we can also see that the percentage of fraud and non fraud among males and females is relatively the same.

Let's check the percentage of each gender in the fraud transactions.

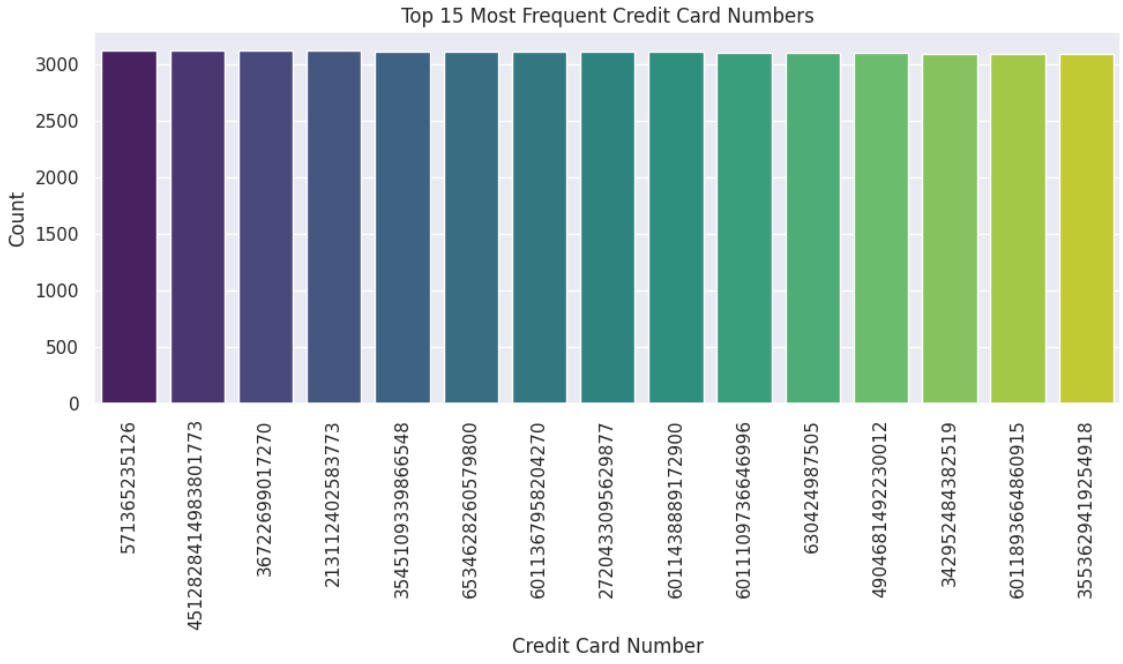


The pie plot shows a balance between the genders in both the fraudulent and non fraudulent transactions.

it's clear that this column won't carry any information concerning the target variable.

-> We will drop this column.

cc$_num$



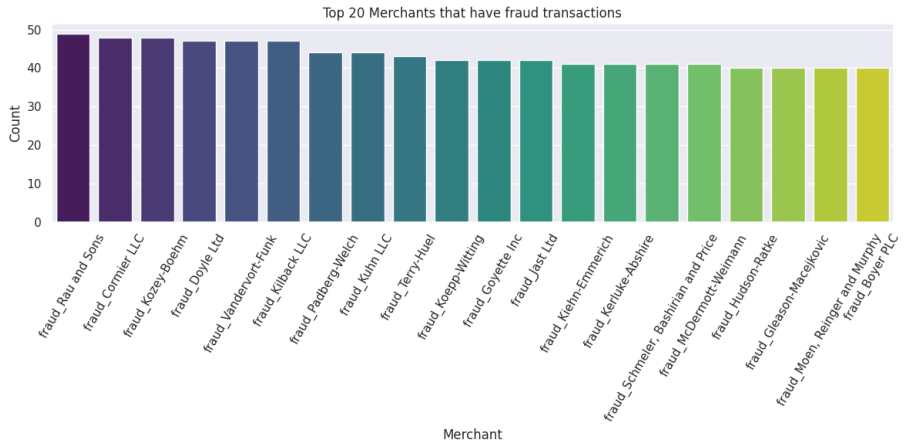
The 15 most frequent credit cards have a similar number of transactions, with a small difference.

Let's check the percentage of fraudulent and not fraudulent transactions on these 15 credit cards.

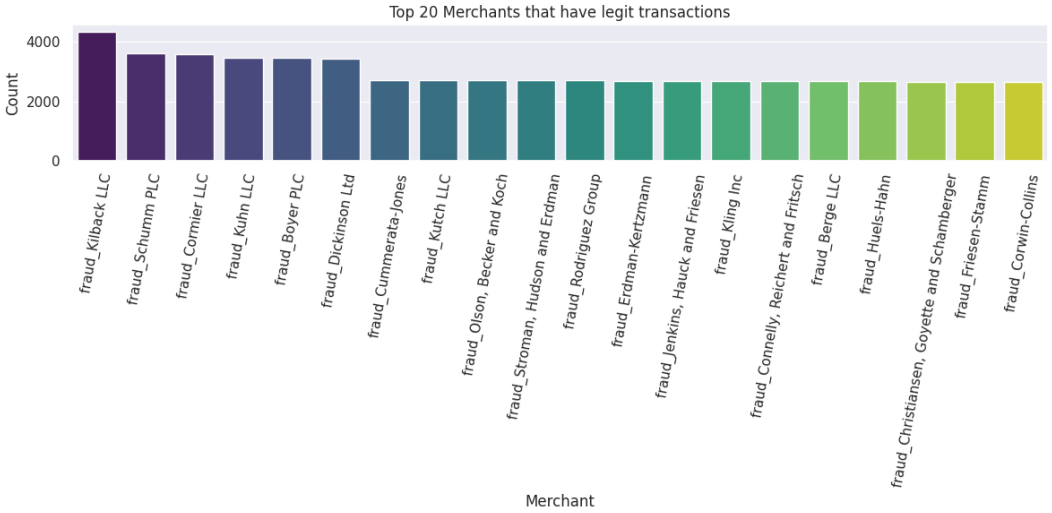
There is a really low percentage of fraudulent transactions, on the 7 most frequent credit cards, and the rest are all legitimate transactions. This column is not going to help with identifying the fraudulent transactions.

-> we will remove this column.

Merchant

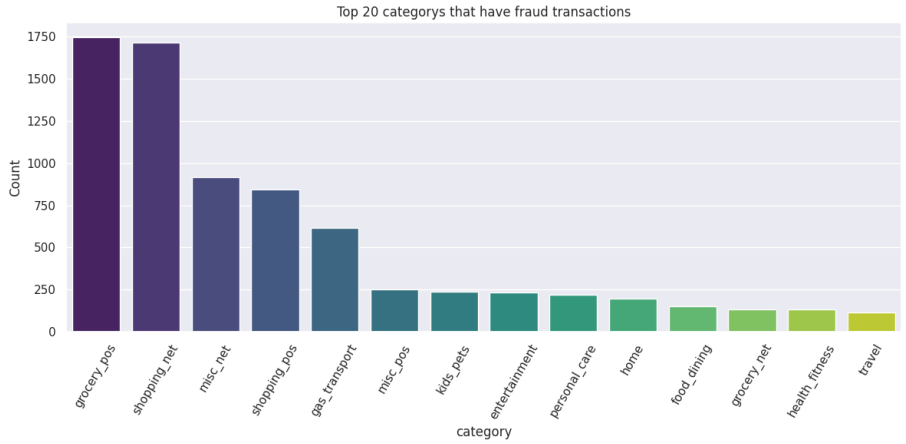


We can see that Rau and Sons have the highest number of fraudulent transactions.



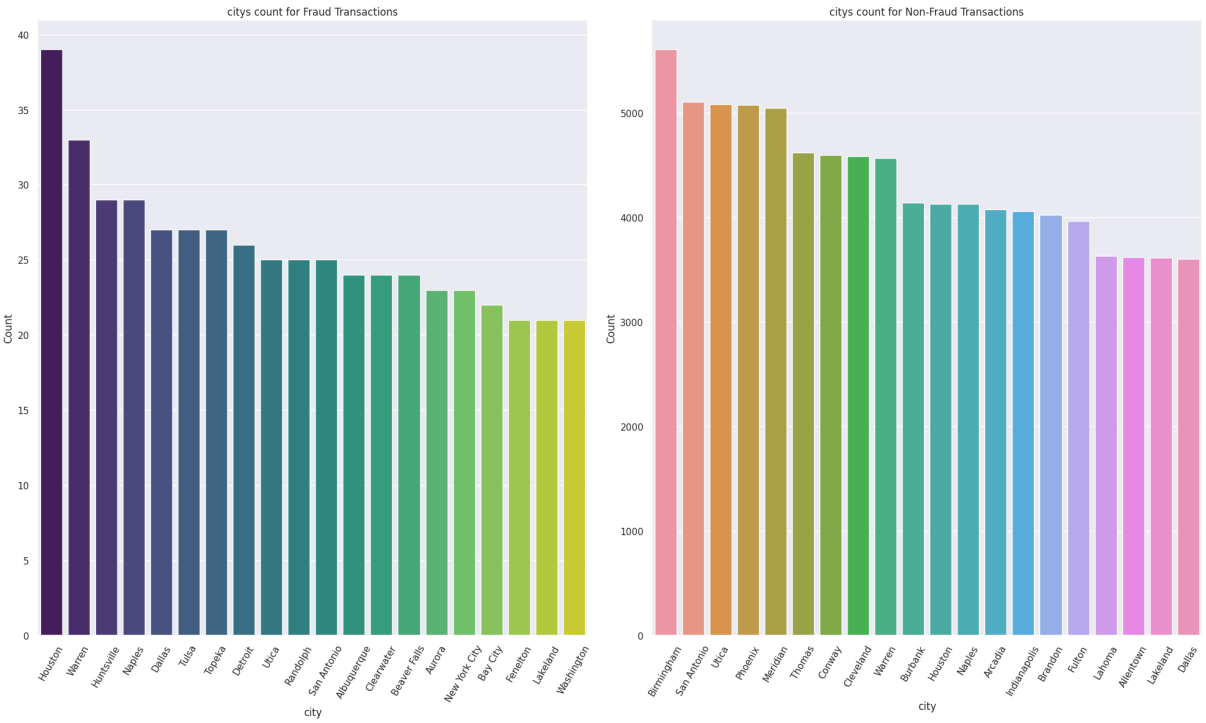
Kilback LLC have the most legitimate transactions.

Category



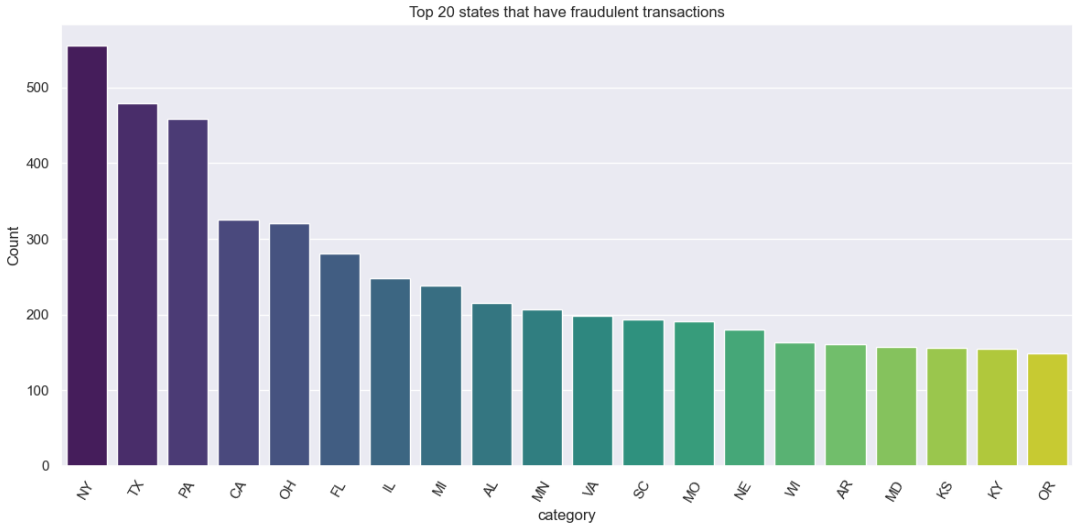
'grocery_pos' and 'shopping_net' have the highest number of fraudulent transactions.

City



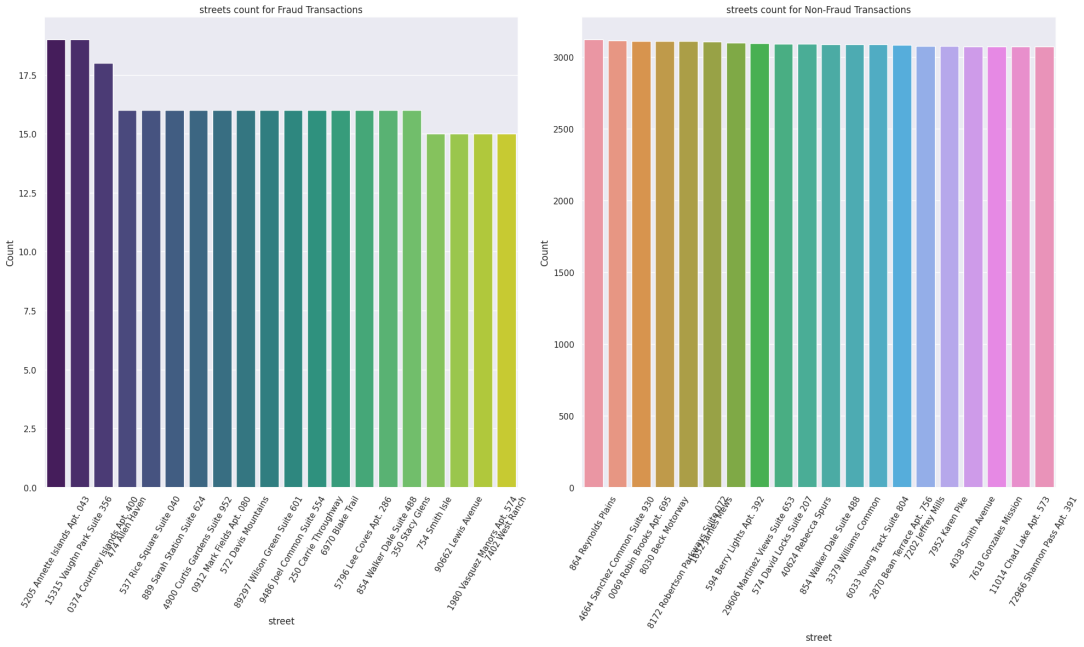
Among the cities analyzed, Houston stands out with the highest number of fraudulent transactions, indicating a potential hot spot for fraudulent activities. On the other hand, Birmingham appears to have the least amount of fraudulent transactions, suggesting a relatively safer environment in comparison.

State



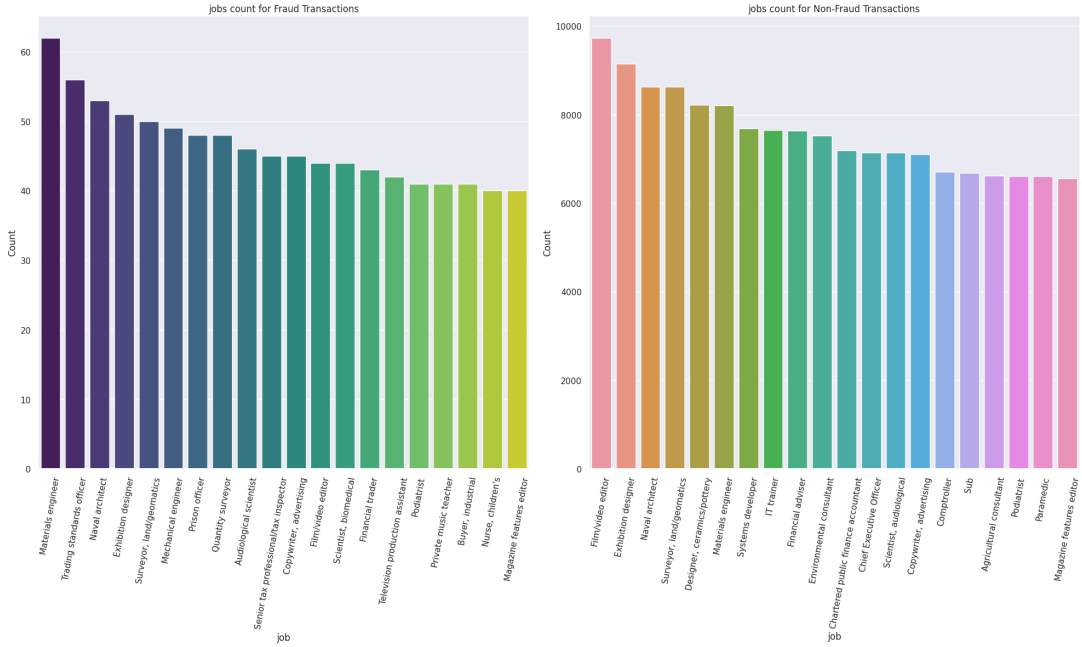
We can see that New York and Texas had the most fraud transactions.

Street



In the analysis of the "street" column, we observe notable differences between non-fraudulent and fraudulent transactions. For non-fraudulent transactions, the distribution of streets shows a relatively uniform count, with the streets having similar occurrences. However, in the case of fraudulent transactions, there is a noticeable gap between the streets, indicating a more uneven distribution. Due to the lack of significant discriminatory patterns and the potential risk of introducing noise into our model, **We will Remove this Column**

Job



Materials engineer is the job with the most fraudulent transactions.

Film/video editing is the job with the most non fraudulent transactions.

Remove the unnecessary columns

```
df = df.drop(['gender', 'cc_num', 'street'], axis=1)
```

In the end these are the column we will work with:

```
In [145]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 13 columns):
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   merchant            1296675 non-null  object
 1   category            1296675 non-null  object
 2   amt                 1296675 non-null  float64
 3   city                1296675 non-null  object
 4   state               1296675 non-null  object
 5   lat                 1296675 non-null  float64
 6   long                1296675 non-null  float64
 7   city_pop            1296675 non-null  int64
 8   job                 1296675 non-null  object
 9   is_fraud            1296675 non-null  int64
 10  hour                1296675 non-null  int64
 11  trans_date          1296675 non-null  datetime64[ns]
 12  age_at_transaction  1296675 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(5)
memory usage: 128.6+ MB
```

2.2.4 Data Encoding

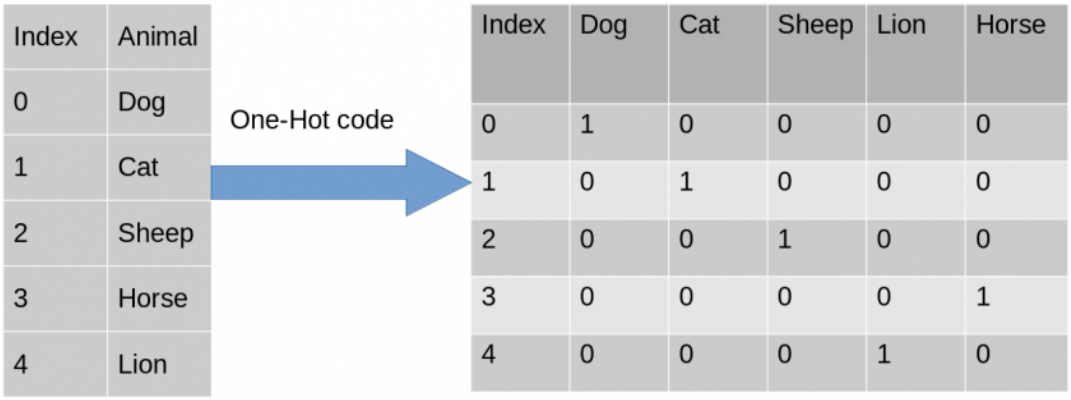
Data Encoding refers to the process of transforming raw data into a suitable format that can be effectively utilized by algorithms for analysis, modeling, and prediction. In

most real-world scenarios, data comes in various types and formats, such as categorical, numerical, or textual data. Each type of data requires a different representation to be processed efficiently by machine learning models.

One Hot Encoding

One-Hot Encoding is a popular data encoding technique used in machine learning to handle categorical data. Categorical data represents discrete values or labels that do not have an inherent numerical order.

In One-Hot Encoding, each unique category is transformed into a binary vector, where only one element is ”hot” or 1, and all other elements are ”cold” or 0. The ”hot” element corresponds to the presence of the category, while the ”cold” elements represent the absence of that category. This encoding effectively converts categorical data into a numerical format, allowing machine learning algorithms to process it.



K-Fold Target Encoding

K-Fold Target Encoding is a technique used to encode categorical variables in a supervised machine learning setting. It is particularly useful when dealing with high-cardinality categorical features, where the number of unique categories is large. The main goal of target encoding is to transform categorical data into numerical values that can be used as input for machine learning algorithms.

The basic idea of the k-fold target encoding originates from the mean-target encoding. In the mean-target encoding, the categorical variables are replaced by the mean of the target corresponding to them. The problem here is Data Leakage, we rely on the target to encode those values, which can cause overfitting.

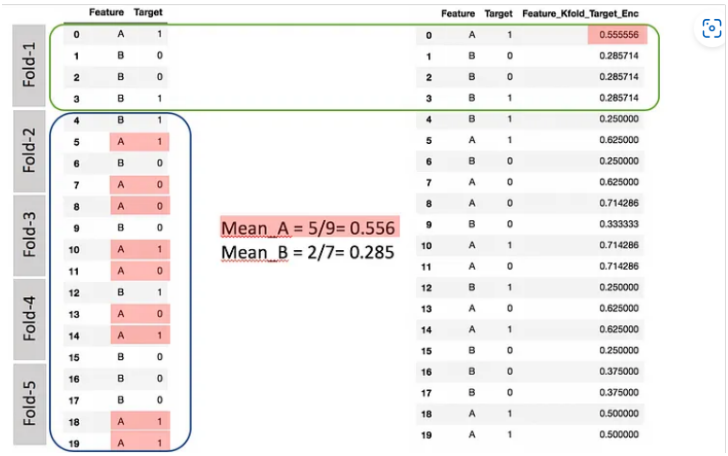
Therefore, k-fold target encoding can be applied to reduce the overfitting. Lets take this example:

	Feature	Target
0	A	1
1	B	0
2	B	0
3	B	1
4	B	1
5	A	1
6	B	0
7	A	0
8	A	0
9	B	0
10	A	1
11	A	0
12	B	1
13	A	0
14	A	1
15	B	0
16	B	0
17	B	0
18	A	1
19	A	1

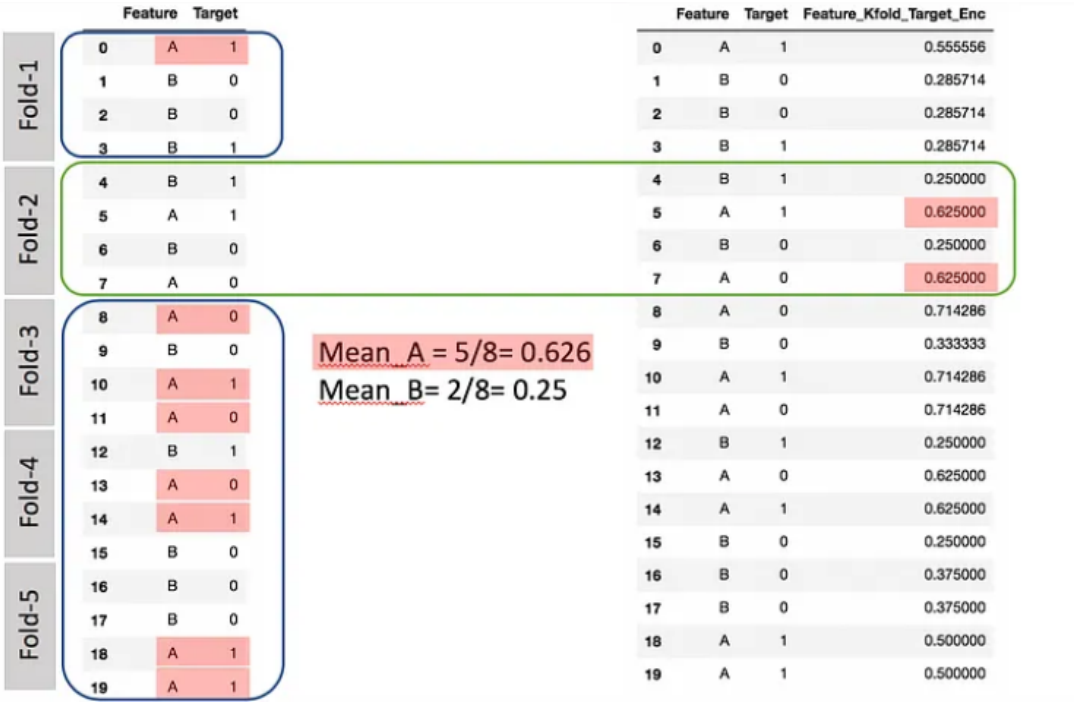
MEAN

Feature	Target
A	0.6
B	0.3

In this method, we divide the dataset into the k-folds, We calculate the mean-target for fold 2, 3, 4 and 5 and we use the calculated values, mean_A = 0.556 and mean_B = 0.285 to estimate mean encoding for the fold-1.



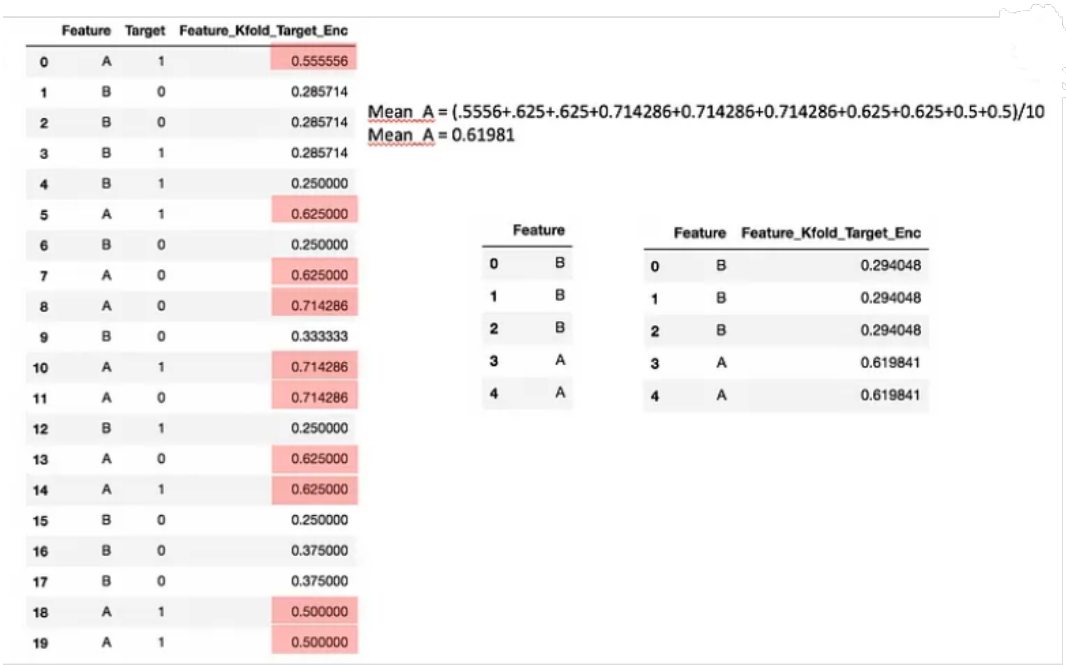
After that, we can calculate for the second fold as follow:



	Feature	Target
0	A	1
1	B	0
2	B	0
3	B	1
4	B	1
5	A	1
6	B	0
7	A	0
8	A	0
9	B	0
10	A	1
11	A	0
12	B	1
13	A	0
14	A	1
15	B	0
16	B	0
17	B	0
18	A	1
19	A	1

	Feature	Target	Feature_Kfold_Target_Enc
0	A	1	0.555556
1	B	0	0.285714
2	B	0	0.285714
3	B	1	0.285714
4	B	1	0.250000
5	A	1	0.625000
6	B	0	0.250000
7	A	0	0.625000
8	A	0	0.714286
9	B	0	0.333333
10	A	1	0.714286
11	A	0	0.714286
12	B	1	0.250000
13	A	0	0.625000
14	A	1	0.625000
15	B	0	0.250000
16	B	0	0.375000
17	B	0	0.375000
18	A	1	0.500000
19	A	1	0.500000

Now the remaining part is creating "Feature_Kfold_Target_Enc" column in the test dataset. This column values can be obtained from getting mean of "Feature_Kfold_mean_Enc" train column for the categorical variables "A" and "B".



	Feature	Target	Feature_Kfold_Target_Enc
0	A	1	0.555556
1	B	0	0.285714
2	B	0	0.285714
3	B	1	0.285714
4	B	1	0.250000
5	A	1	0.625000
6	B	0	0.250000
7	A	0	0.625000
8	A	0	0.714286
9	B	0	0.333333
10	A	1	0.714286
11	A	0	0.714286
12	B	1	0.250000
13	A	0	0.625000
14	A	1	0.625000
15	B	0	0.250000
16	B	0	0.375000
17	B	0	0.375000
18	A	1	0.500000
19	A	1	0.500000

Mean_A = (.5556+.625+.625+0.714286+0.714286+0.714286+0.625+0.625+0.5+0.5)/10
Mean_A = 0.61981

	Feature
0	B
1	B
2	B
3	A
4	A

	Feature	Feature_Kfold_Target_Enc
0	B	0.294048
1	B	0.294048
2	B	0.294048
3	A	0.619841
4	A	0.619841

Applying the Data Encoding

In our project, we have the columns $['merchant','city','state','job']$ contains a high number of unique values, the choice of the encoding method will be **k-Fold Target Encoding**, and for the remaining categorical column **category** we will be using **One Hot Encoding**

This is the final form of our data after cleaning, transformation and encoding:

```
new_df.info()
```

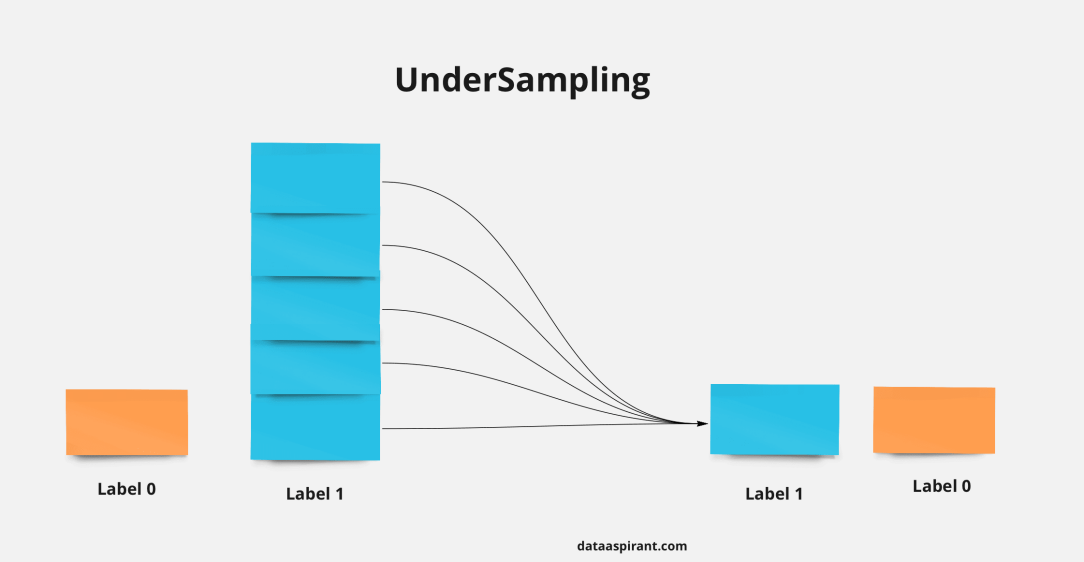
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 28 columns):
 #   Column                      Non-Null Count    Dtype
---  ------                      --------------    -----
 0   amt                         1296675 non-null  float64
 1   lat                         1296675 non-null  float64
 2   long                        1296675 non-null  float64
 3   city_pop                    1296675 non-null  int64
 4   is_fraud                    1296675 non-null  int64
 5   hour                        1296675 non-null  int64
 6   age_at_transaction          1296675 non-null  int64
 7   merchant_Kfold_Target_Enc   1296675 non-null  float64
 8   city_Kfold_Target_Enc       1296675 non-null  float64
 9   state_Kfold_Target_Enc      1296675 non-null  float64
 10  job_Kfold_Target_Enc        1296675 non-null  float64
 11  category_entertainment      1296675 non-null  float64
 12  category_food_dining        1296675 non-null  float64
 13  category_gas_transport      1296675 non-null  float64
 14  category_grocery_net        1296675 non-null  float64
 15  category_grocery_pos        1296675 non-null  float64
 16  category_health_fitness     1296675 non-null  float64
 17  category_home               1296675 non-null  float64
 18  category_kids_pets          1296675 non-null  float64
 19  category_misc_net           1296675 non-null  float64
 20  category_misc_pos           1296675 non-null  float64
 21  category_personal_care      1296675 non-null  float64
 22  category_shopping_net       1296675 non-null  float64
 23  category_shopping_pos       1296675 non-null  float64
 24  category_travel             1296675 non-null  float64
 25  year                        1296675 non-null  int64
 26  month                       1296675 non-null  int64
 27  day                         1296675 non-null  int64
dtypes: float64(21), int64(7)
memory usage: 277.0 MB
```

2.2.5 Balancing the Dataset

When dealing with imbalanced datasets, where one class significantly outnumbers the other, it can lead to biased machine learning models. To address this issue and improve the model's performance, various techniques for balancing the dataset are employed. In our project we will use one of the three most commonly used methods: **Under Sampling**, **Over Sampling**, and **SMOTE**.

Under Sampling

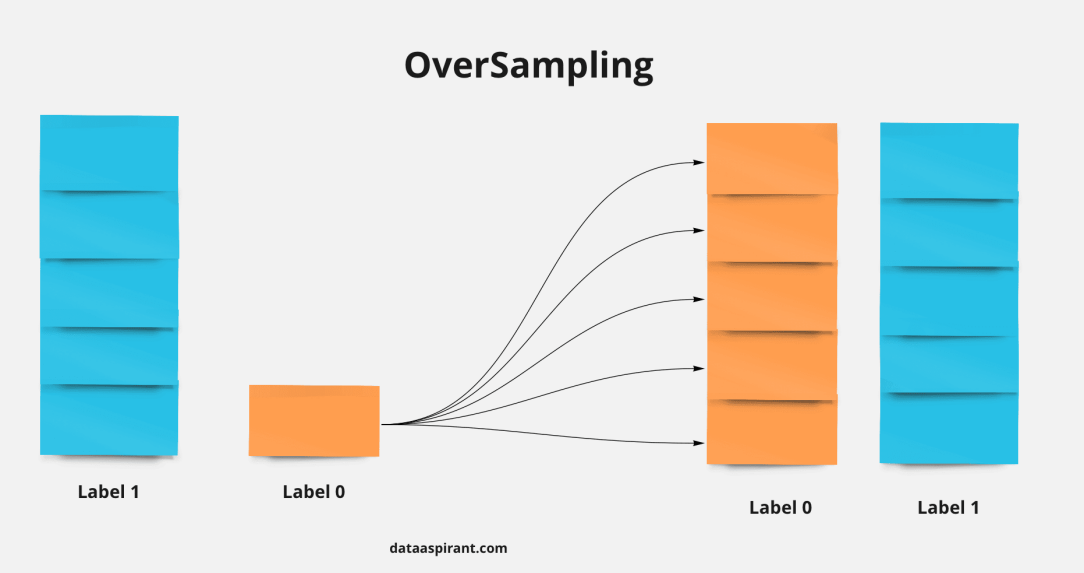
Under sampling involves randomly removing instances from the majority class to balance it with the minority class. The main idea is to reduce the size of the majority class so that it is closer in proportion to the minority class. By doing this, the model is exposed to more examples of the minority class, potentially making it easier to learn and generalize patterns associated with that class. However, under sampling comes with the risk of losing valuable information from the majority class, which may lead to reduced overall performance.



Over Sampling

Over sampling is a technique used to increase the number of instances in the minority class by generating synthetic data points. This process involves replicating existing samples from the minority class or creating new synthetic samples that closely resemble the existing ones. By doing so, the proportion of the minority class is artificially increased, making it more balanced with the majority class.

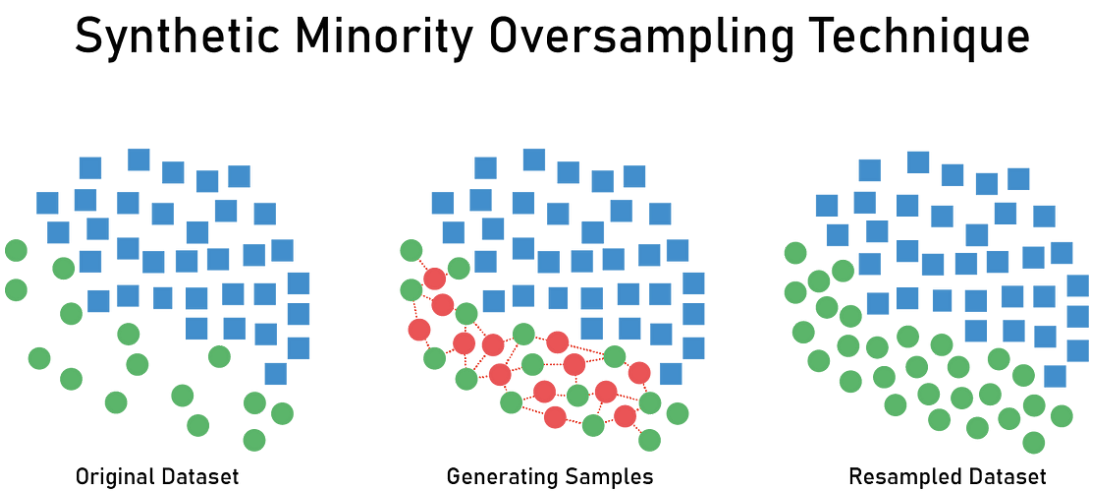
Over sampling ensures that the model has more exposure to the minority class during the training phase, allowing it to learn and recognize the underlying patterns and characteristics of that class better. This may lead to improved model generalization and better performance, but it can also cause overfitting due to duplicated instance in the over sampled dataset.



SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) is a popular and effective technique used in over sampling. It generates synthetic samples by selecting a random instance from the minority class and then finding its k nearest neighbors in the feature

space. New samples are then created by taking a linear combination of the original instance and its neighbors. SMOTE helps to address the overfitting issue that arises with simple over sampling by introducing more diversity in the synthesized samples.



2.2.6 Splitting the data

In this section, we will discuss the process of splitting our dataset for training and testing purposes. The dataset we are working with has already been divided into two separate files: "fraudTrain.csv" and "fraudTest.csv". All the data preprocessing steps we performed earlier in the previous sections have been applied to both the training and testing sets to ensure consistency and fairness in our analysis.

For the training phase, we will employ cross-validation with stratified k-fold. Cross-validation is a robust technique that allows us to make the most of our available data by repeatedly partitioning the training set into subsets for training and validation. In stratified k-fold cross-validation, the data is divided into k-folds with $k = 5$ while preserving the class distribution, ensuring that each fold has a proportional representation of both fraudulent and non-fraudulent transactions. This approach is particularly useful when dealing with imbalanced datasets like ours, where one class is significantly rarer than the other.

By using stratified k-fold cross-validation during training, we can obtain a more reliable estimation of our model's performance. The results will be less sensitive to the initial train-test split and provide us with a better understanding of how well our model generalizes to new, unseen data.

On the other hand, for testing our model's final performance, we will use the separate testing set "fraudTest.csv". This dataset has not been used during model training or cross-validation, ensuring a fair evaluation of the model's ability to generalize to completely new and unseen data.

2.3 Modeling

In this section, we will explore various machine learning techniques for the classification task. Our goal is to build a robust and accurate model that can effectively distinguish between fraudulent and non-fraudulent transactions. To achieve this, we will consider a diverse set of classifiers, including Decision Tree Classifier, CatBoost Classifier, Extra Trees Classifier, K Neighbors Classifier, Extreme Gradient Boosting, Light Gradient Boosting Machine, Gradient Boosting Classifier, and Logistic Regression.

Before diving into the modeling process, we need to address the challenge of dealing with an imbalanced dataset, where the number of fraudulent transactions is significantly lower than the non-fraudulent ones. To tackle this issue and ensure fair evaluation, we will follow several scenarios during the training, testing, and evaluation phases.

Firstly, we will work with the imbalanced dataset that we prepared in the data-preparation section, and apply each machine learning algorithm for training, testing, and evaluation. This will give us insights into how these algorithms perform on the data in its natural distribution.

Next, we will explore the impact of under-sampling, by applying the Under-Sampling technique on the imbalanced dataset. Under this scenario, we will repeat the training, testing, and evaluation process to observe how the models adapt to the balanced data.

Similarly, we will consider the over-sampling technique to handle the class imbalance. By oversampling the minority class, we aim to create a balanced dataset and assess the models' performance under this scenario.

Lastly, we will utilize SMOTE (Synthetic Minority Over-sampling Technique), to generating synthetic samples of the minority class to balance the data.

To facilitate the modeling process and streamline the training, testing, and evaluation of multiple algorithms across different scenarios, we will leverage the "PyCaret" library. PyCaret provides a powerful and user-friendly interface to work with various machine learning models. It automates many aspects of the machine learning pipeline, including data preprocessing, feature engineering, model selection, hyperparameter tuning, and performance evaluation.

2.3.1 What is PyCaret?

PyCaret is a powerful and user-friendly Python library designed to simplify and expedite the entire machine learning workflow. It provides a high-level interface that automates various stages of the machine learning process, including data preparation, feature engineering, model selection, hyperparameter tuning, and model evaluation. With PyCaret, data scientists and machine learning practitioners can efficiently experiment with multiple algorithms and streamline the development of robust and accurate predictive models.

One of the key advantages of PyCaret is its ability to reduce the amount of code needed to perform complex machine learning tasks. It offers an extensive collection of preprocessing functions and transformation techniques, making it easy to handle diverse data types and formats. Additionally, PyCaret comes equipped with a vast selection of machine learning algorithms, ranging from traditional methods like linear regression and decision trees to advanced techniques such as gradient boosting and deep learning.

2.3.2 Modeling with the Original Imbalanced Dataset

In this section, we will start by using the original clean dataset that we prepared in the previous sections. We will apply various classification algorithms for training, then testing, and evaluating these models on the Testing dataset, we aim to identify the most suitable classifier for detecting fraudulent transactions.

Environment setup

The first step is to run the setup function of pycaret, this function initializes the experiment in PyCaret and creates the transformation pipeline based on all the parameters passed in the function. The setup function must be called before executing any other function. It takes two required parameters: data and target.

```
from pycaret.classification import *

s = setup(data=new_df,
          target='is_fraud',
          test_data=new_test,
          index=False,
          normalize=True,
          normalize_method="minmax",
          transformation=True,
          transformation_method = 'yeo-johnson',
          fold=5,
          use_gpu=True
          )
```

The parameters we passed to the setup function:

1. **data**: The data that pycaret will use to train the machine learning algorithms.
2. **target**: The target variable.
3. **test_data**: The testing set .
4. **index**: to ignore the index column .
5. **normalize**: set to True, to use normalisation.
6. **normalize_method**: Min-Max Normalization.
7. **transformation**: to use Transformation to change the distribution of the features in our dataset.
8. **transformation_method**: ”yeo-johnson” transformation method.
9. **fold**: the number of fold will pycaret uses in the stratified k-fold.
10. **use_gpu**: set to True to train with GPU.

Compare Models

we will tun the function compare_models() of pycaret and we will pass ”F1” as a parameter for pycaret to choose the best model based on the best F1 score during Cross-validation.

the Choice of F1 is because of the balance it gives between recall and precision. Accuracy cannot be a useful indicator because if the model mispredicted all the True Positive cases, we will get an accuracy of 99.4%.

Train & *TestScores*

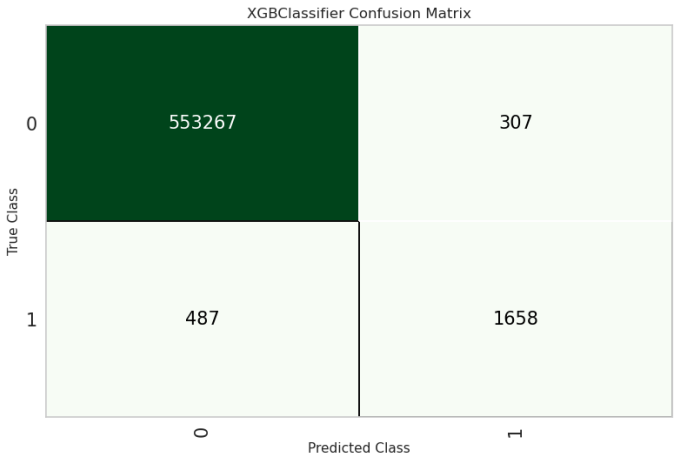
Crosss-Validation Score

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.9977	0.9962	0.7389	0.8790	0.7925	0.7914	0.7996	12.2900
dt	Decision Tree Classifier	0.9975	0.8968	0.7948	0.7837	0.7890	0.7878	0.7879	22.0220
gbc	Gradient Boosting Classifier	0.9977	0.9848	0.7052	0.8706	0.7790	0.7778	0.7823	276.6880
catboost	CatBoost Classifier	0.9962	0.9969	0.7535	0.8048	0.7406	0.7389	0.7576	88.0620
lightgbm	Light Gradient Boosting Machine	0.9967	0.8886	0.7068	0.7252	0.7148	0.7131	0.7137	15.5860
et	Extra Trees Classifier	0.9973	0.9775	0.5441	0.9879	0.7002	0.6990	0.7313	129.8020
ada	Ada Boost Classifier	0.9962	0.9886	0.4781	0.7736	0.5898	0.5880	0.6058	91.2660
knn	K Neighbors Classifier	0.9948	0.0000	0.2701	0.7278	0.3743	0.3723	0.4292	35.4340
lda	Linear Discriminant Analysis	0.9909	0.8368	0.4219	0.3013	0.3505	0.3461	0.3516	13.0280
lr	Logistic Regression	0.9949	0.0000	0.1809	0.7447	0.2889	0.2873	0.3634	31.9700
qda	Quadratic Discriminant Analysis	0.9746	0.9293	0.7220	0.1600	0.2595	0.2524	0.3299	10.6200
nb	Naive Bayes	0.8188	0.8229	0.6952	0.0221	0.0429	0.0320	0.1013	6.8060
svm	SVM - Linear Kernel	0.5448	0.0000	0.8602	0.0109	0.0215	0.0102	0.0616	175.8460
ridge	Ridge Classifier	0.9942	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	7.5380
dummy	Dummy Classifier	0.9942	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	11.6780

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
[I] [18:08:54.141993] Unused keyword parameter: n_jobs during cuML estimator initialization
```

Testing Score

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extreme Gradient Boosting	0.9986	0.9986	0.7730	0.8438	0.8068	0.8061	0.8069



As we can see, the model had a good score on the true negatives, but an average score on the True Positive. let's explore different modeling strategies and check if we can get better results.

2.3.3 Using Under-Sampling

Balancing the Dataset

```
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

rus = RandomUnderSampler()
x_train = new_df.drop("is_fraud", axis=1)
y_train = new_df["is_fraud"]
```

```
rus_x_train, rus_y_train = rus.fit_resample(x_train, y_train)
under_sampled_df = pd.concat([pd.DataFrame(rus_x_train, columns=x_train.columns), pd.Series(rus_y_train,␣
↪name='is_fraud')], axis=1)
```

```
In [ ]: under_sampled_df['is_fraud'].value_counts()
Out[67]: 0    7506
         1    7506
         Name: is_fraud, dtype: int64
```

Environment Setup

```
from pycaret.classification import *

s = setup(data=under_sampled_df,
          target='is_fraud',
          test_data=new_test,
          index=False,
          normalize=True,
          normalize_method="minmax",
          transformation=True,
          transformation_method = 'yeo-johnson',
          fold=5,
          use_gpu=True
          )
```

Train & *TestScores*

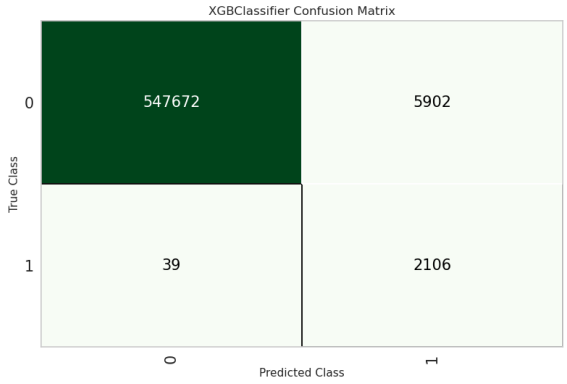
Cross-Validation Score

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.9308	0.9891	0.8824	0.9769	0.9267	0.8616	0.8663	0.6640
lightgbm	Light Gradient Boosting Machine	0.9204	0.9879	0.8612	0.9767	0.9138	0.8408	0.8483	1.9880
catboost	CatBoost Classifier	0.9184	0.9860	0.8597	0.9737	0.9106	0.8368	0.8449	112.4760
dt	Decision Tree Classifier	0.9177	0.9177	0.8642	0.9673	0.9076	0.8355	0.8452	0.3340
gbc	Gradient Boosting Classifier	0.9141	0.9825	0.8574	0.9664	0.9048	0.8283	0.8369	3.5840
et	Extra Trees Classifier	0.9081	0.9811	0.8351	0.9771	0.8955	0.8161	0.8285	1.5520
ada	Ada Boost Classifier	0.8860	0.9066	0.8156	0.9487	0.8717	0.7720	0.7839	1.2280
qda	Quadratic Discriminant Analysis	0.8315	0.9073	0.7330	0.9131	0.8104	0.6629	0.6780	0.3360
lda	Linear Discriminant Analysis	0.7824	0.8490	0.7383	0.8096	0.7722	0.5647	0.5671	0.3860
lr	Logistic Regression	0.7826	0.0000	0.7358	0.8116	0.7713	0.5653	0.5682	2.3160
knn	K Neighbors Classifier	0.7929	0.0000	0.7038	0.8540	0.7696	0.5858	0.5962	0.7440
svm	SVM - Linear Kernel	0.7395	0.0000	0.7190	0.7497	0.7339	0.4790	0.4795	6.5960
nb	Naive Bayes	0.7064	0.8071	0.7366	0.6946	0.7150	0.4129	0.4137	0.1800
dummy	Dummy Classifier	0.4999	0.5000	0.2000	0.1000	0.1333	0.0000	0.0000	0.1800
ridge	Ridge Classifier	0.5332	0.0000	0.0673	0.9852	0.1255	0.0664	0.1814	0.3500

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
[I] [20:12:25.743510] Unused keyword parameter: n_jobs during cuML estimator initialization
```

Testing Score

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extreme Gradient Boosting	0.9893	0.9990	0.9818	0.2630	0.4149	0.4113	0.5053



For under sampling we got a really good recall, but a bad precision. Let's see if we can improve our result by Over Sampling.

2.3.4 Over Sampling

Balancing the dataset

```
from imblearn import over_sampling

X = new_df.drop('is_fraud', axis=1)
y = new_df['is_fraud']

oversampler = over_sampling.RandomOverSampler()

x_train, y_train = oversampler.fit_resample(X, y)

over_sampled_df = pd.concat([pd.DataFrame(x_train, columns=X.columns), pd.Series(y_train, name='is_fraud
 ↪')], axis=1)

over_sampled_df["is_fraud"].value_counts()
```

```
0    1289169
1    1289169
Name: is_fraud, dtype: int64
```

Environment Configuration

```
from pycaret.classification import *

s = setup(data=over_sampled_df,
          target='is_fraud',
          test_data=new_test,
          index=False,
          normalize=True,
          normalize_method="minmax",
          transformation=True,
          transformation_method = 'yeo-johnson',
          fold=5,
          use_gpu=True
          )
```

Train & *TestScores*

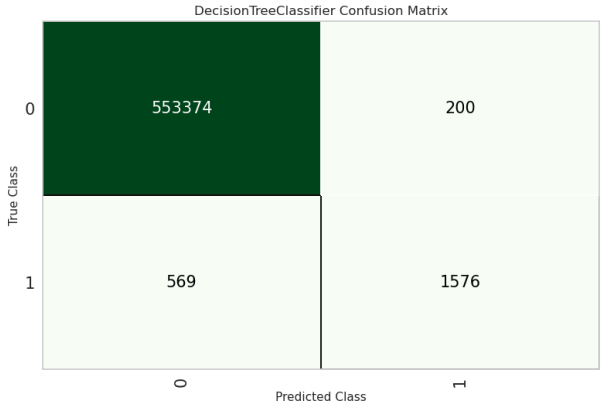
Cross-Validation Score

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
dt	Decision Tree Classifier	0.9957	0.9957	1.0000	0.9915	0.9957	0.9914	0.9915	56.6500
catboost	CatBoost Classifier	0.9693	0.9964	0.9986	0.9435	0.9702	0.9386	0.9403	152.3400
et	Extra Trees Classifier	0.9595	1.0000	1.0000	0.9344	0.9638	0.9190	0.9266	371.7820
knn	K Neighbors Classifier	0.9478	0.0000	1.0000	0.9130	0.9526	0.8956	0.9042	191.4180
xgboost	Extreme Gradient Boosting	0.9437	0.9955	1.0000	0.9020	0.9476	0.8874	0.8946	52.5440
lightgbm	Light Gradient Boosting Machine	0.9406	0.9942	0.9985	0.9014	0.9458	0.8811	0.8902	53.6140
gbc	Gradient Boosting Classifier	0.9437	0.9890	0.9692	0.9232	0.9454	0.8875	0.8891	600.5020
ada	Ada Boost Classifier	0.8727	0.8884	0.9505	0.8409	0.8879	0.7453	0.7604	242.5060
qda	Quadratic Discriminant Analysis	0.8490	0.9221	0.7989	0.8887	0.8412	0.6980	0.7019	34.9880
lr	Logistic Regression	0.7534	0.0000	0.7981	0.7434	0.7668	0.5067	0.5118	71.1440
svm	SVM - Linear Kernel	0.7466	0.0000	0.7946	0.7347	0.7608	0.4932	0.4982	368.3320
lda	Linear Discriminant Analysis	0.7514	0.8473	0.7738	0.7480	0.7586	0.5028	0.5058	46.5340
nb	Naive Bayes	0.6951	0.8068	0.7434	0.6785	0.7093	0.3902	0.3922	59.3440
ridge	Ridge Classifier	0.5565	0.0000	0.1151	0.9846	0.2054	0.1131	0.2404	50.5940
dummy	Dummy Classifier	0.5000	0.5000	0.2000	0.1000	0.1333	0.0000	0.0000	35.7040

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
[I] [20:44:44.445047] Unused keyword parameter: n_jobs during cuML estimator initialization
```

Testing Score

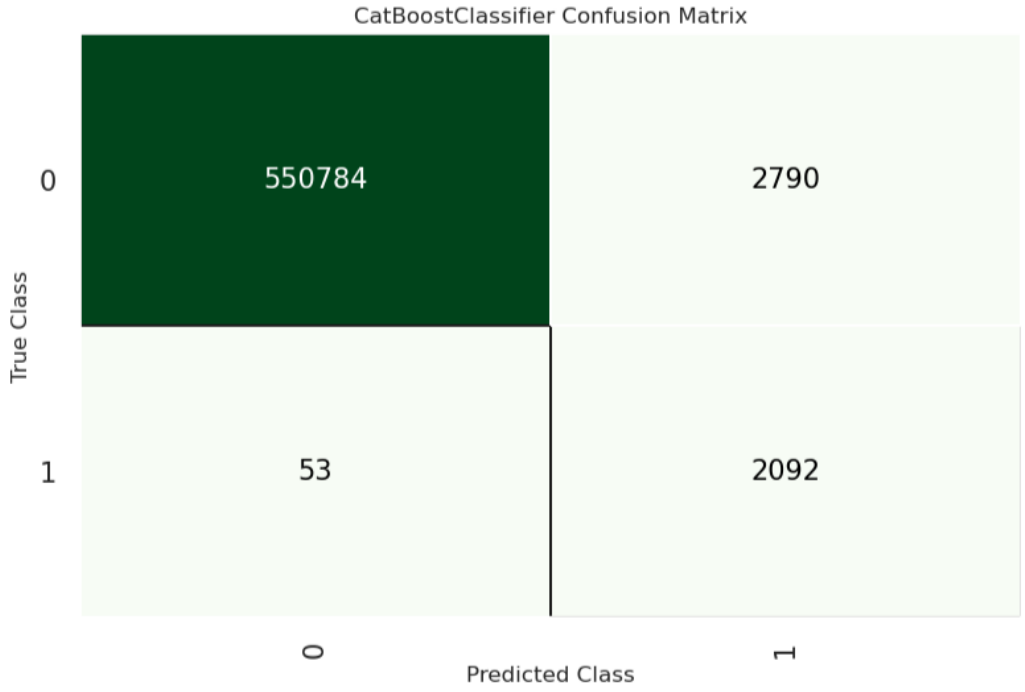
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Decision Tree Classifier	0.9986	0.8672	0.7347	0.8874	0.8039	0.8032	0.8068



This model gave use really good precision, but an average recall, this model doesn't predict a lot of false positives, but for the cost of a high number of false negatives. let's check the result of other algorithms on the testing sets:

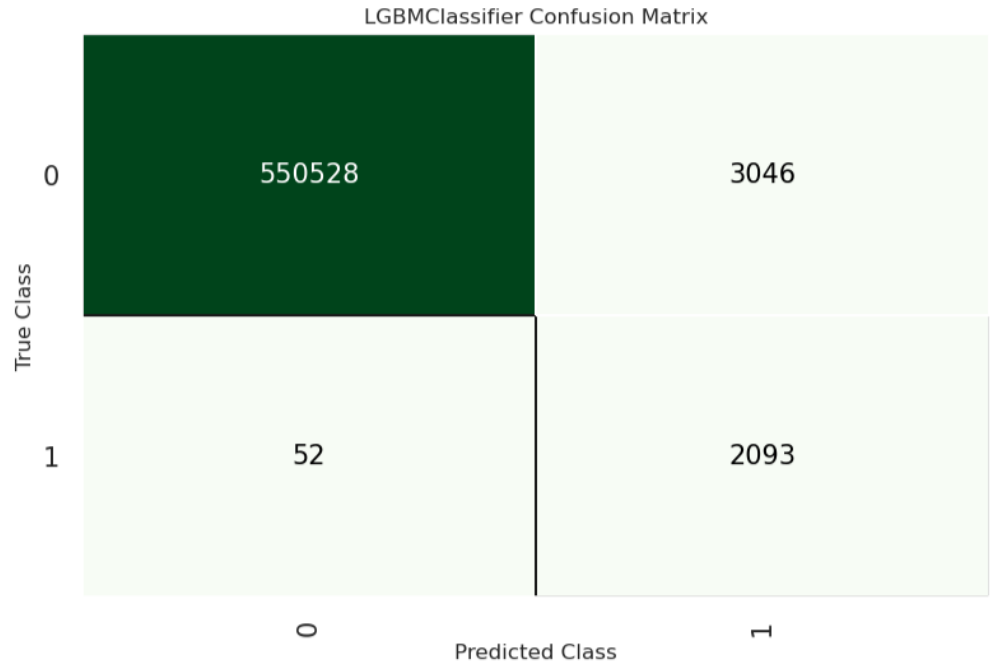
Catboost

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	CatBoost Classifier	0.9949	0.9994	0.9753	0.4285	0.5954	0.5932	0.6447



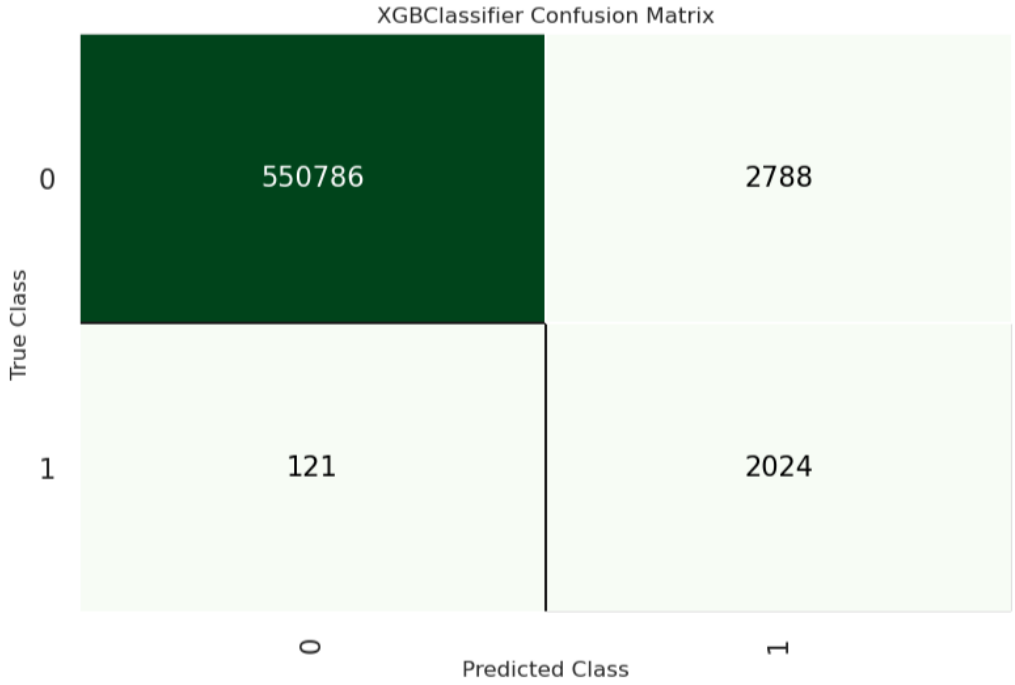
LightGBM

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Light Gradient Boosting Machine	0.9944	0.9993	0.9758	0.4073	0.5747	0.5724	0.6285



XGBoost

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extreme Gradient Boosting	0.9948	0.9986	0.9436	0.4206	0.5819	0.5796	0.6281



From the previous results, CatBoost had the best results by having the lowest number of fraud transactions predicted as non-fraud, and relatively low number of non fraud predicted as fraud.

Let's see if we can have better results with the SMOTE technique.

2.3.5 SMOTE

Balancing the dataset

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=27)

X = new_df.drop('is_fraud', axis=1)
y = new_df['is_fraud']

x_train, y_train = smote.fit_resample(X, y)

smote_df = pd.concat([pd.DataFrame(x_train, columns=X.columns), pd.Series(y_train, name='is_fraud')],␣
↪axis=1)

smote_df["is_fraud"].value_counts()
```

```
0    1289169
1    1289169
Name: is_fraud, dtype: int64
```

Environment Setup

```
from pycaret.classification import *
```

```
s = setup(data=smote_df,
          target='is_fraud',
          test_data = new_test,
          index=False,
          normalize=True,
          normalize_method="minmax",
          transformation=True,
          transformation_method = 'yeo-johnson',
          fold=5,
          use_gpu=True
          )
```

Train & *TestScores*

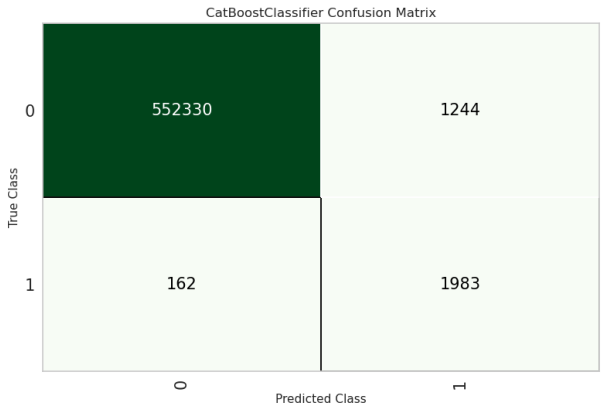
Cross-Validation Scores

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.9730	0.9985	0.9944	0.9544	0.9738	0.9460	0.9473	155.0240
gbc	Gradient Boosting Classifier	0.9668	0.9929	0.9690	0.9651	0.9669	0.9335	0.9337	886.3920
lightgbm	Light Gradient Boosting Machine	0.9601	0.9983	0.9960	0.9326	0.9624	0.9203	0.9242	56.2380
dt	Decision Tree Classifier	0.9526	0.9526	0.9990	0.9330	0.9605	0.9052	0.9178	77.8880
xgboost	Extreme Gradient Boosting	0.9535	0.9986	0.9985	0.9198	0.9566	0.9071	0.9126	38.5140
et	Extra Trees Classifier	0.9427	0.9997	0.9997	0.9197	0.9529	0.8854	0.9011	440.7760
knn	K Neighbors Classifier	0.9317	0.0000	0.9994	0.8898	0.9390	0.8634	0.8759	198.8500
lr	Logistic Regression	0.8997	0.0000	0.9216	0.8919	0.9045	0.7994	0.8037	72.1420
ada	Ada Boost Classifier	0.8851	0.8814	0.9600	0.8553	0.8998	0.7703	0.7862	284.1840
lda	Linear Discriminant Analysis	0.8884	0.9485	0.9002	0.8835	0.8908	0.7768	0.7787	47.4560
svm	SVM - Linear Kernel	0.8749	0.0000	0.9146	0.8534	0.8814	0.7498	0.7545	360.1280
nb	Naive Bayes	0.7207	0.8106	0.7448	0.7113	0.7275	0.4414	0.4421	59.3060
qda	Quadratic Discriminant Analysis	0.7559	0.9622	0.5119	1.0000	0.6771	0.5119	0.5865	33.6540
ridge	Ridge Classifier	0.6280	0.0000	0.2562	0.9991	0.4078	0.2560	0.3828	50.0280
dummy	Dummy Classifier	0.5000	0.5000	0.2000	0.1000	0.1333	0.0000	0.0000	28.5200

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
[I] [09:24:47.907841] Unused keyword parameter: n_jobs during cuML estimator initialization
```

Testing Score

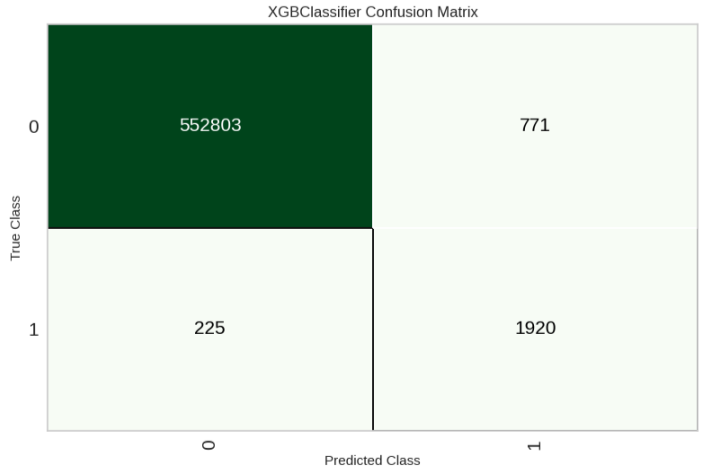
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	CatBoost Classifier	0.9975	0.9990	0.9245	0.6145	0.7383	0.7371	0.7526



Let's check the results of some other algorithms on the testing set:

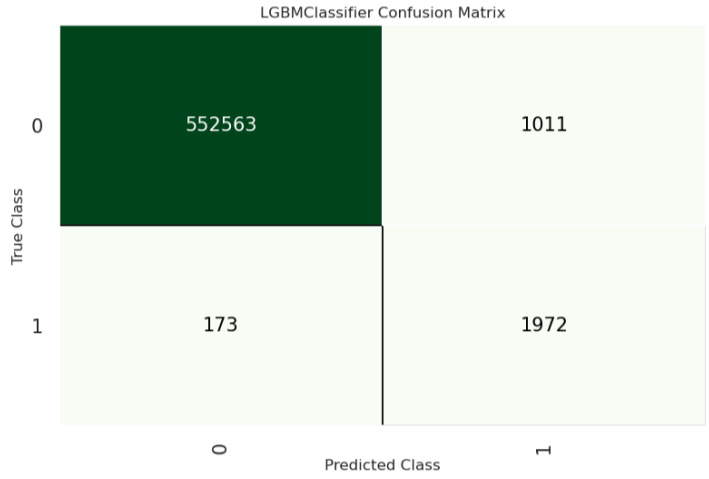
XGBoost

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extreme Gradient Boosting	0.9982	0.9988	0.8951	0.7135	0.7940	0.7932	0.7983



LightGBM

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Light Gradient Boosting Machine	0.9979	0.9991	0.9193	0.6611	0.7691	0.7681	0.7786



2.3.6 Model Selection

From the all the models we created, the models used with SMOTE gave the best results, especially catboost and xgboost.

Now we are left with two choices, The choice of having less fraudulent transactions using the catboost model, or less non fraudulent transactions using the xgboost model.

In our case, we chose the catboost model, for less fraudulent transactions. In the end it's a business decision, which means the choice is left for the company.

2.3.7 Model Tuning

By defining a grid of values, for the most important hyper parameters of CatBoost, we can tune the model with pycaret using the function **tune_model()**.

```
                "iterations":[50,100,250,500,750,1000,1500,2000],
                "learning_rate":[0.001,0.03,0.01,0.1],
                "depth":[4,5,6,7,8,9,10],
                "l2_leaf_reg":[2,3,5,7,10],
                "random_strength":[0.05,0.1,1,3,5,7,10]
                }

tuned_catboost = tune_model(catboost, custom_grid=catboost_grid,search_
  ↪library="scikit-optimize",
                            n_iter=10, tuner_verbose=10)
```

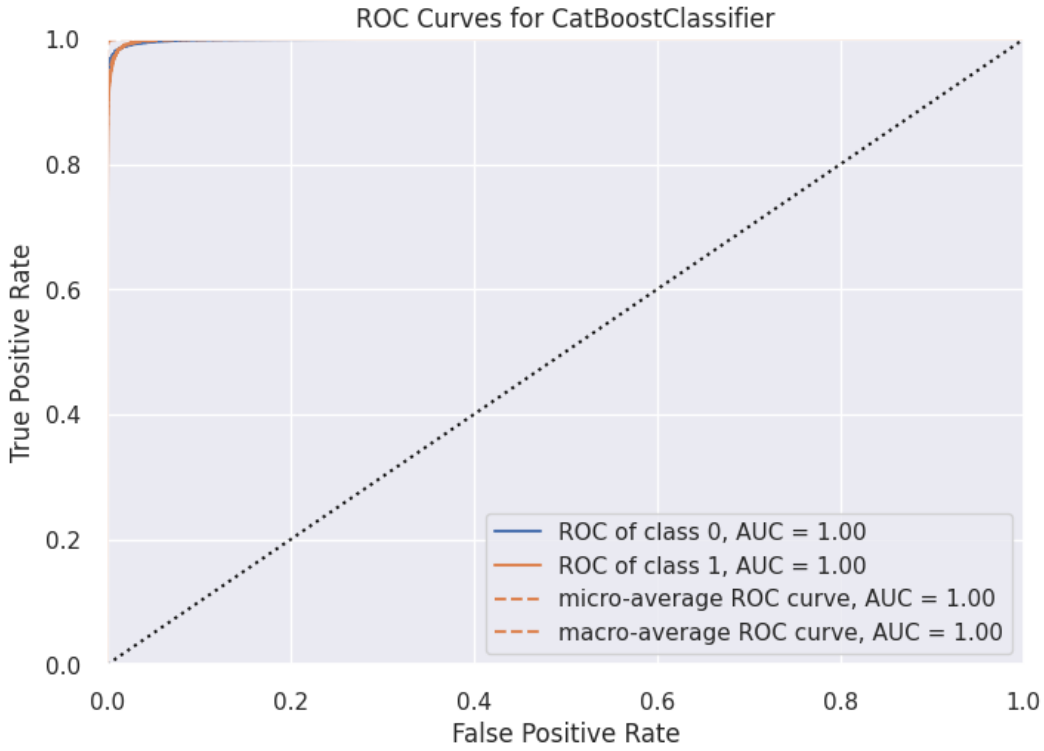
The tuning takes a really long time, and we coudn't complete it in time, for that we are going to use the catboost model in our deployment.

2.3.8 Performance Analysis of the chosen Model

PyCaret Provides a function "plot_model" which allows us to analyze and interpret the performance of a machine learning model. With plot_model, we can generate a variety of informative plots, each highlighting different aspects of the model's behavior and results.

Using plot_model, we can visualize essential performance metrics such as the ROC curve, feature importance, confusion matrix, and more.

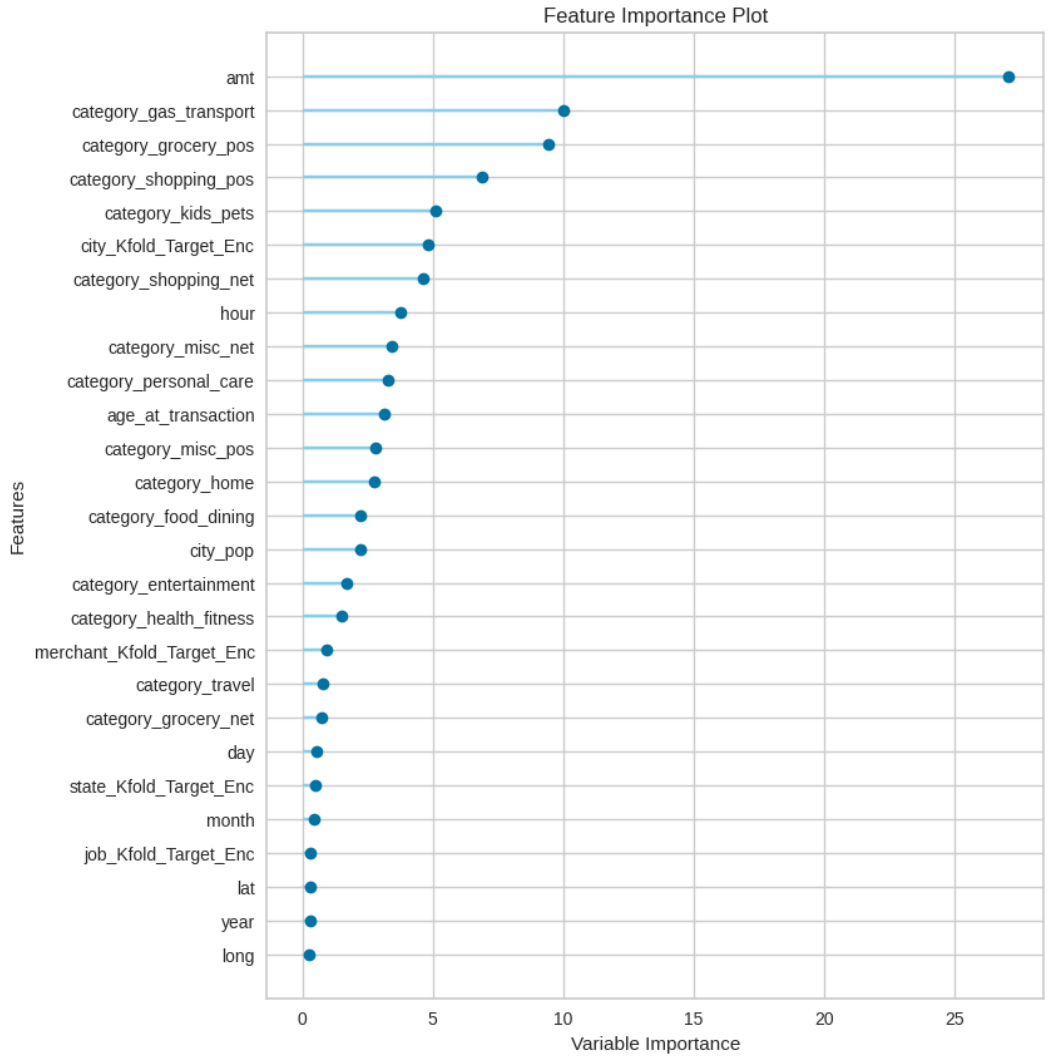
ROC Curve



The ROC (Receiver Operating Characteristic) curve is a graphical representation of the performance of a binary classifier at various classification thresholds. It shows the trade-off between the true positive rate (sensitivity or recall) and the false positive rate (1 - specificity) as the decision threshold is varied.

In our case we are looking at positive results, which means that our model achieved a high true positive rate and a low false positive rate.

Feature Importance



We can see that the most important feature to the catboost mode is the amount of money in each transaction, which means that the column 'amt' has a big impact to the model to predict if the transaction is fraud or not fraud. there is also some important features, like if the category is gas_transport or grocery_pos. we can also notice that lon/lat/year/job/month have a small importance to the model, which indicate we can remove this column and train the model again. in our case we will work with all the features.

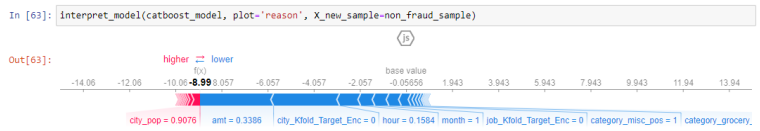
2.3.9 Model Interpretation

We can use the function **interpret_model**, which relies on the **SHAP** Library.

SHAP, which stands for **SHapley Additive exPlanations**, is a popular method for interpreting machine learning models. It is used to explain individual predictions made by complex models. SHAP values provide a way to attribute the prediction of a specific instance (data point) to its input features.

The fundamental idea behind SHAP is based on cooperative game theory, specifically the concept of Shapley values. In cooperative games, Shapley values are used to fairly distribute the ”payoff” among the players. Applied to machine learning, SHAP values assign each feature’s contribution to a prediction in a way that respects the feature interactions and ensures consistency across different predictions.

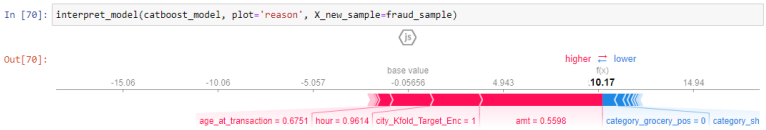
interpreting a non fraud prediction



The model predicted a 0 for a given non fraud sample, and upon interpretation, we obtained a SHAP value of -8.99. This value indicates that the combination of feature values in this instance significantly contributed to a prediction of 0 instead of 1. as we can see, ”amt” and the encoded city column contributed the most in our prediction.

interpreting a fraud prediction

For the fraud transaction, the encoded ”city” column, and the ”amt” contributed the most in our prediction.



2.3.10 Finalize Model

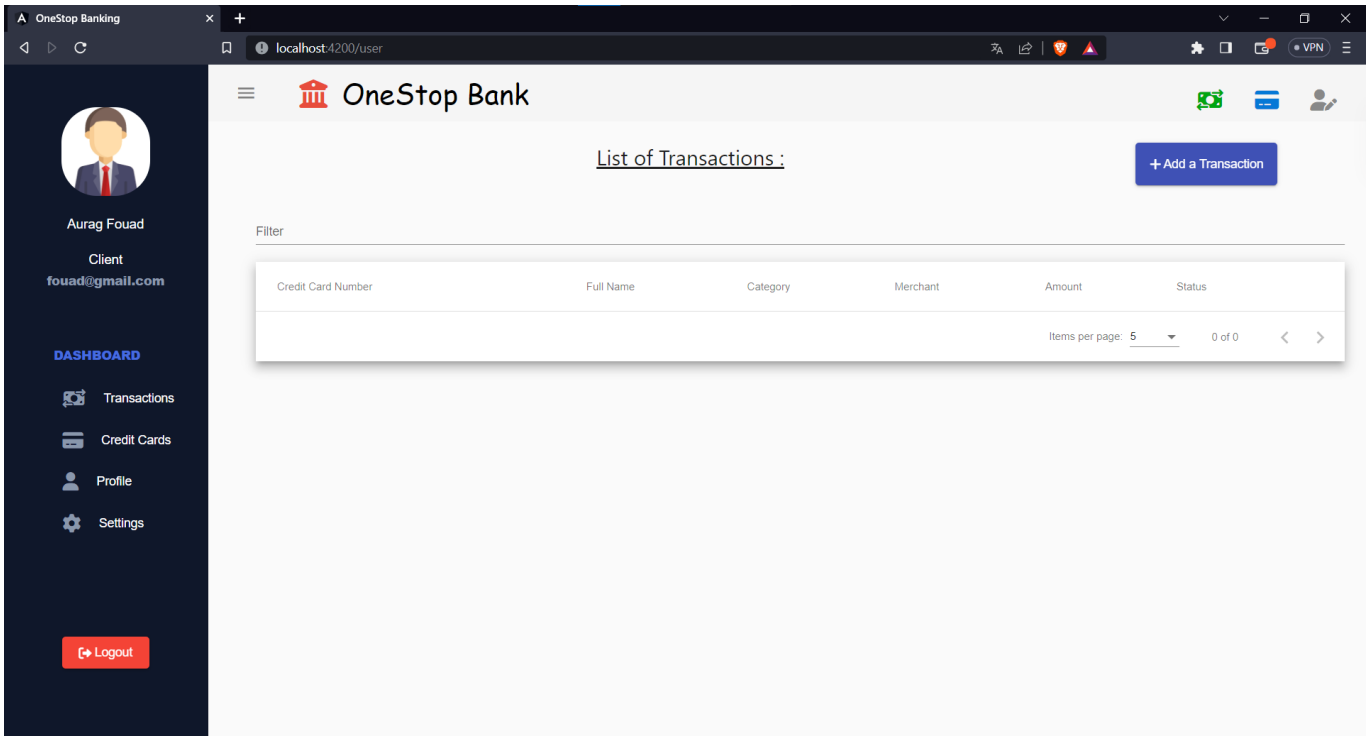
After choosing the best model, it’s time to give that model all the available data, from training and testing sets, because the model is likely to perform better if trained on the whole set of available data than on the single subset used to estimate the model performance.

But for the case of our project, we wont do that, because we still need to deploy the model and test with samples from the testing set. Finalizing our model is going to be done when we make the model for public use.

2.4 Deployment

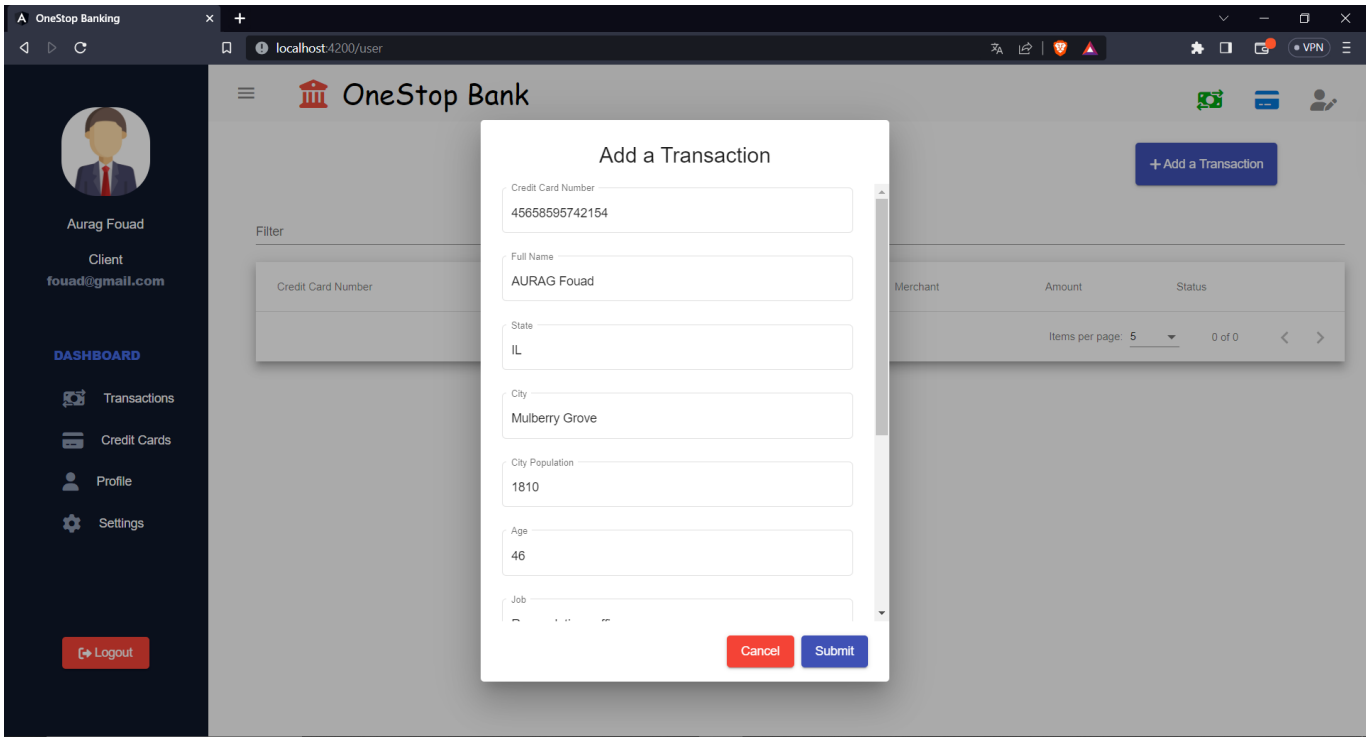
In the final stage of our project, we focused on deploying our trained machine learning model to make it accessible to end-users. For this purpose, we chose to deploy the model within an Angular application to provide a user-friendly interface. To achieve this, we utilized Flask as the back-end API to handle incoming user data. When a user interacts with the Angular application and submits their data for prediction, the Angular front-end sends this data to the Flask back*end through API requests. The Flask back-end then processes the incoming data and feeds it to the deployed machine learning model. After making the prediction, the model’s output is sent back to the Angular front-end via the API, allowing the user to view the prediction results in real-time.

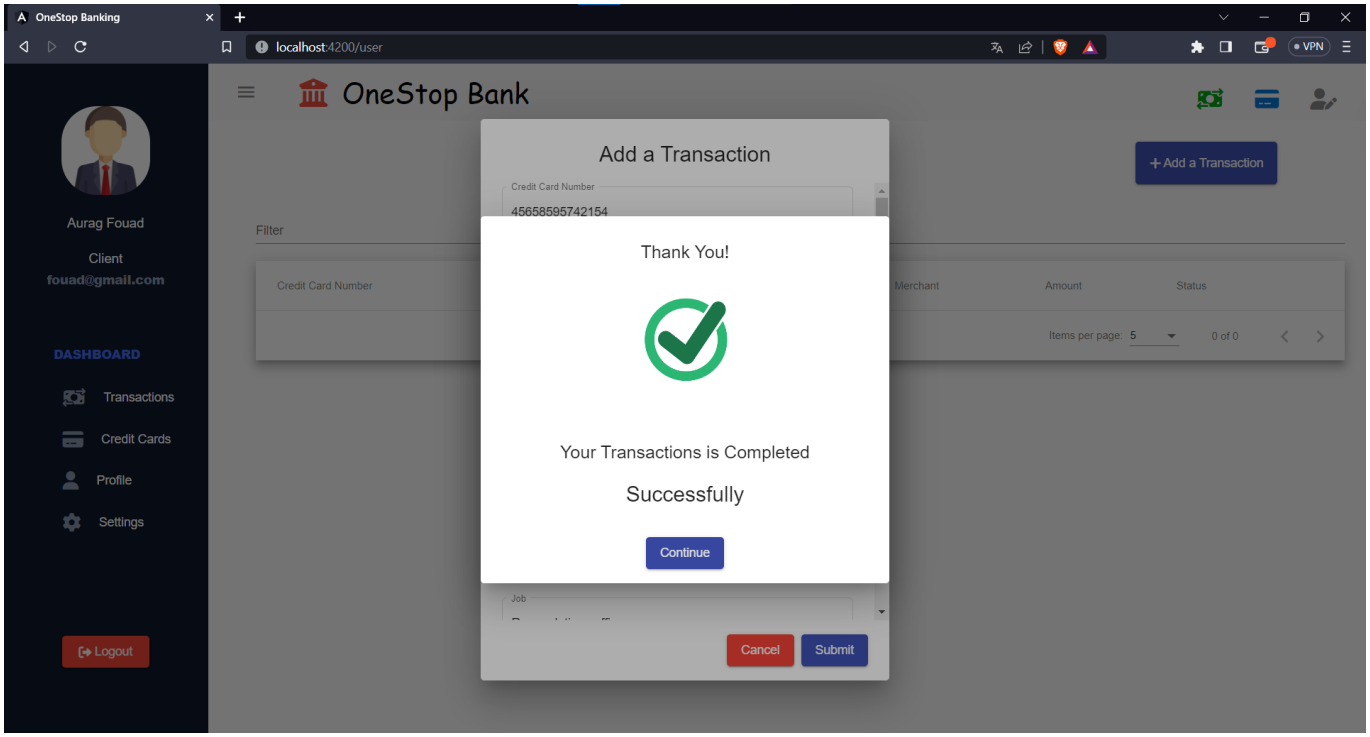
2.4.1 User Interface



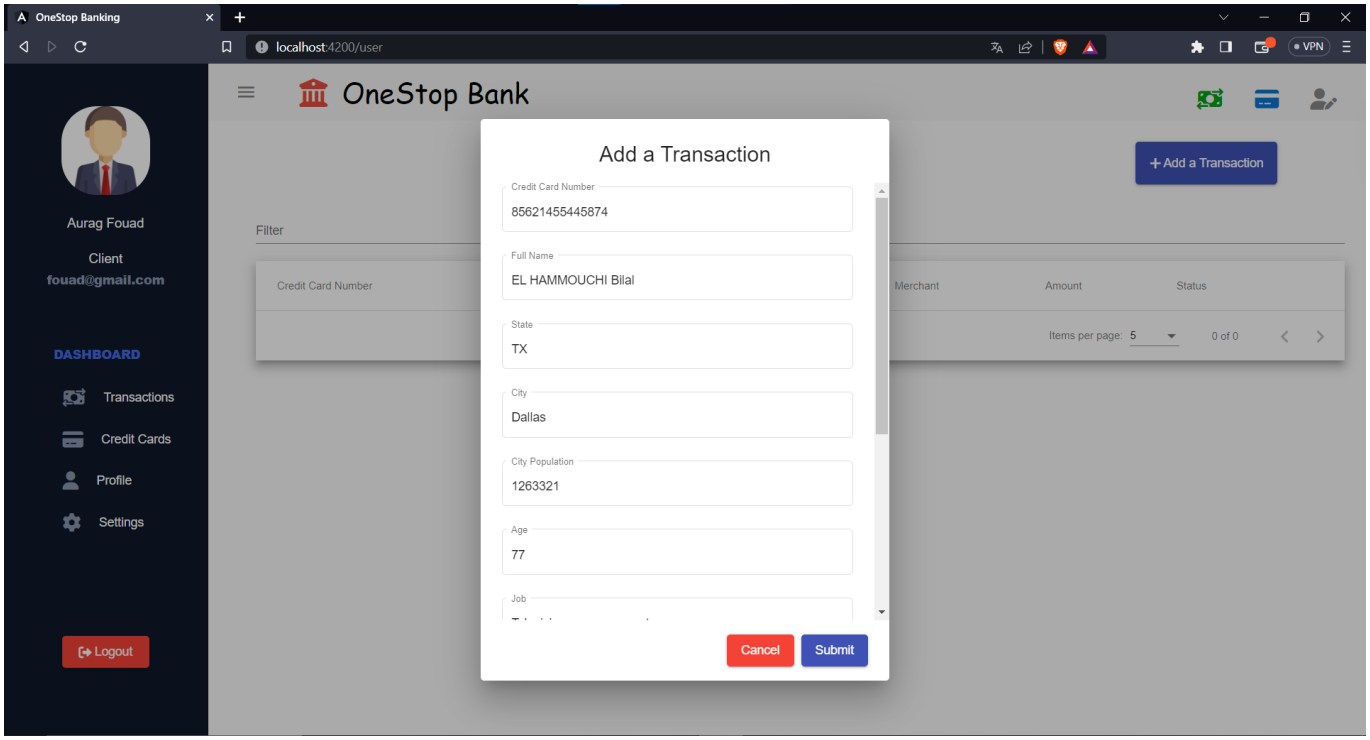
For Testing Purpose please refer to the notebook at the en, you will find the sample i used bellow, you can run the notebook and have new samples as well, make sure to run all the code (except the machine learning chapter) so you can get valid samples from the test dataframe to test with.

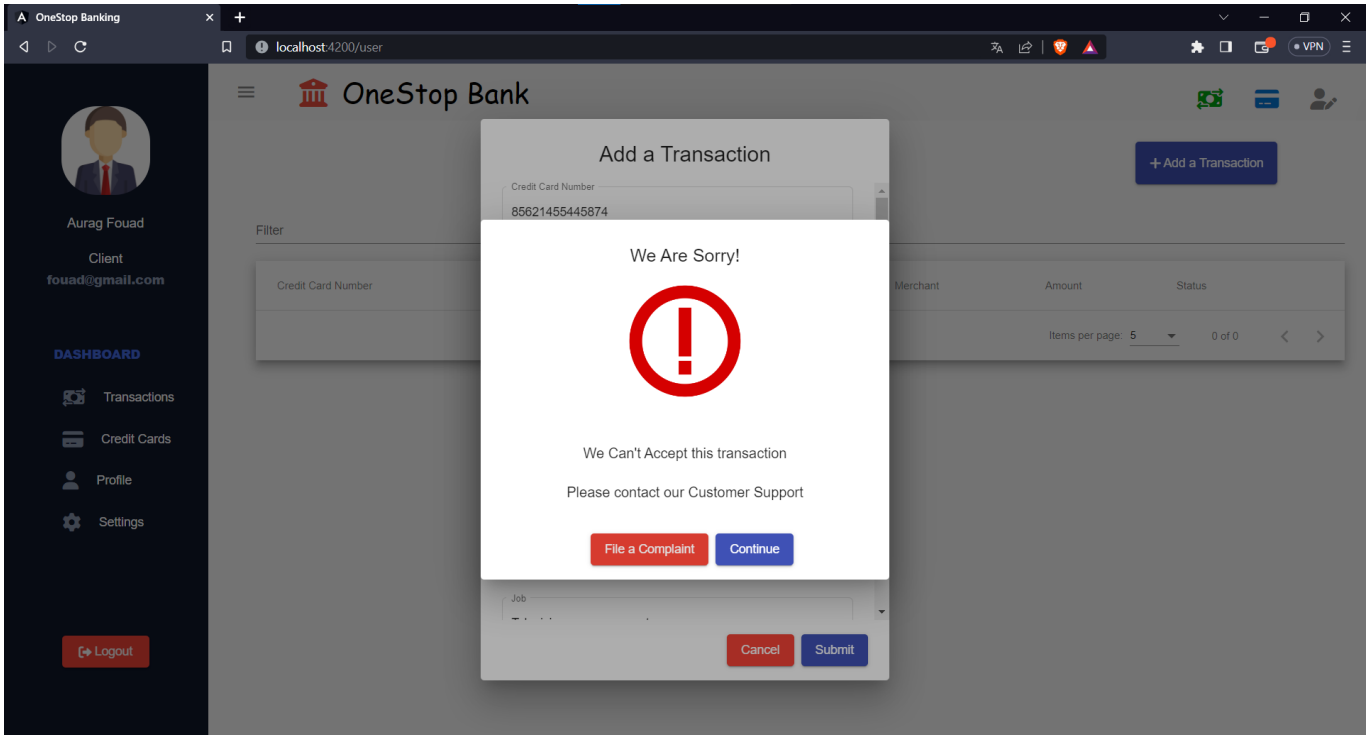
2.4.2 Testing a Non Fraud Transaction





2.4.3 Testing a Fraud Transaction




OneStop Banking
localhost:4200/user
OneStop Bank
Aurag Fouad
Client
fouad@gmail.com
DASHBOARD
Transactions
Credit Cards
Profile
Settings
Logout
+ Add a Transaction
Filter
Credit Card Number
Merchant
Amount
Status
Items per page: 5
0 of 0
Add a Transaction
Credit Card Number
85621455445874
We Are Sorry!
We Can't Accept this transaction
Please contact our Customer Support
File a Complaint
Continue
Job
Cancel
Submit

Conclusion

Throughout this project, I embarked on a data-driven journey to develop an effective fraud detection model. I tackled data preprocessing challenges, handled class imbalance issues, and evaluated various machine learning algorithms for optimal performance. The adoption of PyCaret streamlined the model selection and tuning process, enhancing my ability to compare and analyze different scenarios effectively. Ultimately, the deployed Angular application provides users with a user-friendly and efficient tool to assess the likelihood of fraud in their transactions.

Key Takeaways:

This project has been a valuable learning experience for me. I gained a deeper understanding of data preprocessing techniques, the importance of addressing class imbalance, and the significance of model selection and evaluation. Additionally, the utilization of PyCaret has been enlightening, revealing the potential of this library in facilitating the end-to-end machine learning pipeline. I also improved my skills in deploying machine learning models in real-world applications, fostering a stronger connection between data science and user-friendly interfaces. Overall, this project has enriched my knowledge and expertise in fraud detection, data analysis, and model deployment, leaving me prepared and better equipped for future data-driven projects.