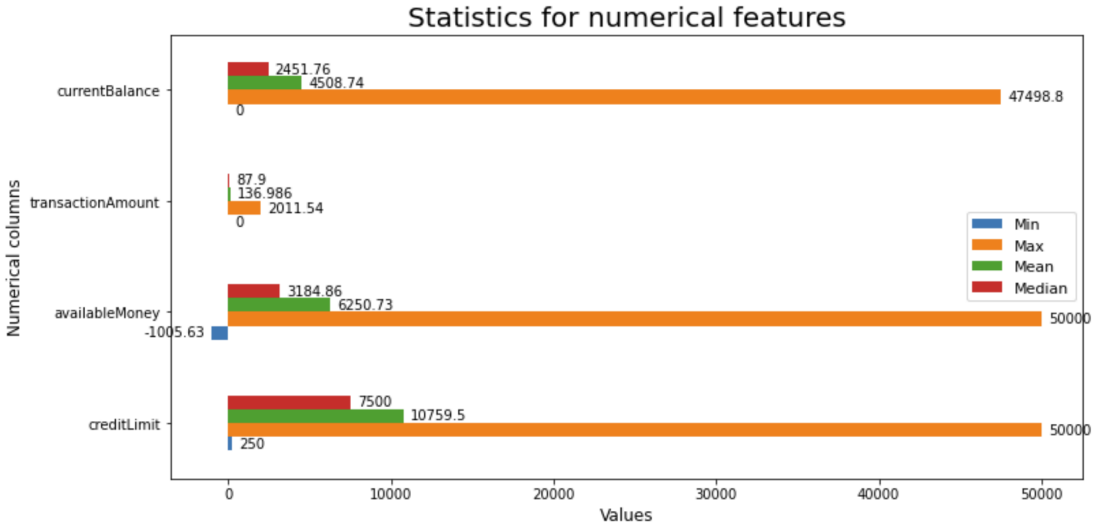
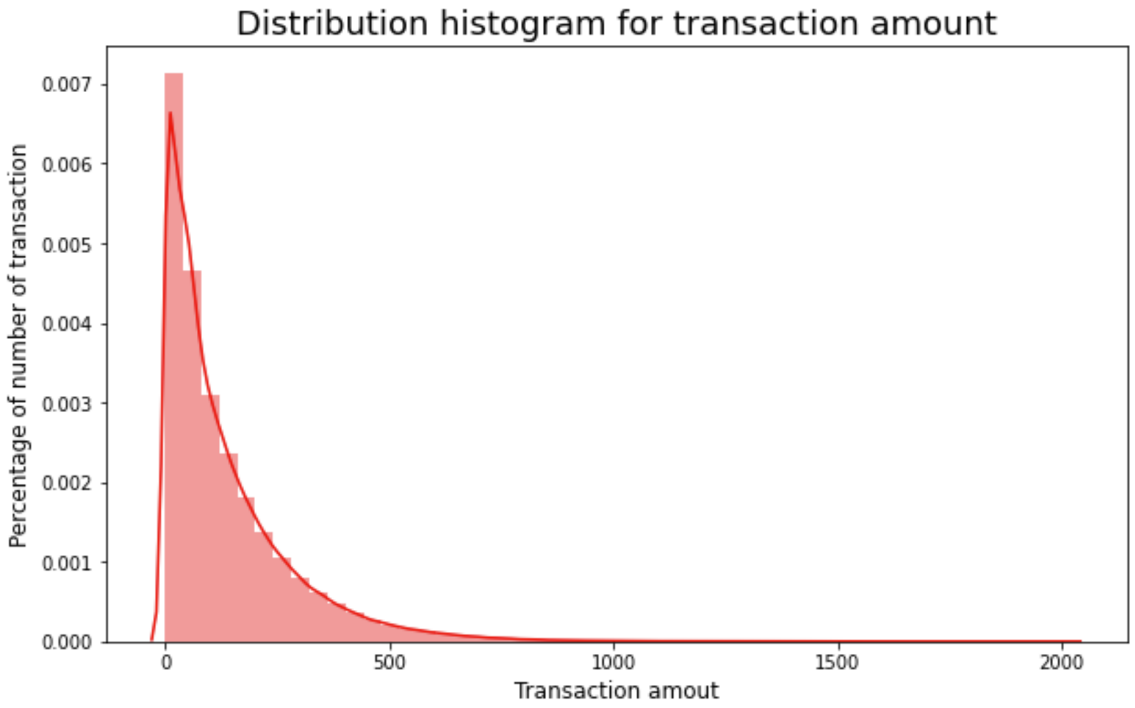
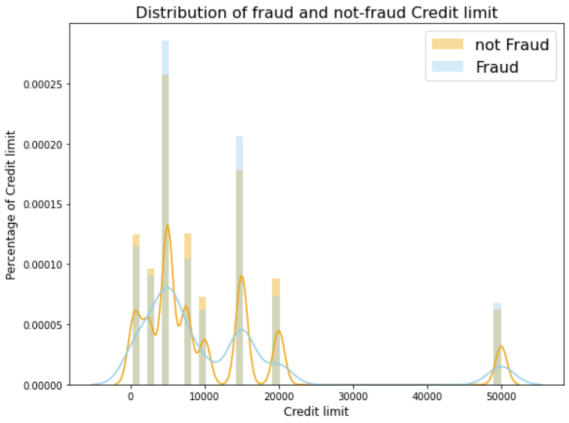
In this task, I used a dataset about card transactions and contained information about whether they were fraudulent to build some models based on the data science techniques I learned to better predict for any fraudulent transaction case for later use. My general ideas and methods are:

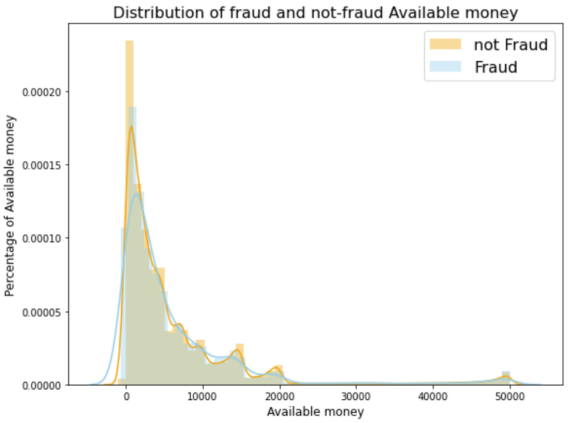
First of all, I used the provided json type data of txt. file into a Python dataframe. I then used this dateFrame to look at the structure of the entire data and understand the data type and meaning of each column. The dataframe is 786363 rows × 29 columns, which means that there are 786363 transactions collected. I quickly identified that the 'isFraud' is the target column. I also realized that there are 4 numerical columns and I looked into some statistics of the features, as well as visualized the histogram distribution of the numerical features with and without considering the fraud case. Besides, there are few columns that are 100% missing data so I decide to leave them out.

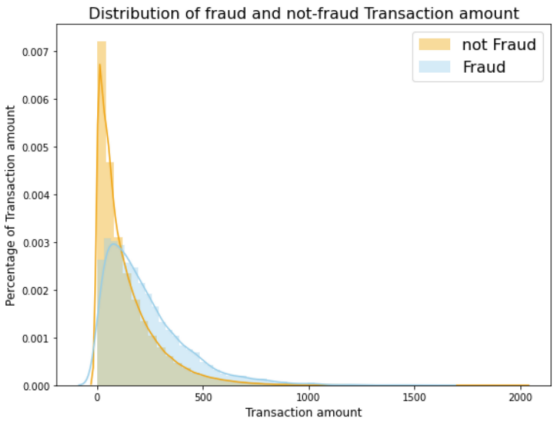


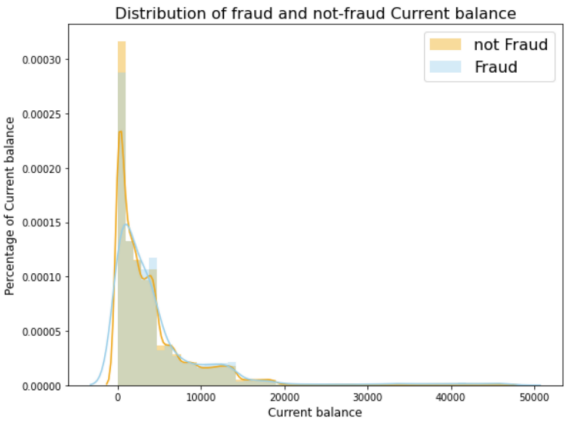
Secondly, I try to visualize some of my findings. The most intuitive thing is to look at the distribution of each transaction amount. This allows me to intuitively understand the transaction value distribution of the data set. I have observed that there is some correlation between numerical information, such as credit limit, transaction amount and current balance. But that wasn't enough, because I wanted to find out if they had anything to do with the fraud case. So I also visualized their distribution under fraud and non-fraud situations. In terms of fraud, I visualized the total number and percentage of non-fraud and fraud for the entire dataset. Finally, I used a strip plot to understand the relationship between fraud and transaction amount from two directions of intensity and distribution. If I have time, I would like to look deeper on the relation between fraud and other categorical features.


Distribution histogram for transaction amount
Percentage of number of transaction
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
500
1000
1500
2000
Transaction amout

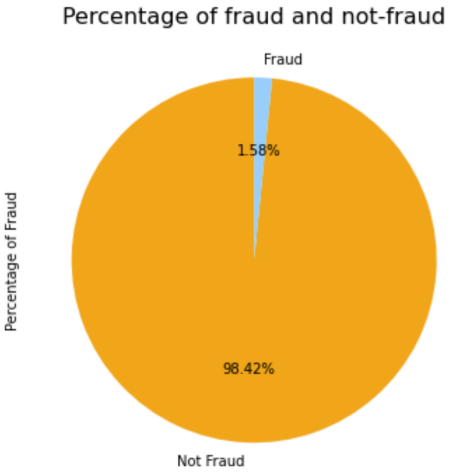

Distribution of fraud and not-fraud Credit limit
not Fraud
Fraud
Percentage of Credit limit
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
10000
20000
30000
40000
50000
Credit limit

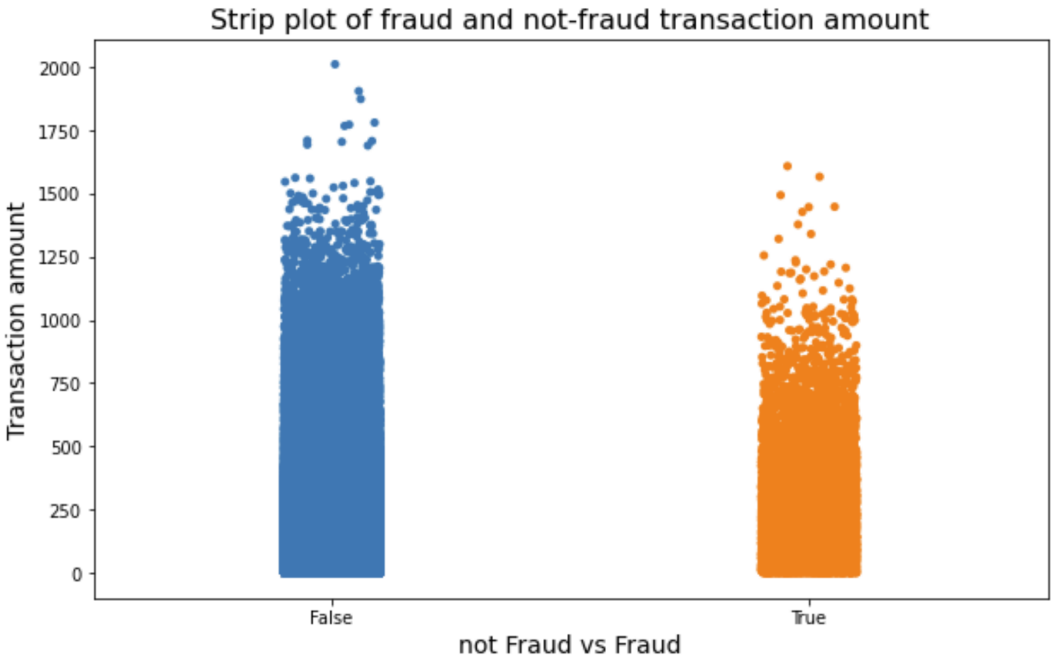

Distribution of fraud and not-fraud Available money
not Fraud
Fraud
Percentage of Available money
0.00020
0.00015
0.00010
0.00005
0.00000
0
10000
20000
30000
40000
50000
Available money


Distribution of fraud and not-fraud Transaction amount
not Fraud
Fraud
Percentage of Transaction amount
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
500
1000
1500
2000
Transaction amount


Distribution of fraud and not-fraud Current balance
not Fraud
Fraud
Percentage of Current balance
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
10000
20000
30000
40000
50000
Current balance

```
False    773946
True      12417
Name: isFraud, dtype: int64
```





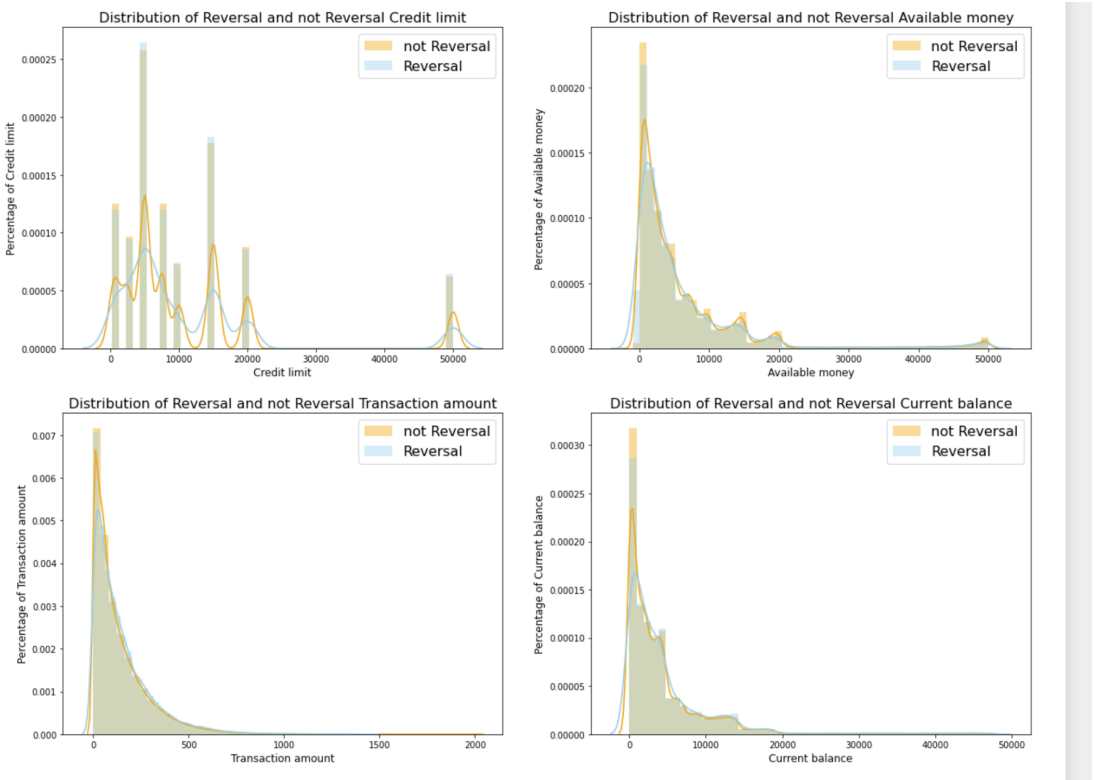
Then comes the most important part, which is Data Wrangling. First of all, according to the question clues, there are some duplicate transaction cases, which are reversal and multi-swipe transactions.

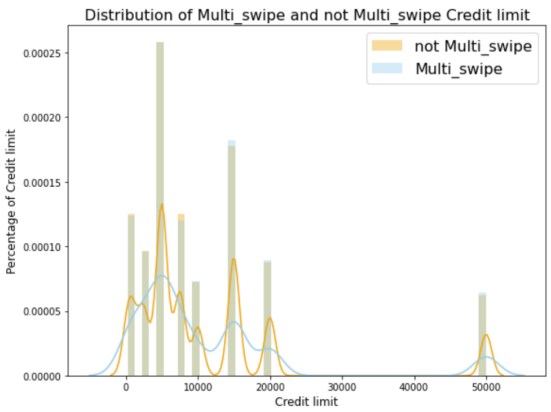
First of all, I decided to deal with reversal transactions. Under a column, the dataset has made a clear distinction whether the transaction is a 'PURCHASE', 'REVERSAL' or 'ADDRESS_VERIFICATION'. However, according to my common sense and personal experience, a reversal must be preceded by at least one purchase. Since the question does not clearly indicate which previous purchase corresponds to reversal, I personally decided to unify all the **'same transactions'** as reversal, and labeled them as True for the created 'isReversal' column. That is, although it is marked as True for the 'isReversal' column, that

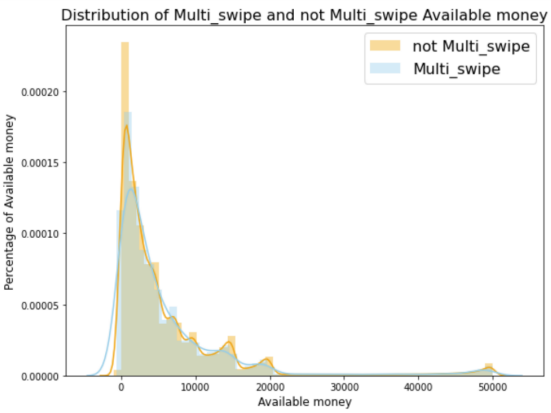
particular transaction may also be a ‘PURCHASE’ transaction type. Here, the meaning of ‘same transactions’ is that if the customer, the merchant, the card used in the transaction (the last four digits) and the amount of transactions are all the same in between several transactions, I will recognize and group them as the same transactions and label a unique index for purpose of identifying duplicate transactions. The code along with more detailed explanations that I mentioned above and below in the writeup are written in the code file (Markdown).

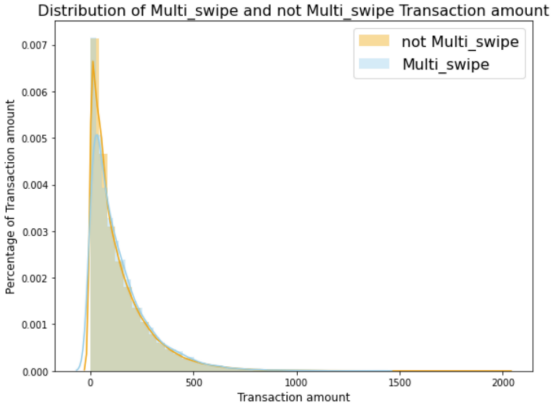
Next, I'm going to deal with some multi-swipe situations. This is also a common duplicated swipe card action. From my personal experience, this usually happens at very short intervals. Since the task didn't define the interval time, I personally defined it as less than 5 minutes. Using the ‘same transaction’ principle I described above, I used a similar approach to find and group several potential ‘same transactions’ and compare the time spans between them. My algorithm compares two neighboring transactions, not with the first one. I think comparing with the most recent transaction can reflect whether it is a multi-swipe behavior. If all the requirements meet, I will mark that transaction as ‘True’ and store it in the created ‘isMulti_swipe’ column, labeled as False otherwise.

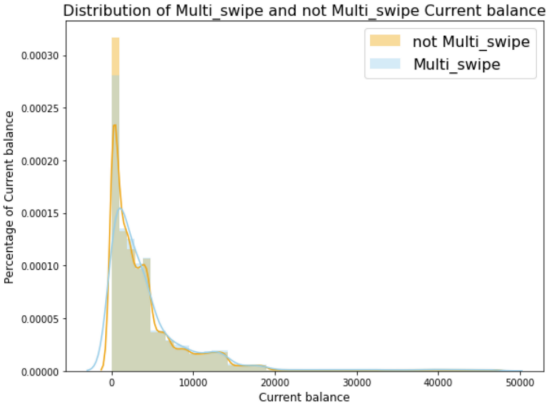
It's not enough to only identify duplicate transactions, because our goal is to find out the cause of fraud. Therefore, I not only visualized the number and percentage of reversal and multi-swipe transaction among the total transaction, but also visualized and discovered the relationship between reversal, multi-swipe and both situations that occur and with the fraudulent transactions, including some histograms and pie charts. I tried to understand the number and percentage of duplicate transactions in all fraudulent transactions, and the number and probability of fraud if any duplicate transactions have occurred. Finally, I discovered the total amount of reversal and multi-swipe transactions, which are 2821792.5 and 1933949.11 respectively.

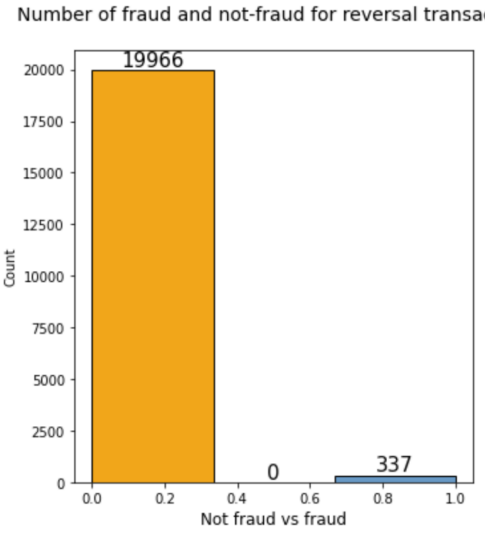



Distribution of Multi_swipe and not Multi_swipe Credit limit
not Multi_swipe
Multi_swipe
Percentage of Credit limit
Credit limit

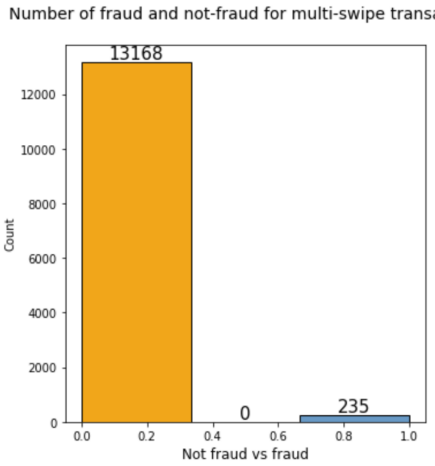

Distribution of Multi_swipe and not Multi_swipe Available money
not Multi_swipe
Multi_swipe
Percentage of Available money
Available money

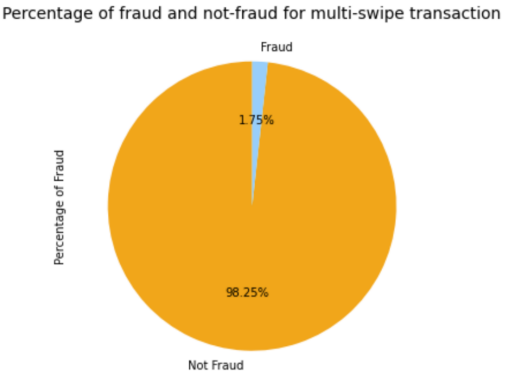

Distribution of Multi_swipe and not Multi_swipe Transaction amount
not Multi_swipe
Multi_swipe
Percentage of Transaction amount
Transaction amount


Distribution of Multi_swipe and not Multi_swipe Current balance
not Multi_swipe
Multi_swipe
Percentage of Current balance
Current balance

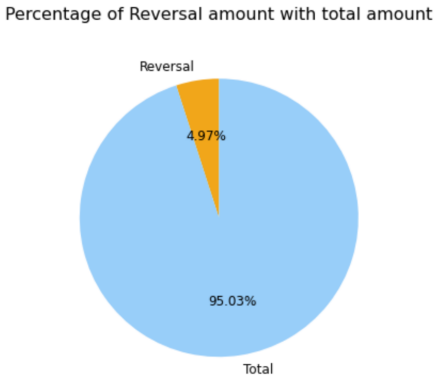

Number of fraud and not-fraud for reversal transaction
19966
0
337
Count
Not fraud vs fraud

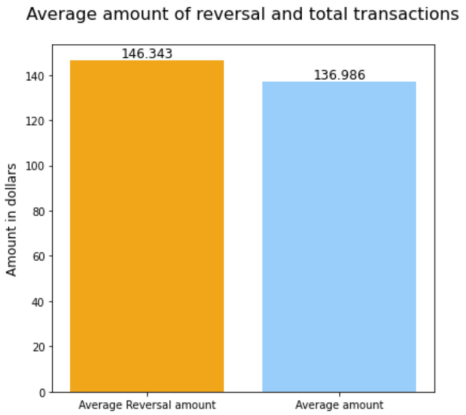

Percentage of fraud and not-fraud for reversal transaction
Fraud
1.66%
98.34%
Not Fraud
Percentage of Fraud

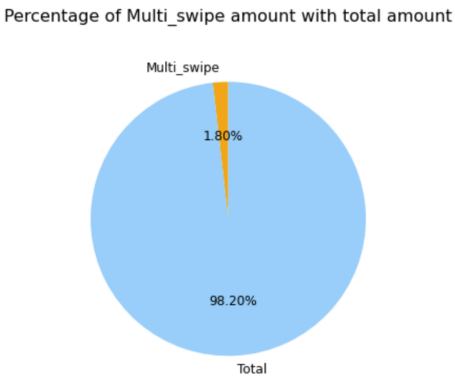


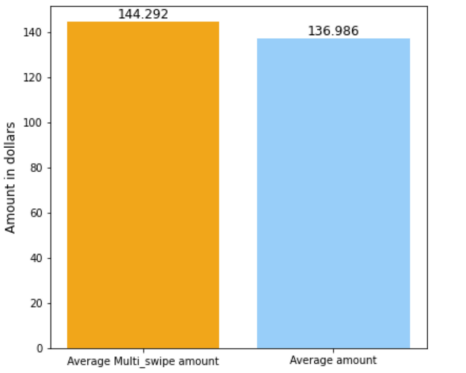


Although the result was in one-sided, that is, the phenomenon of duplicate transaction did not make me conclude any direct relation between duplicate transaction, either reversal or multi-swipe and fraud, this process is still very necessary, because it is closer to the processing of real data and combines the real-life thinking in dealing with real-life world. In addition, it is really interesting to find out that the distribution of fraud, reversal and multi-swipe transactions have pretty similar results in terms of the 4 numerical features. The average transaction amount of reversal and multi-swipe are similar: 146.343 and 144.292, and the average total transaction amount is 136.986. It seems like duplicate transactions happen less often, but with higher amounts each time than normal transactions.









Besides finding duplicate transactions, it is also important to do some feature engineering and augmentation for later modeling use. For this process, I decide to look at each column step by step to discover any potential method to apply to the column, either to transfer, augment or delete the column. There are very detailed instructions explained in the code. Here, I would like to give brief explanations of each step I made.

I combine the result of acqCountry and merchantCountryCode columns because they all deal with country information. I combine posEntryMode and posConditionCode as they also share some correlation. I create a new column to store information whether the card CVV and entered CVV are the same. I transfer some of the datetime information to a better understanding information, such as the day differences between transaction date and account open date. I also store information of whether the available money and credit limit are the same during each transaction. And finally, I delete some unnecessary columns and encode object type to numerical features and bool features to numeric type as well.

The last part is to select appropriate classification models and apply parameter tuning to find out some of the best models and parameters. Before fitting into the models, I apply SMOTE and standardScaler to avoid imbalanced data situations and higher range values that dominate the feature selection, which all help to get a better performance of the model. I am really out of time, so I did not finish a complete pipeline for parameter tuning. I will definitely finish with parameter tuning if I get extra time. However, I still write down the code for some necessary steps to parameter tuning and lastly, I test the model using Logistic regression and Random forest. The parameters I chose are:

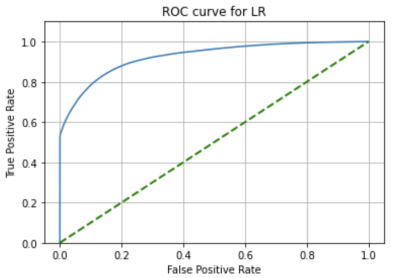
LogisticRegression(random_state=0, solver='liblinear', C=1.0, penalty='l2')

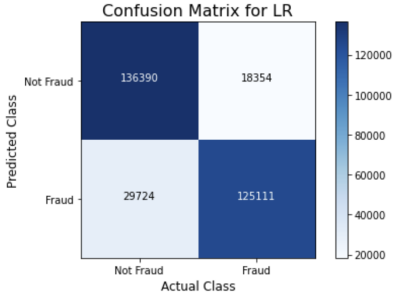
RandomForestClassifier(random_state=0, max_depth=4, n_estimators=50)

and the model performances are shown below.

	precision	recall	f1-score	support
0	0.82	0.88	0.85	154744
1	0.87	0.81	0.84	154835
accuracy			0.84	309579
macro avg	0.85	0.84	0.84	309579
weighted avg	0.85	0.84	0.84	309579

```
[[136390  18354]
 [ 29724 125111]]
```





	precision	recall	f1-score	support
0	0.80	0.83	0.81	154744
1	0.82	0.79	0.81	154835
accuracy			0.81	309579
macro avg	0.81	0.81	0.81	309579
weighted avg	0.81	0.81	0.81	309579

```
[[128037  26707]
 [ 32235 122600]]
```