Realtime Fraud Detection on Credit Card Dataset

Marjan Rezvani, Ayub ali Sarker, Maryam Akrami

Abstract

- The financial industries and services that required financial transactions are suffering from fraud related loss. The challenge of financial industry is to development of real-time claim assessment and improve the accuracy of the fraud detection.

- In this project work, we will development a real time machine learning model that will identify a transaction is fraud or not and improve accuracy of fraud detection.

Introduction

In this work we will preprocess the dataset. We will do features selections and features extraction by PCA. We will train machine learning models (SVM, Random Forests, K-nearest, Voting-classifier) with best parameters and will selection best model best performance. Best Model will be deployed as REST Service.

Dataset: transactions.txt
Total number of rows: 641,914
Total number of columns: 29

Data Visualization

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection  import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import cross_val_score
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics



%matplotlib inline
SEED = 42
```

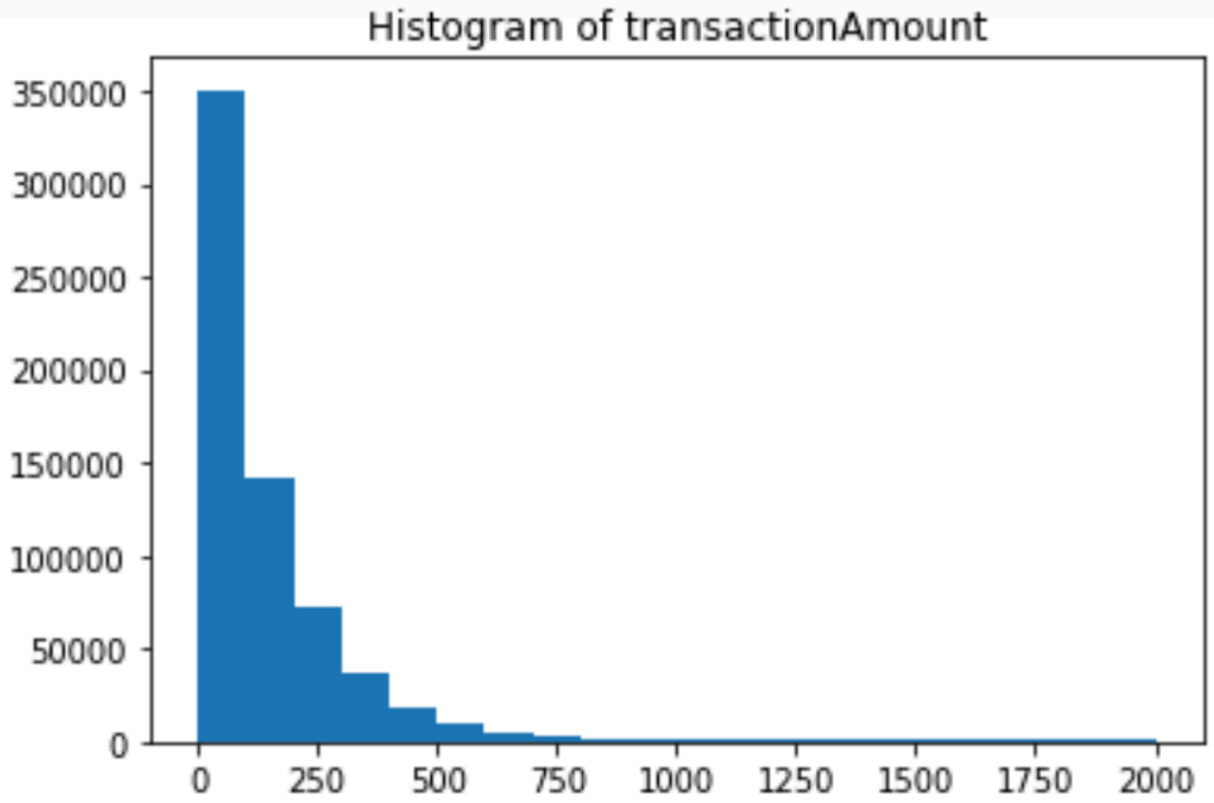
In [2]:
```
df = pd.read_csv('processed_transaction.csv')
df.head()
```

Out[2]:

	accountNumber	acqCountry	availableMoney	cardCVV	cardLast4Digits	cardPresent	creditLimit	currentBalance	custo
0	733493772	3	5000.00	492	9184	0	5000.0	0.00	7334
1	733493772	3	4888.67	492	9184	0	5000.0	111.33	7334
2	733493772	3	4863.92	492	9184	0	5000.0	136.08	7334
3	733493772	3	4676.52	492	9184	0	5000.0	323.48	7334
4	733493772	3	4449.18	492	9184	0	5000.0	550.82	7334

5 rows × 34 columns

Data Visualization




```
bins are:
[   0  100  200  300  400  500  600  700  800  900 1000 2000]
counts for each bin are:
[350445. 142390.  72474.  37202.  19001.   9918.   5127.   2539.   1398.
    682.    738.]


mean value of transactionAmount is 135.16249698556928
median value of transactionAmount is  85.8
```

In [3]:
```
df.columns
```

Out[3]:
```
Index(['accountNumber', 'acqCountry', 'availableMoney', 'cardCVV',
       'cardLast4Digits', 'cardPresent', 'creditLimit', 'currentBalance',
       'customerId', 'enteredCVV', 'expirationDateKeyInMatch', 'isFraud',
       'merchantCategoryCode', 'merchantCountryCode', 'merchantName',
       'posConditionCode', 'posEntryMode', 'transactionAmount',
       'transactionType', 'accountOpenDate_year', 'accountOpenDate_month',
       'accountOpenDate_day', 'transactionDateTime_year',
       'transactionDateTime_month', 'transactionDateTime_day',
       'transactionDateTime_hour', 'transactionDateTime_minute',
       'transactionDateTime_second', 'currentExpDate_year',
       'currentExpDate_month', 'currentExpDate_day',
       'dateOfLastAddressChange_year', 'dateOfLastAddressChange_month',
       'dateOfLastAddressChange_day'],
      dtype='object')
```

Data Visualization

Fraud or Not Fraud

In [5]:
```
sns.countplot(x='isFraud', data=df, palette='hls')
plt.show()
df['isFraud'].value_counts()
```

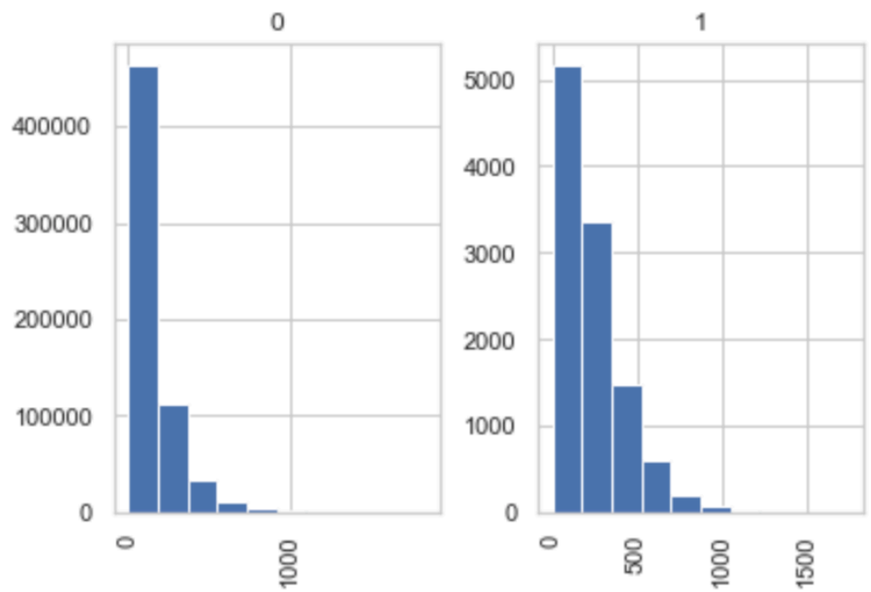
Out[5]:
```
0    622954
1     10892
Name: isFraud, dtype: int64
```

Data Visualization

Fraud by transaction amount

In [6]:
```
df['transactionAmount'].hist(by=df['isFraud'])
```

Out[6]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x1a2502dac8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a25c5ea90>],
      dtype=object)
```

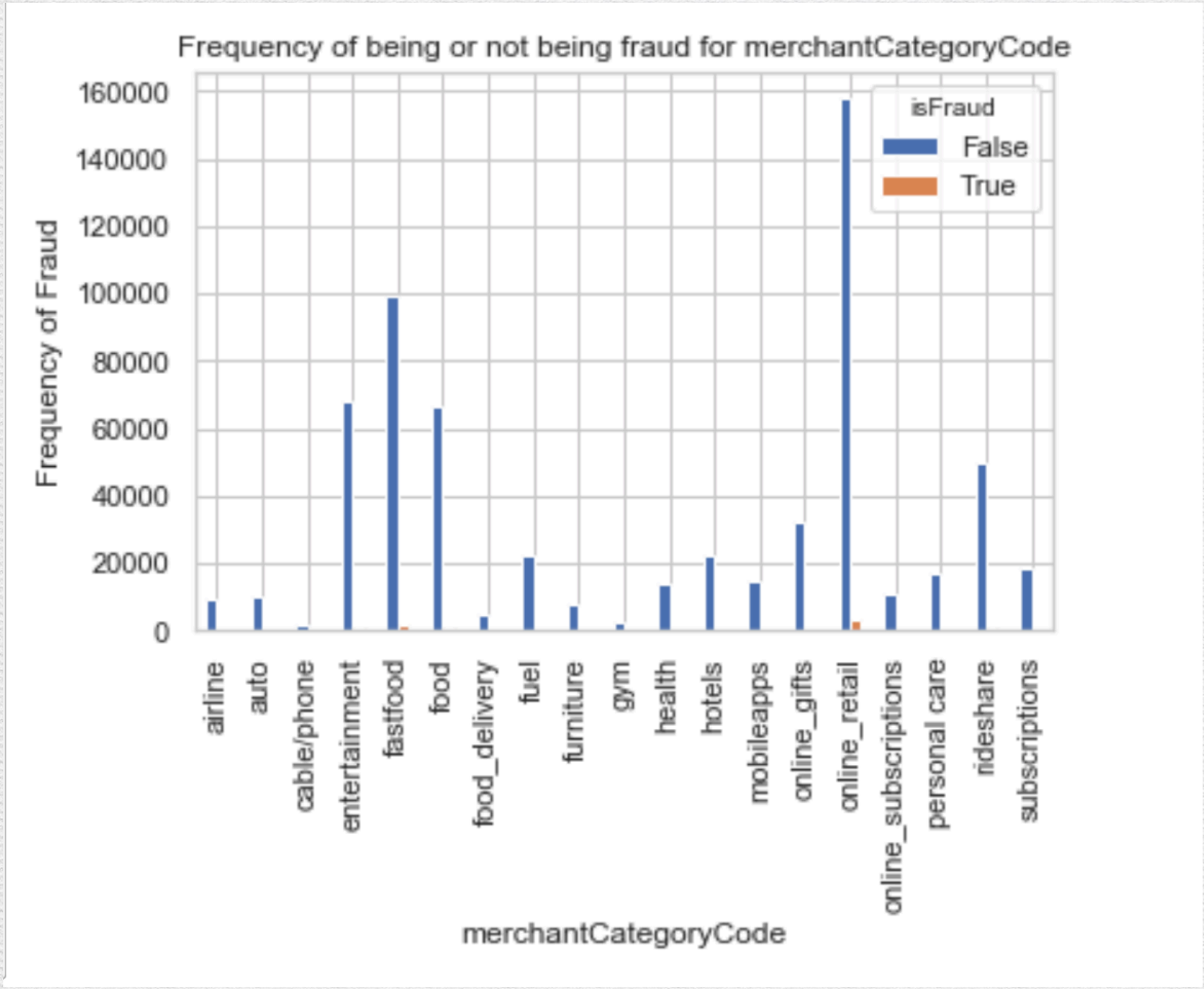


As we can see in the plot, most of the fraud transactions happen in low amount. it means there is not very high amount of fraud transactions. it's usaually because The bank would block these kind of high transactions easily for safety. so, it makes sense that the fraud transactions happen consistently in low prices.

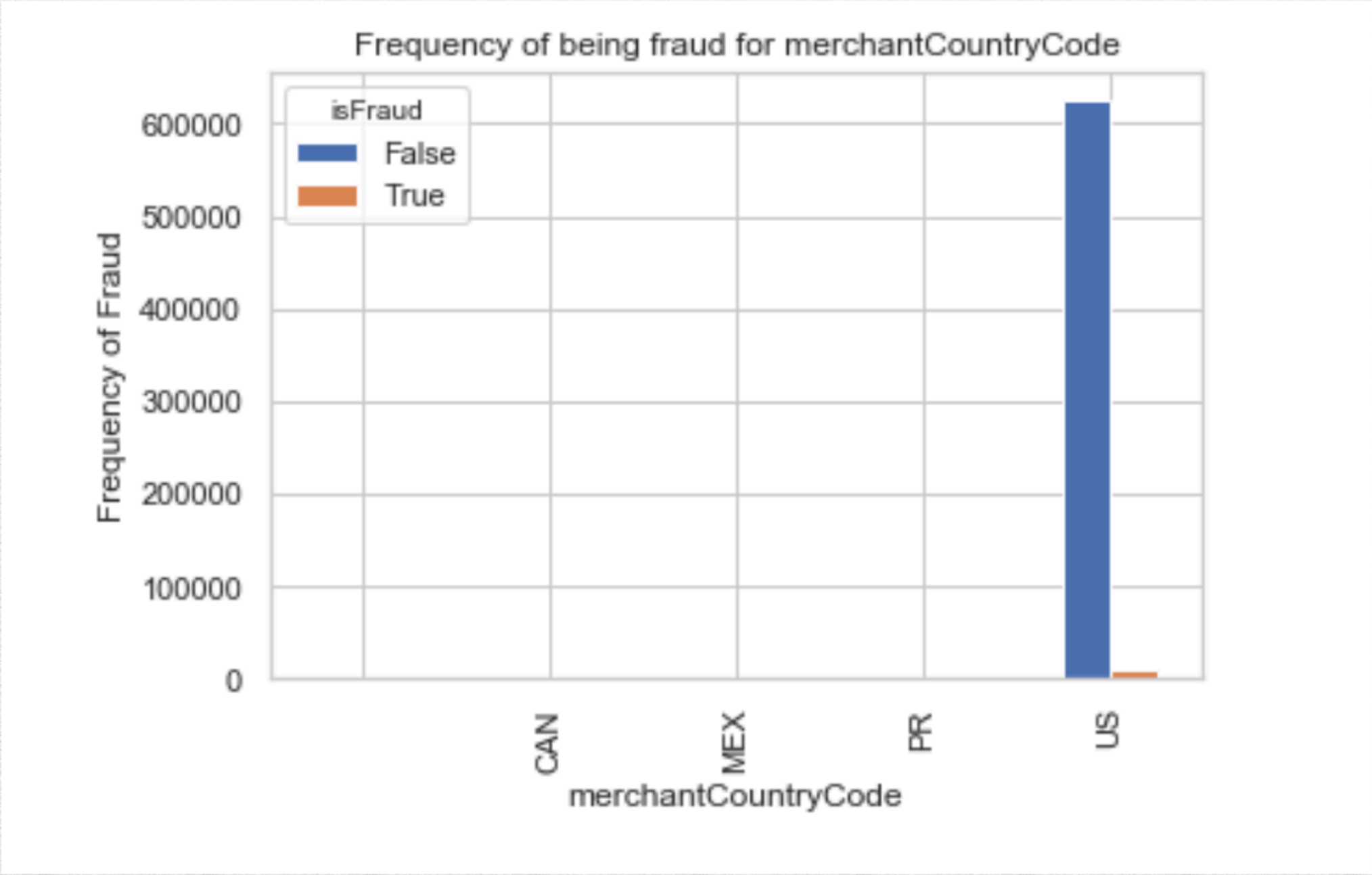
In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=SEED)
```

NOTE : As we can see in the plot, most of the fraud transactions happen in low amount. it means there is not very high amount of fraud transactions. it's usually because The bank would block these kind of high transactions easily for safety. so, it makes sense that the fraud transactions happen consistently in low prices.

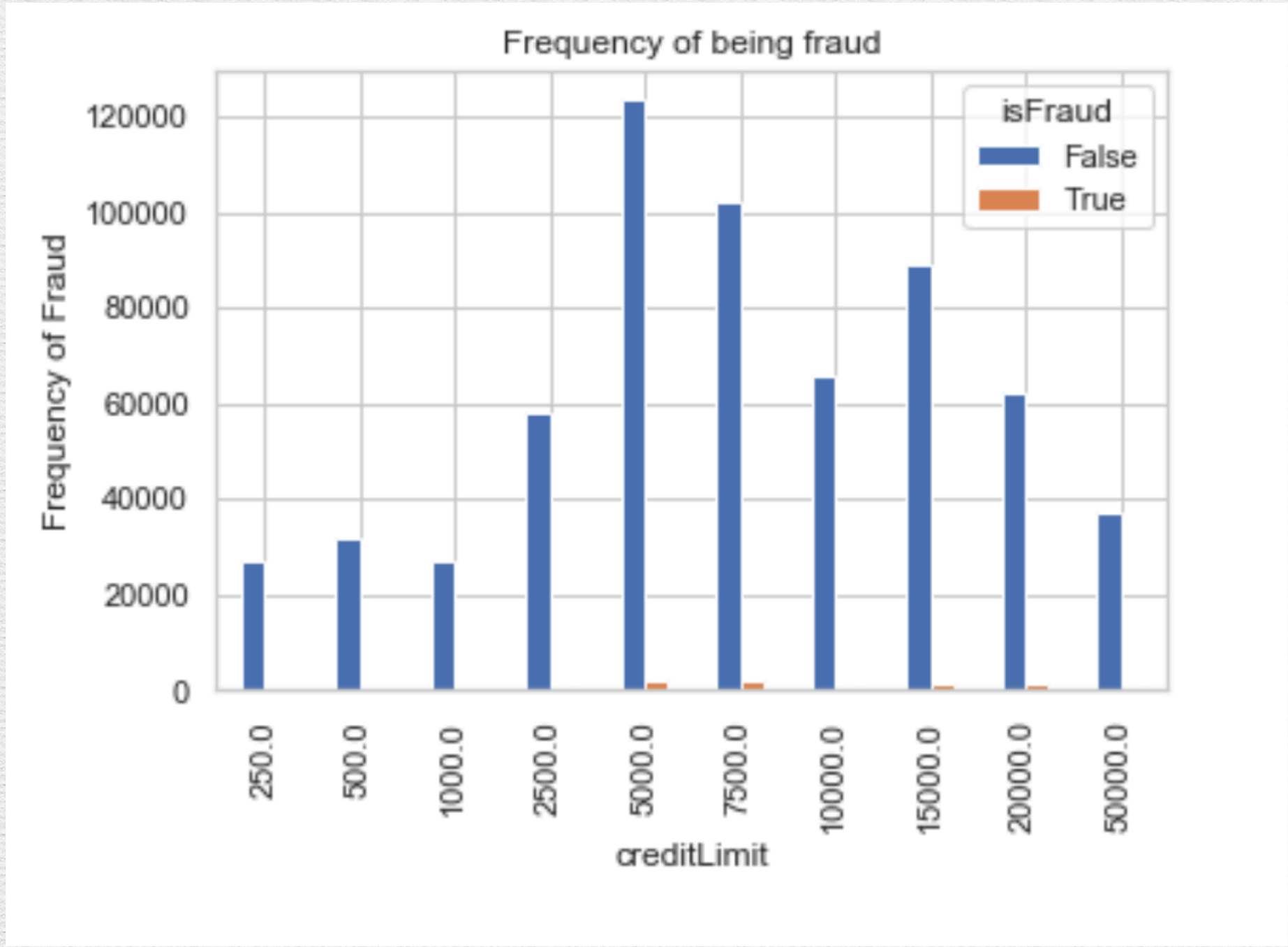
Data Visualization



Data Visualization



Data Visualization



Dataset preparation and preprocessing

Lets read dataset and cleanup missing value

```
df = read_data('../data/transactions.txt')
# Replacing blank values with nan.
df.replace(r'^\s*$', np.nan, regex=True, inplace=True)
# lets inspect missing values now
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 641914 entries, 0 to 641913
Data columns (total 29 columns):
accountNumber                641914 non-null object
accountOpenDate              641914 non-null object
acqCountry                   638001 non-null object
availableMoney               641914 non-null float64
cardCVV                      641914 non-null object
cardLast4Digits              641914 non-null object
cardPresent                  641914 non-null bool
creditLimit                  641914 non-null float64
currentBalance               641914 non-null float64
currentExpDate               641914 non-null object
customerId                   641914 non-null object
dateOfLastAddressChange      641914 non-null object
echoBuffer                   0 non-null float64
enteredCVV                   641914 non-null object
expirationDateKeyInMatch     641914 non-null bool
isFraud                      641914 non-null bool
merchantCategoryCode         641914 non-null object
merchantCity                 0 non-null float64
merchantCountryCode          641290 non-null object
merchantName                 641914 non-null object
merchantState                0 non-null float64
merchantZip                  0 non-null float64
posConditionCode             641627 non-null object
posEntryMode                 638569 non-null object
posOnPremises                0 non-null float64
recurringAuthInd             0 non-null float64
transactionAmount            641914 non-null float64
transactionDateTime          641914 non-null object
transactionType              641325 non-null object
dtypes: bool(3), float64(10), object(16)
memory usage: 129.2+ MB
```

Observations:¶

There are 641914 instances in the dataset. We have some numerical attributes like 'availableMoney', 'creditLimit', and some categorical attributes like 'merchantCategoryCode', 'merchantName'. We have few attributes which totally have missing values. These attributes are:

- echoBuffer
- merchantCity
- merchantState
- merchantZip
- posOnPremises
- recurringAuthInd

We can drop these columns.

Dataset preparation and preprocessing

Handeling missing values

We have few options:

- **totally drop those attributes from data.**
- **Drop those records (remove rows where these attributes are missing)**
- **Set the missing to some values. For numerical attributes, we can set them to the mean/ median, and for categorical attributes we can set them to the most frequent category.**

but this case we are dropping all the row that has missing value

[5]:
```
df_clean = df_clean.dropna()
```

Now let convert **isFraud** to numberic value.

[6]:
```
df_clean['isFraud'] = df_clean['isFraud'].apply(lambda x: 1 if x == True else 0)
```

Dataset preparation and preprocessing

Convert our date columns to datetime

[7]:
```
## temporally remove date column
date_columns=['accountOpenDate', 'transactionDateTime', 'currentExpDate' , 'dateOfLastAddressChange']
df_date_columns = df_clean[date_columns]

## convert this columns to datetime
df_date_columns['accountOpenDate'] = pd.to_datetime(df_date_columns['accountOpenDate'], format = "%Y-%m-%d")
df_date_columns['transactionDateTime'] = pd.to_datetime(df_date_columns['transactionDateTime'], format = "%Y-%m-%dT%H:%M:%S")
df_date_columns['currentExpDate'] = pd.to_datetime(df_date_columns['currentExpDate'], format = "%m/%Y")
df_date_columns['dateOfLastAddressChange'] = pd.to_datetime(df_date_columns['dateOfLastAddressChange'], format = "%Y-%m-%d")
df_date_columns.head()
```

[7]:

	accountOpenDate	transactionDateTime	currentExpDate	dateOfLastAddressChange
0	2014-08-03	2016-01-08 19:04:50	2020-04-01	2014-08-03
1	2014-08-03	2016-01-09 22:32:39	2023-06-01	2014-08-03
2	2014-08-03	2016-01-11 13:36:55	2027-12-01	2014-08-03
3	2014-08-03	2016-01-11 22:47:46	2029-09-01	2014-08-03
4	2014-08-03	2016-01-16 01:41:11	2024-10-01	2014-08-03

Dataset preparation and preprocessing

Now we are converting categorial column to numeric using LabelEncoder

```
# instantiate labelencoder object
le = LabelEncoder()
```

```
category_columns =['acqCountry','cardPresent','expirationDateKeyInMatch','isFraud','merchantCategoryCode',
          'merchantCountryCode','merchantName','posConditionCode','posEntryMode','transactionType'
]
```

```
df_clean[category_columns] = df_clean[category_columns].apply(lambda col: le.fit_transform(col))
df_clean.head()
```

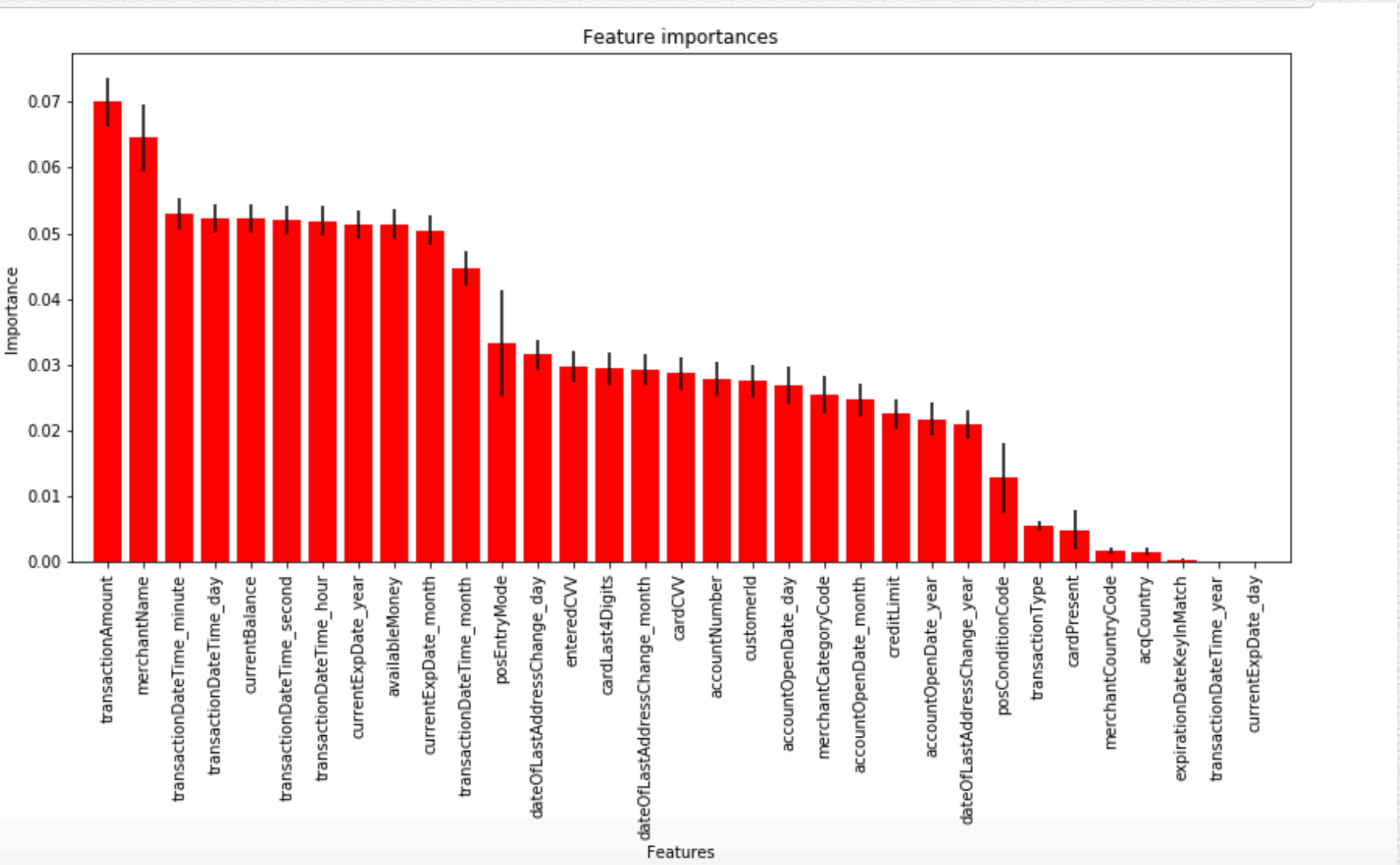
	accountNumber	acqCountry	availableMoney	cardCVV	cardLast4Digits	cardPresent	creditLimit	currentBalance	cust
0	733493772	3	5000.00	492	9184	0	5000.0	0.00	7334
1	733493772	3	4888.67	492	9184	0	5000.0	111.33	7334
2	733493772	3	4863.92	492	9184	0	5000.0	136.08	7334
3	733493772	3	4676.52	492	9184	0	5000.0	323.48	7334
4	733493772	3	4449.18	492	9184	0	5000.0	550.82	7334

5 rows × 34 columns

Save clean data

```
]: df_clean.to_csv('../data/processed_transaction.csv', index=False)
```

Feature-selection



Important features¶

So The followings are the high importance feature

- transactionAmount
- merchantName
- transactionDateTime_minute
- transactionDateTime_day
- currentBalance
- transactionDateTime_second
- transactionDateTime_hour
- currentExpDate_year
- availableMoney
- currentExpDate_month
- transactionDateTime_month

In [60]:
```
# let write this importance to csv. so that we can use later.

import csv
columns_importance = [(X.columns[f], importances[f]) for f in indices]
with open('../data/feature_importance.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    for row in columns_importance:
        writer.writerow(row)
```

Refernce code from

DecisionTreeClassifier

Create a DessionTree classifier and fit the model and report accuracy

```
dt_classifier = DecisionTreeClassifier(max_depth=6, random_state=SEED)
dt_classifier.fit(X_train, y_train)
y_pred = dt_classifier.predict(X_test)

accuracy = round(accuracy_score(y_test, y_pred),5)
precision = round(precision_score(y_test, y_pred),5)
recall = round(recall_score(y_test, y_pred),5)
print('acc: {} precision: {} recal: {}'.format(accuracy, precision, recall))
```

```
acc: 0.98262 precision: 0.86957 recal: 0.00602
```

Cross validation

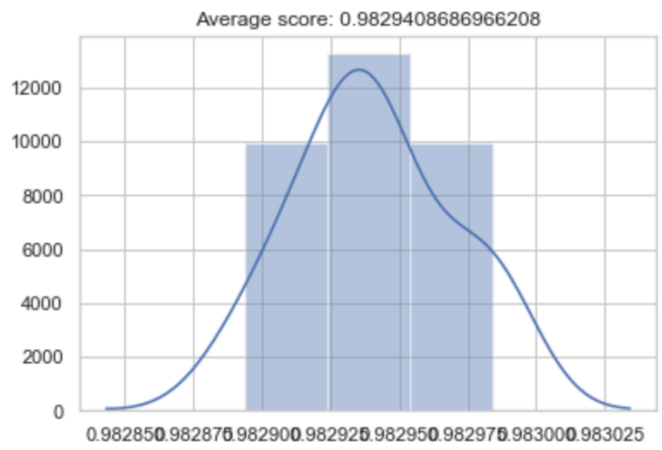
To eliminate over-fitting, we can apply cross-validation. we are going to apply k-fold cross-validation.

- It will split the original data set into k subsets and use one of the subsets as the testing set and the remaining as the training sets. This process iterated k times until every subset have been used as the testing set. Since 10-fold cross-validation is the most popular one, we are going to use that one.

```
dt_classifier = DecisionTreeClassifier(max_depth=5, random_state=SEED)

cv_scores = cross_val_score(dt_classifier, X_train, y_train, cv=10)
sns.distplot(cv_scores)
plt.title('Average score: {}'.format(np.mean(cv_scores)))
```

```
Text(0.5,1,'Average score: 0.9829408686966208')
```



It looks there is no improvement using cross-validation. it seems like we are not suffering from overfit

Logistic Regression

Create a logistic regression classifier and fit the model and report accuracy

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
```

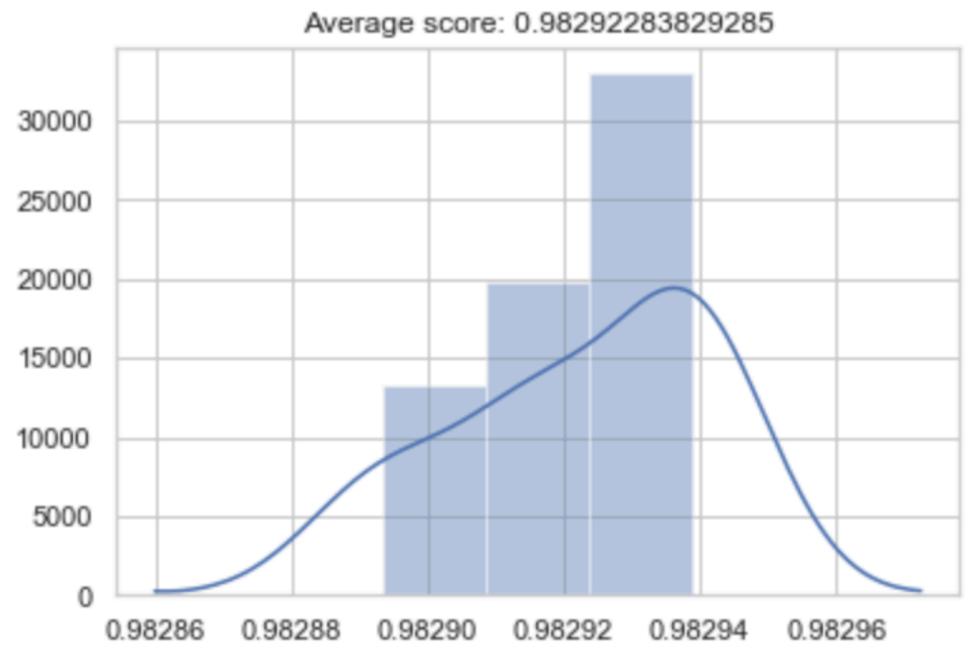
```
accuracy = round(accuracy_score(y_test, y_pred),5)
precision = round(precision_score(y_test, y_pred),5)
recall = round(recall_score(y_test, y_pred),5)
print('acc: {} precision: {} recal: {}'.format(accuracy, precision, recall))
```

```
acc: 0.98249 precision: 0.0 recal: 0.0
```

Cross validation

```
cv_scores = cross_val_score(lr, X_train, y_train, cv=10)
sns.distplot(cv_scores)
plt.title('Average score: {}'.format(np.mean(cv_scores)))
```

```
Text(0.5, 1.0, 'Average score: 0.98292283829285')
```



It looks there is no improvement using cross-validation, as both accuracy score and cross validation look simillar! it seems like we are not suffering from overfit

8 Issues closed by **3** people

Closed #5 **Handle date, datetime columns in dataset** 3 days ago

Closed #10 **Create DecisionTreeClassifier, fit with processed clean dataset and report accuracy** 3 days ago

Closed #4 **Convert categorial columns into numeric** 3 days ago

Closed #6 **Select important features from datasets** 3 days ago

Closed #9 **Finalize preprocessing** 3 days ago

Closed #2 **Clean up dataset** 5 days ago

Closed #3 **Project proposal** 5 days ago

Closed #8 **create necessary project folder structures** 7 days ago