VIETNAM NATIONAL UNIVERSITY - HCMC
UNIVERSITY OF ECONOMICS AND LAW FACULTY
OF FINANCE AND BANKING

FINAL PROJECT
CREDIT CARD FRAUD DETECTION USING RANDOM FOREST

Lecturer:

Phd. Nguyen Anh Phong

Mr. Phan Huy Tam

Author:

Le Nguyen Phu Loc K194141730

Ho Chi Minh city, June 2022

Contents

Project Overview

1. Credit card

- A credit card is a thin rectangular piece of plastic or metal issued by a bank or financial services company, that allows cardholders to borrow funds with which to pay for goods and services with merchants that accept cards for payment. Credit cards impose the condition that cardholders pay back the borrowed money, plus any applicable interest, as well as any additional agreed-upon charges, either in full by the billing date or over time.

2. Credit card fraud

- Credit card fraud is when somebody makes unauthorized purchases using a stolen or misappropriated credit card (or card number).
- In the U.S., millions of credit card numbers are stolen each year accounting for billions of dollars in illegal purchases

3. Random Forest

- Random forest is a consensus algorithm used in supervised machine learning (ML) to solve regression and classification problems. Each random forest is comprised of multiple decision trees that work together as an ensemble to produce one prediction.
- A decision tree is a logical construct that resembles a flowchart and illustrates a series of if-else statements. An important purpose of using random forest is to compensate for the limitations of decision tree algorithms by mapping multiple trees and using the forest's average output (statistical mean).
- Random forest algorithms can produce acceptable predictions even if individual trees in the forest have incomplete data. Statistically, increasing the number of trees in the ensemble will correspondingly increase the precision of the outcome.

I. Objective of project

- I made this project mostly because of my final project. Though, I hope this project can be a guide line for others to hand on the credit card fraud by using Machine Learning algorithms.

II. Code Explanation

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
import warnings
warnings.filterwarnings("ignore")
df= pd.read_csv('card_transdata.csv')
```

```
df
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order	fraud
0	57.877857	0.311140	1.945940	1.0	1.0	0.0	0.0	0.0
1	10.829943	0.175592	1.294219	1.0	0.0	0.0	0.0	0.0
2	5.091079	0.805153	0.427715	1.0	0.0	0.0	1.0	0.0
3	2.247564	5.600044	0.362663	1.0	1.0	0.0	1.0	0.0
4	44.190936	0.566486	2.222767	1.0	1.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...
999995	2.207101	0.112651	1.626798	1.0	1.0	0.0	0.0	0.0
999996	19.872726	2.683904	2.778303	1.0	1.0	0.0	0.0	0.0
999997	2.914857	1.472687	0.218075	1.0	1.0	0.0	1.0	0.0
999998	4.258729	0.242023	0.475822	1.0	0.0	0.0	1.0	0.0
999999	58.108125	0.318110	0.386920	1.0	1.0	0.0	1.0	0.0

1000000 rows × 8 columns

- First I import the libraries and import the dataset. Then I print out to have a look at the df:

- The dataset contains 8 columns with 1 million rows in each column.

Features Explanation:

- distancefromhome, numeric - the distance from home where the transaction happened.
- distancefromlast_transaction, numeric - the distance from last transaction happened.
- ratiotomedianpurchaseprice, numeric - Ratio of purchased price transaction to median purchase price.
- repeat_retailer, binary - Is the transaction happened from same retailer.
- used_chip, binary - Is the transaction through chip (credit card).
- used_pin_number, binary - Is the transaction happened by using PIN number.
- online_order, binary - Is the transaction an online order.
- fraud, binary - Is the transaction fraudulent. 0 is not fraud, 1 is fraud,

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 8 columns):
 #   Column                          Non-Null Count    Dtype  
---  ------                          --------------    -----  
 0   distance_from_home              1000000 non-null  float64
 1   distance_from_last_transaction  1000000 non-null  float64
 2   ratio_to_median_purchase_price  1000000 non-null  float64
 3   repeat_retailer                 1000000 non-null  float64
 4   used_chip                       1000000 non-null  float64
 5   used_pin_number                 1000000 non-null  float64
 6   online_order                    1000000 non-null  float64
 7   fraud                           1000000 non-null  float64
dtypes: float64(8)
memory usage: 61.0 MB
```

```
df.isna().sum()
```

```
distance_from_home                0
distance_from_last_transaction    0
ratio_to_median_purchase_price    0
repeat_retailer                   0
used_chip                         0
used_pin_number                   0
online_order                      0
fraud                             0
dtype: int64
```

```
df.describe()
```

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order	fraud
count	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000
mean	26.628792	5.036519	1.824182	0.881536	0.350399	0.100608	0.650552	0.087403
std	65.390784	25.843093	2.799589	0.323157	0.477095	0.300809	0.476796	0.282425
min	0.004874	0.000118	0.004399	0.000000	0.000000	0.000000	0.000000	0.000000
25%	3.878008	0.296671	0.475673	1.000000	0.000000	0.000000	0.000000	0.000000
50%	9.967760	0.998650	0.997717	1.000000	0.000000	0.000000	1.000000	0.000000
75%	25.743985	3.355748	2.096370	1.000000	1.000000	0.000000	1.000000	0.000000
max	10632.723672	11851.104565	267.802942	1.000000	1.000000	1.000000	1.000000	1.000000

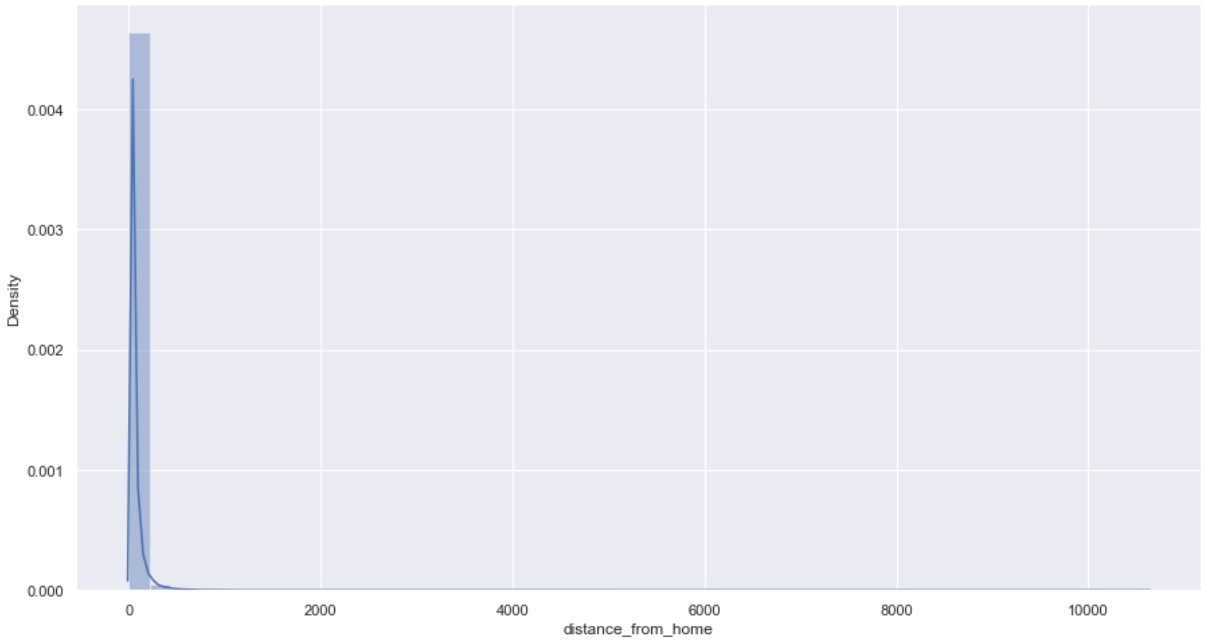
- I did a few check on the df. The df data was float type and there are no NaN. From the mean of the fraud, I can tell that the dataset is highly imbalanced. So I calculate the fraud to not fraud percentages.

```
no = len(df[df['fraud'] == 0]['fraud'])
yes = len(df[df['fraud'] == 1]['fraud'])
no_perc = round(no/len(df)*100,1)
yes_perc = round(yes/len(df)*100,1)

print('Number of transactions aren\'t fraud: {} ({}%)'.format(no, no_perc))
print('Number of transactions are fraud: {} ({}%)'.format(yes, yes_perc))
```

```
Number of transactions aren't fraud: 912597 (91.3%)
Number of transactions are fraud: 87403 (8.7%)
```

- As I expected, the dataset was majorly possessed by the transactions aren’t fraud.
- Then I plot 1 variables to check on the distribution.

```
sns.set(style ="darkgrid")
sns.distplot(df.distance_from_home)
```



- The dataset not only imbalances but also contains a lot of outliers. For the continous variables, I will remove the outliers for better views of the distributions.

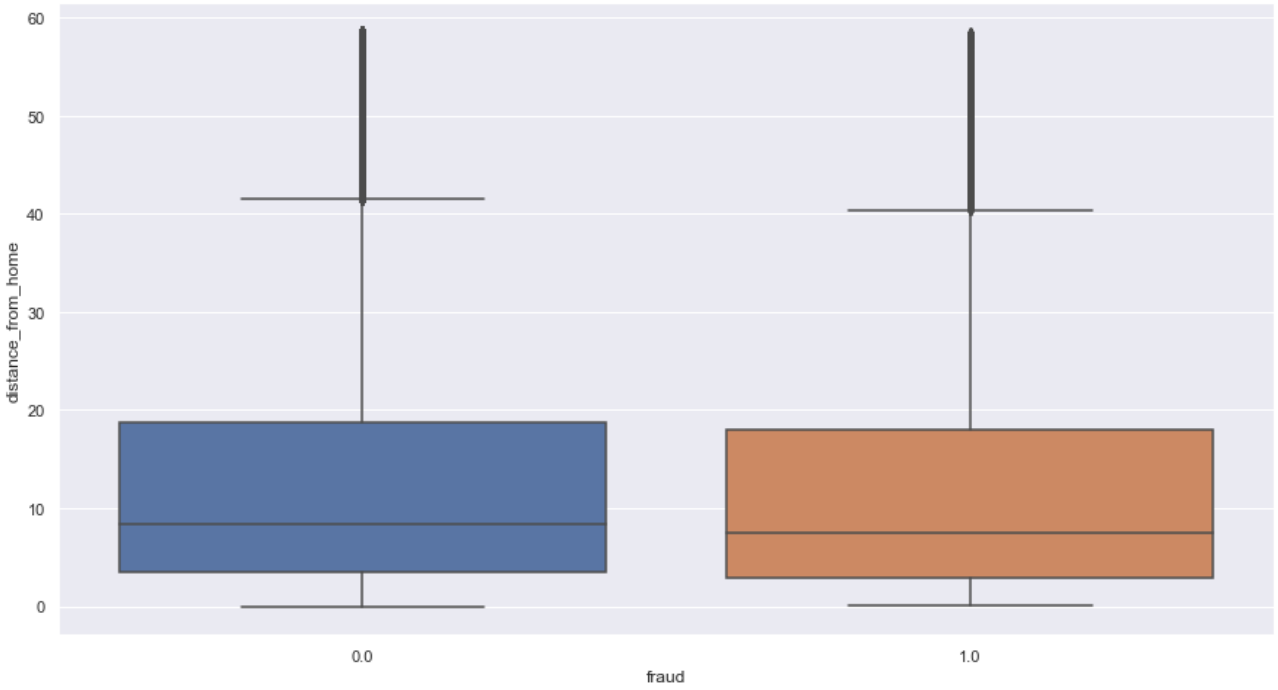
```
# EDA without outlier
def outlier(df_in, col_name):
    q1 = df_in[col_name].quantile(0.25)
    q3 = df_in[col_name].quantile(0.75)
    iqr = q3-q1 #Interquartile range
    fence_low  = q1-1.5*iqr
    fence_high = q3+1.5*iqr
    df_out = df_in.index[(df_in[col_name] < fence_low) | (df_in[col_name] > 
fence_high)]
    return df_out

index_list= []
for i in 
['distance_from_home','distance_from_last_transaction','ratio_to_median_purchase_p
rice']:
    index_list.extend(outlier(df,i))

def remove(df, ls):
    ls= sorted(set(ls))
    df= df.drop(ls)
    return df

eda = remove(df,index_list)
eda = eda.iloc[:,[0,1,2,7]]
eda
```

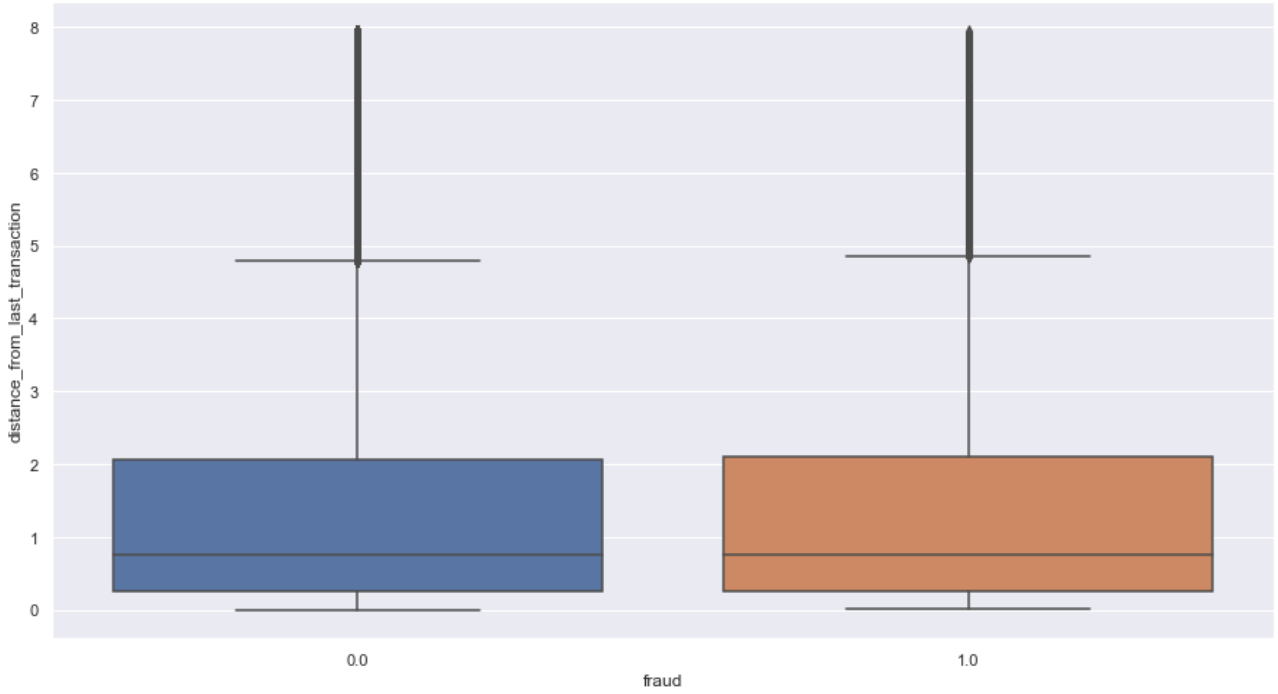
	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	fraud
0	57.877857	0.311140	1.945940	0.0
1	10.829943	0.175592	1.294219	0.0
2	5.091079	0.805153	0.427715	0.0
3	2.247564	5.600044	0.362663	0.0
4	44.190936	0.566486	2.222767	0.0
...	...	...	...	...
999995	2.207101	0.112651	1.626798	0.0
999996	19.872726	2.683904	2.778303	0.0
999997	2.914857	1.472687	0.218075	0.0
999998	4.258729	0.242023	0.475822	0.0
999999	58.108125	0.318110	0.386920	0.0

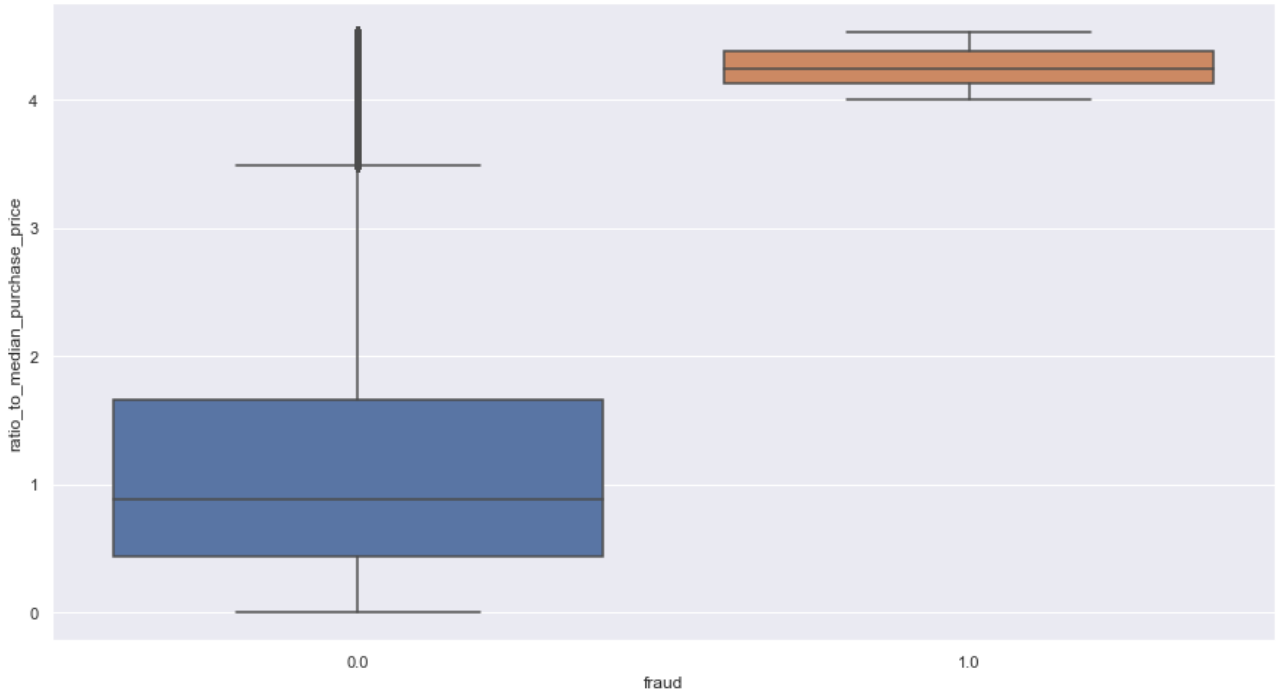
718750 rows × 4 columns

- I choose using Interquartile for detecting outliers. First I calculate the 1st and 3rd quartile, then I calculate the range of the data isn't outliers. After that I extract the outliers indexes and put them into a list. I remove the outliers and assign the data into a new df call eda to use for Explatory Data Analysis. The data reduce more than ¼ of the rows in each column.

```
sns.set(rc = {'figure.figsize':(15,8)})
for col in
['distance_from_home','distance_from_last_transaction','ratio_to_median_purchase_p
rice']:
    plt.figure(col)
    sns.set(style ="darkgrid")
    sns.boxplot(x=eda['fraud'],y=eda[col],data=eda)
```

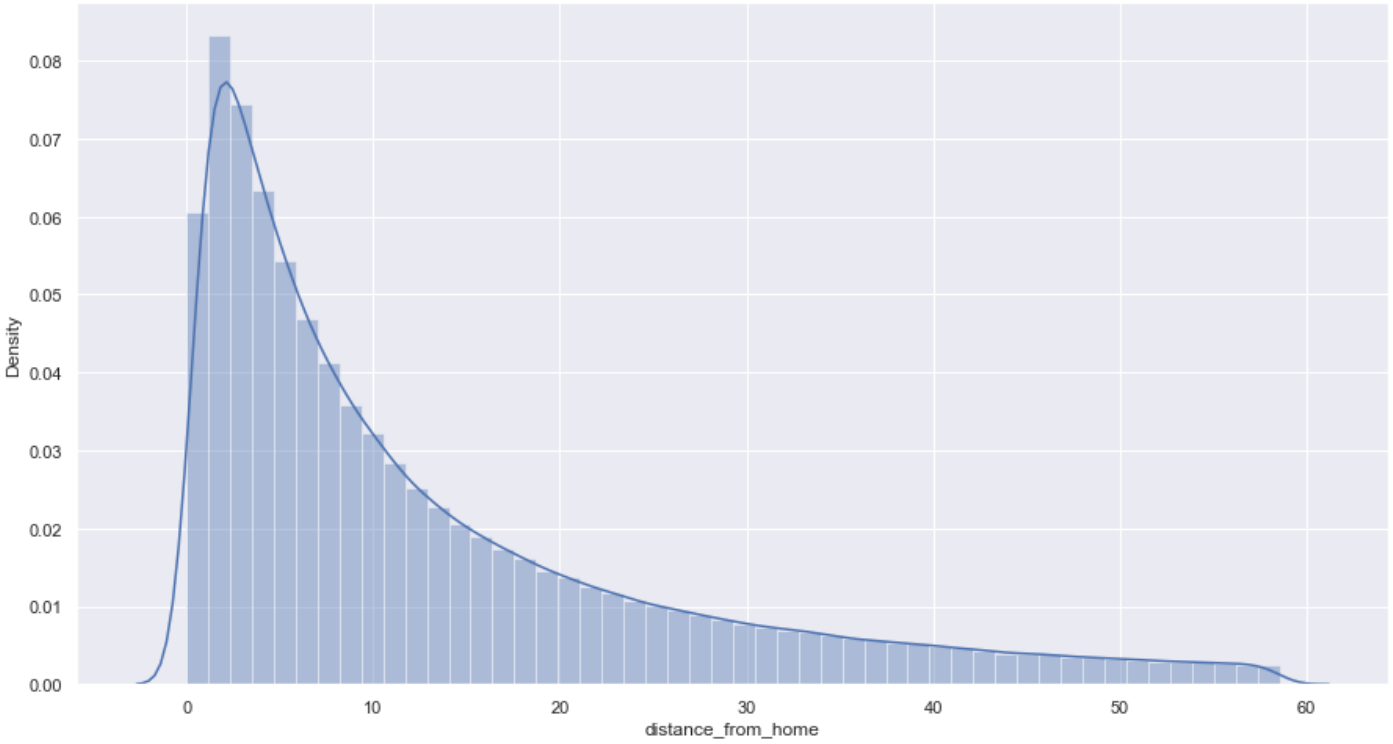


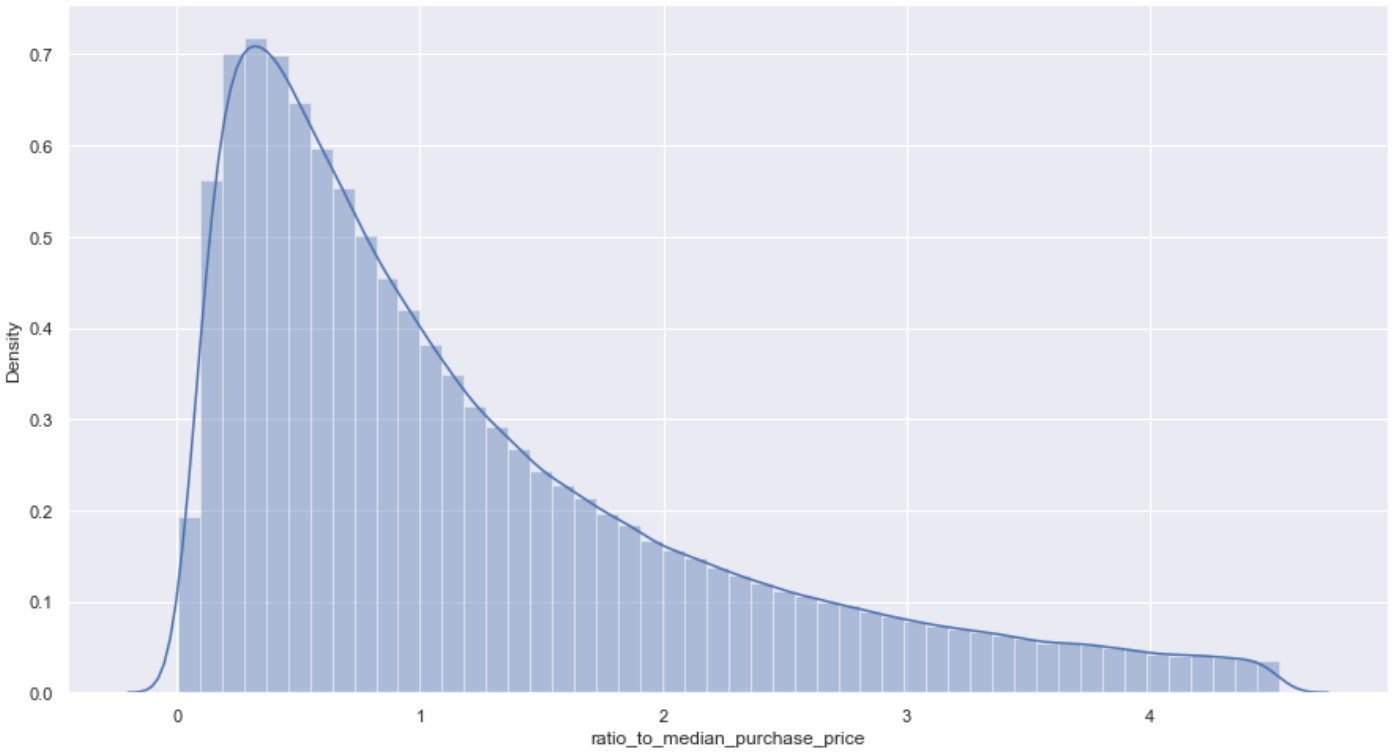


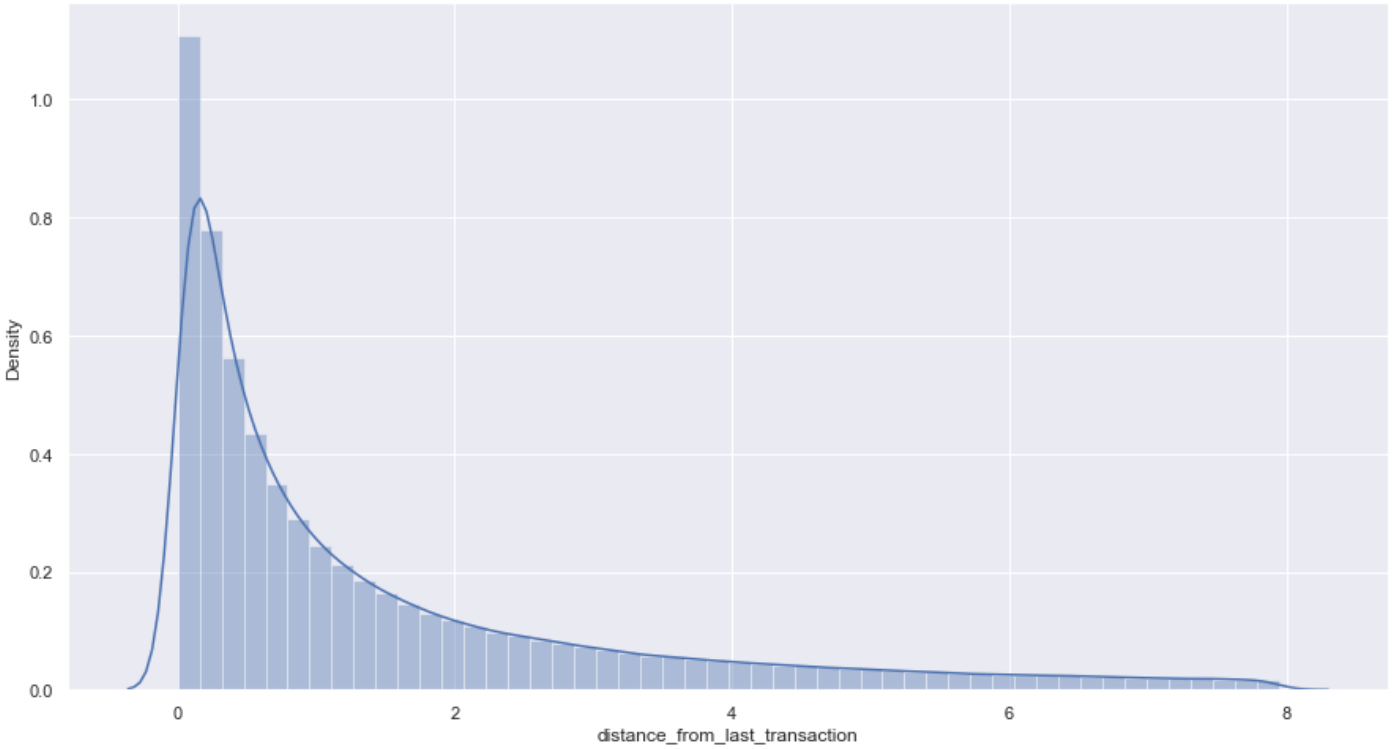


- I visualize the boxplots to see if there are differences between not fraud and fraud. The 2 distances of 2 classes are slightly different but it's hard to see. The ratio to mean purchase price will be the one that distinguish 2 classes, I think.

```
for i in
['distance_from_home','distance_from_last_transaction','ratio_to_median_purchase_p
rice']:
    plt.figure(i)
    sns.set(style ="darkgrid")
    sns.distplot(eda[i])
```

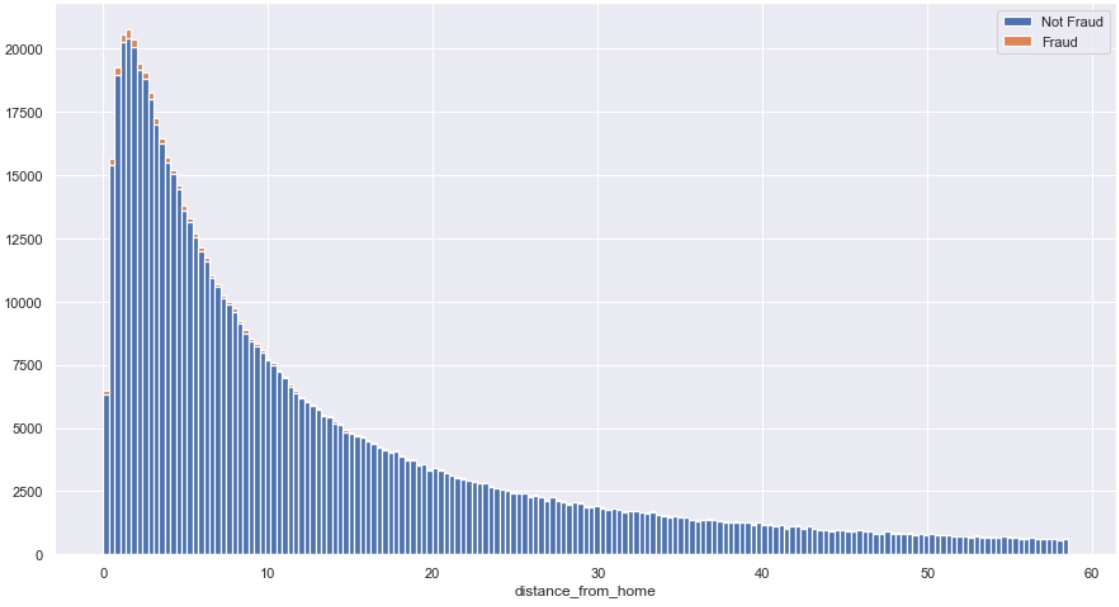

0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
0
10
20
30
40
50
60
distance_from_home

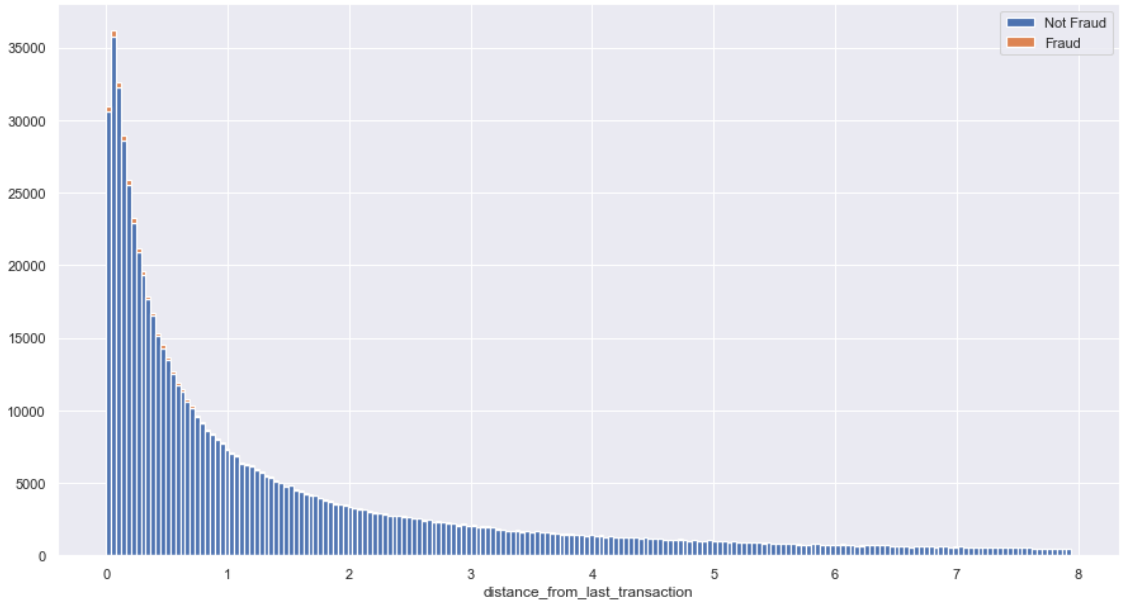

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Density
0
1
2
3
4
ratio_to_median_purchase_price

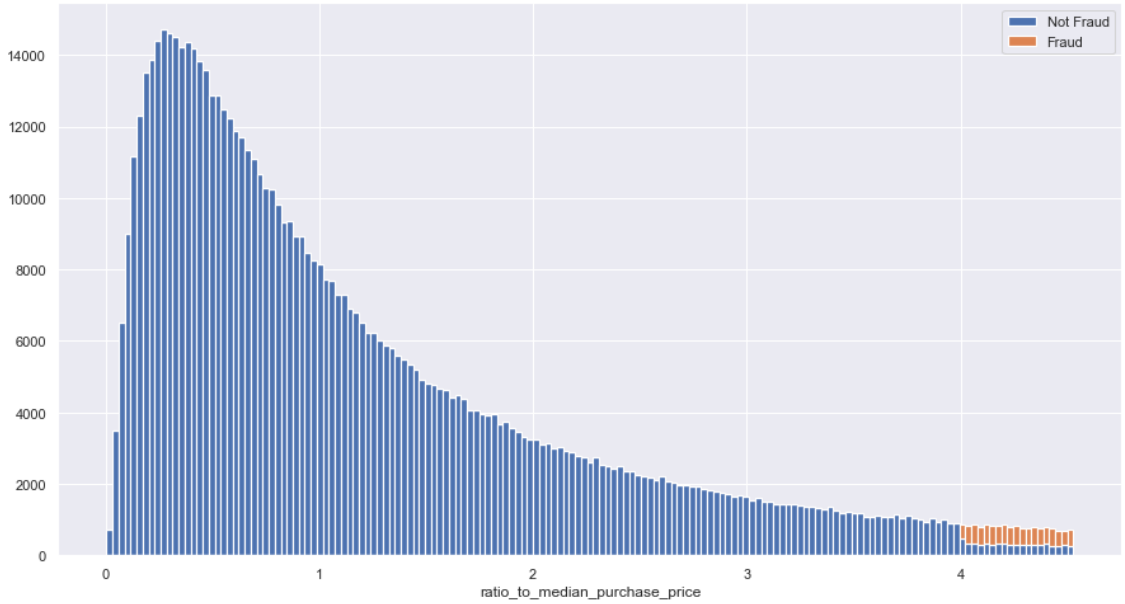


- The distribution was right-skewed. This can be explained by the fraud column, because the data was majorly not fraud so the others variables was characteristics of not fraud.

```
for col in
['distance_from_home','distance_from_last_transaction','ratio_to_median_purchase_p
rice']:
    plt.rcParams["figure.figsize"] = (15,8)
    plt.hist([eda[(eda.fraud==0)][col],eda[(eda.fraud==1)][col]],stacked=True,
bins = 'auto',label = ['Not Fraud','Fraud'],linewidth=1.2)
    plt.xlabel(col)
    plt.ylabel('')
    plt.legend()
    plt.show()
```

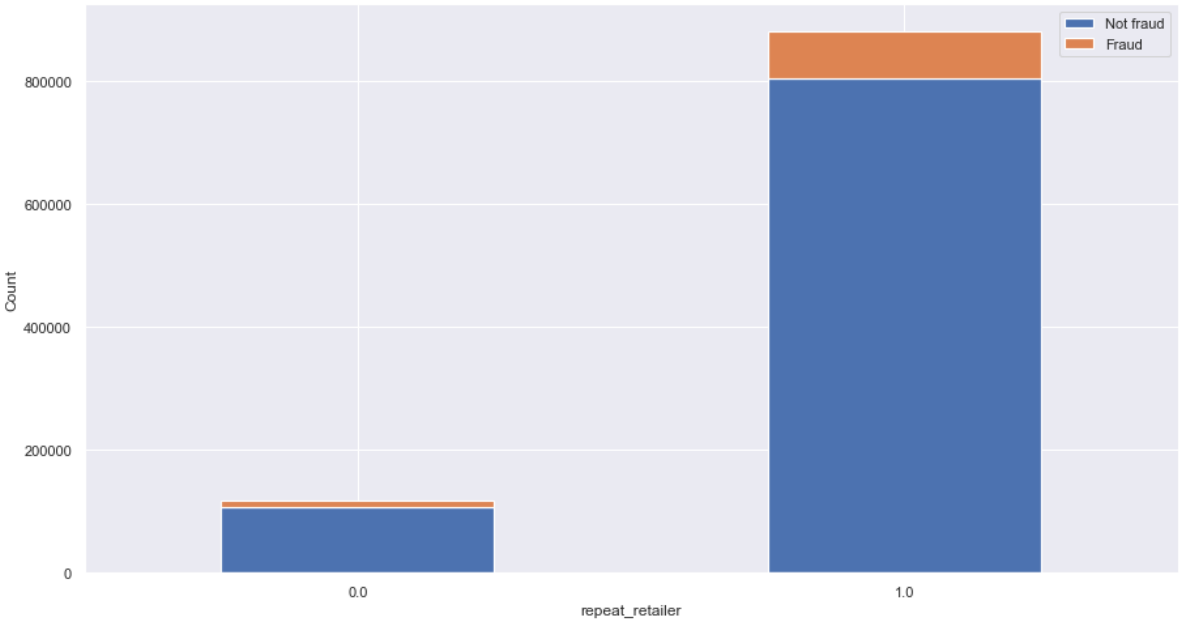


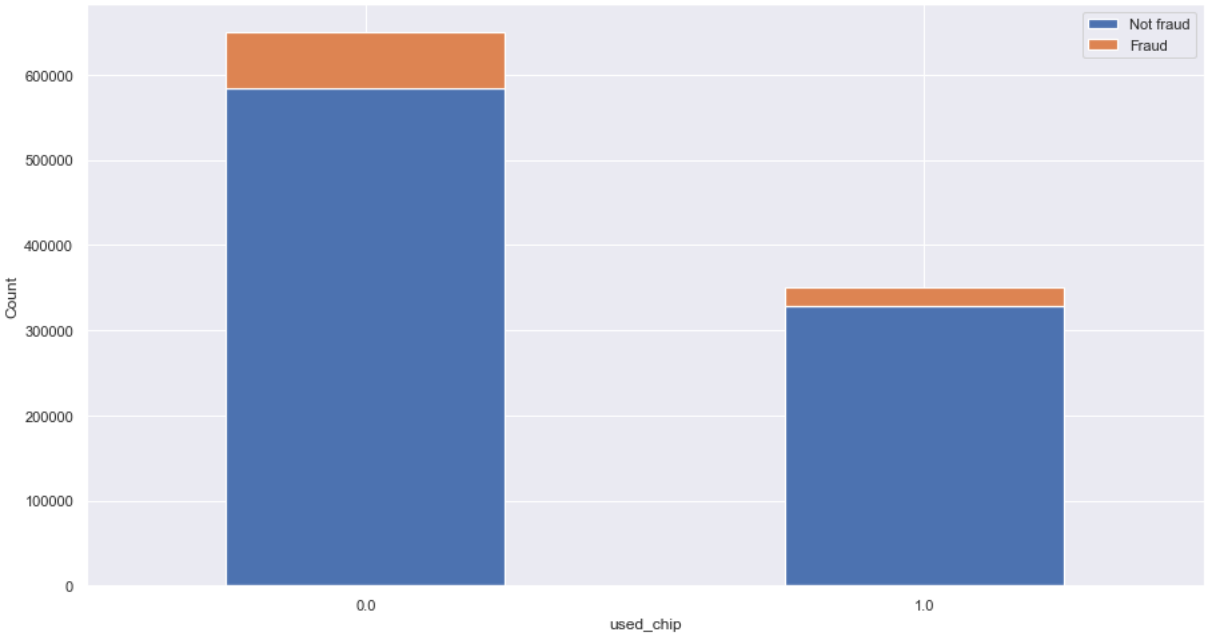


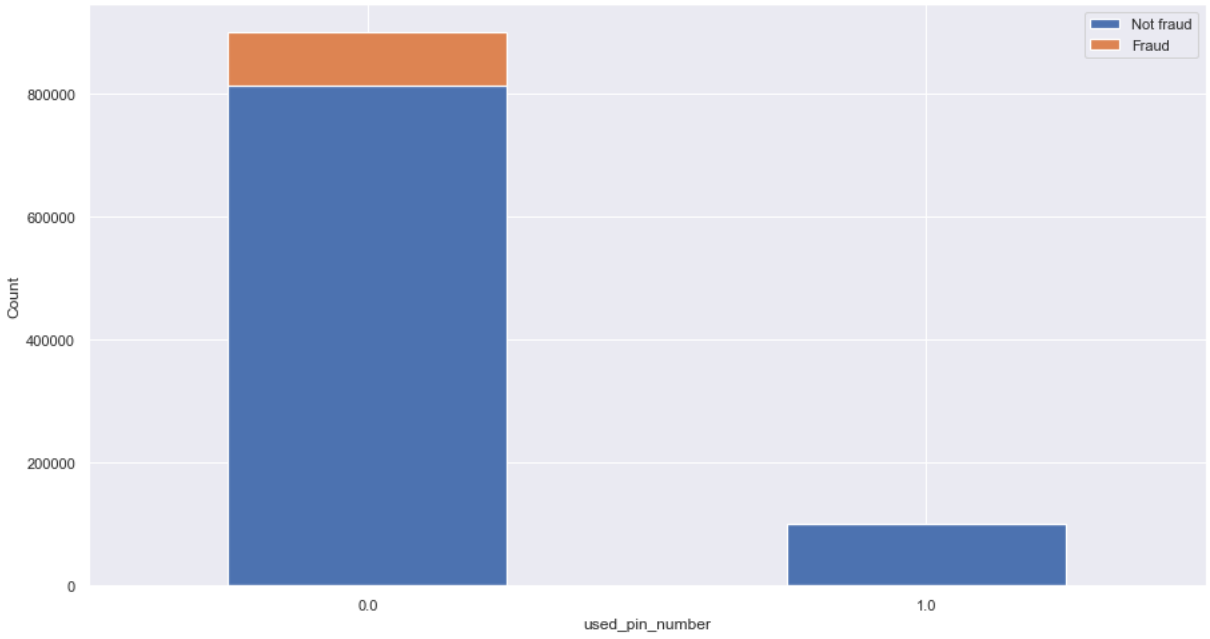


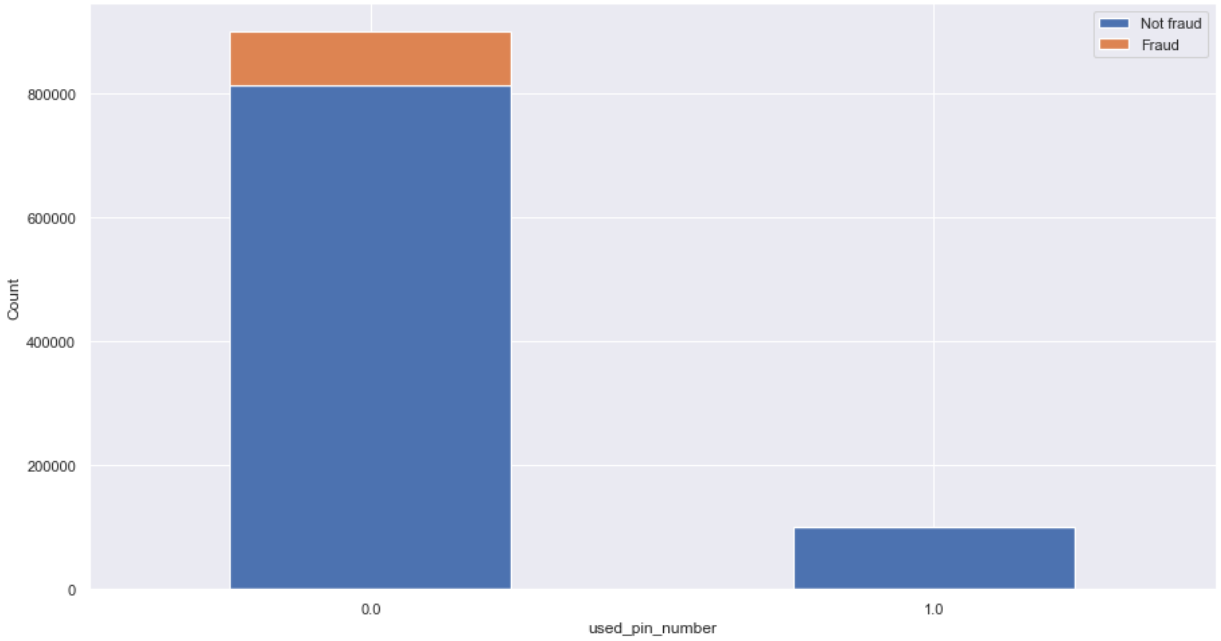
- As we can see in the distribution, the fraud in 2 distances variables was evenly spread. But the ratio to median purchase was completely different, the ratio that equal or larger than 4 was mostly fraud.

```
column= ['repeat_retailer','used_chip','used_pin_number','online_order']
for col in column:
    object = df.groupby([col,'fraud'])['fraud'].count().unstack(level=1)
    object.plot(kind = 'bar', stacked = True,rot = 0,
figsize=(15,8),linewidth=1.2)
    plt.legend(['Not fraud','Fraud'])
    plt.ylabel('Count')
    plt.xlabel(col)
    plt.show()
```


Not fraud
Fraud
800000
600000
400000
200000
0
Count
0.0
1.0
repeat_retailer


Not fraud
Fraud
600000
500000
400000
300000
200000
100000
0
Count
0.0
1.0
used_chip





- Because these 4 variables was binary so I visualize them in stacked bar for an easier view between 2 classes.
- First, the fraud tend to happens at the same retailer that the credit card was used. For example, the cashier steal your information on the card and then use it.
- The fraud transactions have used chip to pay for the transactions but it's less than the fraud that didn't use the chip.
- The transaction didn't use the card pin number. This is easy to understand because who did the fraud can't know your personal pin number.
- The fraud was mostly online orders. I think this is because online orders are the best way to use the credit card.

```
# Selecting input and output for the models
X = df.drop('fraud',axis=1)
y = df['fraud']
```

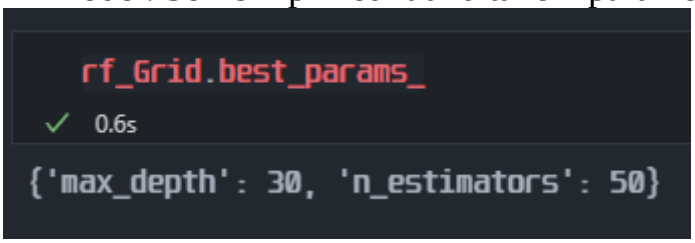
```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
under= RandomUnderSampler()
over = SMOTE()
X,y = under.fit_resample(X, y)

# Summarize class distribution
from collections import Counter
counter = Counter(y)
print(counter)

# Splitting the data to train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 42)
```

- Now to the model, I divide the dataset in to independent variables and dependent variable. Because the data was highly imbalanced, so I use a function called Random Under Sampler. It'll divide the big data part into small pieces and gather them to 1 piece. The data is now balanced between not fraud and fraud.

```
# Number of trees in random forest
# n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 50, num = 5)]
# # Number of features to consider at every split
# max_features = ['auto', 'sqrt']
# # Maximum number of levels in tree
# max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth = [int(x) for x in np.linspace(10, 50, num = 5)]
max_depth.append(None)
# # Minimum number of samples required to split a node
# min_samples_split = [2, 5, 10]
# # Minimum number of samples required at each leaf node
# min_samples_leaf = [1, 2, 4]
# # Method of selecting samples for training each tree
# bootstrap = [True, False]
# # Create the random grid
# param_grid = {'n_estimators': n_estimators,
#                'max_features': max_features,
#                'max_depth': max_depth,
#                'min_samples_split': min_samples_split,
#                'min_samples_leaf': min_samples_leaf,
#                'bootstrap': bootstrap}
param_grid = {'n_estimators': n_estimators,
              'max_depth': max_depth}
print(param_grid)

from sklearn.ensemble import RandomForestClassifier
rf= RandomForestClassifier()
from sklearn.model_selection import GridSearchCV
rf_Grid= GridSearchCV(estimator=rf,param_grid= param_grid,cv=3,verbose=2,n_jobs=4)
```

```
from sklearn.ensemble import RandomForestClassifier
rf= RandomForestClassifier()
from sklearn.model_selection import GridSearchCV
rf_Grid= GridSearchCV(estimator=rf,param_grid= param_grid,cv=3,verbose=2,n_jobs=4)

rf_Grid.fit(X_train, y_train)

rf_Grid.best_params_
```

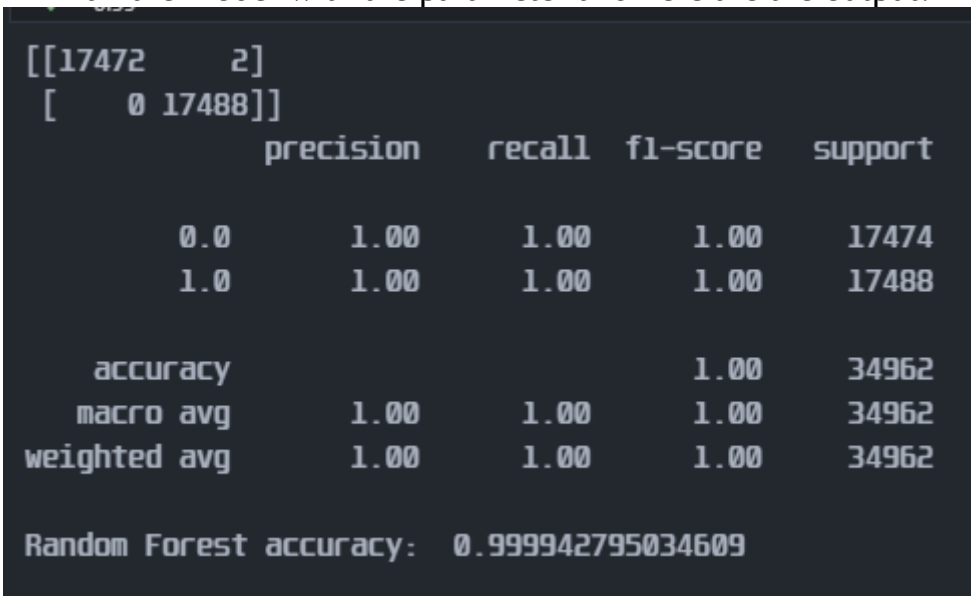
- Next I use the GridSearchCV for the hyper parameters tuning for the input of Random Forest. Unfortunately, my PC took too long to calculate the best parameters for the model. So I simplified it and take 2 parameters with small data point.



- Then I use these parameter to input the model.

```
# Training the data
classifier = RandomForestClassifier(n_estimators = 50,max_depth = 30, criterion =
'entropy', random_state = 42)
classifier.fit(X_train, y_train)

# Predicting the y and arrange the predicted y and actual y for confusion matrix
y_pred = classifier.predict(X_test)

y_pred = classifier.predict(X_test)
print(confusion_matrix(y_test,classifier.predict(X_test)))
print(classification_report(y_test,classifier.predict(X_test)))
print('Decision Tree accuracy: ',
accuracy_score(y_test,classifier.predict(X_test)))
```

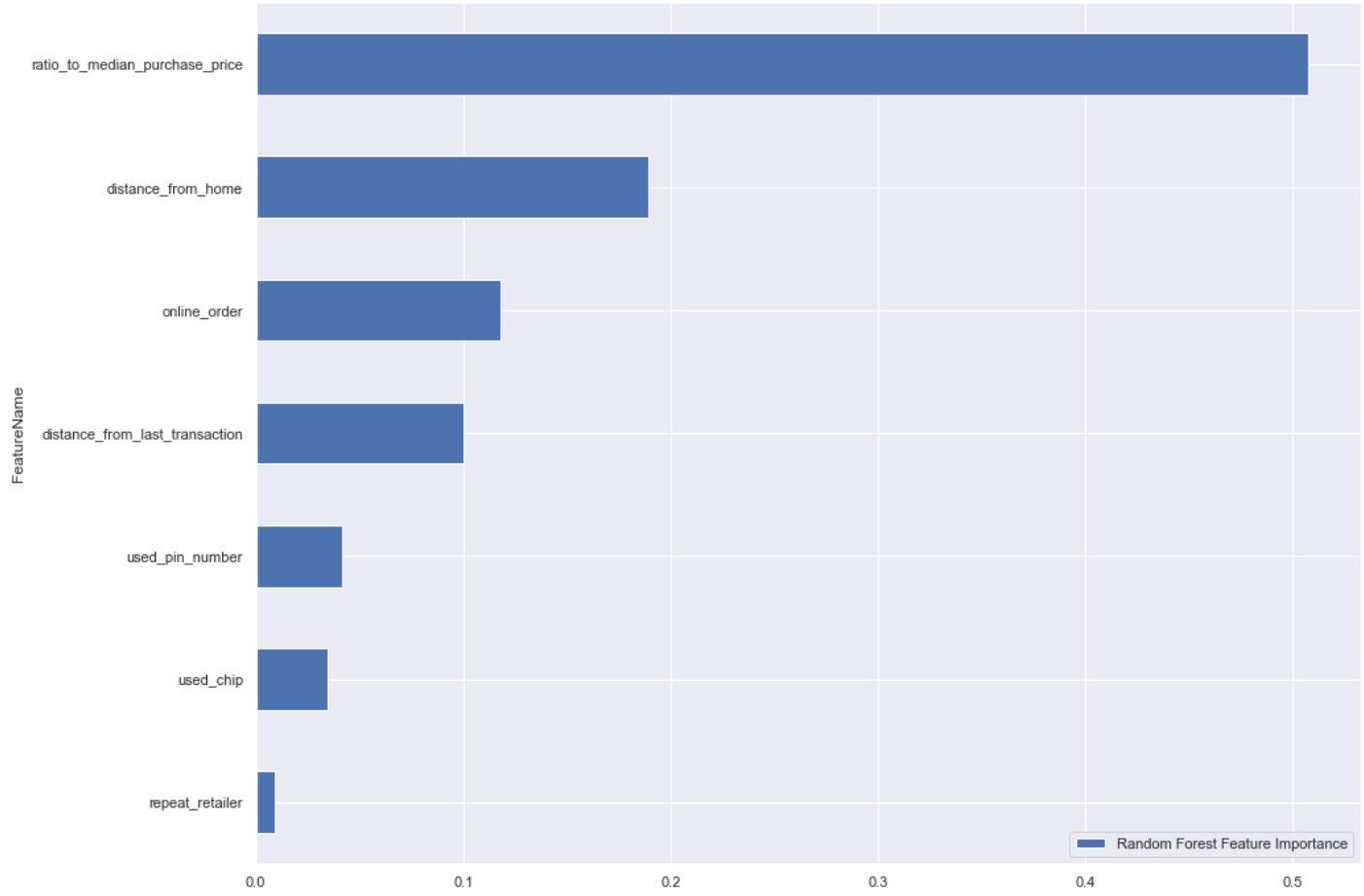
- I ran the model with the parameter and here are the output:



- The confusion matrix was so great. But this also mean that the model are overfitted. And the best way I know to reduce the overfitting was using the hyper parameter tuning. But my PC can compute them so I commented for those who possess the ability to calculate them and output a better model.

```
importance = classifier.feature_importances_
FT = pd.DataFrame({'FeatureName': X.columns, 'Random Forest Feature Importance':
importance})
FT.sort_values(by=['Random Forest Feature Importance'], ascending=False)

FT.sort_values("Random Forest Feature Importance").plot(figsize=(15,12),
x="FeatureName", y=["Random Forest Feature Importance"], kind="barh")
```

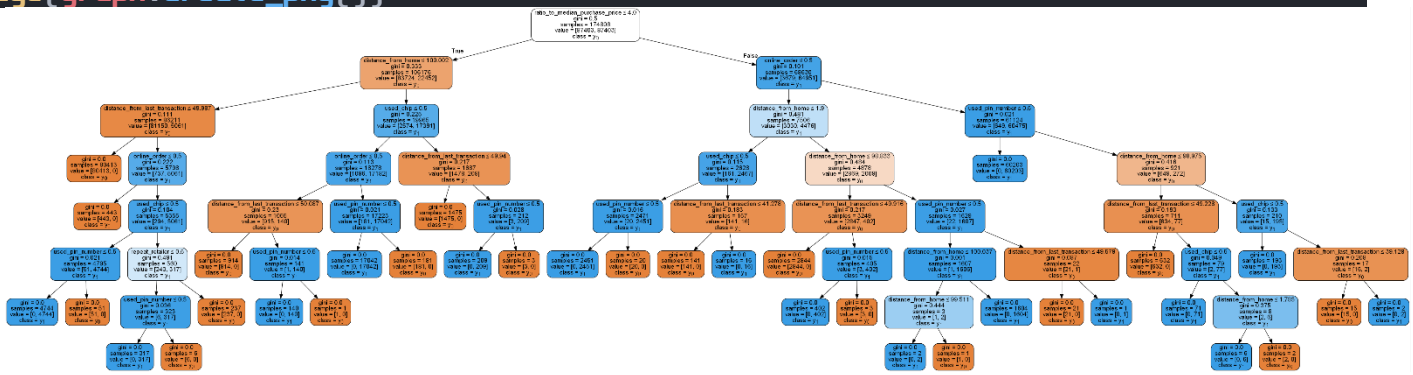
	FeatureName	Random Forest Feature Importance
2	ratio_to_median_purchase_price	0.507141
0	distance_from_home	0.189275
6	online_order	0.117987
1	distance_from_last_transaction	0.099862
5	used_pin_number	0.041561
4	used_chip	0.034883
3	repeat_retailer	0.009291



- As I expected, the ratio was the most important feature to see if a transaction are fraud

or not. But we also have to look at other feature such as the distance from home that the transaction was made, it's only order or not,…

```
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
import pydotplus
from graphviz import Digraph
dot_data = StringIO()
clf = DecisionTreeClassifier()
clf.fit(X.values, y.ravel())
export_graphviz(clf,
out_file=dot_data,feature_names=X.columns,class_names=True,  filled=True,
rounded=True, special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



- Finally I plot the Decision Tree of the fraud cases. This is the way to make the guide lines for the bank or financial instituitions that want to detect the fraud, but the data input must be reliable and big enough for a better approach to this topic.
- The final DT cases can be post-prunned to reduce the leaf and reduce the length of the tree. And the RF contains many DT like above. The best way to use this I think is to input the data and make the model predict if it is fraud or not.
- In conclusion, the RF go great with the classification. If I did the hyper parameters tuning, the model wouldn't be overfited.