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 2](#_Toc106721700)

[1. Credit card 2](#_Toc106721701)

[2. Credit card fraud 2](#_Toc106721702)

[3. Random Forest 2](#_Toc106721703)

[I. Objective of project 2](#_Toc106721704)

[II. Code Explanation 2](#_Toc106721705)

# Project Overview

## 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.

## 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

## 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.

# 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.

# Code Explanation

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

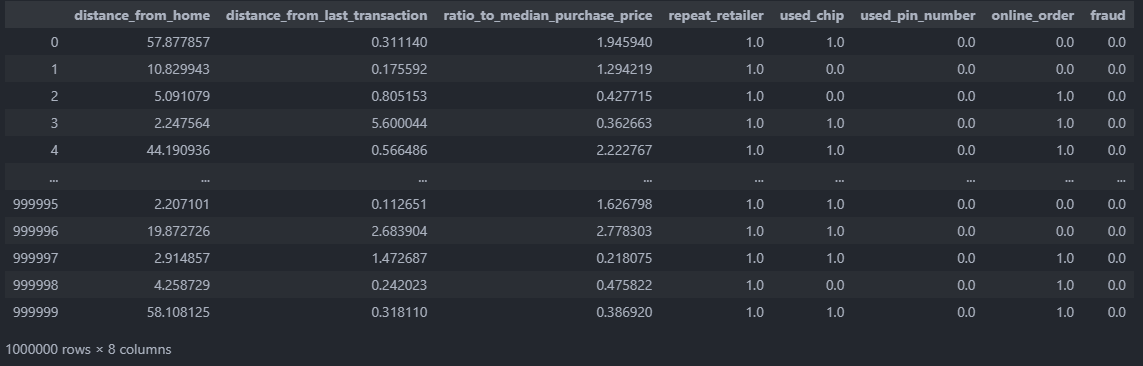
from sklearn import preprocessing

from sklearn.metrics import confusion\_matrix, accuracy\_score,classification\_report

import warnings

warnings.filterwarnings("ignore")

df= pd.read\_csv('card\_transdata.csv')

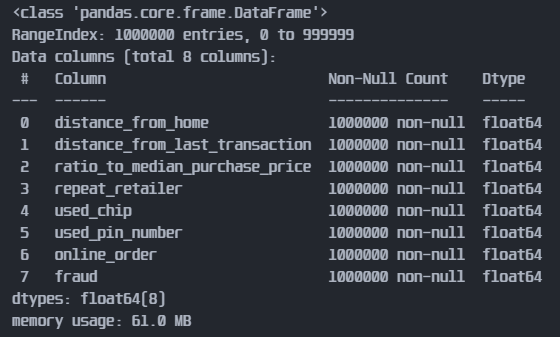
df

* 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.

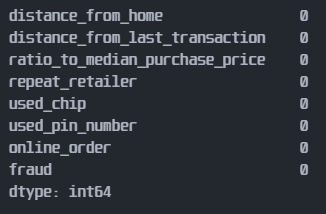
Features Explanation:

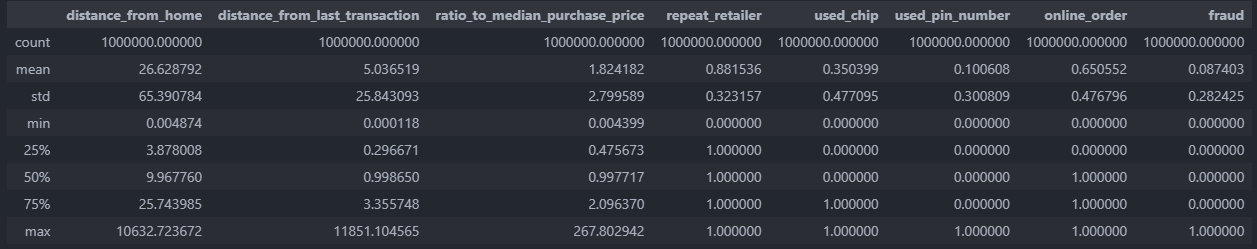
* distancefromhome, numeric - the distance from home where the transaction happened.
* distancefromlast\_transaction, numeric - the distance from last transaction happened.
* ratiotomedianpurchaseprice, numeric - Ratio of purchased price transaction to median purchase price.
* repeat\_retailer, binary - 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()



df.isna().sum()

****

df.describe()

* 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'])

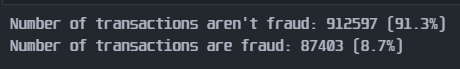
yes = len(df[df['fraud'] == 1]['fraud'])

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))

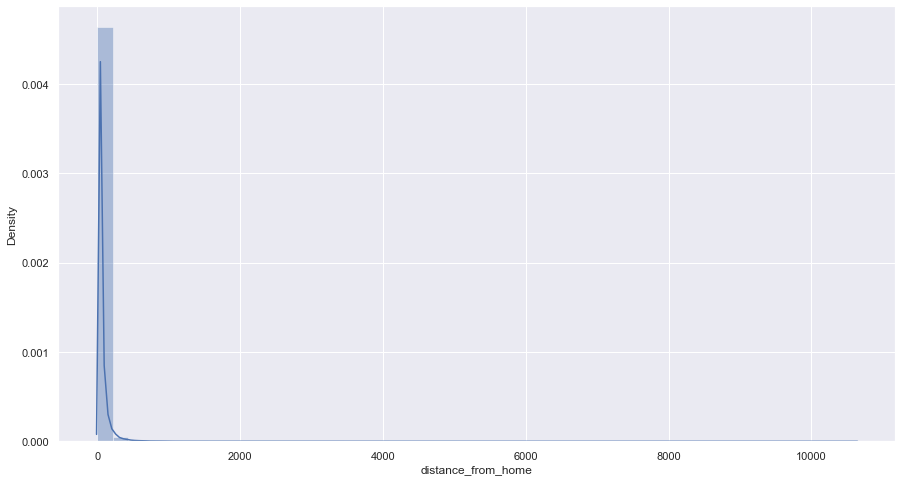
print('Number of transactions are fraud: {} ({}%)'.format(yes, yes\_perc))



* 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\_price']:

    index\_list.extend(outlier(df,i))

def remove(df, ls):

    ls= sorted(set(ls))

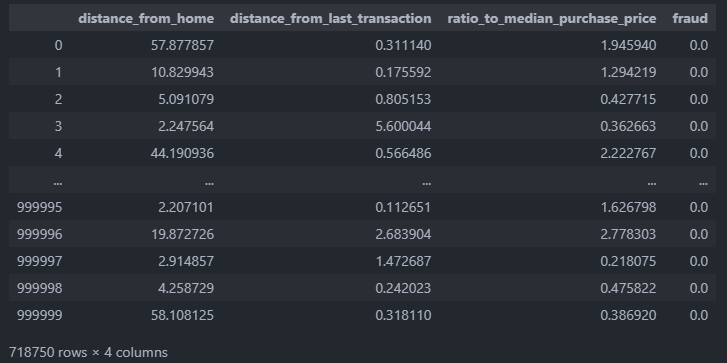
    df= df.drop(ls)

    return df

eda = remove(df,index\_list)

eda = eda.iloc[:,[0,1,2,7]]

eda



* 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.

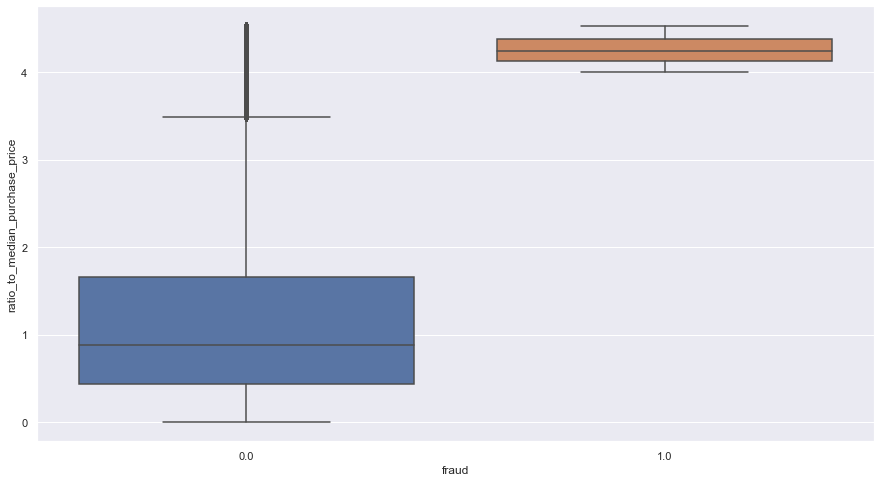
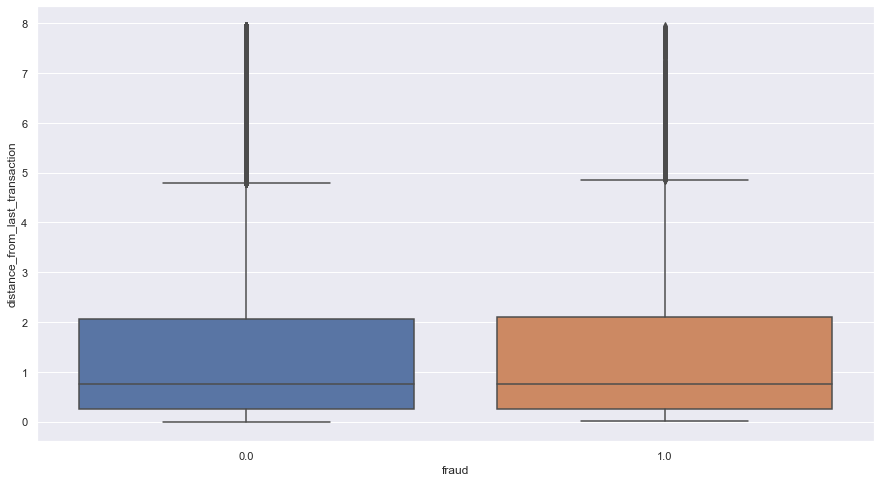
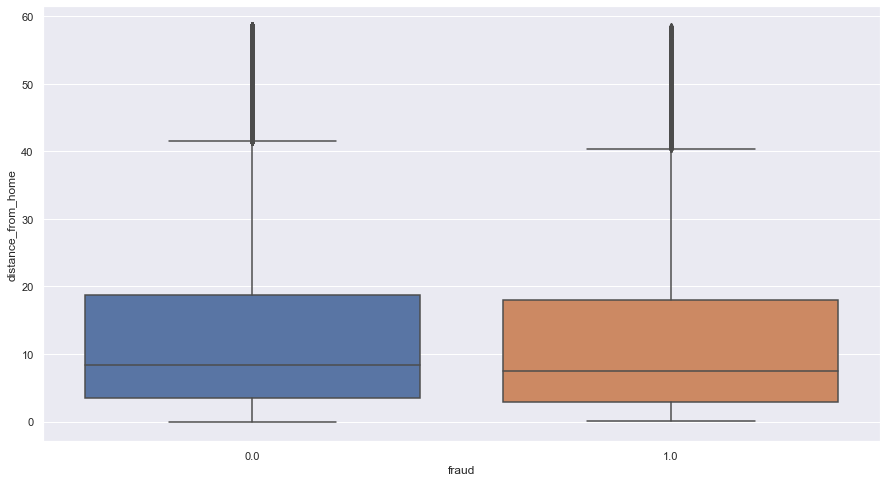
sns.set(rc = {'figure.figsize':(15,8)})

for col in ['distance\_from\_home','distance\_from\_last\_transaction','ratio\_to\_median\_purchase\_price']:

    plt.figure(col)

    sns.set(style ="darkgrid")

    sns.boxplot(x=eda['fraud'],y=eda[col],data=eda)



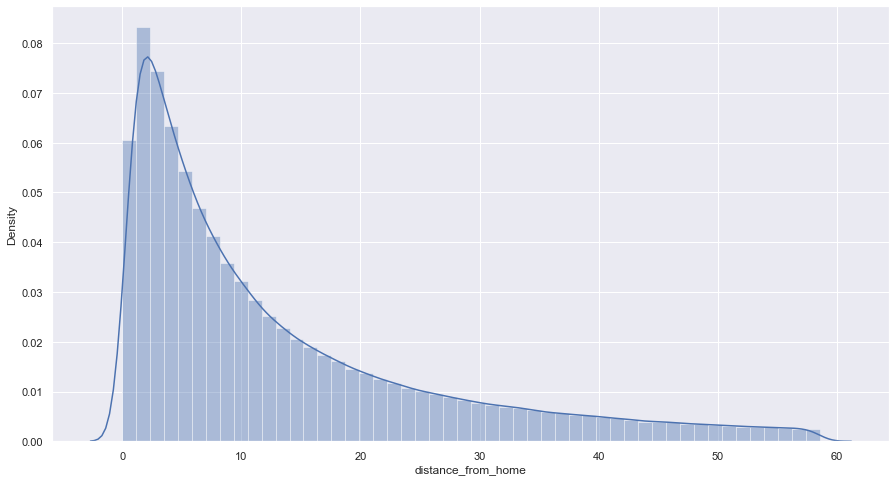
* I visualize the boxplots to see if there are differences between not fraud and fraud. The 2 distances of 2 classes are slightly different but it’s hard to see. The ratio to mean purchase price will be the one that distinguish 2 classes, I think.

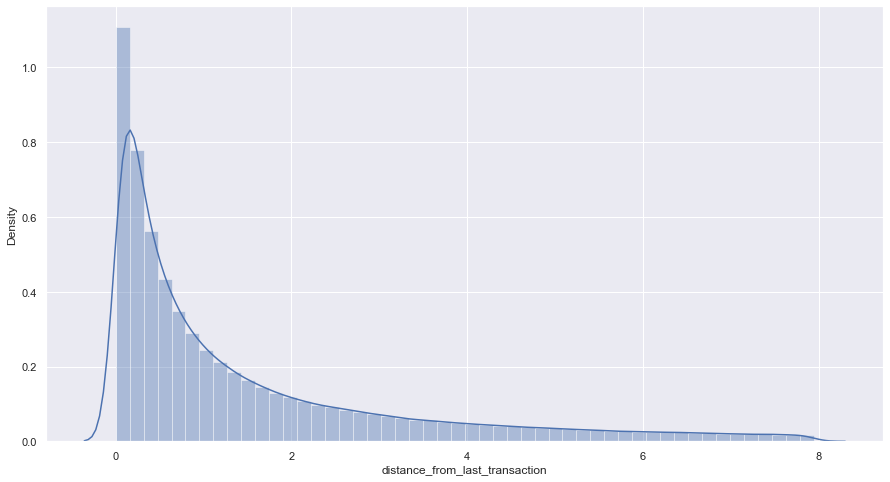
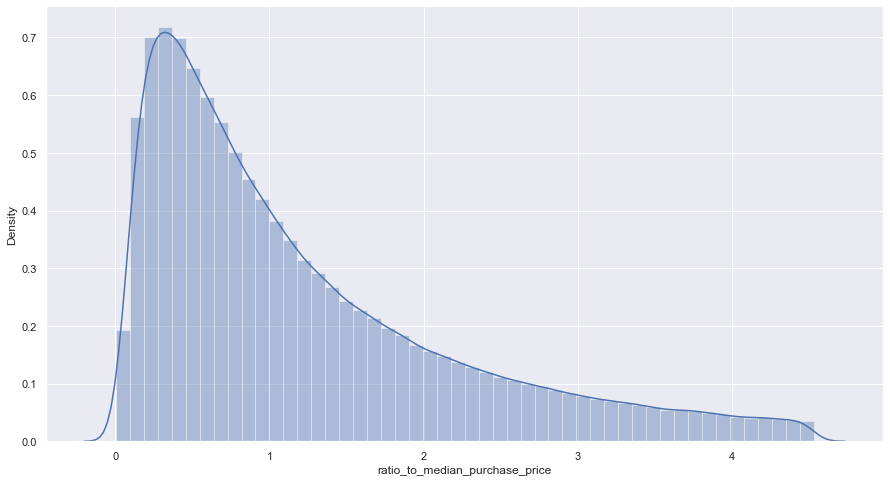
for i in ['distance\_from\_home','distance\_from\_last\_transaction','ratio\_to\_median\_purchase\_price']:

    plt.figure(i)

    sns.set(style ="darkgrid")

    sns.distplot(eda[i])





* 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\_price']:

    plt.rcParams["figure.figsize"] = (15,8)

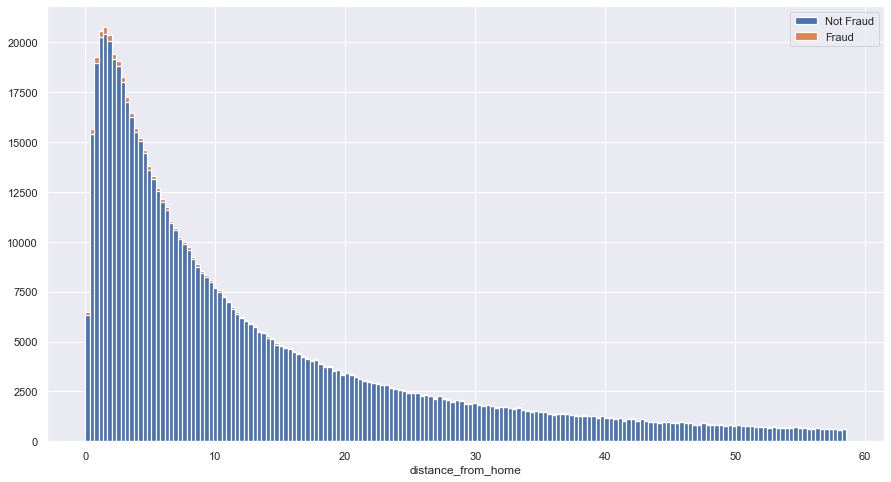
    plt.hist([eda[(eda.fraud==0)][col],eda[(eda.fraud==1)][col]],stacked=True, bins = 'auto',label = ['Not Fraud','Fraud'],linewidth=1.2)

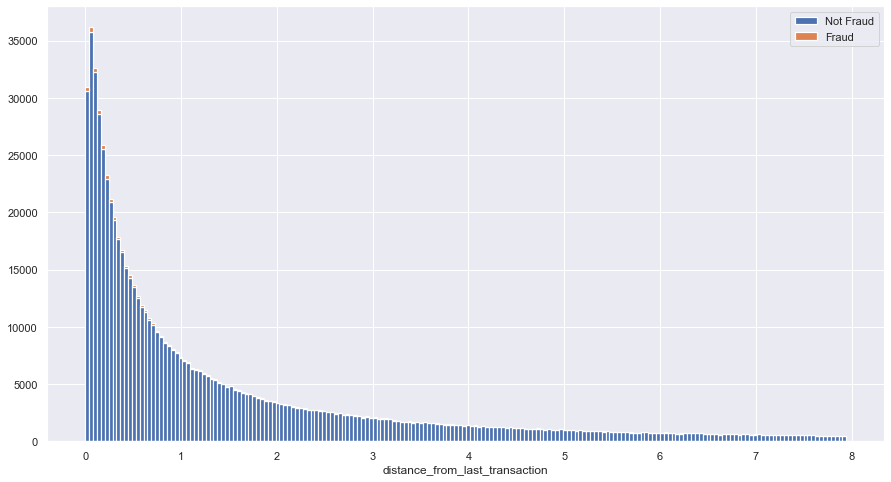
    plt.xlabel(col)

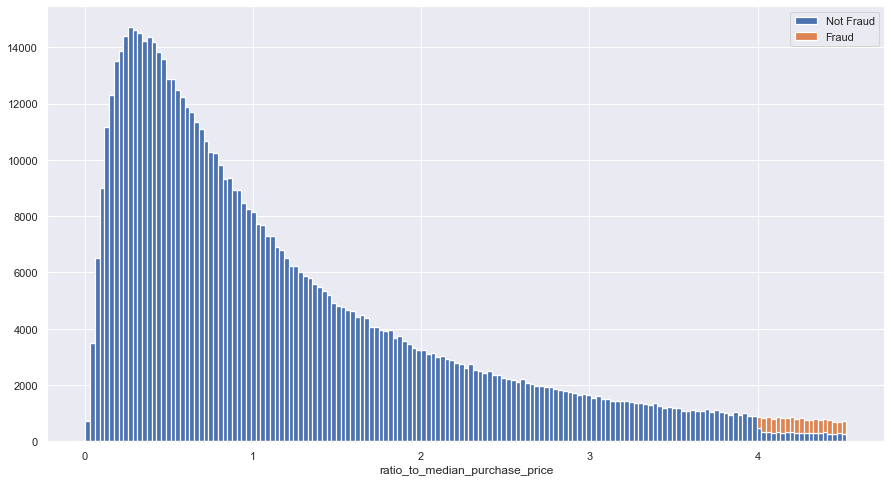
    plt.ylabel('')

    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)

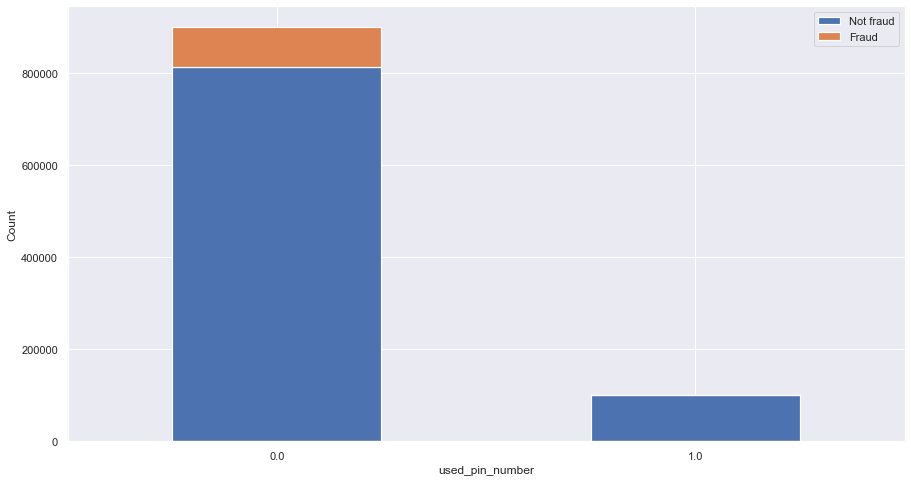
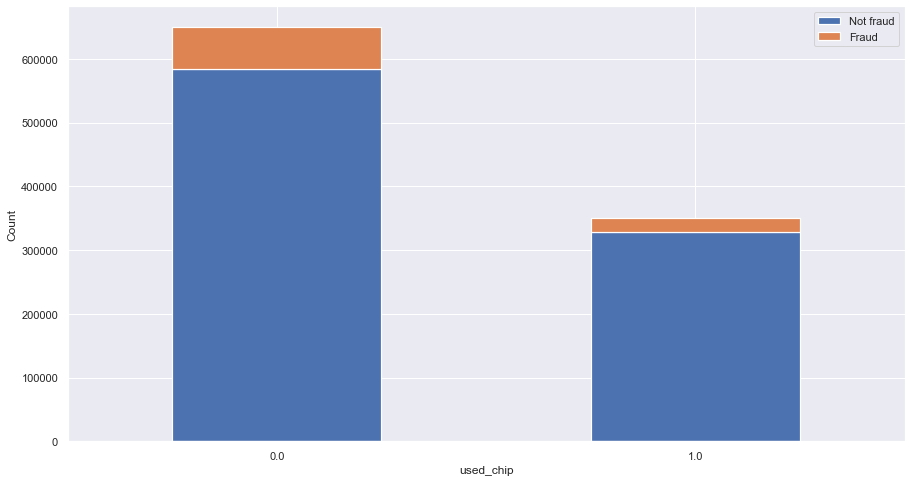
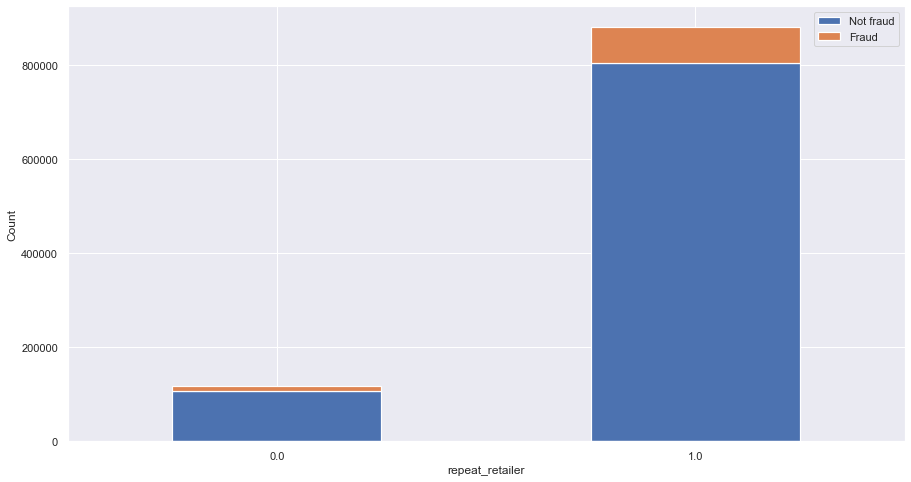
    object.plot(kind = 'bar', stacked = True,rot = 0, figsize=(15,8),linewidth=1.2)

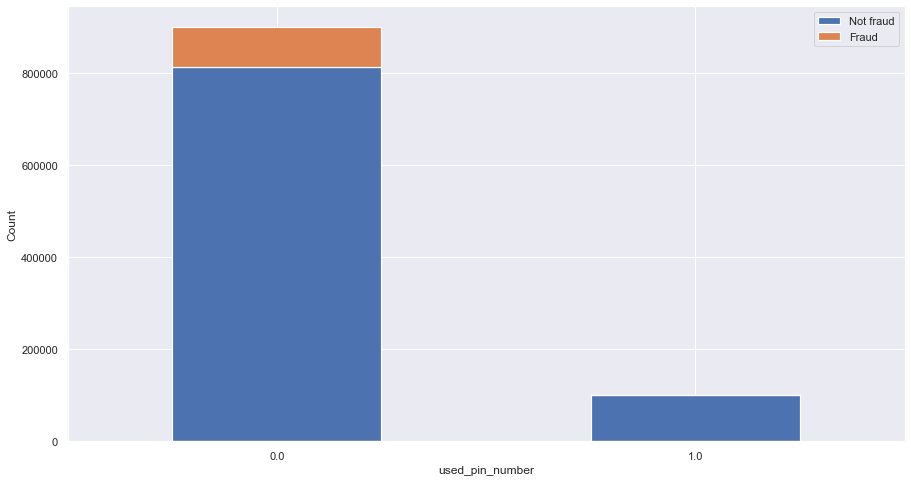
    plt.legend(['Not fraud','Fraud'])

    plt.ylabel('Count')

    plt.xlabel(col)

    plt.show()





* 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()

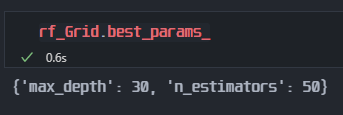
from sklearn.model\_selection import GridSearchCV

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 to 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)

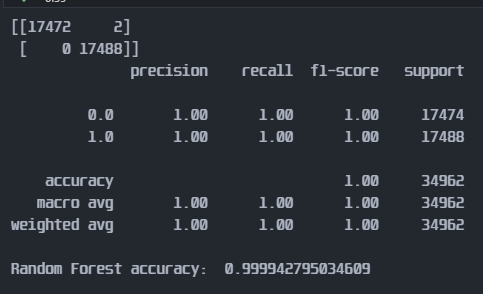
y\_pred = classifier.predict(X\_test)

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:



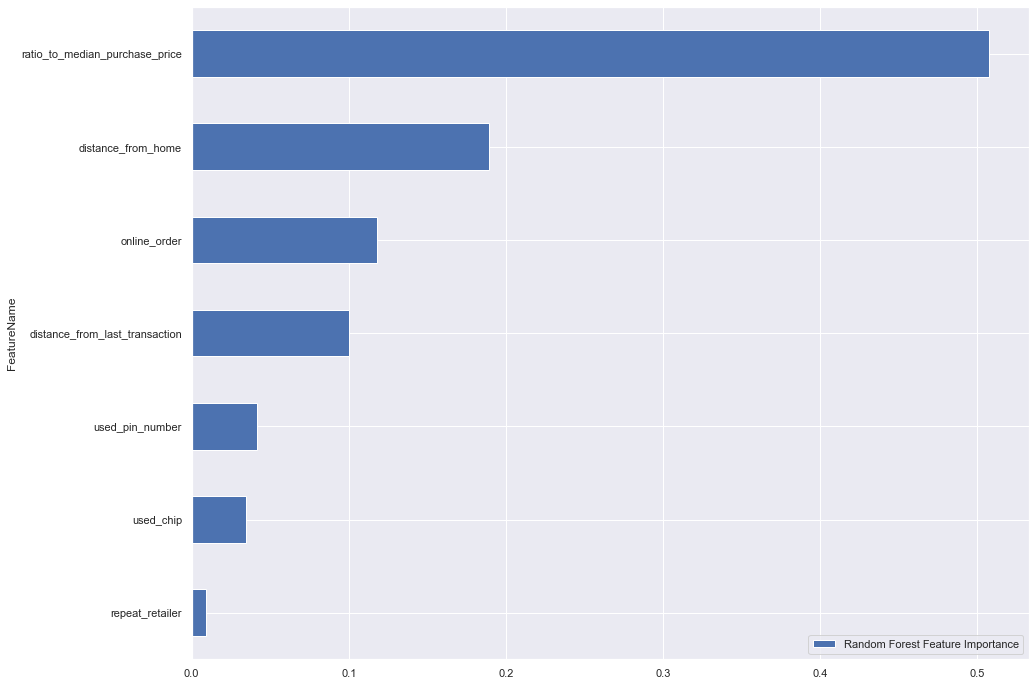
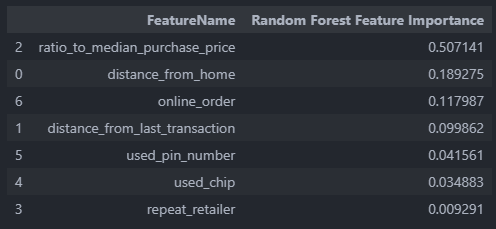
* The confusio matrix was so great. But this also mean that the model are overfitted. And the best way I know to reduce the overfitting was using the hyper parameter tuning. But my PC can compute them so I commented for those who possess the ability to calculate them and output a better model.

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")



* 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()

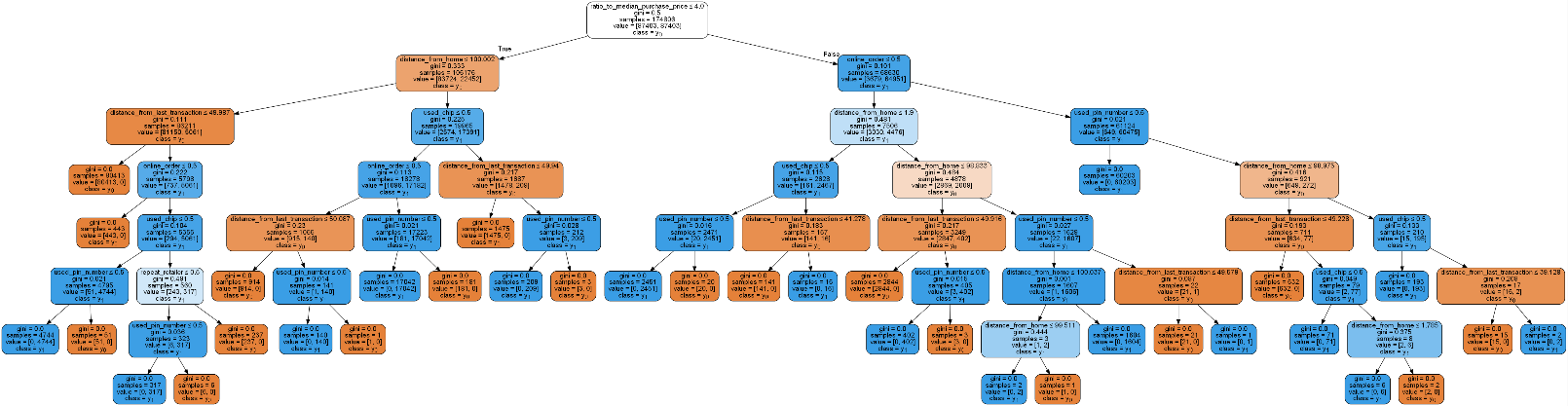
clf = DecisionTreeClassifier()

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.