CREDIT CARD FRAUD DETECTION

ÖZGE GÜNEY
2100365

Bahçeşehir University

In recent years, because of increasing in online transactions, credit card frauds have increased. Therefore, banks had to deal with this problem using data mining techniques. In this data mining project, I am going to use python to create classification algorithms such as to detect credit card fraud by analyzing the old data.
Dataset which I used in this project is
https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?select=creditcard.csv
About Dataset:
The dataset contains transactions made by credit cards in September 2013 by European cardholders.
This dataset presents transactions that occurred in two days.
It contains only numerical input variables which are the result of a PCA transformation.
Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'.
Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.
The feature 'Amount' is the transaction Amount
Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

The dataset have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

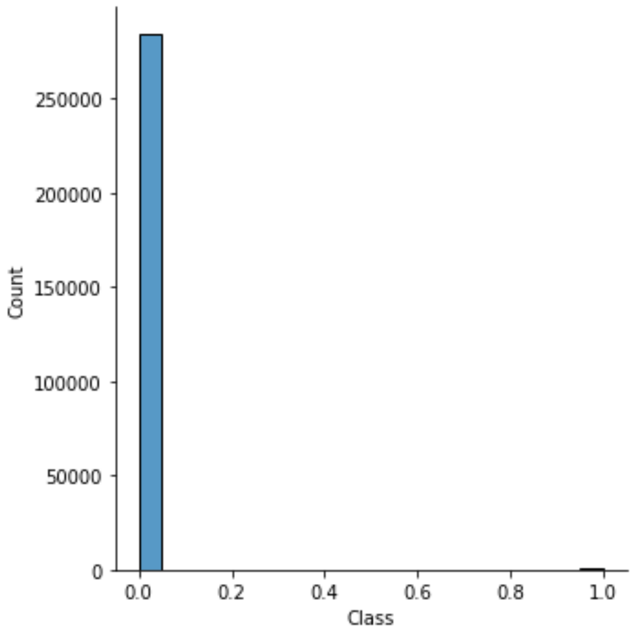


Figure 1: Numbers of Class 0 for normal transactions and Class1 for fraud transactions

If the data is unbalanced, accuracy is not important and minority class' precision and recall are important. If the data is balanced, I can use accuracy.

Dataset describe:
Here, our result is unbalanced, we can see it in Figure 1.
I used Area Under the Precision-Recall Curve (AUPRC) . As we see below in Figure 2, Random Forest is the best algorithm.

Unbalanced Dataset Describe

count	284807.000000
mean	0.001727
std	0.041527
min	0.000000
%25	0.000000
%50	0.000000
%75	0.000000
max	1.000000

Majority and minority class shape

(284315,31)	class2.shape
(492, 31)	class1.shape

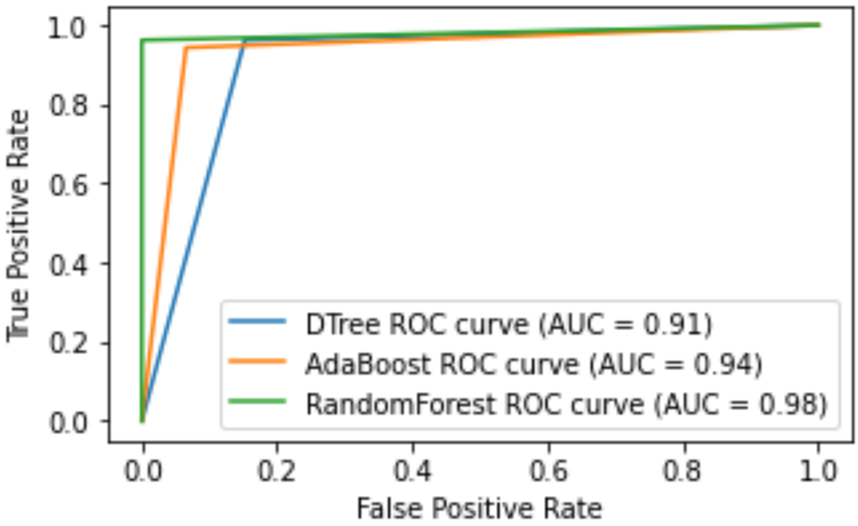


Figure 2: Accuracy of the different algorithms

I used in this project downsample: Now, they includes 492 rows.

Correlations:
We can see the correlation between features in Figure 3.

get top correlations with "Class"

Class	1.000000
V4	0.712453
V11	0.681193
V2	0.491064

Balanced Dataset describe

count	984.000000
mean	0.500000
std	0.500254
min	0.000000
%25	0.000000
%50	0.500000
%75	1.000000
max	1.000000

Majority and minority class

(492,31)	class2.shape
(492, 31)	class1.shape

Accuracy of Different Algorithms

Decision Tree accuracy	0.91
AdaBoost accuracy	0.94
RandomForest accuracy	0.98

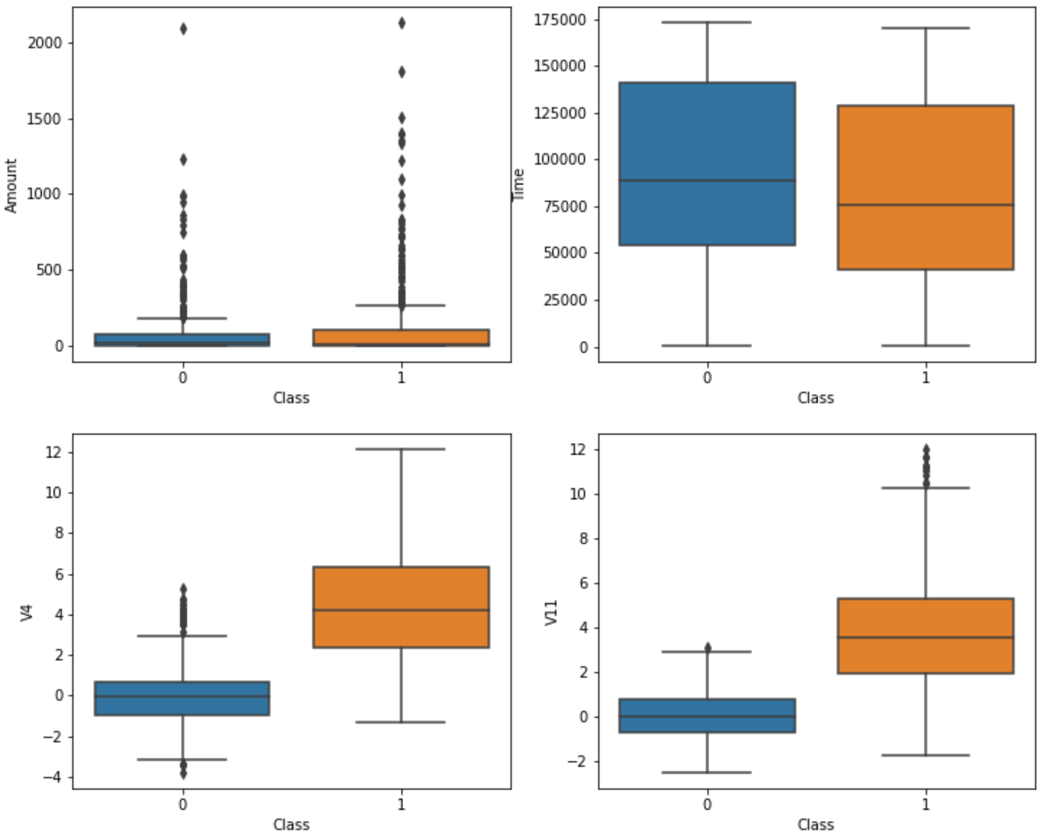
V19	0.253659
V20	0.165434
V21	0.120668
V28	0.110680
V27	0.082108
V8	0.061467

According to this result, these features (V4,V11,V2, V19, V20, V21, V28, V27, V8) effect highly Class output.

Anova F-value:
Below is the result of one-way anova test: As we see below, V4 and V11 features are very effective to the output.

AMOUNT : 8.477407838623522 0.0036771657911981483
V4: 1035.9695475616334 9.20042927677307e-156
V11: 859.8292895258559 2.8839456864094746e-136
Time : 11.578647456592767 0.0006940682737631966

As we see below, time and amount features are not effective to the output Class.



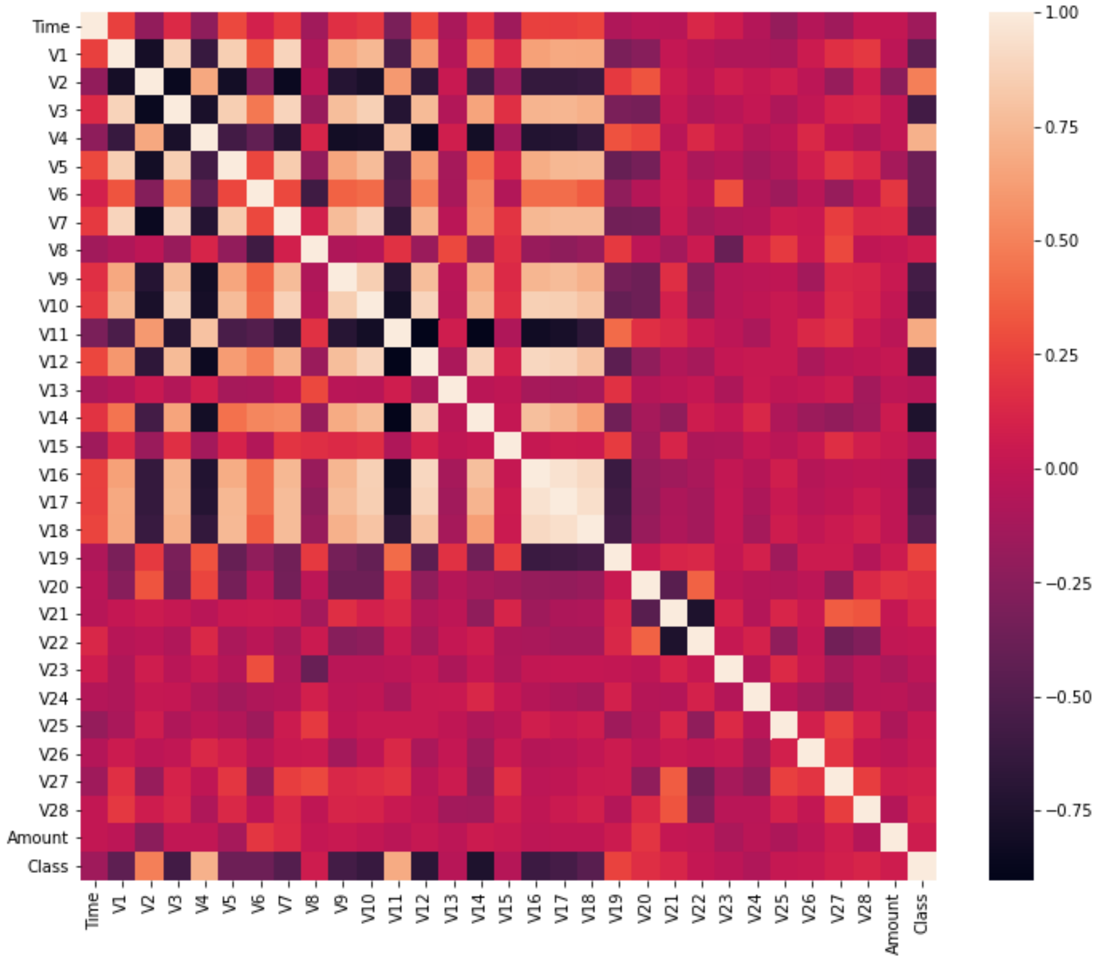


Figure 3:The Correlation of Balanced credit card

We can use SelectKBest to find features with the highest scores. By using SelectKBest I got this result.

features		values	p-values
0	V14	1259.408993	3.560270e-178
1	V4	1002.311300	3.580733e-152
2	V11	872.700886	9.402612e-138
3	V12	863.859359	9.848618e-137
4	V10	635.362783	1.622826e-108
5	V16	553.260524	2.177498e-97
6	V3	462.535291	2.250863e-84
7	V9	452.888546	6.084922e-83
8	V17	441.992780	2.589774e-81
9	V2	326.563603	3.064403e-63
10	V7	283.725974	4.045118e-56

As a final example, I want to show you a part of decision tree which is used here.

```
|--- feature_14 <= -2.19
|   |--- feature_14 <= -3.38
|   |   |--- feature_12 <= 0.19
|   |   |   |--- class: 1
|   |   |--- feature_12 >  0.19
|   |   |   |--- feature_16 <= 0.76
|   |   |   |   |--- class: 0
|   |   |   |--- feature_16 >  0.76
|   |   |   |   |--- class: 1
|   |--- feature_14 >  -3.38
|   |   |--- feature_17 <= 2.21
|   |   |   |--- feature_11 <= -0.06
|   |   |   |   |--- feature_29 <= 260.36
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_29 >  260.36
|   |   |   |   |   |--- class: 1
|   |   |   |--- feature_11 >  -0.06
|   |   |   |   |--- class: 1
|   |   |--- feature_17 >  2.21
|   |   |   |--- class: 0
```

As we see, again feature V14 is effective feature. Similarly, V12 were chosen by using SelectKBest and we see it here.

To sum up,

Fraud detection dataset is unbalanced dataset. With unbalanced dataset, we can see the result by using Precision-Recall Curve (AUPRC). In addition to this, we can use downsample method and now we can use accuracy to compare the algorithms. Both of them gave me that random forest is the best algorithm for this problem.

In data mining project, explanatory data analysis is very important. There are some methods. Here, I used SelectKBest algorithm. And I compared the other methods. Many of them gave similar result.

Code:

```
import pandas as pd
import seaborn as sns



credit_card = pd.read_csv("/Users/ozgeguney/.spyder-py3/DataMining/creditcard.csv")


print(credit_card.describe(include="all").loc[:,"Class"])

credit_card_class1 =  credit_card[credit_card.Class==1]
print(credit_card_class1.shape)

print(credit_card.Class.value_counts())

# sns.displot(credit_card.Class)

cc_majority = credit_card[credit_card.Class==0]
cc_minority = credit_card[credit_card.Class==1]



from sklearn.utils import resample
cc_majority_downsampled = resample(cc_majority,
                                   replace=False,
                                   n_samples=492)

cc_balanced = pd.concat([cc_minority, cc_majority_downsampled])
print(cc_balanced.Class.value_counts())
print(cc_balanced.describe(include="all").loc[:,"Class"])
```

```
X = cc_balanced.loc[:,'Time':'Amount']
y = cc_balanced.loc[:,'Class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
X_train = X_train.fillna(X_train.mean())
X_test = X_test.fillna(X_test.mean())

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
clf1 = DecisionTreeClassifier()
clf2 = AdaBoostClassifier(n_estimators=200)
clf3 = RandomForestClassifier(n_estimators=100, bootstrap=True)

clf1.fit(X_train, y_train);
clf2.fit(X_train, y_train);
clf3.fit(X_train, y_train);

y_pred1 = clf1.predict(X_test)
y_pred2 = clf2.predict(X_test)
y_pred3 = clf3.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred1))
print(classification_report(y_test,y_pred2))
print(classification_report(y_test,y_pred3))

# from sklearn.metrics import recall_score
# recall_acc_DecisionTree = recall_score (y_test,y_pred1)
# print(recall_acc_DecisionTree)
# recall_acc_AdaBoost = recall_score (y_test,y_pred2)
# print(recall_acc_AdaBoost)
# recall_acc_RandomForest = recall_score (y_test,y_pred3)
# print(recall_acc_RandomForest)

from sklearn.metrics import roc_curve, auc
fpr1, tpr1, thresholds1 = roc_curve(y_test, y_pred1,drop_intermediate=False)
fpr2, tpr2, thresholds2 = roc_curve(y_test, y_pred2,drop_intermediate=False)
fpr3, tpr3, thresholds3 = roc_curve(y_test, y_pred3,drop_intermediate=False)


import matplotlib.pyplot as plt

auc1 = auc(fpr1, tpr1)
auc2 = auc(fpr2, tpr2)
auc3 = auc(fpr3, tpr3)
```

```
plt.plot(fpr1,tpr1,label='DTree ROC curve (AUC = %0.2f)' % auc1);
plt.plot(fpr2,tpr2,label='AdaBoost ROC curve (AUC = %0.2f)' % auc2);
plt.plot(fpr3,tpr3,label='RandomForest ROC curve (AUC = %0.2f)' % auc3);
plt.legend(loc="lower right")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')



plt.figure(figsize=(12,10));
sns.heatmap(cc_balanced.corr());



# get top correlations with Class
cors = cc_balanced.corr();
cors.loc[:, "Class"].sort_values(ascending = False ).head(10)


sns.displot(credit_card.Class);


sns.catplot(x='Class', y='Amount', data=cc_balanced, jitter=0.2);


sns.catplot(x='Class', y='Amount', kind="box", data=cc_balanced);

f, ax = plt.subplots(2,2)
f.set_size_inches(12,10)
sns.boxplot(y="Amount", x="Class", data= cc_balanced, ax = ax[0,0]);
sns.boxplot(y="Time", x="Class", data= cc_balanced, ax = ax[0,1]);
sns.boxplot(y="V4", x="Class", data= cc_balanced, ax = ax[1,0]);
sns.boxplot(y="V11", x="Class", data= cc_balanced, ax = ax[1,1]);


cc_balanced["Amount"].hist(by=cc_balanced["Class"])
cc_balanced["V4"].hist(by=cc_balanced["Class"])


import scipy.stats as stats
group1 = cc_balanced[cc_balanced["Class"]==1].Amount
group2 = cc_balanced[cc_balanced["Class"]==0].Amount
fvalue, pvalue = stats.f_oneway(group1, group2)
print(fvalue, pvalue)


group1 = cc_balanced[cc_balanced["Class"]==1].V4
```

```
group2 = cc_balanced[cc_balanced["Class"]==0].V4
fvalue, pvalue = stats.f_oneway(group1, group2)
print(fvalue, pvalue)

group1 = cc_balanced[cc_balanced["Class"]==1].Time
group2 = cc_balanced[cc_balanced["Class"]==0].Time
fvalue, pvalue = stats.f_oneway(group1, group2)
print(fvalue, pvalue)

from sklearn.feature_selection import  SelectKBest, f_classif
# from sklearn.feature_selection import mutual_info_classif
selector = SelectKBest(f_classif, k='all')
# selector = SelectKBest(score_func=mutual_info_classif, k='all')
selector.fit(X, y)

import numpy as np
sorted_idx = np.argsort(selector.scores_)[::-1]
sorted_vals = np.sort(selector.scores_)[::-1]

d = {"features":X.columns[sorted_idx], "values":sorted_vals, "p-
values":selector.pvalues_[sorted_idx]}
df = pd.DataFrame(d)
df

from sklearn import tree
text_representation = tree.export_text(clf1)
print(text_representation)



from sklearn.tree import plot_tree
feature_names = cc_balanced.columns


plot_tree(clf1,
        feature_names = feature_names,
        class_names = "Class",
        filled = True,
        rounded = True)

plt.savefig('tree_visualization.png')
```

PART 2

The second dataset : https://www.kaggle.com/datasets/kartik2112/fraud-detection?resource=download

This dataset includes 555719 rows. This dataset is unbalanced, so I used downsampled. Majority class includes 553574 rows and minority class 2145 rows. I continued with based on minority class. Total dataset includes 4290 rows.
Now, my dataset is :
1   2145
0   2145
This dataset includes 23 rows initially. Fraud rows : (2145, 23)
These columns are as below.

Data Dictionary

- trans_date_trans_time -> Transaction time stamp
- **cc_num ->** Credit card number
- **merchant ->** merchant name
- **category ->** transaction category
- **amt ->** Transaction amount
- **first ->** First name of card holder
- **last ->** Last name of card holder
- **gender ->** Sex of card holder
- **street ->** transaction address
- **city ->** transaction city
- **state ->** transaction state
- **zip ->** transaction zipcode
- **lat ->** transaction lattitude
- **long ->** transaction longitude
- **city_pop ->** Population of the city
- **job ->** job of the card holder
- **dob ->** date of birth of card holder
- **trans_num ->** transaction number of transaction
- **unix_time ->** time in unix format
- **merch_lat ->** lattitude of the merchant
- **merch_long ->** longitude of merchant
- **is_fraud ->** nature of transaction (fraud or not fraud)

Data pre-processing

1-) #converting trans_date_trans_time into datetime
cc_balanced['trans_date_trans_time'] = pd.to_datetime(cc_balanced[‘trans_date_trans_time'])
I separated the column into hour, day, and Month-year, because I need to get more information from the column and add these additional columns to original dataframe.

cc_balanced['trans_hour'] = cc_balanced['trans_date_trans_time'].dt.hour
#deriving 'day of the week'
cc_balanced['trans_day_of_week'] = cc_balanced['trans_date_trans_time'].dt.day_name()

#deriving 'year_month'
cc_balanced['trans_year_month'] = cc_balanced['trans_date_trans_time'].dt.to_period('M')

2-) Then I calculated the age of the customer at the time of transacting. cc_balanced['age'] = np.round((cc_balanced['trans_date_trans_time'] -
cc_balanced['dob'])/np.timedelta64(1, 'Y'))

I can drop trans_date_trans_time, because I have already divide into some part. I can drop dob, because. I calculated age and I use it.

3-) dropping unique variables
For example; name, last name, trans_num features are special and it does not give us any important

Table 1-1

trans_date_trans_time	4290
trans_num	4290
unix_time	4290
merch_long	4290
merch_lat	4289
amt	3883
cc_num	810
street	810
dob	802
zip	801
lat	799
long	799
city	749
city_pop	737
merchant	675
job	449
last	436
first	321

result. It is unnecessary to use these data. Table 1 shows us unique vales of features.

#dropping unique variables
cc_balanced.drop(['trans_date_trans_time','unix_time', 'trans_num','merch_long','merch_lat', 'cc_num', 'lat', 'long',
'first', 'last', 'dob','street'] , axis=1, inplace=True)

4-) Look other features: Some of these features look like important, because their distributions are different. As we see below Figure 1, trans_hour and art (amount) features are important.

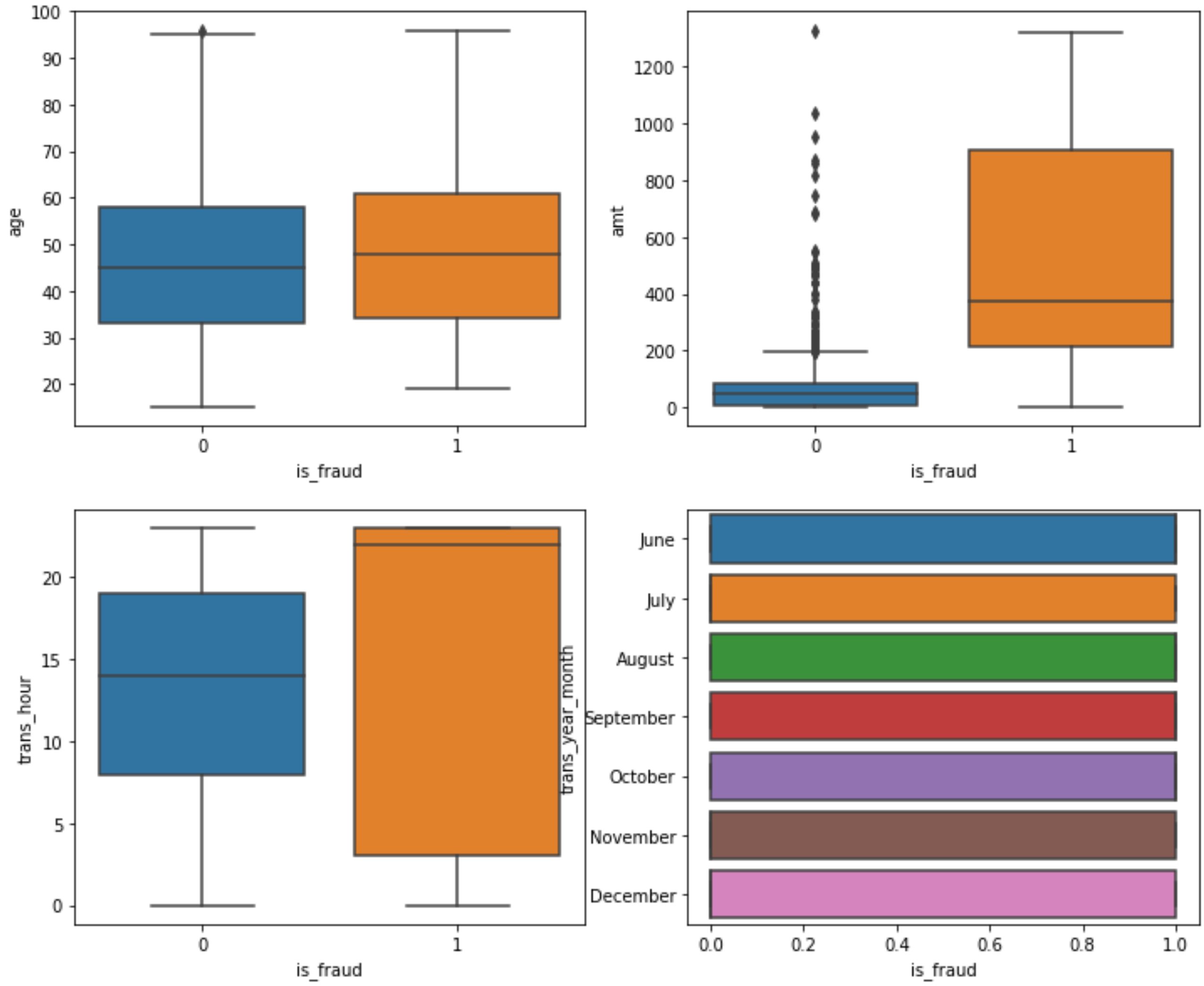


Figure 1: Some Features vs Fraud Transaction

5-) With SelectKbest, select numeric best columns.
X1 = cc_balanced.loc[:,'merchant':'age']
X1 = cc_balanced.select_dtypes(include=np.number) # select numeric columns

Table 2 shows us feautures and their efficient values and p values. **If p value is greater than 0.05**, it means we need collect data. Therefore, act, trans_hour, age and city_pop features are important than zip feature. We can drop it.

Table 2

	features	values	p-values
0	is_fraud	inf	0.000000e+00
1	amt	2804.196380	0.000000e+00
2	trans_hour	30.137082	4.258150e-08
3	age	16.629627	4.625995e-05
4	city_pop	7.300185	6.921886e-03
5	zip	1.600961	2.058358e-01

6-) With chi2, select best string columns.
I use here labelEncoder.
le = LabelEncoder()
X2 = cc_balanced.loc[:,'merchant':'age']
X2 = X2.select_dtypes(include=object) # select string columns
X2 = X2.apply(LabelEncoder().fit_transform)
y2 = le.fit_transform(cc_balanced.loc[:,'is_fraud'])
chi2, pval=feature_selection.chi2(X2, y2).
Table 3 shows us job, category, state, trans_year_month, merchant features are more important than trans_day_of_week, gender and city.

Table 3

	features	values	(sklearn)	p-values
0	job		601.790788	6.827934e-133
1	category		164.398429	1.238008e-37
2	state		81.278102	1.960874e-19
3	trans_year_month		60.548307	7.179483e-15
4	merchant		24.546889	7.252334e-07
5	trans_day_of_week		6.491746	1.083765e-02
6	gender		2.090047	1.482615e-01
7	city		0.392579	5.309467e-01

In summary, job, amt, trans_hour, category features are important columns. Firstly, I tried some algorithms without these parameter and then I added one by one. I calculated accuracy.
Here, my new dataset is balanced, I can use accuracy.

7-) Prepared X and y input for algorithms. Then I used **Knn, DecisionTreeClassifier, AdaBoostClassifier, RandomForestClassifier** algorithms and I shared the accuracy results.

# 'job','amt', 'trans_hour','category', 'age',
X = cc_balanced.loc[:,['zip','city','merchant', 'gender', 'state',
        'city_pop',   'trans_day_of_week']]
y = cc_balanced.loc[:,'is_fraud']

```
knn                   precision    recall  f1-score   support

           0       0.98      0.62      0.76       210
           1       0.73      0.99      0.84       219

    accuracy                           0.81       429
   macro avg       0.85      0.80      0.80       429
weighted avg       0.85      0.81      0.80       429

/opt/anaconda3/lib/python3.8/site-packages/sklearn/feature_selection/_univariate_selection.py
113: RuntimeWarning: divide by zero encountered in true_divide
  f = msb / msw
DecisionTreeClassifier              precision    recall  f1-score   support

           0       0.97      0.80      0.88       210
           1       0.84      0.97      0.90       219

    accuracy                           0.89       429
   macro avg       0.90      0.89      0.89       429
weighted avg       0.90      0.89      0.89       429

AdaBoostClassifier              precision    recall  f1-score   support

           0       0.65      0.58      0.61       210
           1       0.64      0.70      0.67       219

    accuracy                           0.64       429
   macro avg       0.64      0.64      0.64       429
weighted avg       0.64      0.64      0.64       429

RandomForestClassifier              precision    recall  f1-score   support

           0       0.98      0.82      0.89       210
           1       0.85      0.99      0.91       219

    accuracy                           0.90       429
   macro avg       0.92      0.90      0.90       429
```

Add **trans_year_month** feature:

```
knn                   precision    recall  f1-score   support

           0       0.96      0.60      0.74       220
           1       0.70      0.97      0.81       209

    accuracy                           0.78       429
   macro avg       0.83      0.79      0.77       429
weighted avg       0.83      0.78      0.77       429

DecisionTreeClassifier              precision    recall  f1-score   support

           0       0.95      0.82      0.88       220
           1       0.83      0.96      0.89       209

    accuracy                           0.89       429
   macro avg       0.89      0.89      0.89       429
weighted avg       0.89      0.89      0.89       429

AdaBoostClassifier              precision    recall  f1-score   support

           0       0.78      0.85      0.81       220
           1       0.83      0.75      0.78       209

    accuracy                           0.80       429
   macro avg       0.80      0.80      0.80       429
weighted avg       0.80      0.80      0.80       429

RandomForestClassifier              precision    recall  f1-score   support

           0       0.99      0.91      0.95       220
           1       0.92      0.99      0.95       209

    accuracy                           0.95       429
   macro avg       0.95      0.95      0.95       429
weighted avg       0.95      0.95      0.95       429
```

Add **job** and **amt** features:

knn	precision	recall	f1-score	support
0	0.87	0.76	0.81	238
1	0.74	0.85	0.79	191
accuracy			0.80	429
macro avg	0.80	0.81	0.80	429
weighted avg	0.81	0.80	0.80	429

DecisionTreeClassifier	precision	recall	f1-score	support
0	0.89	0.89	0.89	238
1	0.86	0.87	0.86	191
accuracy			0.88	429
macro avg	0.88	0.88	0.88	429
weighted avg	0.88	0.88	0.88	429

AdaBoostClassifier	precision	recall	f1-score	support
0	0.91	0.92	0.91	238
1	0.89	0.88	0.89	191
accuracy			0.90	429
macro avg	0.90	0.90	0.90	429
weighted avg	0.90	0.90	0.90	429

RandomForestClassifier	precision	recall	f1-score	support
0	0.97	0.96	0.96	238
1	0.95	0.96	0.96	191
accuracy			0.96	429
macro avg	0.96	0.96	0.96	429
weighted avg	0.96	0.96	0.96	429

Add **trans_hour**:

knn	precision	recall	f1-score	support
0	0.88	0.77	0.82	211
1	0.80	0.90	0.85	218
accuracy			0.84	429
macro avg	0.84	0.84	0.84	429
weighted avg	0.84	0.84	0.84	429

DecisionTreeClassifier	precision	recall	f1-score	support
0	0.92	0.94	0.93	211
1	0.94	0.92	0.93	218
accuracy			0.93	429
macro avg	0.93	0.93	0.93	429
weighted avg	0.93	0.93	0.93	429

AdaBoostClassifier	precision	recall	f1-score	support
0	0.93	0.94	0.93	211
1	0.94	0.93	0.94	218
accuracy			0.93	429
macro avg	0.93	0.93	0.93	429
weighted avg	0.93	0.93	0.93	429

RandomForestClassifier	precision	recall	f1-score	support
0	0.95	0.97	0.96	211
1	0.97	0.95	0.96	218
accuracy			0.96	429
macro avg	0.96	0.96	0.96	429
weighted avg	0.96	0.96	0.96	429

Add **Category:**

```
knn                  precision    recall  f1-score   support

           0       0.81      0.75      0.78       212
           1       0.77      0.83      0.80       217

    accuracy                           0.79       429
   macro avg       0.79      0.79      0.79       429
weighted avg       0.79      0.79      0.79       429

DecisionTreeClassifier              precision    recall  f1-score   support

           0       0.97      0.95      0.96       212
           1       0.95      0.97      0.96       217

    accuracy                           0.96       429
   macro avg       0.96      0.96      0.96       429
weighted avg       0.96      0.96      0.96       429

AdaBoostClassifier                precision    recall  f1-score   support

           0       0.96      0.97      0.96       212
           1       0.97      0.96      0.96       217

    accuracy                           0.96       429
   macro avg       0.96      0.96      0.96       429
weighted avg       0.96      0.96      0.96       429

RandomForestClassifier              precision    recall  f1-score   support

           0       0.95      0.99      0.97       212
           1       0.99      0.95      0.97       217

    accuracy                           0.97       429
   macro avg       0.97      0.97      0.97       429
weighted avg       0.97      0.97      0.97       429
```

As we see above accuracy results, we get more effective result by adding important features, Firstly, I got 0.90 and finally I got 0.97 accuracy result in Random Forest.
I used some graphically show, SelectKBest, chi2 to select effective features. I used here downsampled dataset. Because my original dataset is unbalanced, and I converted it to unbalanced data by using minority class. Minority class includes fraud transactions. I got that **RandomForest** is better than **KNN**, **AdaBoostClassifier** and **DecisionTree** classifiers. I want to show only confusion matrix of RandomForest. It is like below Figure 2.
[[184 38]
 [7 200]]

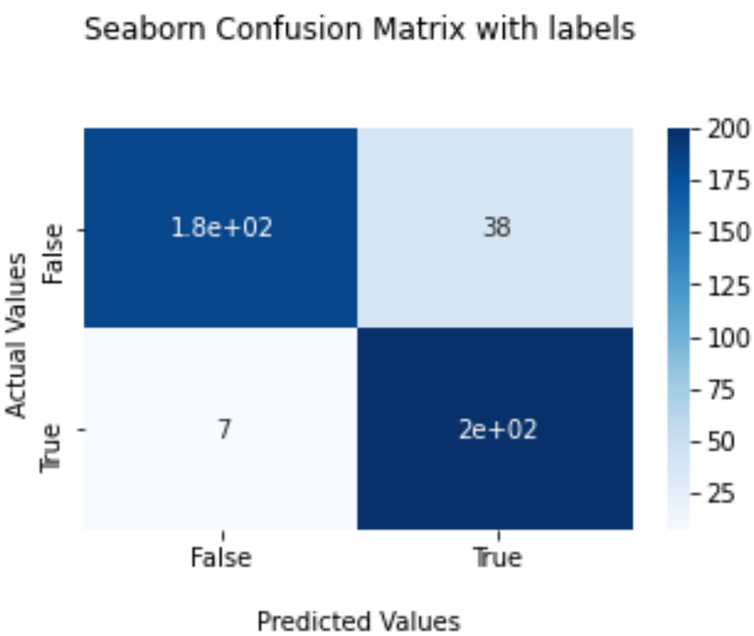


Figure 2 : Random Forest Confusion Matrix

PART 2 - CODE

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import chi2, SelectKBest, f_classif, f_regression
from sklearn.preprocessing import LabelBinarizer, LabelEncoder, OrdinalEncoder
from sklearn.feature_selection import mutual_info_classif
from sklearn import feature_selection
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier



credit_card = pd.read_csv("/Users/ozgeguney/.spyder-py3/DataMining/
fraudTest.csv")


print(credit_card.describe(include="all").loc[:,"is_fraud"])

credit_card_class1 =  credit_card[credit_card.is_fraud==1]
print(credit_card_class1.shape)

print(credit_card.is_fraud.value_counts())


cc_majority = credit_card[credit_card.is_fraud==0]
cc_minority = credit_card[credit_card.is_fraud==1]


from sklearn.utils import resample
cc_majority_downsampled = resample(cc_majority,
                                   replace=False,
                                   n_samples=2145)



cc_balanced = pd.concat([cc_minority, cc_majority_downsampled])
print(cc_balanced.is_fraud.value_counts())
```

```
cc_balanced['trans_date_trans_time'] =
pd.to_datetime(cc_balanced['trans_date_trans_time'])

cc_balanced['trans_hour'] = cc_balanced['trans_date_trans_time'].dt.hour
#deriving 'day of the week'
cc_balanced['trans_day_of_week'] =
cc_balanced['trans_date_trans_time'].dt.day_name()
#deriving 'year_month'
cc_balanced['trans_year_month'] =
cc_balanced['trans_date_trans_time'].dt.month_name()

cc_balanced['age'] = np.round((cc_balanced['trans_date_trans_time'] -
                  pd.to_datetime(cc_balanced['dob']))/np.timedelta64(1, 'Y'))

#dropping unique variables
cc_balanced.drop(['trans_date_trans_time','unix_time',
'trans_num','merch_long','merch_lat', 'cc_num', 'lat', 'long',
'first', 'last', 'dob','street'] , axis=1, inplace=True)


f, ax = plt.subplots(2,2)
f.set_size_inches(12,10)
sns.boxplot(y="age", x="is_fraud", data= cc_balanced, ax = ax[0,0]);
sns.boxplot(y="amt", x="is_fraud", data= cc_balanced, ax = ax[0,1]);
sns.boxplot(y="trans_hour", x="is_fraud", data= cc_balanced, ax = ax[1,0]);
sns.boxplot(y="trans_year_month", x="is_fraud", data= cc_balanced, ax = ax[1,1]);


X1 = cc_balanced.loc[:,'merchant':'age']
X1 = cc_balanced.select_dtypes(include=np.number) # select numeric columns

y1 = cc_balanced.loc[:,'is_fraud']
selector = SelectKBest(f_classif, k=6)
selector.fit(X1, y1)


sorted_idx = np.argsort(selector.scores_)[::-1]
sorted_vals = np.sort(selector.scores_)[::-1]

d = {"features":X1.columns[sorted_idx], "values":sorted_vals, "p-
values":selector.pvalues_[sorted_idx]}
df = pd.DataFrame(d)
print(df)


le = LabelEncoder()
```

```
X2 = cc_balanced.loc[:,'merchant':'age']
X2 = X2.select_dtypes(include=object) # select string columns
X2 = X2.apply(LabelEncoder().fit_transform)
y2 = le.fit_transform(cc_balanced.loc[:,'is_fraud'])
chi2, pval=feature_selection.chi2(X2, y2)

sorted_idx2 = np.argsort(chi2)[::-1]
sorted_vals2 = np.sort(chi2)[::-1]

d2 = {"features":X2.columns[sorted_idx2], "values (sklearn)":sorted_vals2, "p-
values":pval[sorted_idx2]}
df2 = pd.DataFrame(d2)
print(df2)


X = cc_balanced.loc[:,[  'state',
       'city_pop', 'trans_day_of_week']]
y = cc_balanced.loc[:,'is_fraud']

X = pd.get_dummies(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
X_train = X_train.fillna(X_train.mean())
X_test = X_test.fillna(X_test.mean())


knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('knn     '+classification_report(y_test,pred))


clf1 = DecisionTreeClassifier()
clf2 = AdaBoostClassifier(n_estimators=200)
clf3 = RandomForestClassifier(n_estimators=100, bootstrap=True)

clf1.fit(X_train, y_train);
clf2.fit(X_train, y_train);
clf3.fit(X_train, y_train);

y_pred1 = clf1.predict(X_test)
y_pred2 = clf2.predict(X_test)
y_pred3 = clf3.predict(X_test)
print('DecisionTreeClassifier'+classification_report(y_test,y_pred1))
print('AdaBoostClassifier'+classification_report(y_test,y_pred2))
print('RandomForestClassifier'+classification_report(y_test,y_pred3))
```

```
from sklearn.metrics import confusion_matrix

#Generate the confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred1)
cf_matrix2 = confusion_matrix(y_test, y_pred2)
cf_matrix3 = confusion_matrix(y_test, y_pred3)

print(cf_matrix)
print(cf_matrix2)
print(cf_matrix3)



ax = sns.heatmap(cf_matrix3, annot=True, cmap='Blues')

ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```