Exploratory Data Analysis for Machine Learning Course

IBM Machine Learning
Professional Specialization

August 2021

PAULINA KOSSOWSKA

Introduction

The goals here was to find a dataset and proceed some data cleaning, data exploration, data visualization together with some feature engineering.

I chose a Kaggle dataset contains information about credit card fraud transaction.

I split my Jupiter notebook into following parts:

1. Introduction
2. Import libraries
3. Load dataset
4. EDA
5. Scaling Amount and Time
6. Hypothesis Testing
7. Limitations

Import libraries

In this section I imported all libraries I was used in this project.

Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import plotly.figure_factory as ff
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
from scipy.stats import shapiro
%matplotlib inline
```

Load dataset

In the next step using basic pandas features **read_csv** I open a csv file and load it into data frame named **df**.

The dataset contains transactions made by credit cards in two days in September 2013 by European cardholders. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. 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, this feature can be used for example-dependant cost-sensitive learning.

Feature **'Class'** is the response variable and it takes value 1 in case of fraud and 0 otherwise.

```
df = pd.read_csv('creditcard.csv')
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.

5 rows × 31 columns

EDA

```
print('This dataset contains:  {} rows and {} columns.'.format(df.shape[0], df.shape[1]))
```

```
This dataset contains:  284807 rows and 31 columns.
```

Using shape() method gave me answer to the questions about how many rows and columns contains my dataset. I also use at the beginning of EDA methods: info() and isnull() to check for missing values.

What I found out is a fact that all variables were numerical and there were no missing data in this dataset.

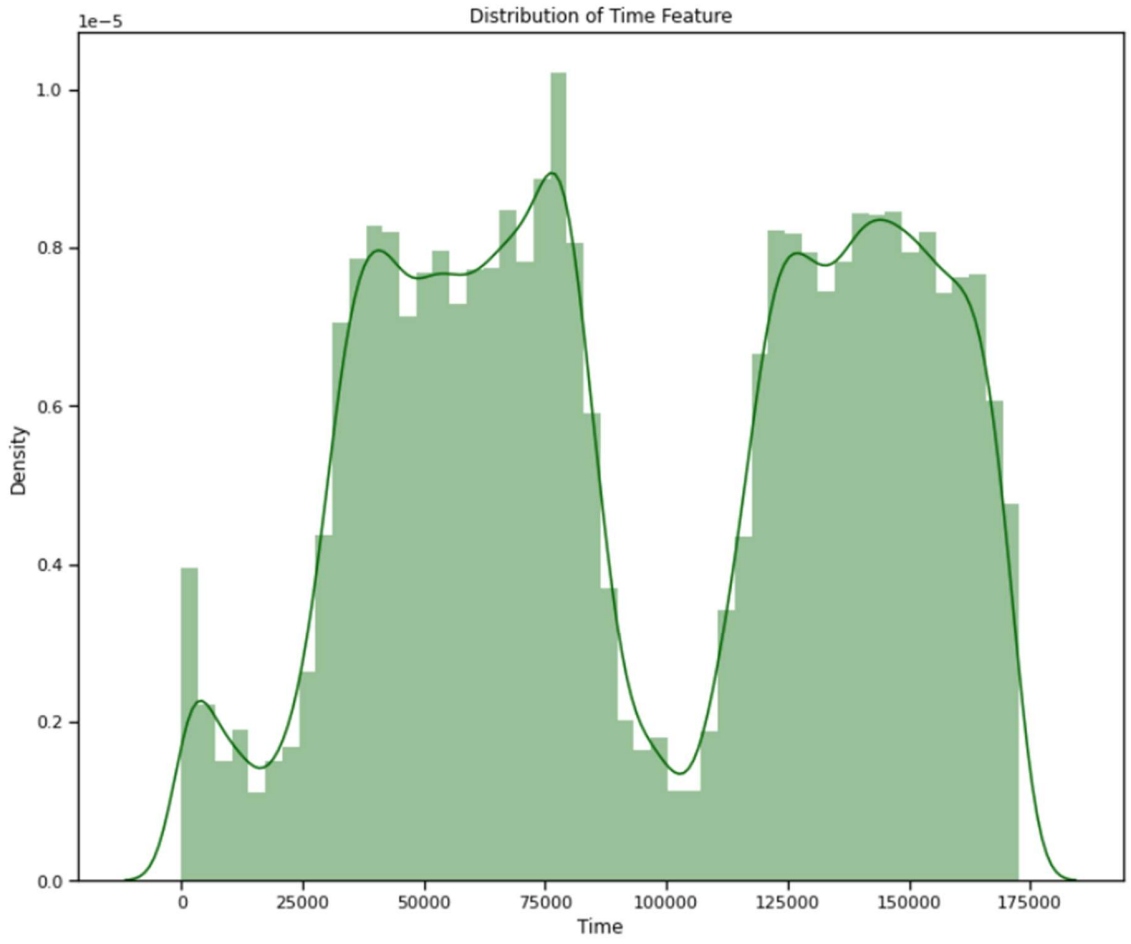
Then, I would like to take closer look at two variables: Time and Amount.

```
col_2 = ['Time', 'Amount']
df.loc[:, col_2].describe()
```

	Time	Amount
count	284807.000000	284807.000000
mean	94813.859575	88.349619
std	47488.145955	250.120109
min	0.000000	0.000000
25%	54201.500000	5.600000
50%	84692.000000	22.000000
75%	139320.500000	77.165000
max	172792.000000	25691.160000

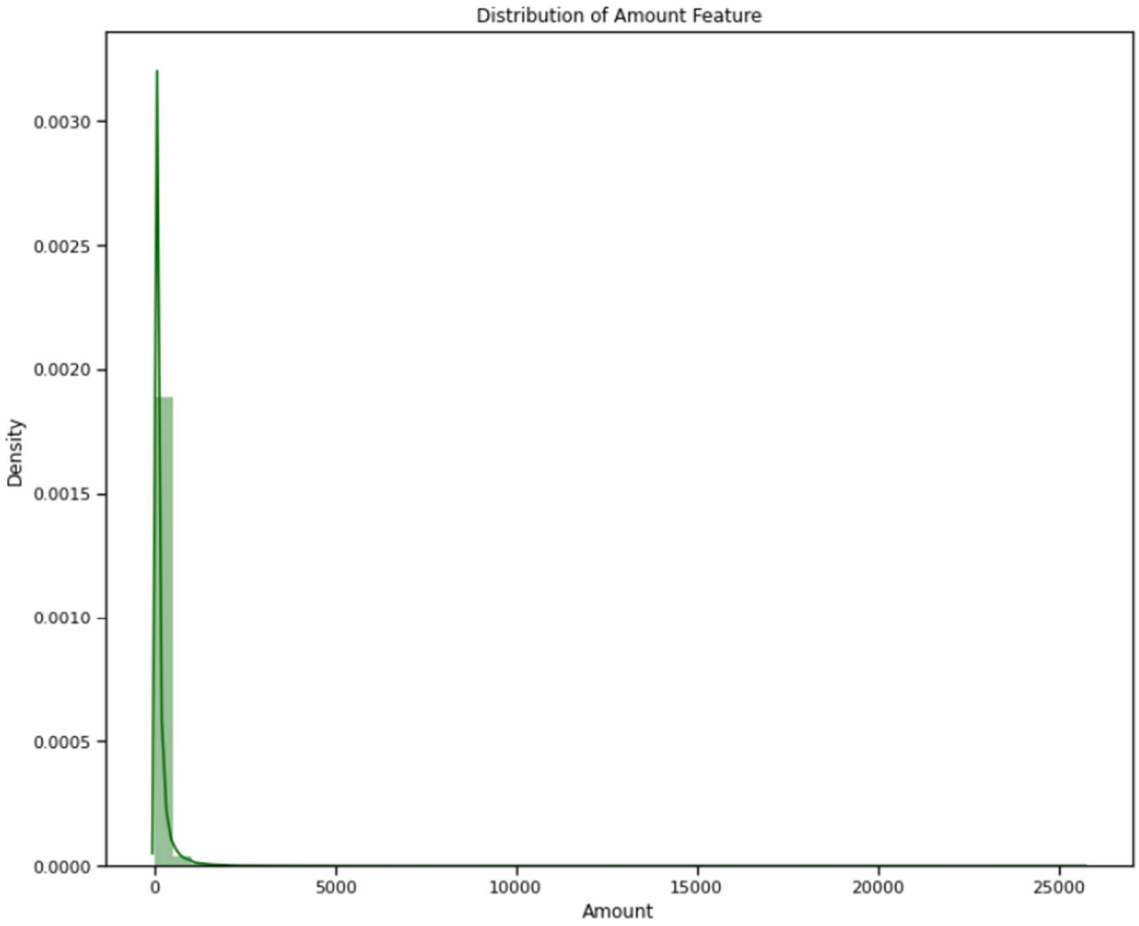
Firstly, we can notice that the range of transaction value is huge. The minimum value per transaction is 0 and the maximum is 25691 (amount is dollars). Secondly, is also clear that the average amount per transaction is 88 dollars.

Feature Time contains the seconds elapsed between each transaction and the first transaction in the dataset but to better understand this variable I will visualize it.

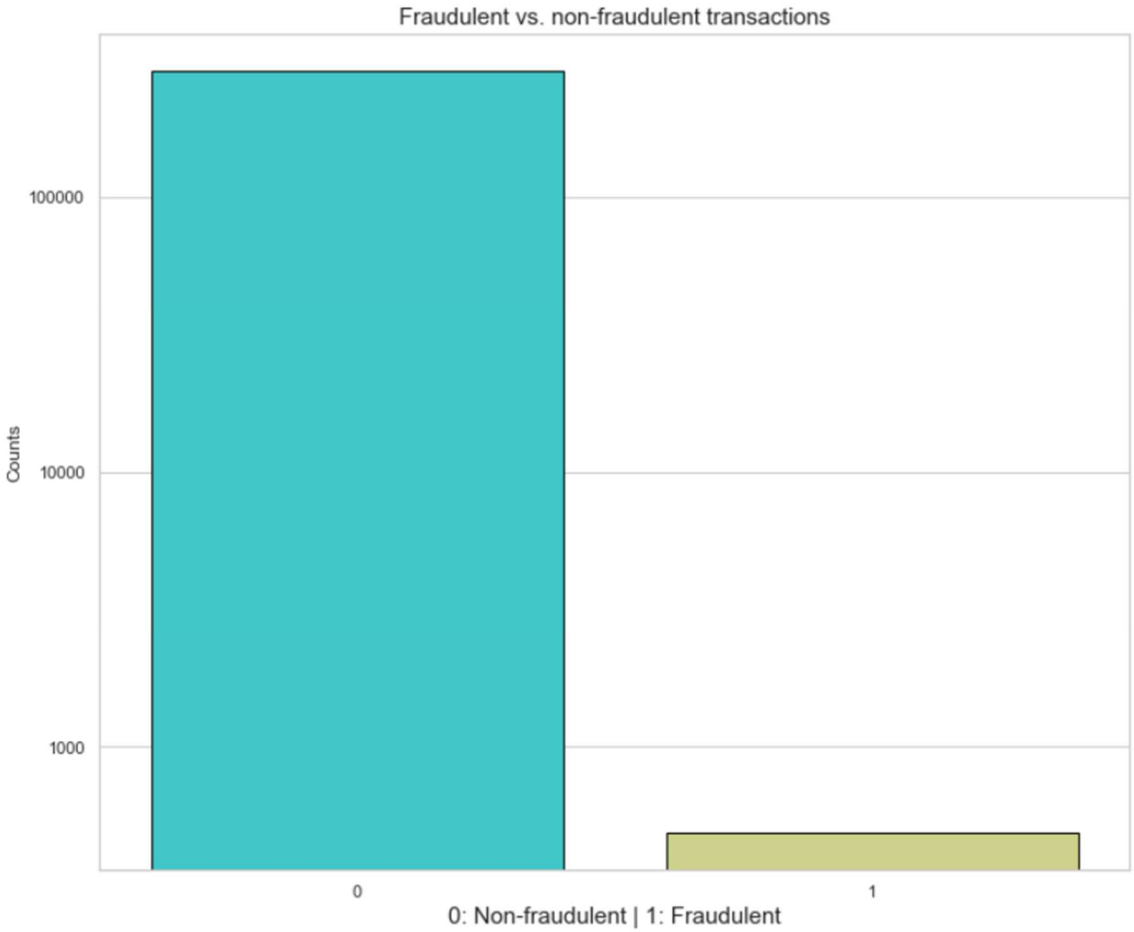


Based on graph provided we can clearly see how distribution of Time feature looks like. It is clear that is a bimodal distribution with two peaks.

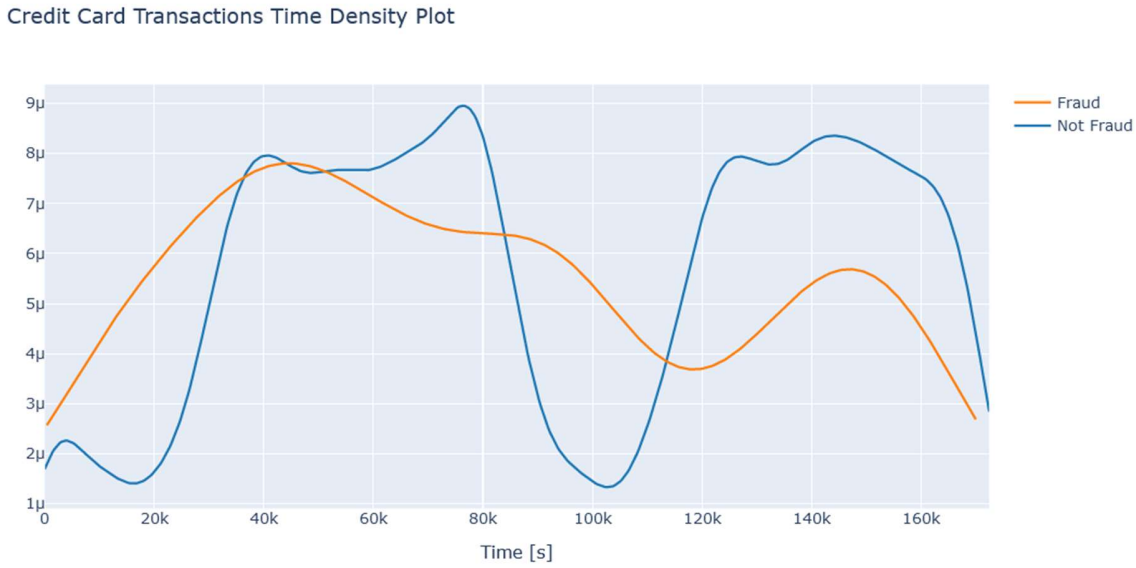
The same graph for Amount feature shows that this is right-skewed distribution where mean > median. It is also clear for us that in this variable we have to deal with outliers.



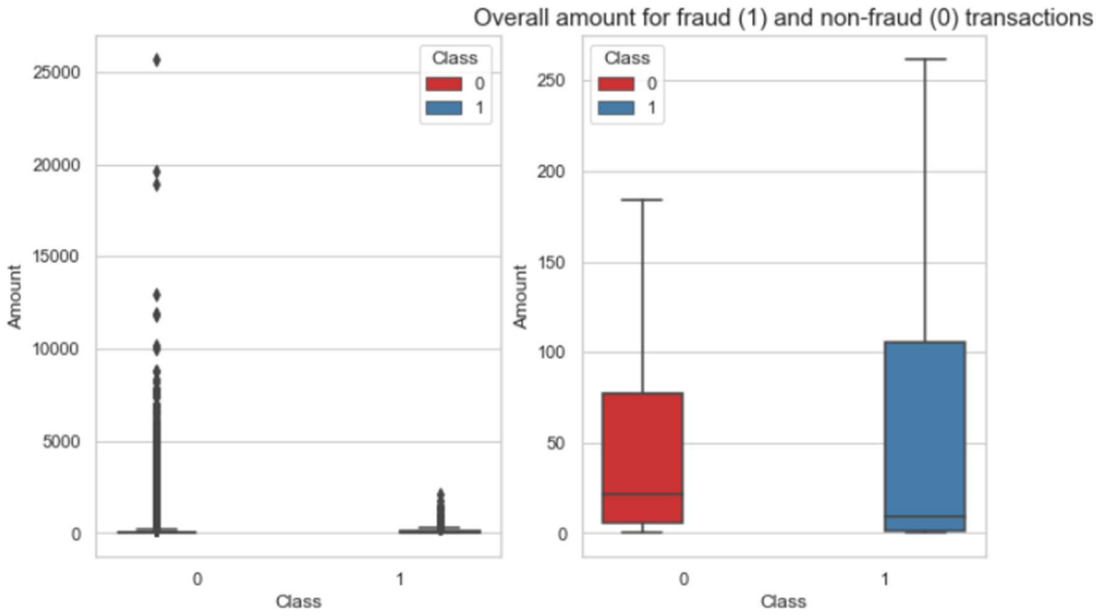
To better understand how the proportion between fraudulent and non-fraudulent transaction looks like I decided to visualize it by barplot graph. To make data more readable I used a logarithmic scale on y axis to at least point out the total amount of fraudulent transactions which are almost invisible using default y axis scale.



Another useful visualization I decided to show is a line graph which contains information's about fraudulent and non-fraudulent transaction over time.



The similar splitting, I made for an Amount column showing this time a boxplot for a fraudulent and non-fraudulent transaction.



To quickly understand what above graph represent I provided some additional information using describe() method.

```
class_0.describe().to_frame()
```

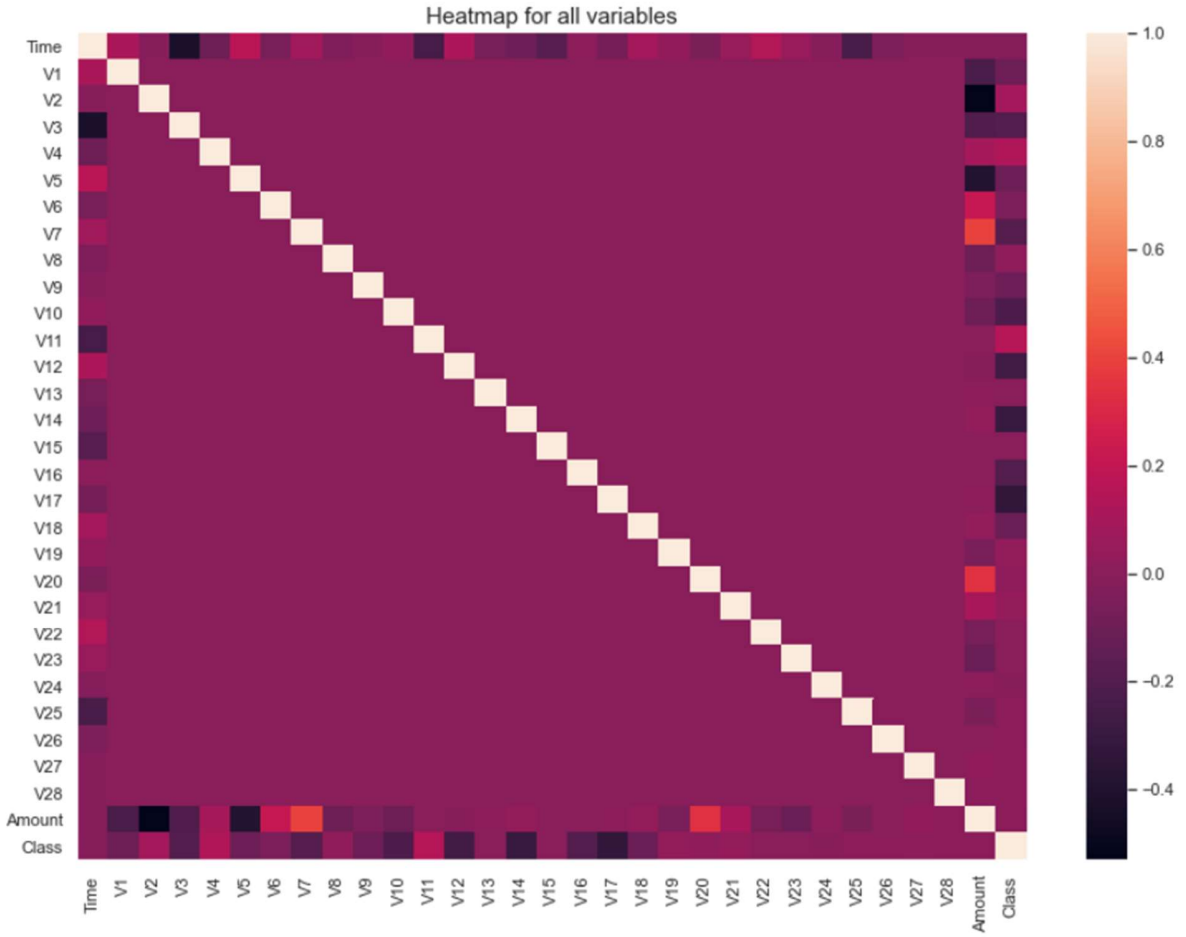
	Amount
count	284315.000000
mean	88.291022
std	250.105092
min	0.000000
25%	5.650000
50%	22.000000
75%	77.050000
max	25691.160000

```
class_1.describe().to_frame()
```

	Amount
count	492.000000
mean	122.211321
std	256.683288
min	0.000000
25%	1.000000
50%	9.250000
75%	105.890000
max	2125.870000

The real transactions (non-fraudulent) have a larger mean and median, together with higher Q1, but smaller Q3 and have also larger outliers.

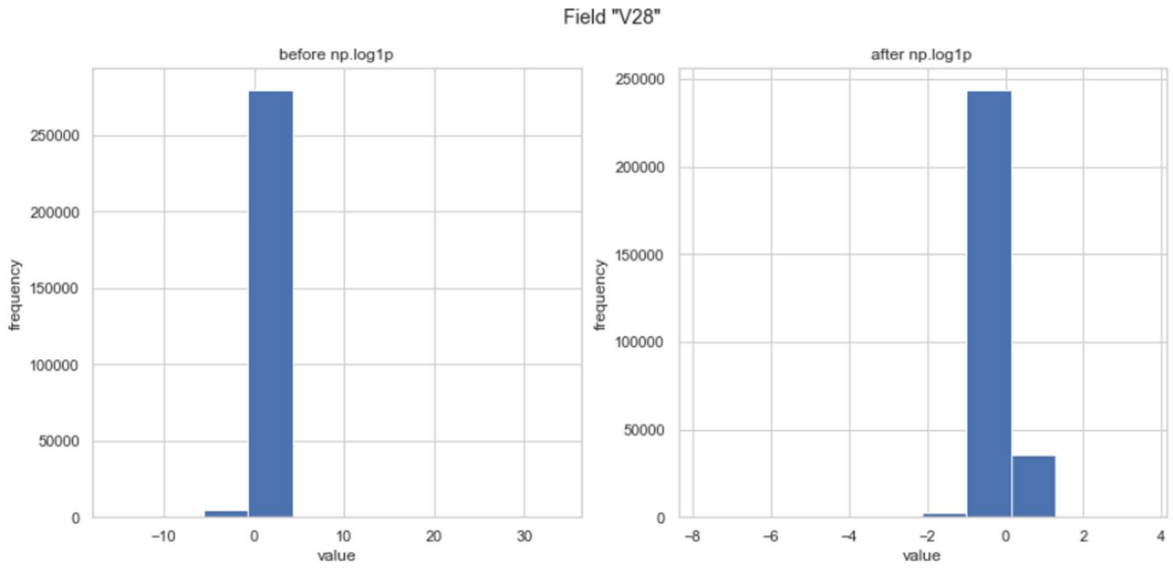
Next step concerned to correlation. To check it I used a corr() method together with heatmap visualization. This give me an overview how variables are correlated and what relationship I can find between them.



My last step in this section was checking how data were skewed. To do that I use a skew() method. I chose a float numerical variable and create a variable which describe a skew limit I decided to chose equal to 0.75. All variables above this limit are represent by below table.

	Skew
Amount	16.977724
V28	11.192091
V21	3.592991
V7	2.553907
V6	1.826581
V10	1.187141
V16	-1.100966
V27	-1.170209
V14	-1.995176
V20	-2.037155
V3	-2.240155
V12	-2.278401
V5	-2.425901
V1	-3.280667
V17	-3.844914
V2	-4.624866
V23	-5.875140
V8	-8.521944

Then, using log transformation I change the skew of this variables into variables with more normal distribution.



Scaling Amount and Time

I decided to use StandardScaler() method from sklearn.preprocessing to handle with Time and Amount columns. Those two variables are original and wasn't changed by PCA during preparing dataset phase.

```
scaler = StandardScaler()
scaler2 = StandardScaler()

#scaling time
scaled_time = scaler.fit_transform(df[['Time']])
flat_list1 = [item for sublist in scaled_time.tolist() for item in sublist]
scaled_time = pd.Series(flat_list1)
```

```
#scaling amount
scaled_amount = scaler2.fit_transform(df[['Amount']])
flat_list2 = [item for sublist in scaled_amount.tolist() for item in sublist]
scaled_amount = pd.Series(flat_list2)
```

```
#concatenating newly created columns w original df
df = pd.concat([df, scaled_amount.rename('scaled_amount'), scaled_time.rename('scaled_time')], axis=1)
df.sample(5)
```

Then, by dropping original columns my dataframe contains only scaled time and amount variables.

V7	V8	V9	V10	...	V22	V23	V24	V25	V26	V27	V28	Class	scaled_amount	scaled_time
.424307	0.347630	1.529731	0.925246	...	0.668112	-0.197363	0.721599	0.239269	0.084573	0.599710	0.328658	0	-1.146549	1.107902
.558234	-0.106938	-0.492603	-0.008616	...	0.461473	-0.150041	0.386209	-0.198937	-0.316247	0.345135	0.204319	0	-0.507437	-0.490731
.292652	0.320868	-1.520382	0.610235	...	-1.290535	0.195098	-0.515501	-0.705176	-0.289697	0.095357	0.087193	0	-0.206189	-1.339913
NaN	0.260783	-0.542465	0.559818	...	1.104509	-0.192460	-0.776698	0.424044	0.016740	0.047137	0.001410	0	-1.075068	-0.483719
.279144	0.617682	-0.319656	-0.965074	...	-1.289110	0.131891	0.600019	0.397099	-0.832160	0.175303	0.066511	0	0.877063	1.008804

Hypothesis Testing

In this part of task I figured out three different hypothesis which could be investigated and only look closer at one of them.

One of hypothesis I can try to answer is if your data has a Gaussian distribution.

H0: the sample has a Gaussian distribution.

H1: the sample does not have a Gaussian distribution.

I performed this hypothesis using Shapiro-Wilk Test.

```
# Shapiro-Wilk Test

mask = df.dtypes == np.float
float_cols = df.columns[mask]
alpha = 0.05 # p-value

stat, p = shapiro(df[float_cols])

print('stat=%.3f, p=%.3f' % (stat, p))
if p > alpha:
    print('Sample looks Gaussian (fail to reject H0)')
else:
    print('Sample does not look Gaussian (reject H0)')
```

```
stat=0.159, p=0.000
Sample does not look Gaussian (reject H0)
```

Another two hypothesis which could be further investigated include:

1. Check if two samples are related:
    a. H0: the two samples are independent
    b. H1: there is a dependency between the samples

2. Tests whether the means of two independent samples are significantly different:
    a. H0: the means of the samples are equal
    b. H1: the means of the samples are unequal

Limitations

One thing important to notice here is a fact that data in this dataset are highly imbalanced.

To work with future hypothesis testing or machine learning algorithms it seems to be a good idea to choose method which will help us with this imbalanced. One of the idea here is to use SMOTE (Synthetic Minority Over-sampling Technique) is an oversampling method.