credit-card-fraud-detection-part1-eda

July 27, 2024

Credit Card Fraud Detection

Part 1. Exploratory Data Analysis (EDA)

Yatharth Gautam

1 Contents

- Introduction
- Visuaizing individual features
- Relationships among the features

1.0.1 NOTE:

Section 2 and Section 3, which contain a large number of *plotly* and *seaborn* diagrams, used for visualizing distributions of the feature variables in the dataset, as well as their interrelationships, may take a minute to load.

2 1. Introduction

[1]:
```
# Importing necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import plotly.express as px
```

2.1 Data

Source: https://www.kaggle.com/mlg-ulb/creditcardfraud

The dataset contains information on the transactions made using credit cards by European cardholders, in two particular days of September 2013. It presents a total of 284807 transactions, of which 492 were fraudulent. Clearly, the dataset is highly imbalanced, the positive class (fraudulent transactions) accounting for only 0.173% of all transactions.

For a particular transaction, the feature **Time** represents the time (in seconds) elapsed between the transaction and the very first transaction, **Amount** represents the amount of the transaction and **Class** represents the status of the transaction with respect to authenticity. The class of an authentic (resp. fraudulent) transaction is taken to be 0 (resp. 1). Rest of the variables (**V1** to **V28**) are obtained from principle component analysis (PCA) transformation on original features that are not available due to confidentiality.

[2]:
```python
# The dataset

data = pd.read_csv('../input/creditcardfraud/creditcard.csv')
data
```

[2]:
```
            Time         V1         V2        V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
…             …          …          …         …         …         …
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  …       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  … -0.018307  0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  … -0.225775 -0.638672
2       1.800499  0.791461  0.247676 -1.514654  …  0.247998  0.771679
3       1.247203  0.237609  0.377436 -1.387024  … -0.108300  0.005274
4       0.095921  0.592941 -0.270533  0.817739  … -0.009431  0.798278
…            …         …         …         …  …        …         …
284802 -2.606837 -4.918215  7.305334  1.914428  …  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  …  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  …  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  …  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  …  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
…            …         …         …         …         …         …       …
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
```

```
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0           0
1           0
2           0
3           0
4           0
…         …
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]
```

[3]:
```python
# Statistical descriptions of the features

features = data.drop(['Class'], axis = 1)
features.describe()
```

[3]:
```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.213481e-16 -2.406331e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       …           V20           V21           V22           V23  \
count  …  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   …  6.406204e-16  1.654067e-16 -3.568593e-16  2.578648e-16
std    …  7.709250e-01  7.345240e-01  7.257016e-01  6.244603e-01
min    … -5.449772e+01 -3.483038e+01 -1.093314e+01 -4.480774e+01
```

```
25%    … -2.117214e-01 -2.283949e-01 -5.423504e-01 -1.618463e-01
50%    … -6.248109e-02 -2.945017e-02  6.781943e-03 -1.119293e-02
75%    …  1.330408e-01  1.863772e-01  5.285536e-01  1.476421e-01
max    …  3.942090e+01  2.720284e+01  1.050309e+01  2.252841e+01

                V24           V25           V26           V27           V28  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   4.473266e-15  5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-16
std    6.056471e-01  5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01
min   -2.836627e+00 -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01
25%   -3.545861e-01 -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02
50%    4.097606e-02  1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02
75%    4.395266e-01  3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02
max    4.584549e+00  7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01

              Amount
count  284807.000000
mean       88.349619
std       250.120109
min         0.000000
25%         5.600000
50%        22.000000
75%        77.165000
max     25691.160000

[8 rows x 30 columns]
```

2.2 Objectives of the project

2.2.1 Primary objective:

Classification of transactions as authentic or fraudulent. To be prcise, given the data on **Time**, **Amount** and transformed features **V1** to **V28** for a particular transaction, our goal is to correctly classify the transaction as **authentic** or **fraudulent**. We employ different techniques to build classification models and compare them by various evaluation metrics.

2.2.2 Secondary objectives:

Answering the following questions using machine learning and statistical tools and techniques.

- When a fraudulent transaction is made, is it followed soon by one or more such fraudulent transactions? In other words, do the attackers make consecutive fraudulent transactions in a short span of time?
- Is the amount of a fraudulent transaction generally larger than that of an authentic transaction?
- Is there any indication in the data that fraudulent transactions occur at high-transaction period?

- It is seen from the data that the number of transactions are high in some time intervals and low in between. Does the occurance of fraudulent transactions related to these time intervals?
- There are a few time-points which exhibits high number of fraud transactions. Is it due to high number of total transactions or due to some other reason?

In this part we carry out exploratory data analysis for the features in the dataset.

3 2. Visualizing individual features

3.1 Class

First we analyze the feature which is the main object of the study: The class variable, which indicates if a particular transaction is authentic or fraudulent.

[4]:
```
# Splitting of the data by authenticity of transactions

data_authentic = data[data['Class'] == 0] # authentic transactions only
data_fraud = data[data['Class'] == 1] # fraud transactions only

# Class frequencies

class_label = ['Authentic', 'Fraud']
class_frequency = [len(data_authentic), len(data_fraud)]

fig1 = px.pie(values = class_frequency,
             names = class_label,
             title = 'Frequency comparison of authentic and fraudulent␣
 ↪transactions',
             template = 'ggplot2'
            )
fig1.show()
```

It is evident that the data is extremely imbalanced with authentic transactions being the majority class and fraudulent transactions being the minority class. Next we analyze the frequency of transactions made over time elapsed starting from the first transaction.

3.2 Time

[5]:
```
# Transaction frequency over time

fig1 = px.histogram(data,
                   x = 'Time',
                   nbins = 200,
                   title = 'Distribution of transactions over time',
                   template = 'ggplot2'
                  )
fig1.show()
```

Observation: The number of transactions are particularly high in certain time intervals and low in between.

[6]:
```
# Histogram for fraudulent transactions

fig1 = px.histogram(data_fraud,
                    x = 'Time',
                    nbins = 200,
                    title = 'Distribution of fraudulent transactions over time',
                    template = 'ggplot2'
                   )
fig1.show()
```

Observation: There are certain spikes in the data that indicates high number of fraud transactions at certain time points.

Next we visualize the distribution of transaction amount. It is seen from the data that this feature is positively skewed to a great extent. Hence we use log-scale in the y-axis to produce a nondegenerate visualization of the same.

3.3 Amount

[7]:
```
# Transaction amount

fig1 = px.histogram(data,
                    x = 'Amount',
                    nbins = 200,
                    title = 'Distribution of transaction amount',
                    #log_y = True,
                    template = 'ggplot2'
                   )
fig1.show()

fig2 = px.histogram(data,
                    x = 'Amount',
                    nbins = 200,
                    title = 'Distribution of transaction amount on logarithmic␣
 ↪scale',
                    log_y = True,
                    template = 'ggplot2'
                   )
fig2.show()
```

The high positive skewness even after taking the log-scale motivates us to map the amount data using log transformation.

[8]:
```
# Transaction amount after log transformation

np.seterr(divide = 'ignore')
```

```
#np.seterr(divide = 'warn')

df = pd.DataFrame(columns = ['Class','log_amount'])
df['Class'] = data['Class']
df['log_amount'] = np.log2(data['Amount'])

fig1 = px.histogram(df,
                   x = 'log_amount',
                   nbins = 200,
                   title = 'Distribution of transaction amount after log␣
 ↪transformation',
                   #log_y = True,
                   template = 'ggplot2'
                  )
fig1.show()
```

Since this gives a more symmetric output, we are motivated to work with this transformed amount data, from which the original amount data can easily be converted back to.

[9]:
```
# Visualizations of authentic and fraudulent transactions after log␣
 ↪transformation

class_list = list(data['Class'])
fraud_status = []
for i in range(len(class_list)):
    fraud_status.append(bool(class_list[i]))

fig1 = px.violin(df,
             x = 'Class',
             y = 'log_amount',
             color = fraud_status,
             title = 'Distribution of Amount for Authentic and Fraudulent␣
 ↪transactions after log transformation',
             template = 'ggplot2'
            )
fig1.show()
```

```
/opt/conda/lib/python3.10/site-packages/plotly/express/_core.py:2065:
FutureWarning:

When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
```

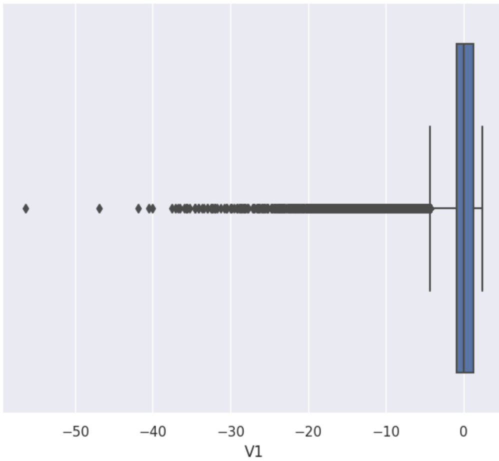
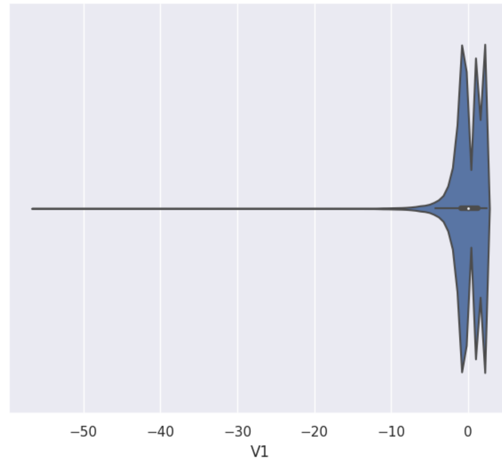
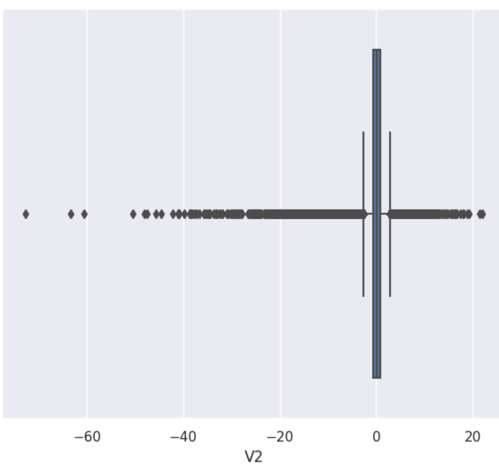
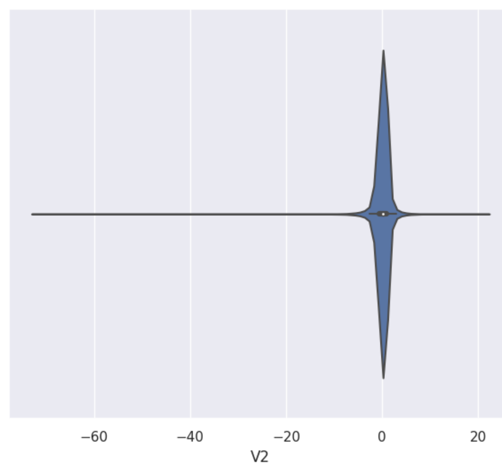
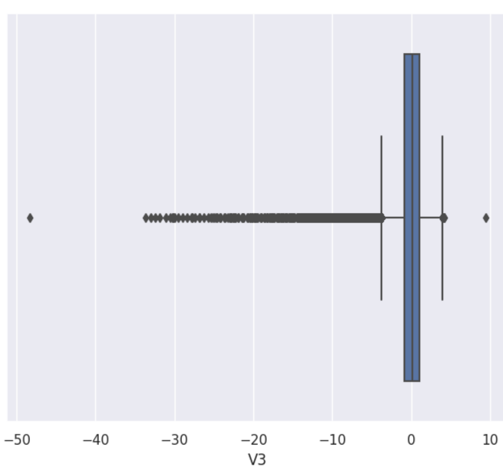
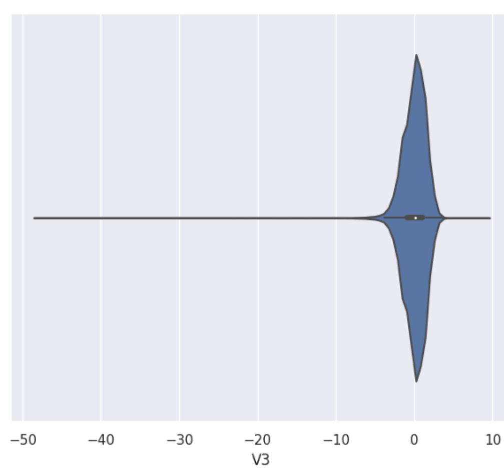
It is clear from the plots that most of the large-amount transactions are authentic, which maybe caused by the extra security measures given to high-amount transactions in form of multiple passwords and OTPs.

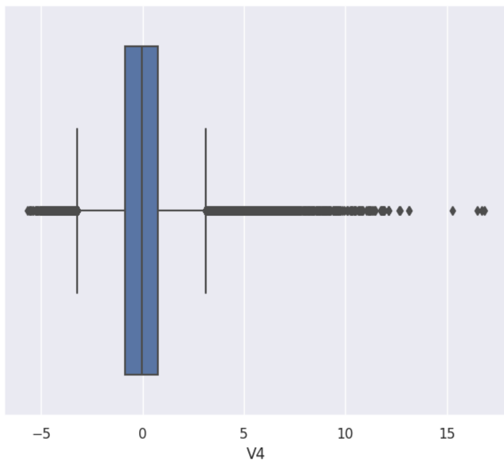
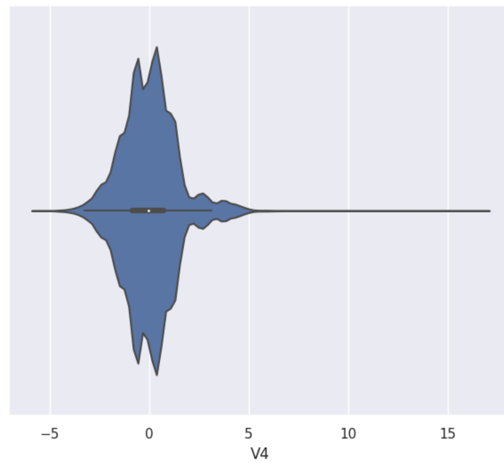
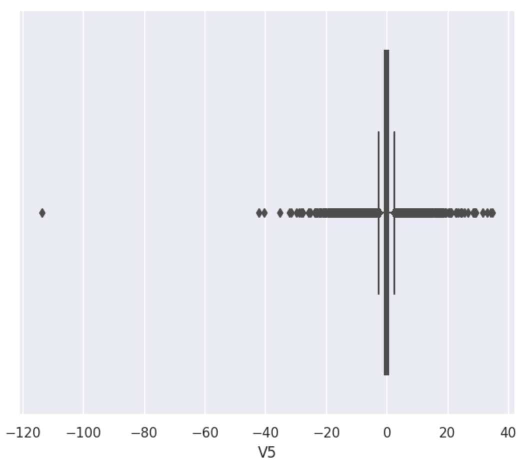
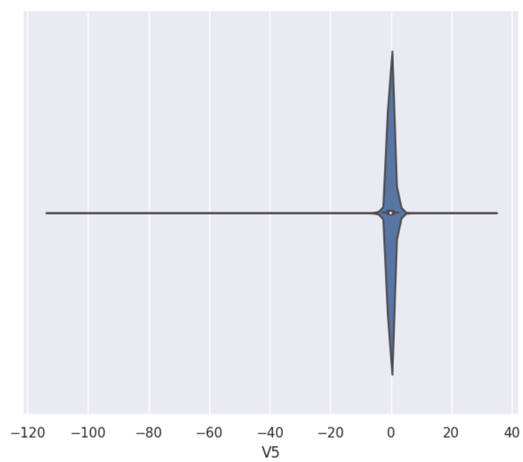
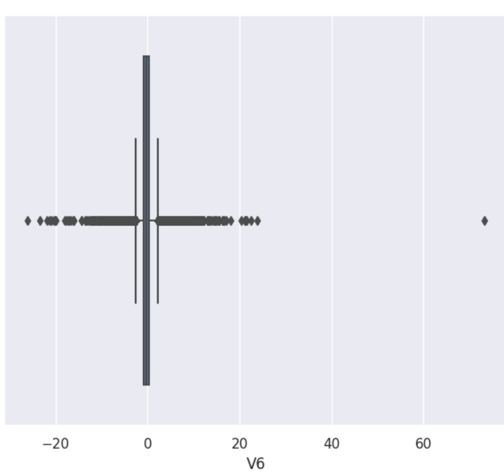
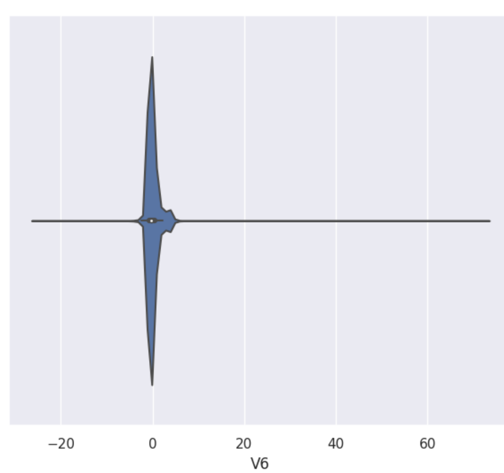
3.4 V1-V28

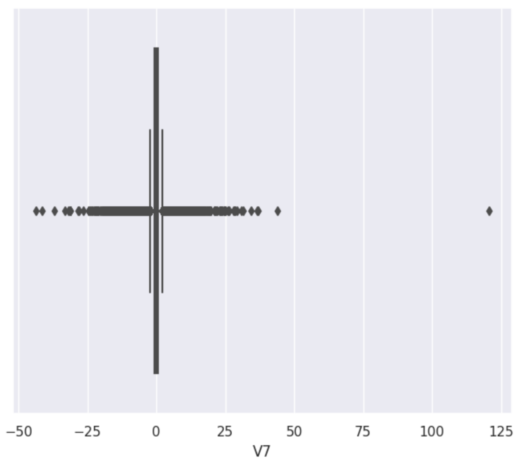
[10]:
```
# Function to print histogram of a chosen feature

def hist(data, feature):
    fig1 = px.histogram(data,
                        x = feature,
                        nbins = 200,
                        title = 'Distribution of {}'.format(feature),
                        template = 'ggplot2'
                       )
    fig1.show()

# Function to print boxplot and violinplot of a chosen feature

def box_violin(data, feature):
    fig, (ax1, ax2) = plt.subplots(1,2, figsize=(16.1, 6))
    sns.boxplot(x = data[feature], ax = ax1)
    sns.violinplot(x = data[feature], ax = ax2)
    plt.show()

# Function to combine the above two functions

def eda(data, feature):
    hist(data, feature)
    box_violin(data, feature)
```

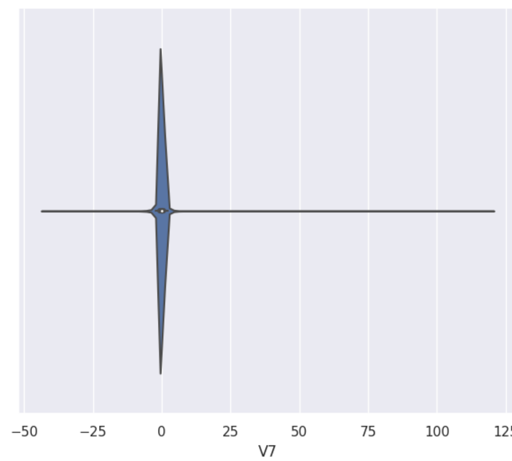
[11]:
```
# List of features obtained by PCA transformation

features_pca = list(data.columns)
features_pca.remove('Time')
features_pca.remove('Amount')
features_pca.remove('Class')
```

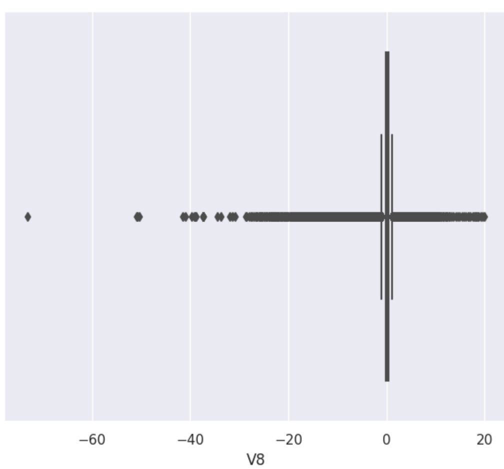
[12]:
```
# Visualizations and statistical descriptions of the features obtained by PCA␣
↪transformation

for feature in features_pca:
    eda(data, feature)
```

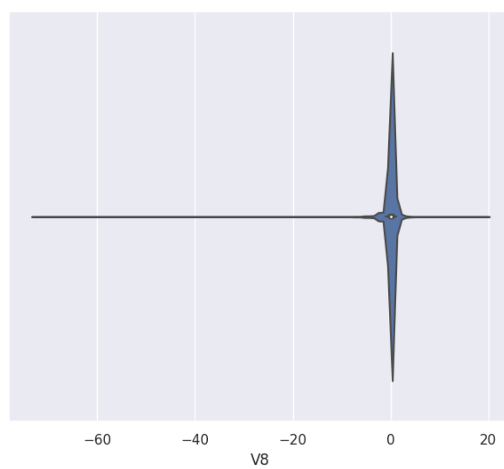

−50 −40 −30 −20 −10 0
V1

−50 −40 −30 −20 −10 0
V1

−60 −40 −20 0 20
V2

−60 −40 −20 0 20
V2

−50 −40 −30 −20 −10 0 10
V3

−50 −40 −30 −20 −10 0 10
V3

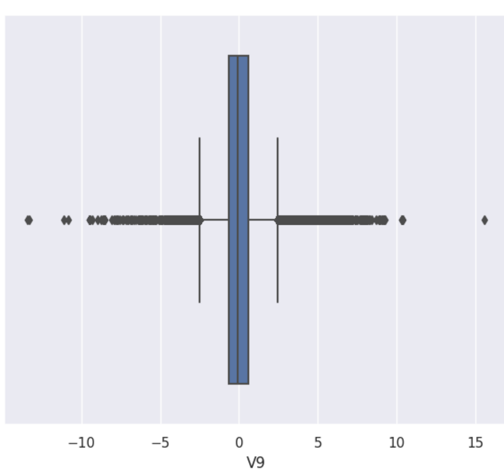

−5 0 5 10 15
V4

−5 0 5 10 15
V4

−120 −100 −80 −60 −40 −20 0 20 40
V5

−120 −100 −80 −60 −40 −20 0 20 40
V5

−20 0 20 40 60
V6

−20 0 20 40 60
V6

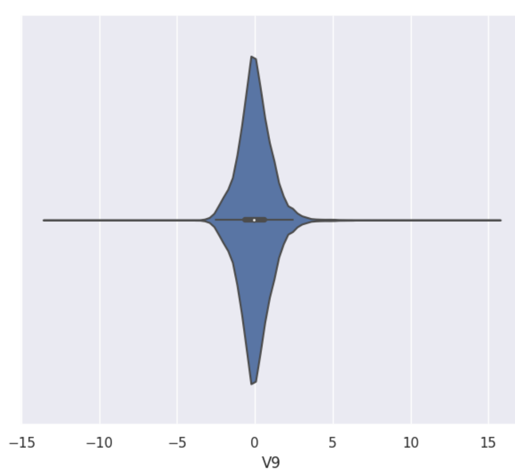

−50 −25 0 25 50 75 100 125
V7

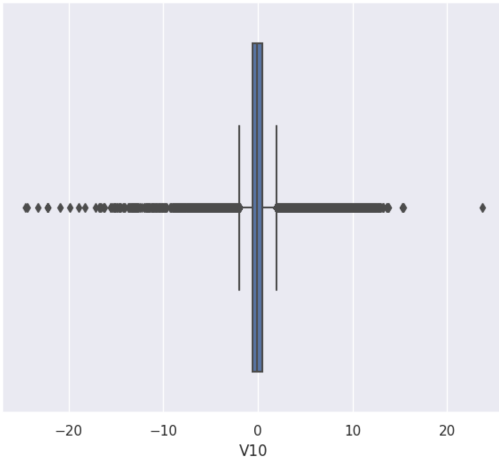
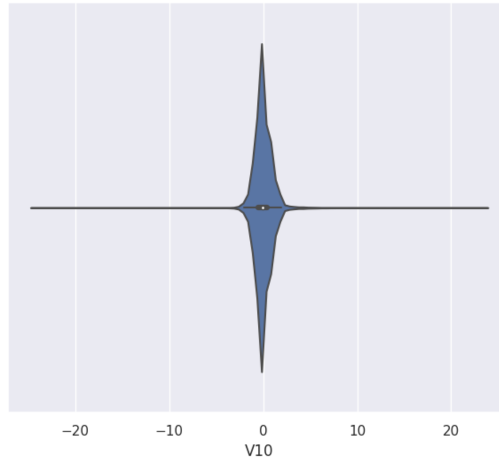
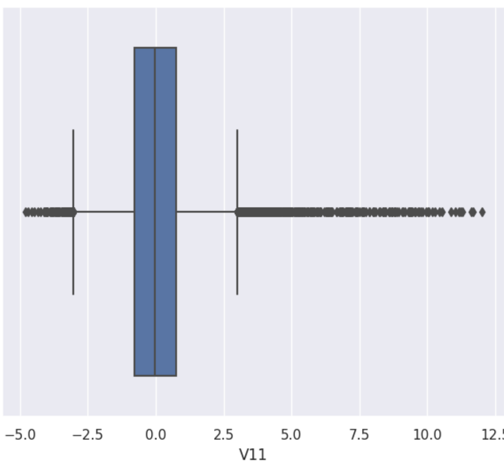
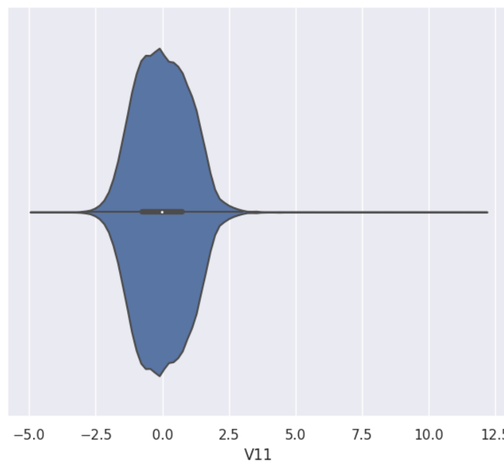
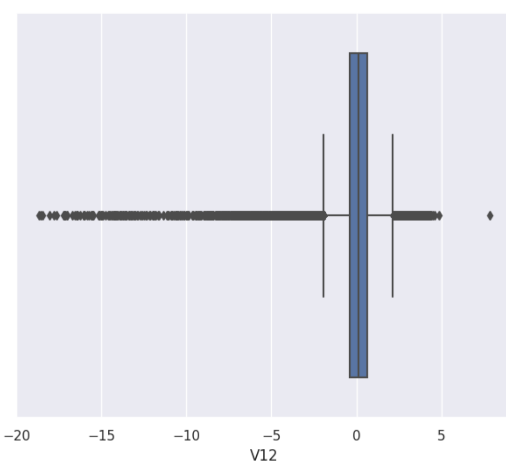
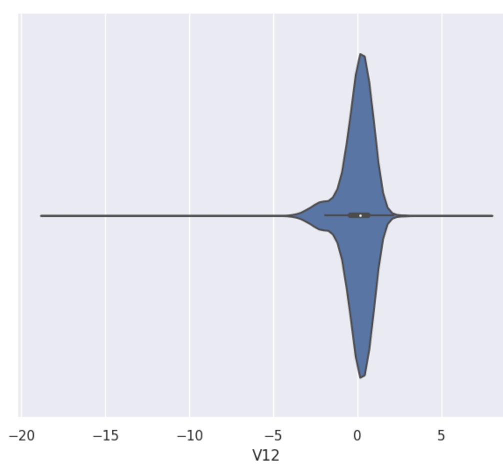

−50 −25 0 25 50 75 100 125
V7

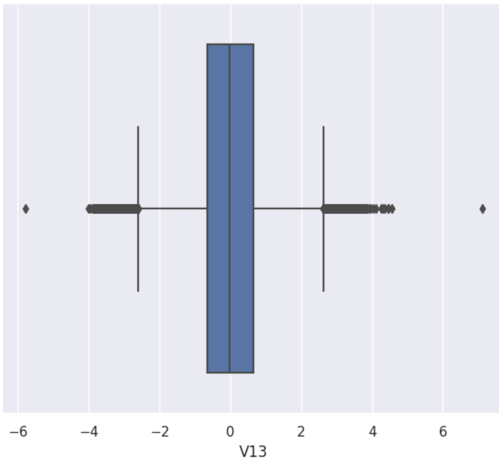

−60 −40 −20 0 20
V8

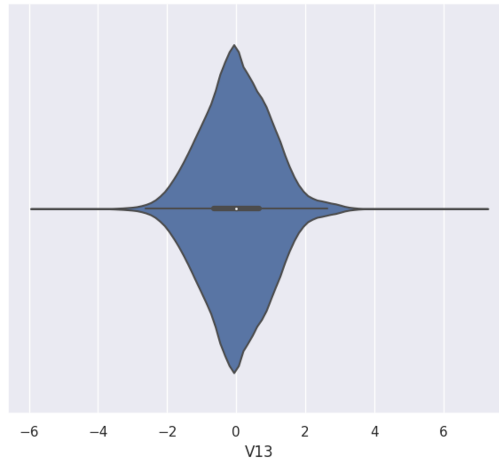

−60 −40 −20 0 20
V8

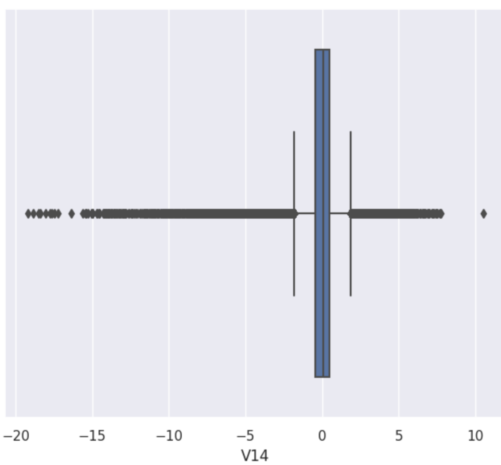

−10 −5 0 5 10 15
V9

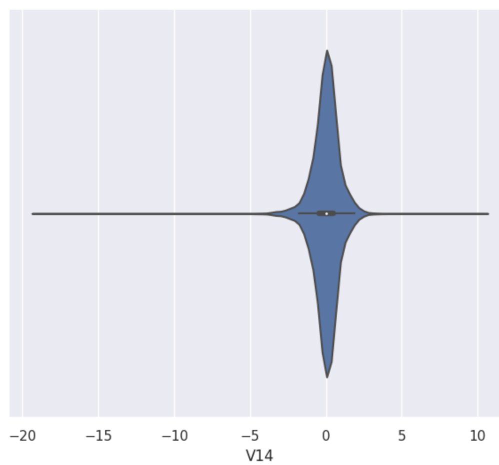

−15 −10 −5 0 5 10 15
V9

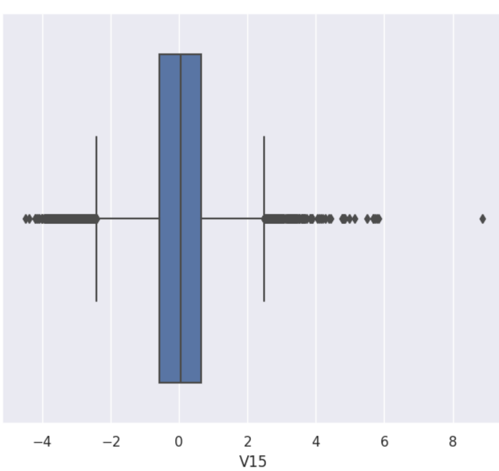

−20 −10 0 10 20
V10

−20 −10 0 10 20
V10

−5.0 −2.5 0.0 2.5 5.0 7.5 10.0 12.5
V11

−5.0 −2.5 0.0 2.5 5.0 7.5 10.0 12.5
V11

−20 −15 −10 −5 0 5
V12

−20 −15 −10 −5 0 5
V12

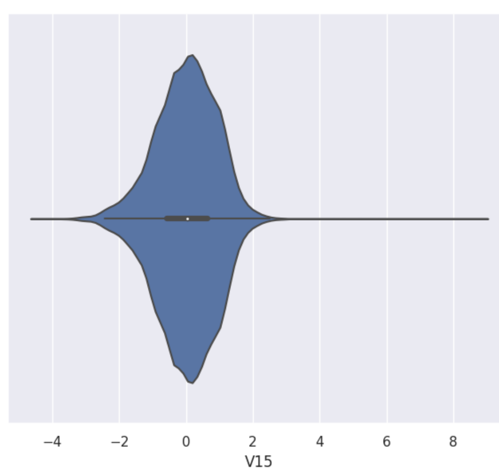

−6 −4 −2 0 2 4 6
V13

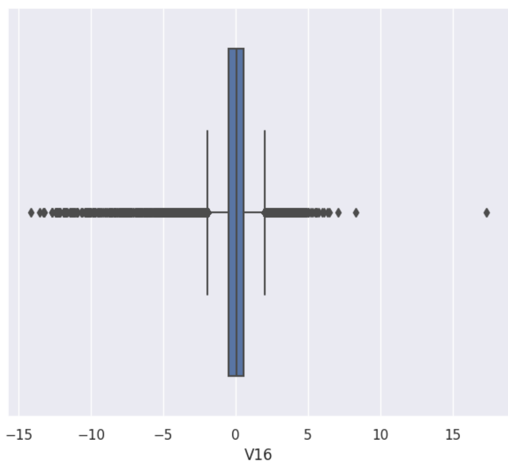

−6 −4 −2 0 2 4 6
V13

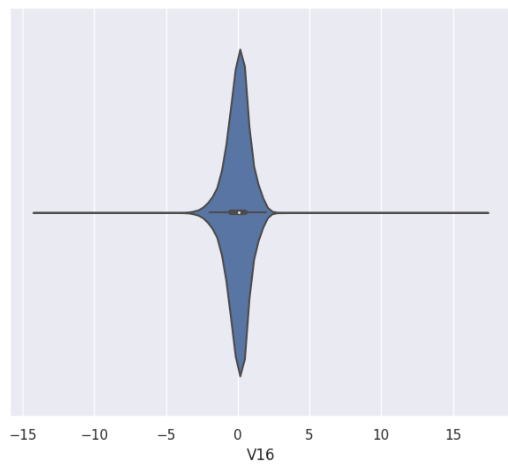

−20 −15 −10 −5 0 5 10
V14

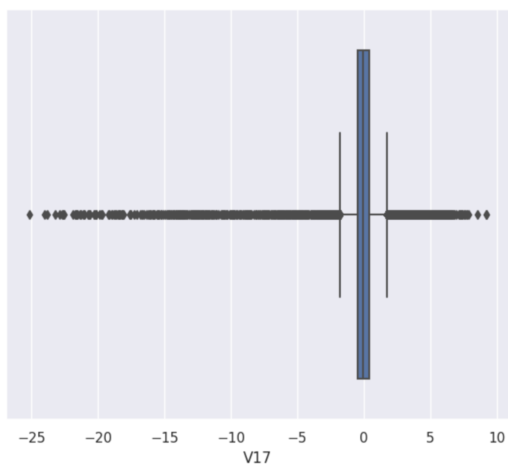

−20 −15 −10 −5 0 5 10
V14

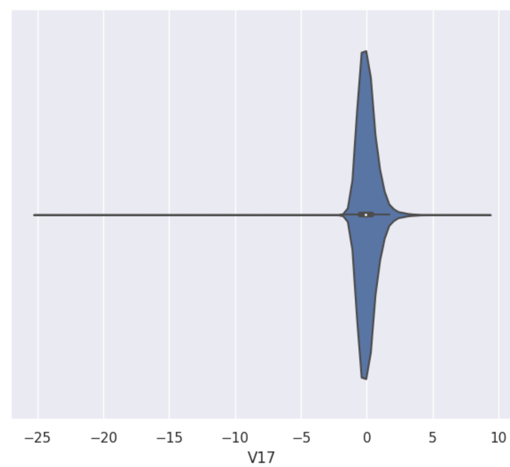

−4 −2 0 2 4 6 8
V15

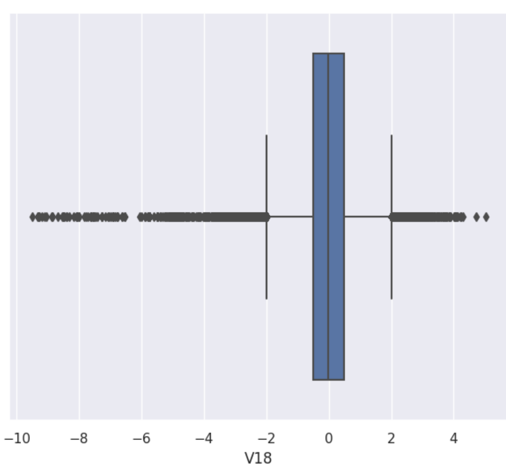

−4 −2 0 2 4 6 8
V15

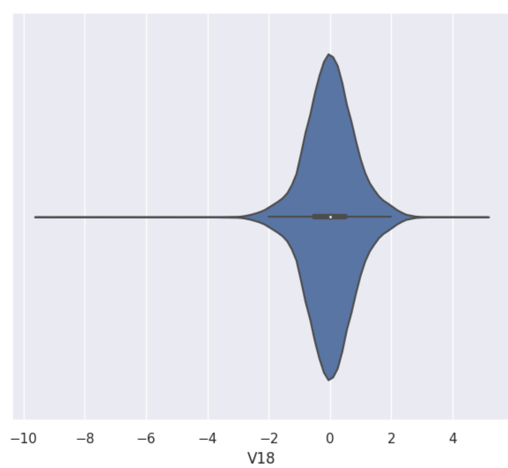

−15 −10 −5 0 5 10 15
V16

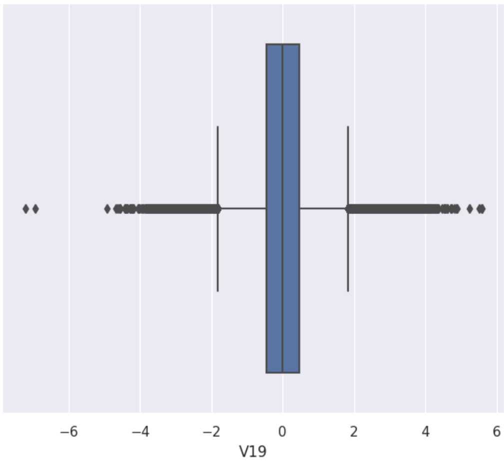

−15 −10 −5 0 5 10 15
V16

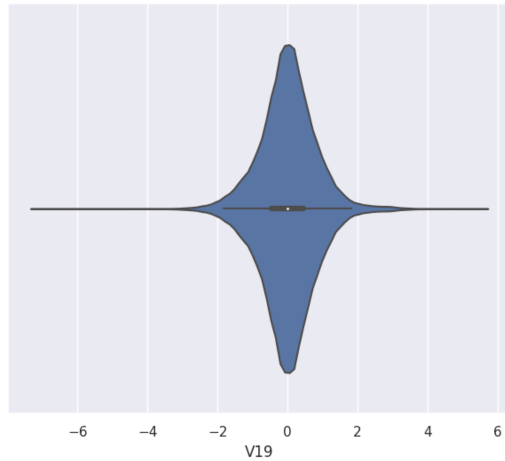

−25 −20 −15 −10 −5 0 5 10
V17

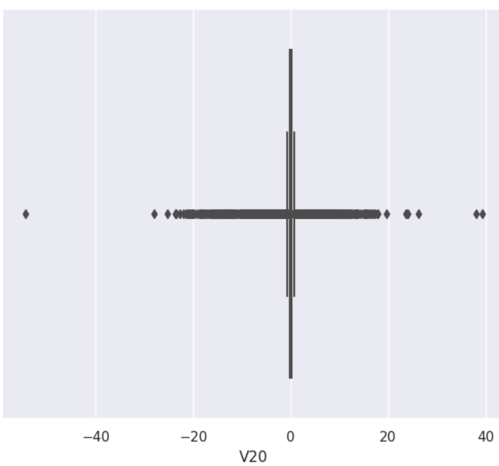

−25 −20 −15 −10 −5 0 5 10
V17

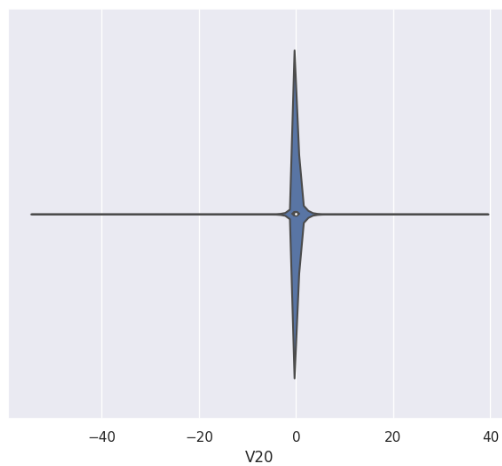

−10 −8 −6 −4 −2 0 2 4
V18

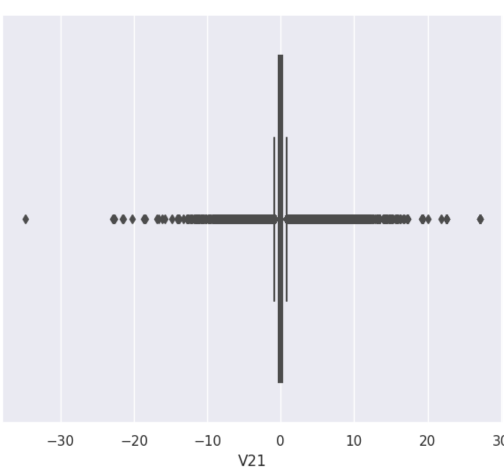

−10 −8 −6 −4 −2 0 2 4
V18

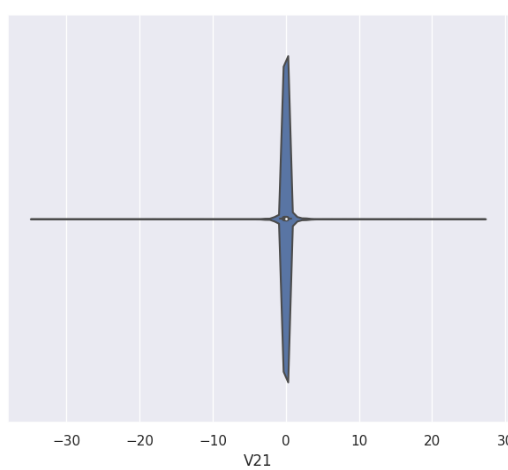

−6 −4 −2 0 2 4 6
V19

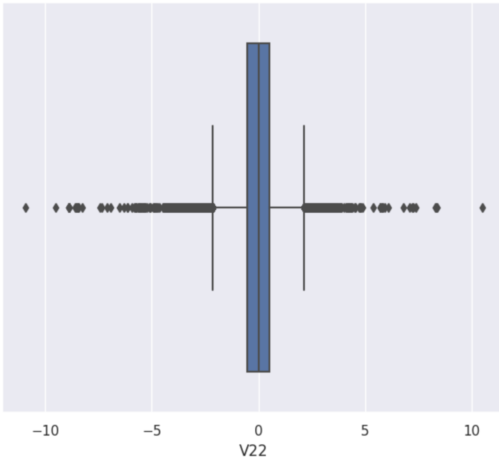

−6 −4 −2 0 2 4 6
V19

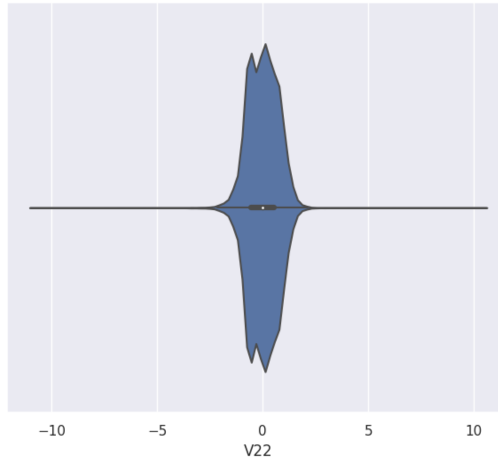

−40 −20 0 20 40
V20

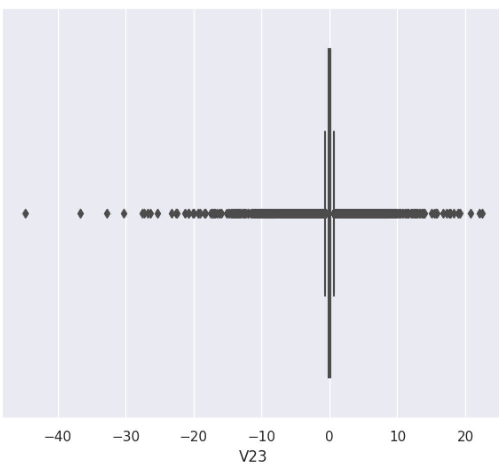

−40 −20 0 20 40
V20

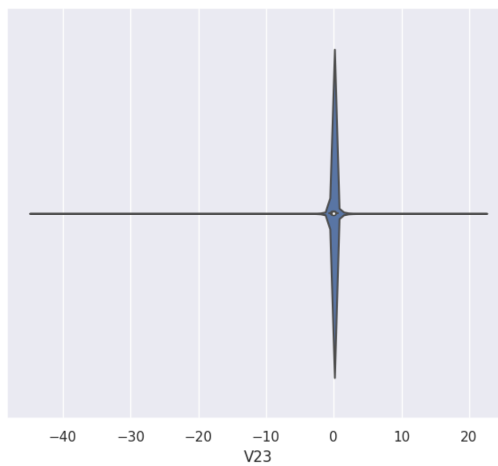

−30 −20 −10 0 10 20 30
V21

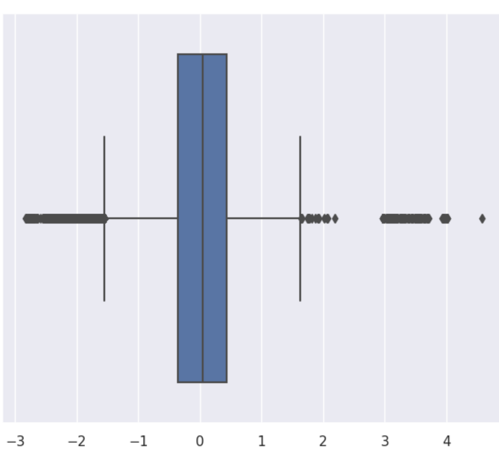

−30 −20 −10 0 10 20 30
V21

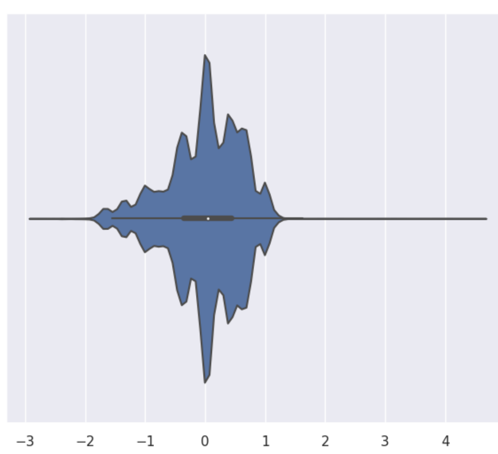

−10 −5 0 5 10
V22

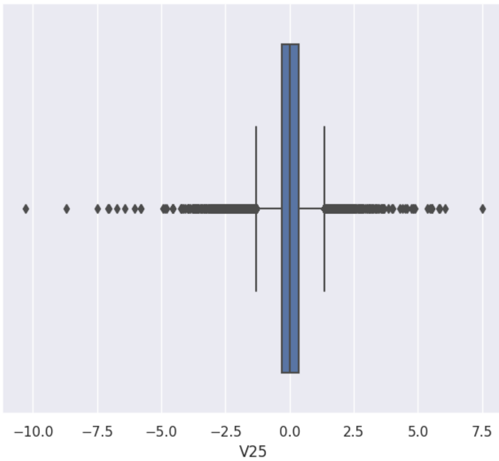

−10 −5 0 5 10
V22

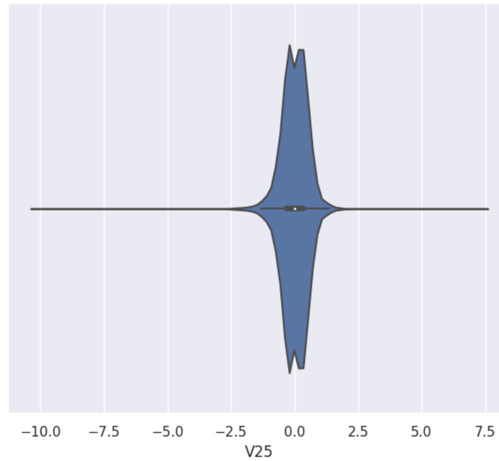

−40 −30 −20 −10 0 10 20
V23

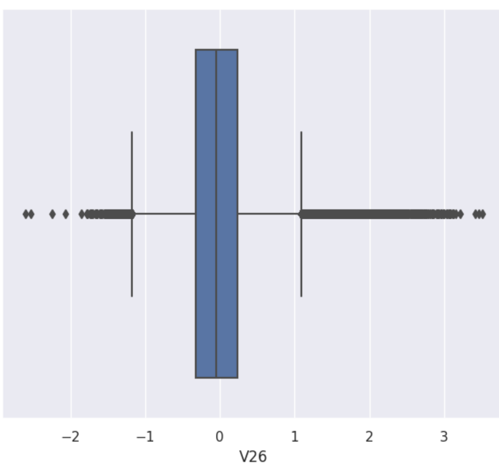

−40 −30 −20 −10 0 10 20
V23


−3 −2 −1 0 1 2 3 4
V24

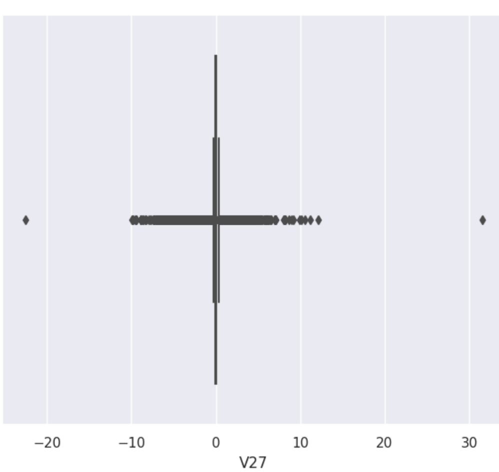

−3 −2 −1 0 1 2 3 4 5
V24

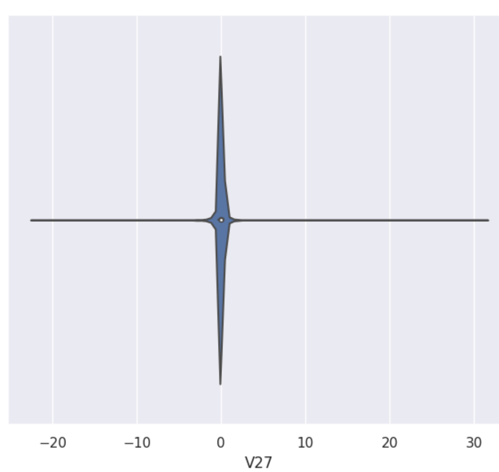

−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5
V25

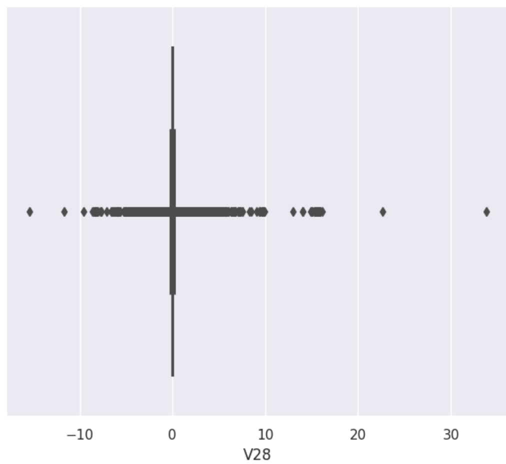

−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5
V25

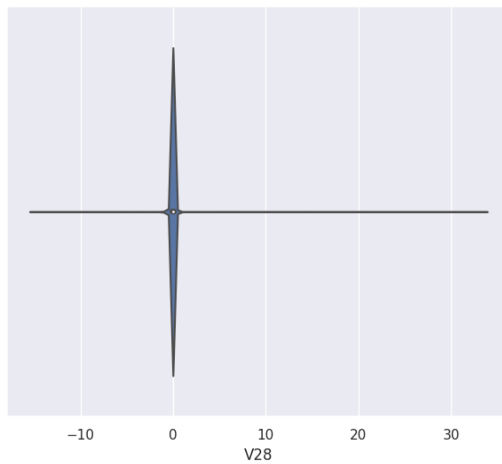

−2 −1 0 1 2 3
V26


−2 −1 0 1 2 3
V26


−20 −10 0 10 20 30
V27


−20 −10 0 10 20 30
V27





4 3. Relationships among the features

First we analyze how the amount of transaction behaves with respect to time.

4.1 Amount vs Time

[13]:
```
fig1 = px.scatter(data,
                  x = 'Time',
                  y = 'Amount',
                  color = fraud_status,
                  #marginal_x = 'rug',
                  #marginal_y = 'rug',
                  #size = class_rescaled,
                  title = 'Amount vs Time',
                  template = 'ggplot2'
                 )
fig1.show()
```

```
/opt/conda/lib/python3.10/site-packages/plotly/express/_core.py:2065:
FutureWarning:

When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
```

We split up the scatterplot into two different subplots, one for authentic transactions and the other for fraudulent tranactions.

```
[14]: fig1 = px.scatter(data,
                      x = 'Time',
                      y = 'Amount',
                      facet_col = fraud_status,
                      color = fraud_status,
                      title = 'Amount vs Time',
                      template = 'ggplot2'
                     )
      fig1.show()
```

Note that *facet_col=False* corresponds to the authentic transactions and *facet_col=True* corresponds to the fraudulent transactions. We zoom into the second subplot a bit to get a clearer picture.

```
[15]: # Amount vs Time for fraudulent transactions - scatterplot

      fig1 = px.scatter(data_fraud,
                      x = 'Time',
                      y = 'Amount',
                      title = 'Amount vs Time for fraudulent transactions',
                      template = 'ggplot2'
                     )
      fig1.show()
```

```
[16]: print('Correlation coefficient between Time and Amount')
      print('\n')
      print('For all transactions: {}'.format(data['Time'].corr(data['Amount'])))
      print('For authentic transactions: {}'.format(data_authentic['Time'].
        ↪corr(data_authentic['Amount'])))
      print('For fraudulent transactions: {}'.format(data_fraud['Time'].
        ↪corr(data_fraud['Amount'])))
```

```
Correlation coefficient between Time and Amount


For all transactions: -0.010596373389029244
For authentic transactions: -0.010633753673009977
For fraudulent transactions: 0.048731876460612104
```

Observation: Time and Amount appear to be approximately uncorrelated, which is echoed even when authentic and fraudulent transactions are considered separately.

Next we examine bivariate scatterplots and linear relationships between certain pairs of feature variables, which exhibit contrasting correlation structures for authentic and fraudulent transactions. Such a phenomenon occurs for a number of pairs, but we analyze 5 specific pairs among these for the sake of brevity.

4.2 V3 vs Time

[17]:
```
fig1 = px.scatter(data,
                 x = 'Time',
                 y = 'V3',
                 facet_col = fraud_status,
                 color = fraud_status,
                 title = 'V3 vs Time',
                 template = 'ggplot2'
                )
fig1.show()
```

[18]:
```
print('Correlation coefficient between V3 and Time')
print('\n')
print('For all transactions: {}'.format(data['V3'].corr(data['Time'])))
print('For authentic transactions: {}'.format(data_authentic['V3'].
 ↪corr(data_authentic['Time'])))
print('For fraudulent transactions: {}'.format(data_fraud['V3'].
 ↪corr(data_fraud['Time'])))
```

```
Correlation coefficient between V3 and Time


For all transactions: -0.41961817221152636
For authentic transactions: -0.441000843827495
For fraudulent transactions: 0.2095967306300109
```

Observations:

- V3 and Time have moderate negative correlation for authentic transactions.
- However, they have slightly positive correlation for fraudulent transactions.

4.3 Amount vs V20

[19]:
```
fig1 = px.scatter(data,
                 x = 'V20',
                 y = 'Amount',
                 facet_col = fraud_status,
                 color = fraud_status,
                 title = 'Amount vs V20',
                 template = 'ggplot2'
                )
fig1.show()
```

[20]:
```
print('Correlation coefficient between Amount and V20')
print('\n')
print('For all transactions: {}'.format(data['Amount'].corr(data['V20'])))
```

```
print('For authentic transactions: {}'.format(data_authentic['Amount'].
 ↪corr(data_authentic['V20'])))
print('For fraudulent transactions: {}'.format(data_fraud['Amount'].
 ↪corr(data_fraud['V20'])))
```

```
Correlation coefficient between Amount and V20


For all transactions: 0.3394034045461746
For authentic transactions: 0.3404290130779015
For fraudulent transactions: 0.045428450514568376
```

Observations:

- V1 and V2 are approximately uncorrelated for authentic transactions.
- However, they have significant negative correlation for fraudulent transactions.

4.4 V2 vs V1

[21]:
```
fig1 = px.scatter(data,
                  x = 'V1',
                  y = 'V2',
                  facet_col = fraud_status,
                  color = fraud_status,
                  title = 'V2 vs V1',
                  template = 'ggplot2'
                 )
fig1.show()
```

[22]:
```
print('Correlation coefficient between V1 and V2')
print('\n')
print('For all transactions: {}'.format(data['V1'].corr(data['V2'])))
print('For authentic transactions: {}'.format(data_authentic['V1'].
 ↪corr(data_authentic['V2'])))
print('For fraudulent transactions: {}'.format(data_fraud['V1'].
 ↪corr(data_fraud['V2'])))
```

```
Correlation coefficient between V1 and V2


For all transactions: -1.3020218771793466e-16
For authentic transactions: 0.022537284217764193
For fraudulent transactions: -0.8192257999187483
```

Observations:

- V1 and V2 are approximately uncorrelated for authentic transactions.
- However, they have significant negative correlation for fraudulent transactions.

4.5 V3 vs V2

[23]:
```
fig1 = px.scatter(data,
                  x = 'V2',
                  y = 'V3',
                  facet_col = fraud_status,
                  color = fraud_status,
                  title = 'V3 vs V2',
                  template = 'ggplot2'
                 )
fig1.show()
```

[24]:
```
print('Correlation coefficient between V2 and V3')
print('\n')
print('For all transactions: {}'.format(data['V2'].corr(data['V3'])))
print('For authentic transactions: {}'.format(data_authentic['V2'].
 ↪corr(data_authentic['V3'])))
print('For fraudulent transactions: {}'.format(data_fraud['V2'].
 ↪corr(data_fraud['V3'])))
```

```
Correlation coefficient between V2 and V3


For all transactions: 6.584315291185336e-17
For authentic transactions: 0.03785508793171424
For fraudulent transactions: -0.876903687461216
```

Observations:

- V2 and V3 are approximately uncorrelated for authentic transactions.
- However, they have significant negative correlation for fraudulent transactions.

4.6 V3 vs V1

[25]:
```
fig1 = px.scatter(data,
                  x = 'V1',
                  y = 'V3',
                  facet_col = fraud_status,
                  color = fraud_status,
                  title = 'V3 vs V1',
                  template = 'ggplot2'
                 )
fig1.show()
```

[26]:
```
print('Correlation coefficient between V1 and V3')
print('\n')
print('For all transactions: {}'.format(data['V1'].corr(data['V3'])))
```

```
print('For authentic transactions: {}'.format(data_authentic['V1'].
    ↪corr(data_authentic['V3'])))
print('For fraudulent transactions: {}'.format(data_fraud['V1'].
    ↪corr(data_fraud['V3'])))
```

```
Correlation coefficient between V1 and V3


For all transactions: -5.503293652784113e-16
For authentic transactions: -0.047510934990898264
For fraudulent transactions: 0.9078750102143118
```

Observations:

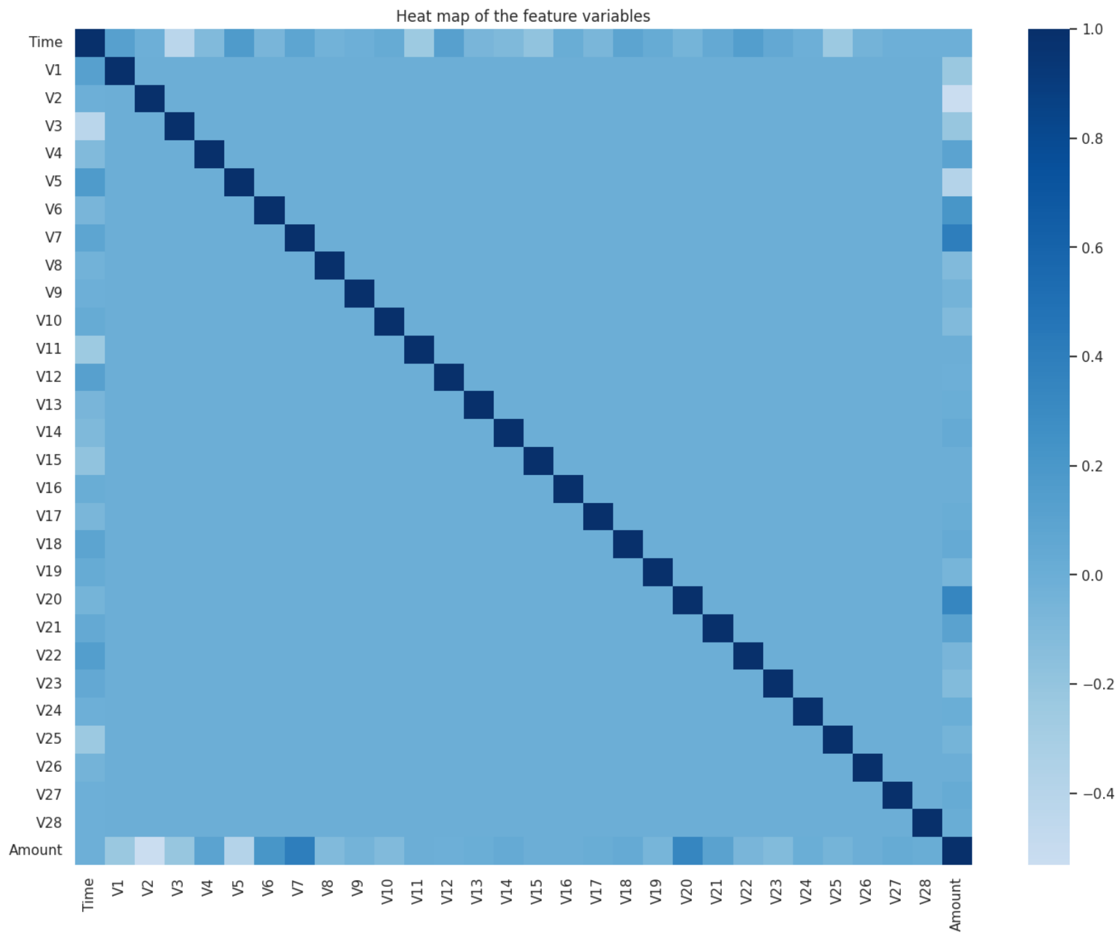
- V1 and V3 are approximately uncorrelated for authentic transactions.
- However, they have significant positive correlation for fraudulent transactions.

4.7 Multicollinearity

We check for multicollinearity among the features through the heat map which plots the correlation coefficient of each pair of features via color density.

[27]:
```
# Heat map of the feature variables

features = data.drop(['Class'], axis = 1)

fig, ax = plt.subplots(figsize=(16, 12))
sns.heatmap(features.corr(), center = 0, cmap = 'Blues')
ax.set_title('Heat map of the feature variables')
```

[27]:
```
Text(0.5, 1.0, 'Heat map of the feature variables')
```



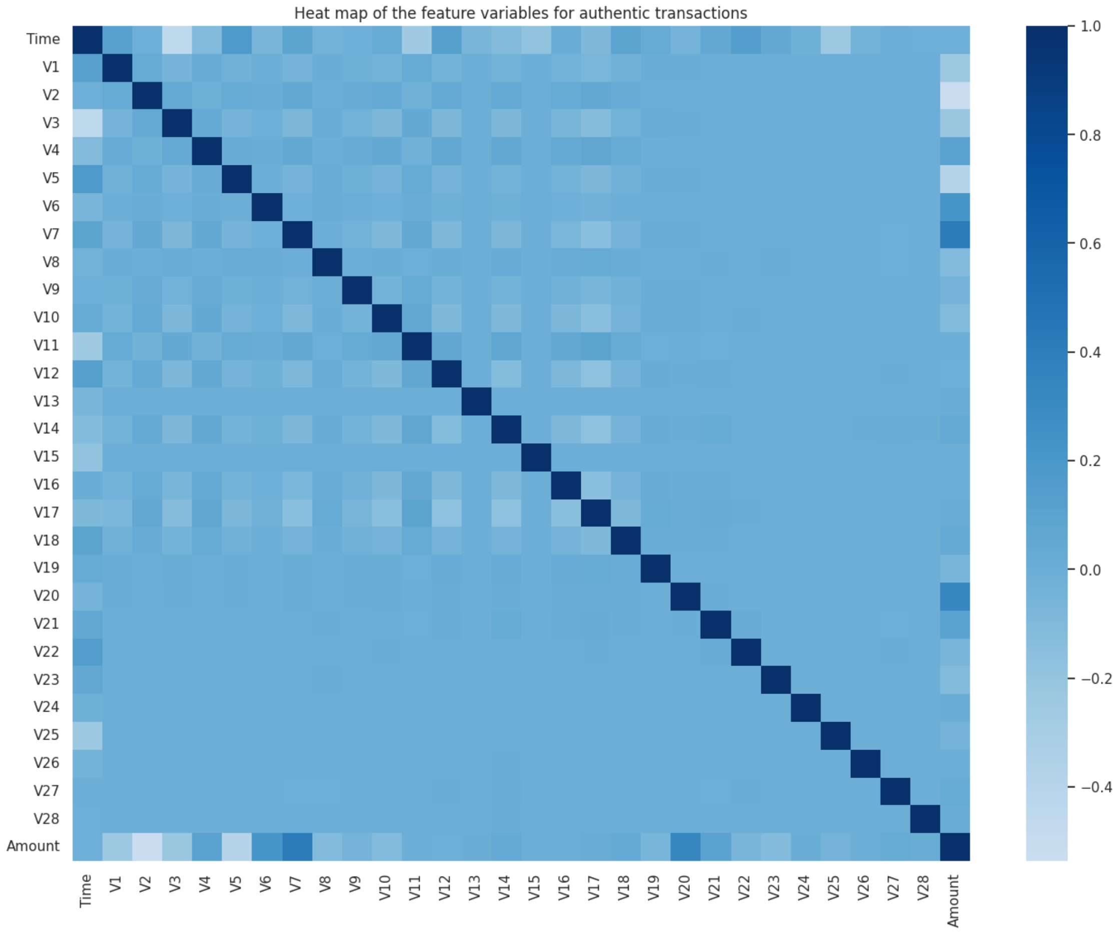
Observations:

- As expected, the PCA-engineered features are uncorrelated.
- There exists non-zero correlations between time and some PCA-engineered features as well as between amount and some PCA-engineered features.

If one considers only the authentic transactions, then the overall structure of the heat map remains the same, although moderate changes are visible.

[28]:
```
# Heat map of the feature variables for fraudulent transactions

features_authentic = data_authentic.drop(['Class'], axis = 1)

fig, ax = plt.subplots(figsize=(16, 12))
sns.heatmap(features_authentic.corr(), center = 0, cmap = 'Blues')
ax.set_title('Heat map of the feature variables for authentic transactions')
```

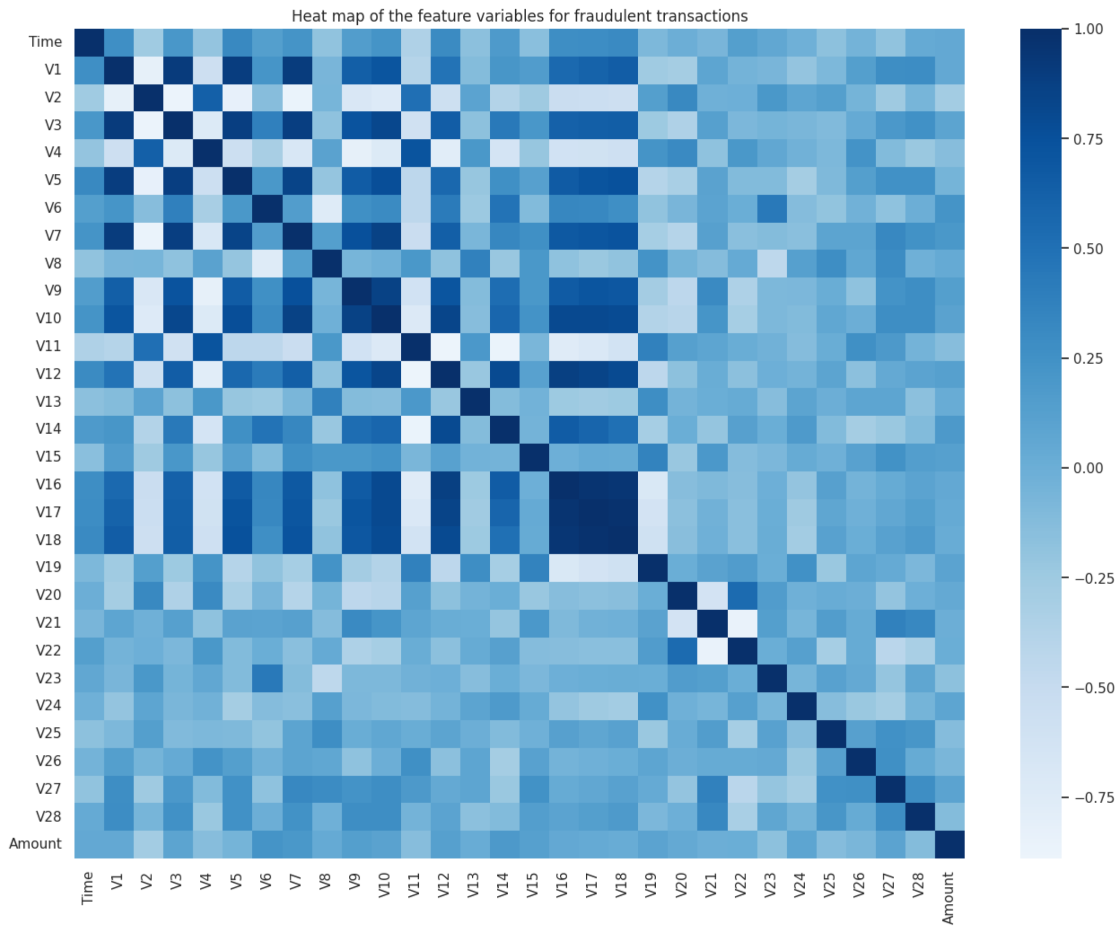
[28]:
```
Text(0.5, 1.0, 'Heat map of the feature variables for authentic transactions')
```



However, the fraudulent transactions show a significantly different correlation structure among the features.

[29]:
```
# Heat map of the feature variables for fraudulent transactions

features_fraud = data_fraud.drop(['Class'], axis = 1)

fig, ax = plt.subplots(figsize=(16, 12))
sns.heatmap(features_fraud.corr(), center = 0, cmap = 'Blues')
ax.set_title('Heat map of the feature variables for fraudulent transactions')
```

[29]:
```
Text(0.5, 1.0, 'Heat map of the feature variables for fraudulent transactions')
```



Note that the analysis involving the transformed variables **V1-V28** do not reflect any relationship among the original variables from which those are engineered. We have included EDA for these variables for the sake of completeness.