CSCI - 6409 - The Process of Data Science - Fall 2022

</center>

Assignment 3

</center>

Meagan Sinclair
B00737317

Samarth Jariwala
B00899380

1. Data Preparation

1.1 Data Quality Report

In [1]:

```
from sklearn import datasets
dataset = datasets.fetch_openml(data_id = 1597, as_frame=True)
```

In [2]:

```
df = dataset.frame
df = df.convert_dtypes()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 30 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   V1      284807 non-null  Float64
 1   V2      284807 non-null  Float64
 2   V3      284807 non-null  Float64
 3   V4      284807 non-null  Float64
 4   V5      284807 non-null  Float64
 5   V6      284807 non-null  Float64
 6   V7      284807 non-null  Float64
 7   V8      284807 non-null  Float64
 8   V9      284807 non-null  Float64
 9   V10     284807 non-null  Float64
 10  V11     284807 non-null  Float64
 11  V12     284807 non-null  Float64
 12  V13     284807 non-null  Float64
 13  V14     284807 non-null  Float64
 14  V15     284807 non-null  Float64
 15  V16     284807 non-null  Float64
 16  V17     284807 non-null  Float64
 17  V18     284807 non-null  Float64
 18  V19     284807 non-null  Float64
 19  V20     284807 non-null  Float64
 20  V21     284807 non-null  Float64
 21  V22     284807 non-null  Float64
 22  V23     284807 non-null  Float64
```

```
 23  V24     284807 non-null  Float64
 24  V25     284807 non-null  Float64
 25  V26     284807 non-null  Float64
 26  V27     284807 non-null  Float64
 27  V28     284807 non-null  Float64
 28  Amount  284807 non-null  Float64
 29  Class   284807 non-null  category
dtypes: Float64(29), category(1)
memory usage: 71.2 MB
```

In [3]:

```
df
```

Out[3]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794
1	1.191857	0.266151	0.16648	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974
2	-1.358354	-1.340163	1.773209	0.37978	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074
...	...	...	...	...	...	...	...	...	...	...
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.35617
284803	-0.732789	-0.05508	2.03503	-0.738589	0.868229	1.058415	0.02433	0.294869	0.5848	-0.975926
284804	1.919565	-0.301254	-3.24964	-0.557828	2.630515	3.03126	-0.296827	0.708417	0.432454	-0.484782
284805	-0.24044	0.530483	0.70251	0.689799	-0.377961	0.623708	-0.68618	0.679145	0.392087	-0.399126
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.41465	0.48618	-0.915427

284807 rows × 30 columns

In [67]:

```
clsses = df['Class'].to_numpy(dtype='int')
print("Number of class 0: ", len(clsses[clsses == 0]))
print("Number of class 1: ", len(clsses[clsses == 1]))
```

```
Number of class 0:  284315
Number of class 1:  492
```

In [4]:

```
# code source: Tutorial [1]
import pandas as pd
import warnings

def build_continuous_features_report(data_df):

    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Min": lambda df: df.min(),
        "1st Qrt.": lambda df: df.quantile(0.25),
        "Mean": lambda df: df.mean(),
        "Median": lambda df: df.median(),
        "3rd Qrt": lambda df: df.quantile(0.75),
        "Max": lambda df: df.max(),
        "Std. Dev.": lambda df: df.std(),
    }
```

```
    contin_feat_names = data_df.select_dtypes("number").columns
    continuous_data_df = data_df[contin_feat_names]

    report_df = pd.DataFrame(index=contin_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df
```

In [5]:

```
build_continuous_features_report(df)
```

Out[5]:

	Count	Miss %	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt	Max	Std. Dev.
V1	284807	0.0	275663	-56.407510	-0.920373	1.168375e-15	0.018109	1.315642	2.454930	1.958696
V2	284807	0.0	275663	-72.715728	-0.59855	3.416908e-16	0.065486	0.803724	22.057729	1.651309
V3	284807	0.0	275663	-48.325589	-0.890365	-1.379537e-15	0.179846	1.027196	9.382558	1.516255
V4	284807	0.0	275663	-5.683171	-0.84864	2.074095e-15	-0.019847	0.743341	16.875344	1.415869
V5	284807	0.0	275663	-113.743307	-0.691597	9.604066e-16	-0.054336	0.611926	34.801666	1.380247
V6	284807	0.0	275663	-26.160506	-0.768296	1.487313e-15	-0.274187	0.398565	73.301626	1.332271
V7	284807	0.0	275663	-43.557242	-0.554076	-5.556467e-16	0.040103	0.570436	120.589494	1.237094
V8	284807	0.0	275663	-73.216718	-0.20863	1.205498e-16	0.022358	0.327346	20.007208	1.194353
V9	284807	0.0	275663	-13.434066	-0.643098	-2.406306e-15	-0.051429	0.597139	15.594995	1.098632
V10	284807	0.0	275663	-24.588262	-0.535426	2.238853e-15	-0.092917	0.453923	23.745136	1.088850
V11	284807	0.0	275663	-4.797473	-0.762494	1.673327e-15	-0.032757	0.739593	12.018913	1.020713
V12	284807	0.0	275663	-18.683715	-0.405571	-1.247012e-15	0.140033	0.618238	7.848392	0.999201
V13	284807	0.0	275663	-5.791881	-0.648539	8.190001e-16	-0.013568	0.662505	7.126883	0.995274
V14	284807	0.0	275663	-19.214325	-0.425574	1.207294e-15	0.050601	0.49315	10.526766	0.958596
V15	284807	0.0	275663	-4.498945	-0.582884	4.887456e-15	0.048072	0.648821	8.877742	0.915316
V16	284807	0.0	275663	-14.129855	-0.468037	1.437516e-15	0.066413	0.523296	17.315112	0.876253
V17	284807	0.0	275663	-25.162799	-0.483748	-3.740237e-16	-0.065676	0.399675	9.253526	0.849337
V18	284807	0.0	275663	-9.498746	-0.49885	9.564149e-16	-0.003636	0.500807	5.041069	0.838176
V19	284807	0.0	275663	-7.213527	-0.456299	1.039917e-15	0.003735	0.458949	5.591971	0.814041
V20	284807	0.0	275663	-54.497720	-0.211721	6.407202e-16	-0.062481	0.133041	39.420904	0.770925
V21	284807	0.0	275663	-34.830382	-0.228395	1.656562e-16	-0.029450	0.186377	27.202839	0.734524
V22	284807	0.0	275663	-10.933144	-0.54235	-3.568593e-16	0.006782	0.528554	10.503090	0.725702
V23	284807	0.0	275663	-44.807735	-0.161846	2.610582e-16	-0.011193	0.147642	22.528412	0.624460

	Count	Miss %	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt	Max	Std. Dev.
V24	284807	0.0	275663	-2.836627	-0.354586	4.473066e-15	0.040976	0.439527	4.584549	0.605647
V25	284807	0.0	275663	-10.295397	-0.317145	5.213180e-16	0.016594	0.350716	7.519589	0.521278
V26	284807	0.0	275663	-2.604551	-0.326984	1.683537e-15	-0.052139	0.240952	3.517346	0.482227
V27	284807	0.0	275663	-22.565679	-0.07084	-3.659966e-16	0.001342	0.091045	31.612198	0.403632
V28	284807	0.0	275663	-15.430084	-0.05296	-1.223710e-16	0.011244	0.07828	33.847808	0.330083
Amount	284807	0.0	32767	0.000000	5.6	8.834962e+01	22.000000	77.165	25691.160000	250.120109

In [6]:

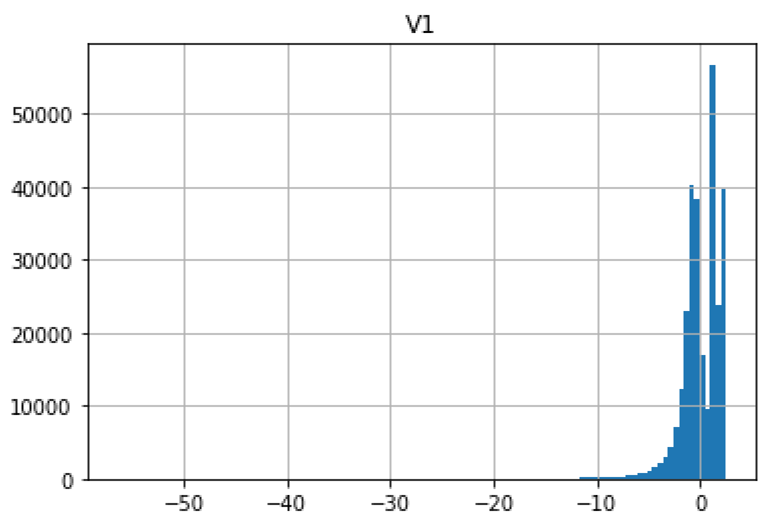
```
from matplotlib import pyplot as plt
```

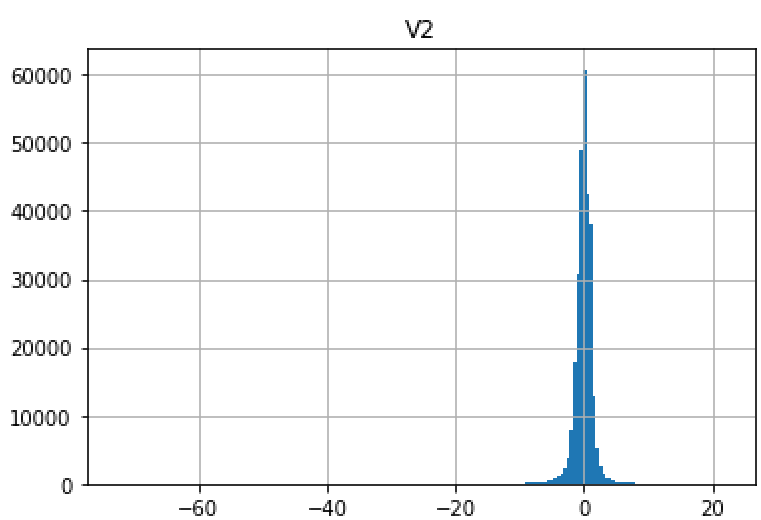
In [7]:

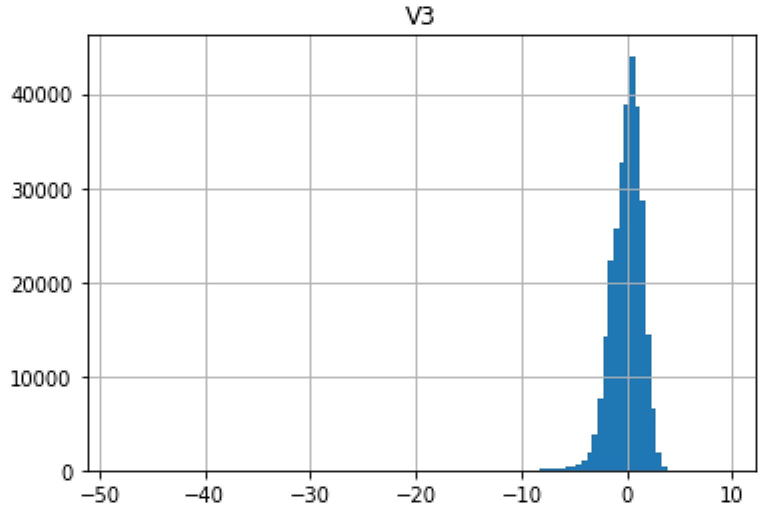
```
contin_feat_names = df.select_dtypes("number").columns
```

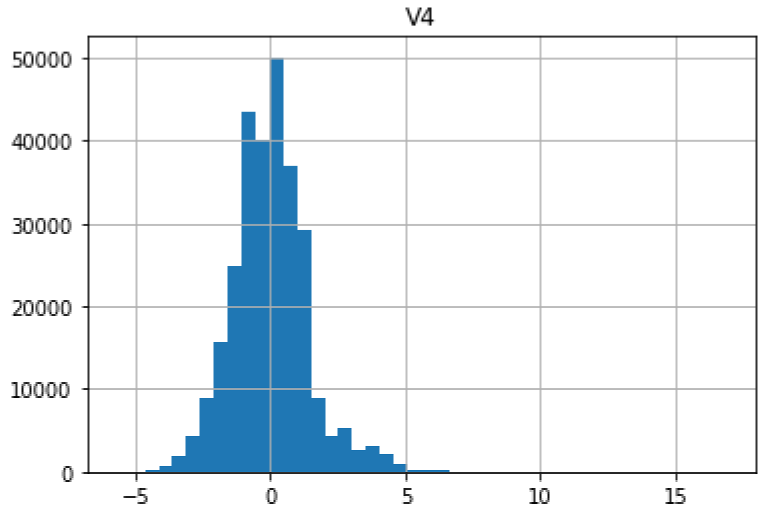
In [8]:

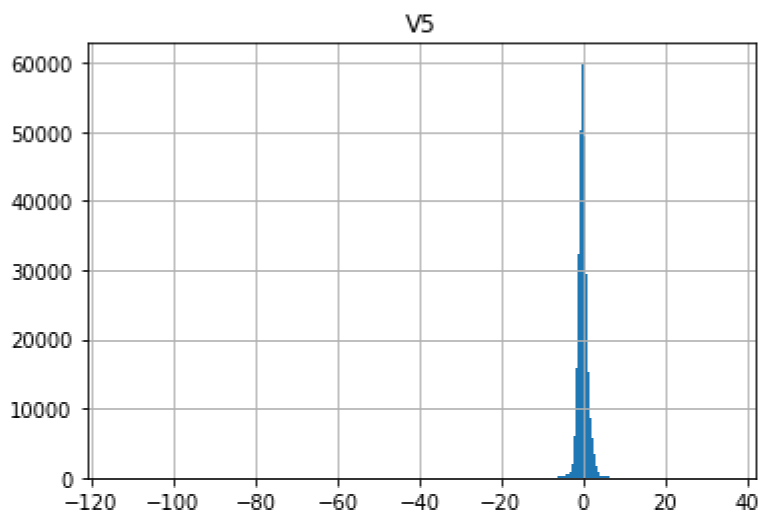
```
for column in contin_feat_names:
    range = int(df[column].max() - df[column].min())
    df.hist(column=column, bins=range*2)
    plt.show()
```

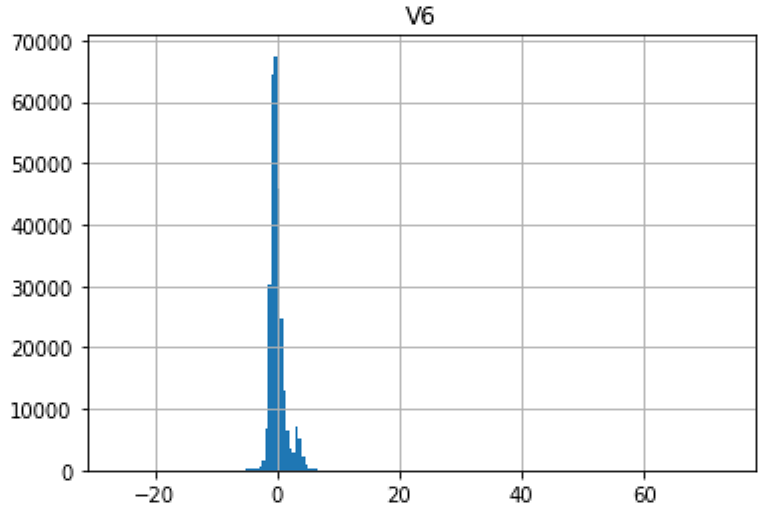


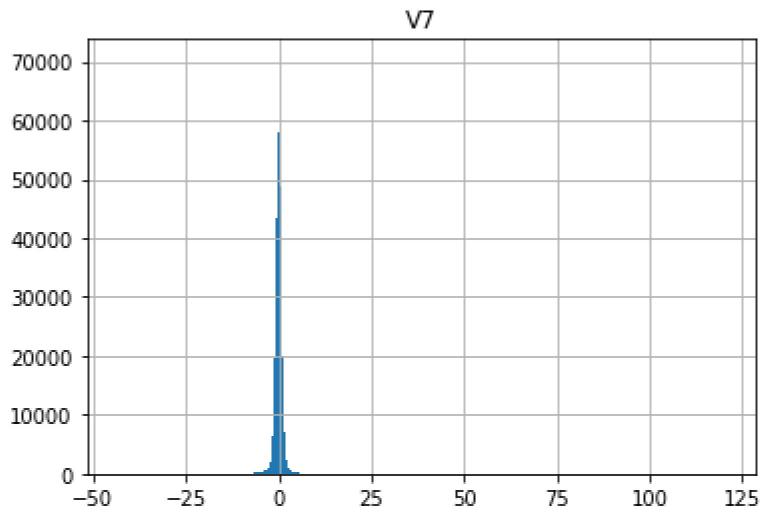


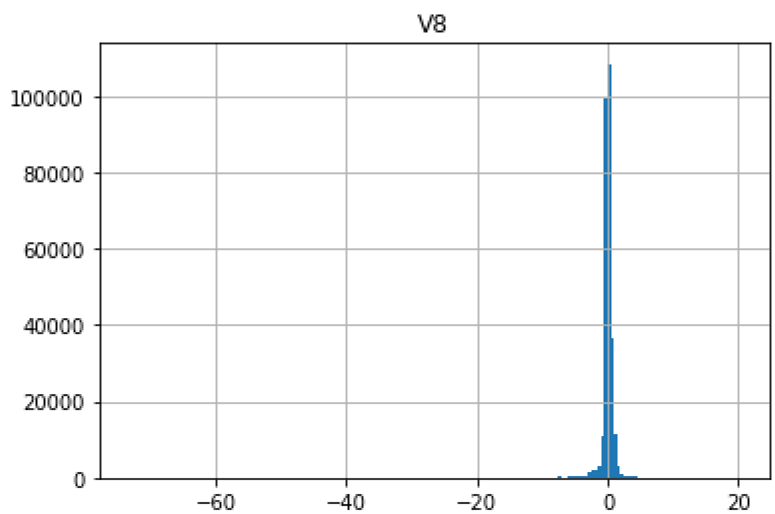

V3
0
10000
20000
30000
40000
−50
−40
−30
−20
−10
0
10

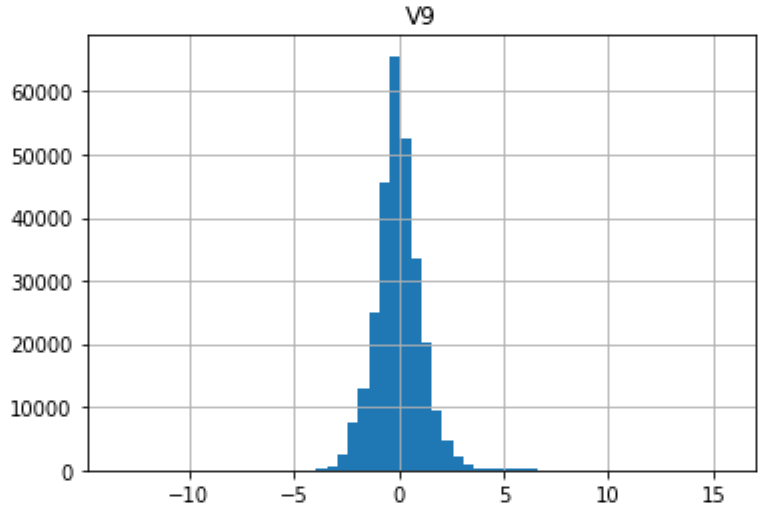

V4
0
10000
20000
30000
40000
50000
−5
0
5
10
15

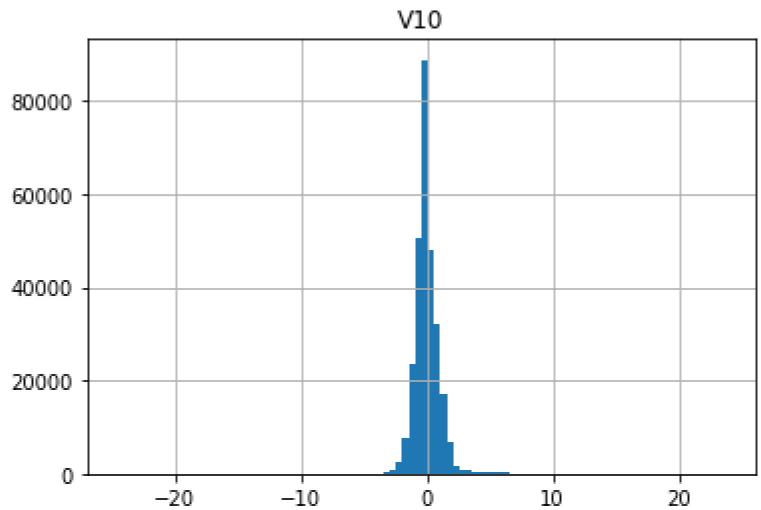

V5
0
10000
20000
30000
40000
50000
60000
−120
−100
−80
−60
−40
−20
0
20
40

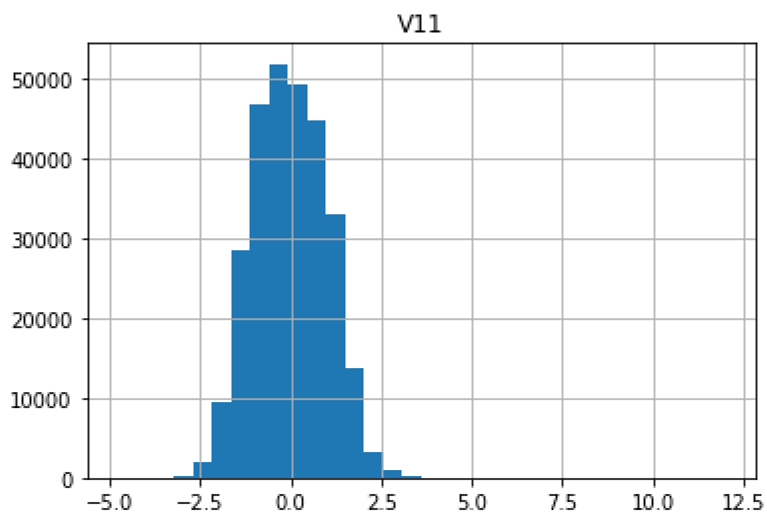

V6
70000
60000
50000
40000
30000
20000
10000
0
−20
0
20
40
60

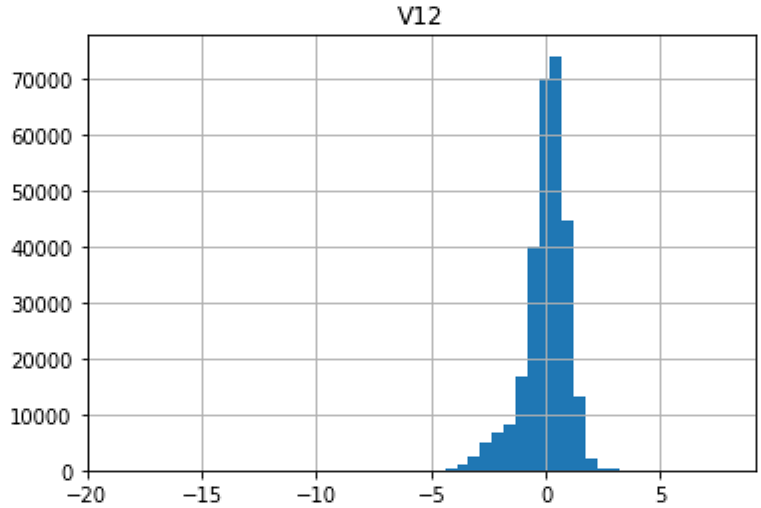

V7
70000
60000
50000
40000
30000
20000
10000
0
−50
−25
0
25
50
75
100
125

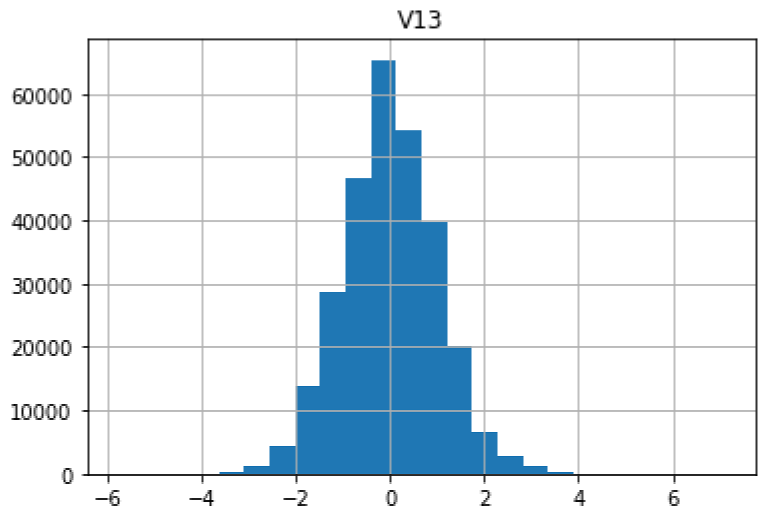

V8
100000
80000
60000
40000
20000
0
−60
−40
−20
0
20

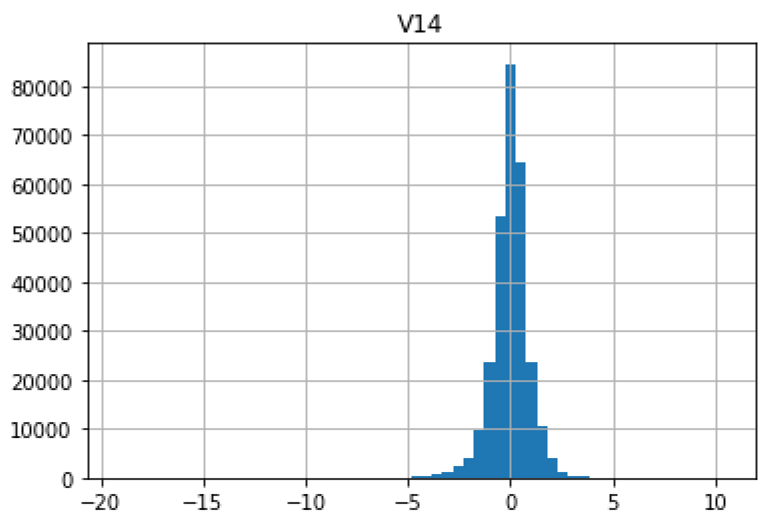

V9
60000
50000
40000
30000
20000
10000
0
−10
−5
0
5
10
15

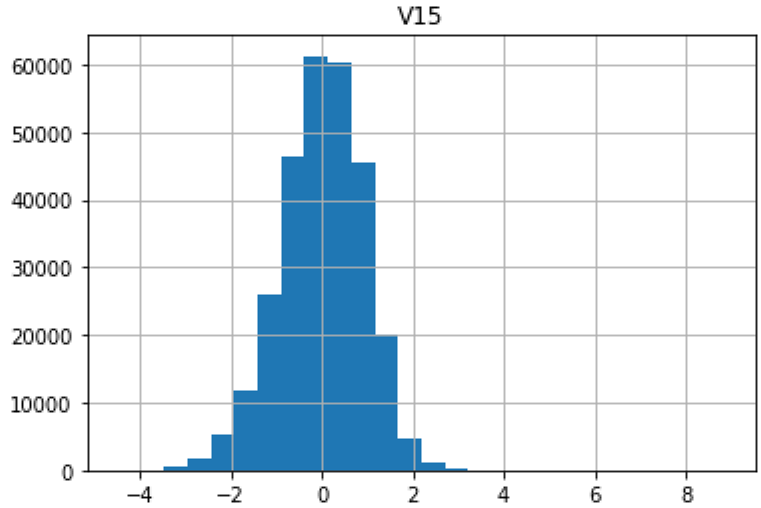

V10
80000
60000
40000
20000
0
−20
−10
0
10
20

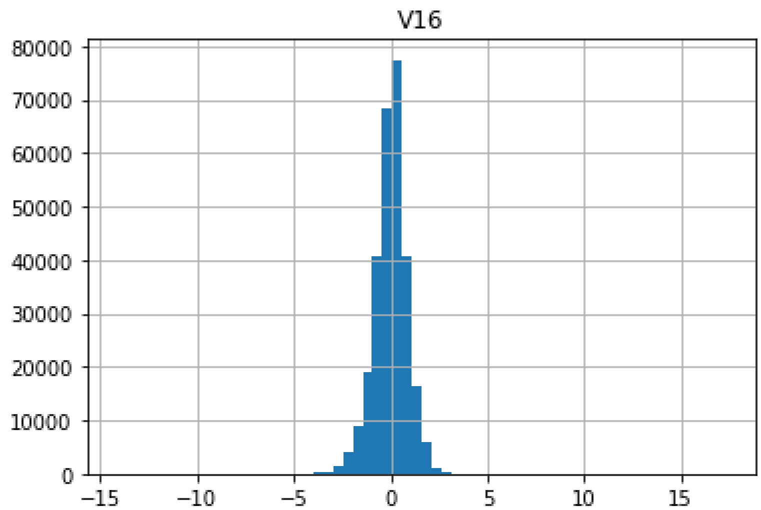

V11
50000
40000
30000
20000
10000
0
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5

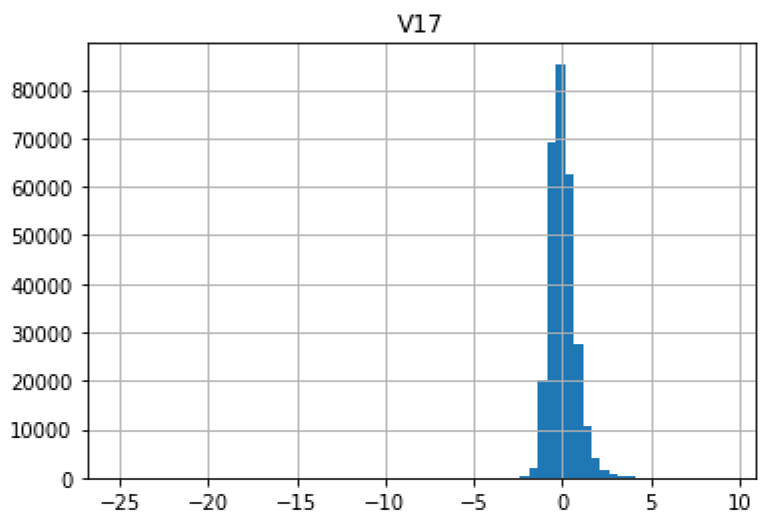

V12
70000
60000
50000
40000
30000
20000
10000
0
−20
−15
−10
−5
0
5

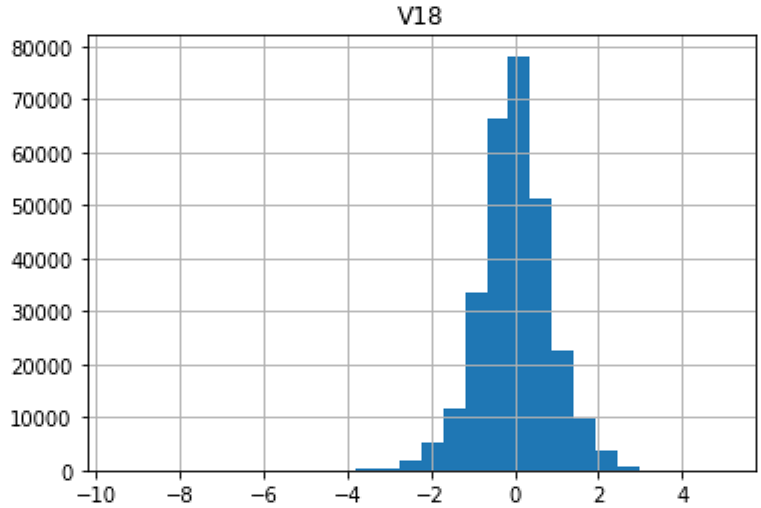

V13
60000
50000
40000
30000
20000
10000
0
−6
−4
−2
0
2
4
6

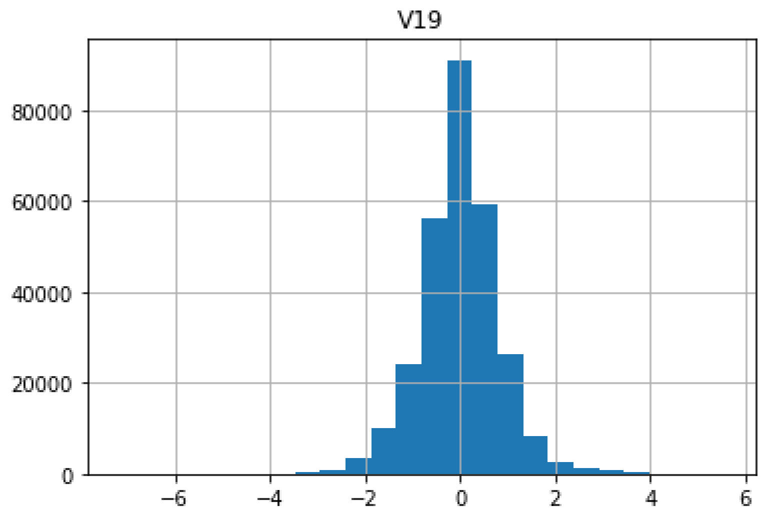

V14
80000
70000
60000
50000
40000
30000
20000
10000
0
−20
−15
−10
−5
0
5
10

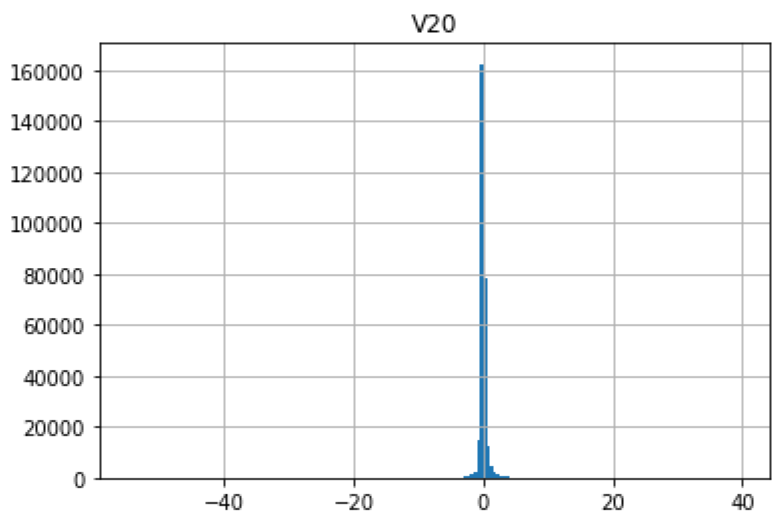

V15
0
10000
20000
30000
40000
50000
60000
−4
−2
0
2
4
6
8

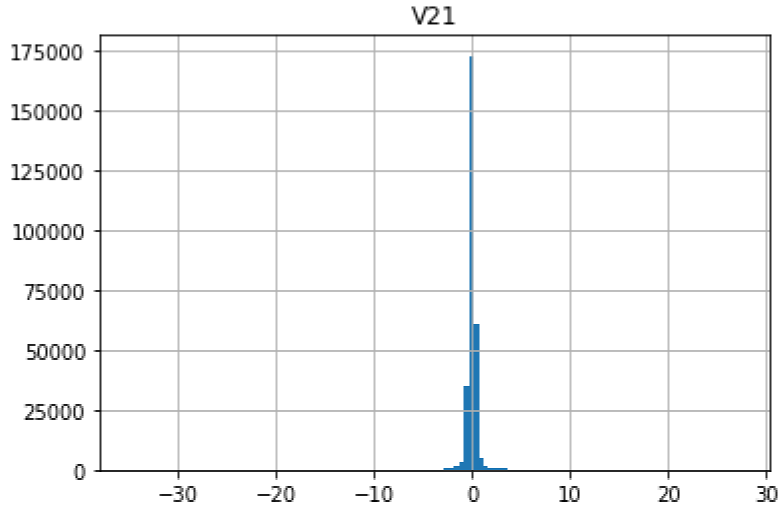

V16
0
10000
20000
30000
40000
50000
60000
70000
80000
−15
−10
−5
0
5
10
15

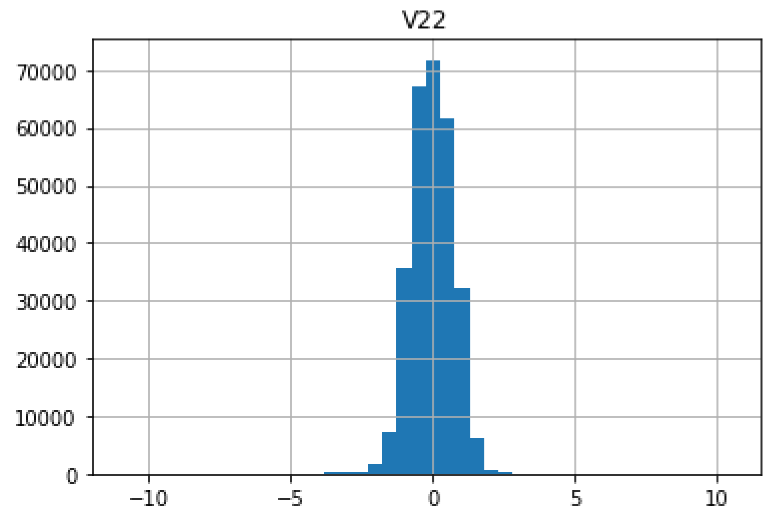

V17
0
10000
20000
30000
40000
50000
60000
70000
80000
−25
−20
−15
−10
−5
0
5
10

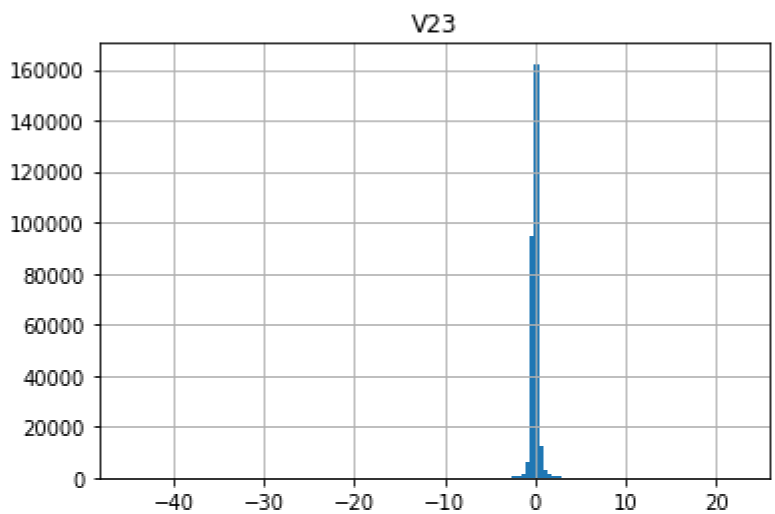

V18
80000
70000
60000
50000
40000
30000
20000
10000
0
−10
−8
−6
−4
−2
0
2
4

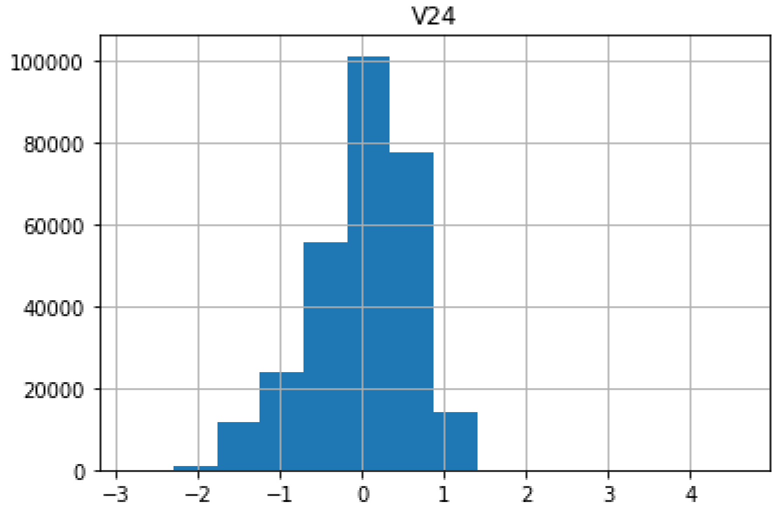

V19
80000
60000
40000
20000
0
−6
−4
−2
0
2
4
6

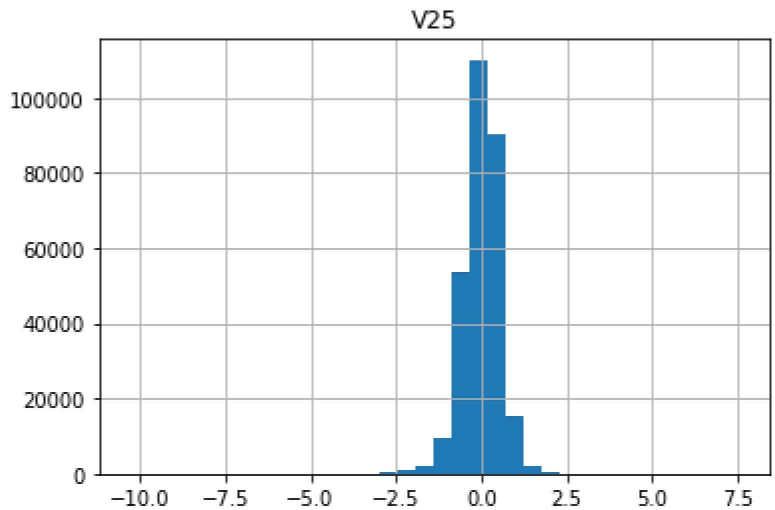

V20
160000
140000
120000
100000
80000
60000
40000
20000
0
−40
−20
0
20
40

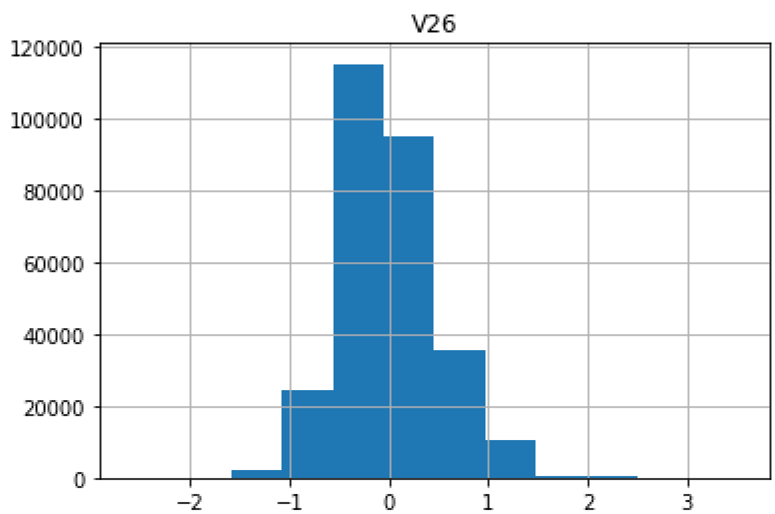

V21
0
25000
50000
75000
100000
125000
150000
175000
−30
−20
−10
0
10
20
30

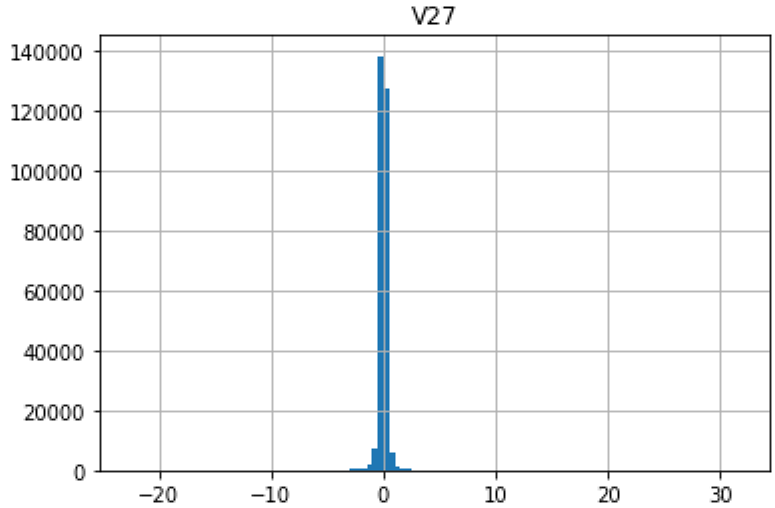

V22
0
10000
20000
30000
40000
50000
60000
70000
−10
−5
0
5
10

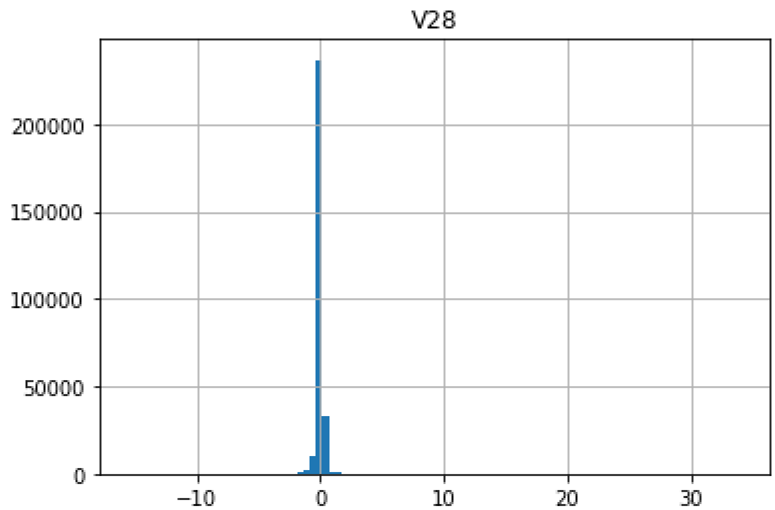

V23
0
20000
40000
60000
80000
100000
120000
140000
160000
−40
−30
−20
−10
0
10
20

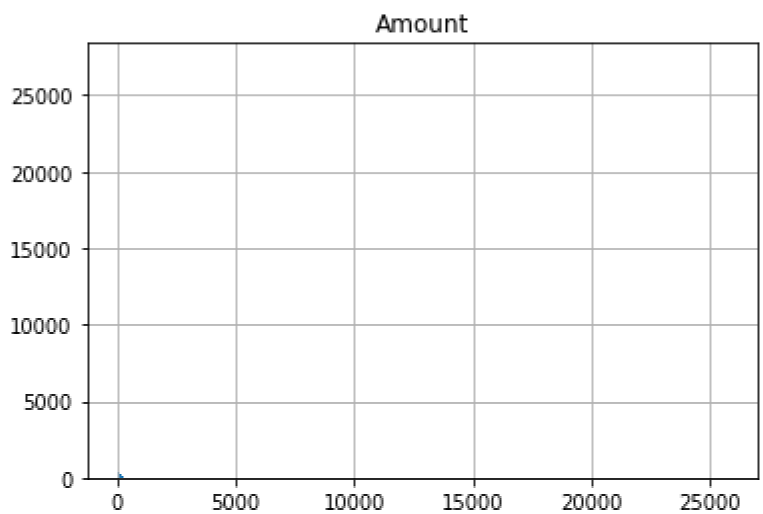

V24
100000
80000
60000
40000
20000
0
−3
−2
−1
0
1
2
3
4


V25
100000
80000
60000
40000
20000
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5


V26
120000
100000
80000
60000
40000
20000
0
−2
−1
0
1
2
3

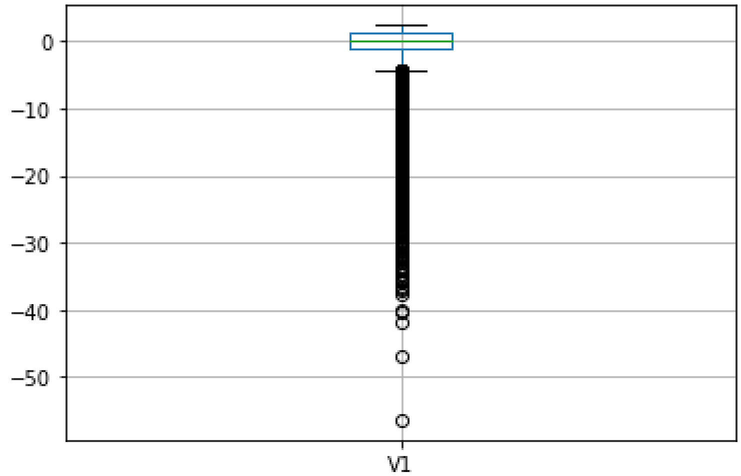


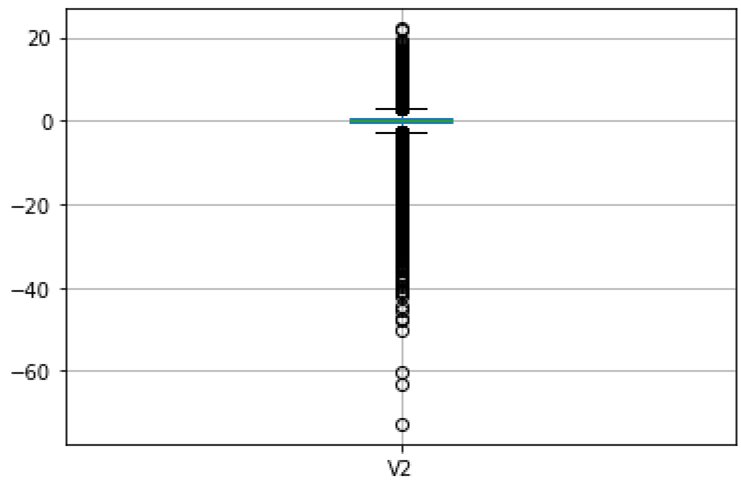


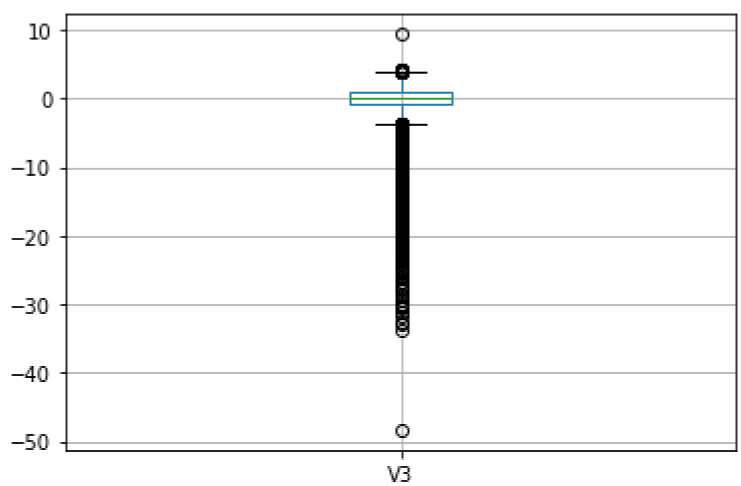


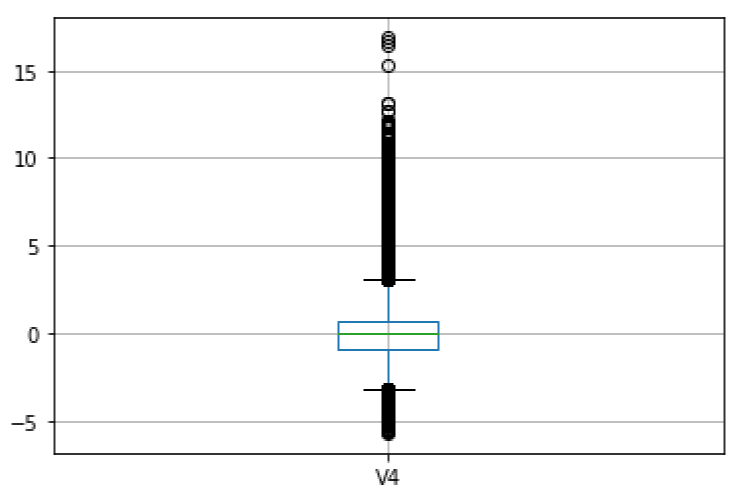
In [9]:

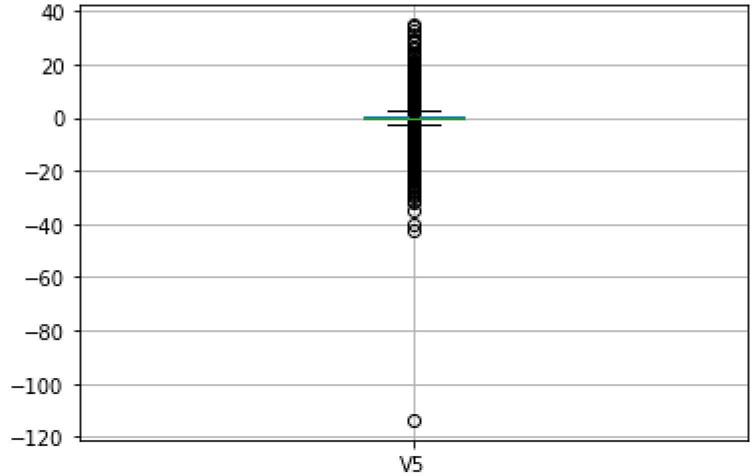
```
for column in contin_feat_names:
    df.boxplot(column)
    plt.show()
```


0
−10
−20
−30
−40
−50
V1

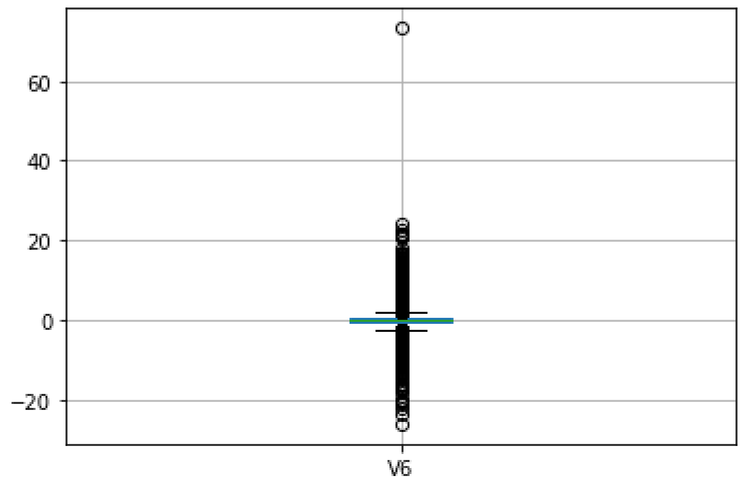

20
0
−20
−40
−60
V2


10
0
−10
−20
−30
−40
−50
V3

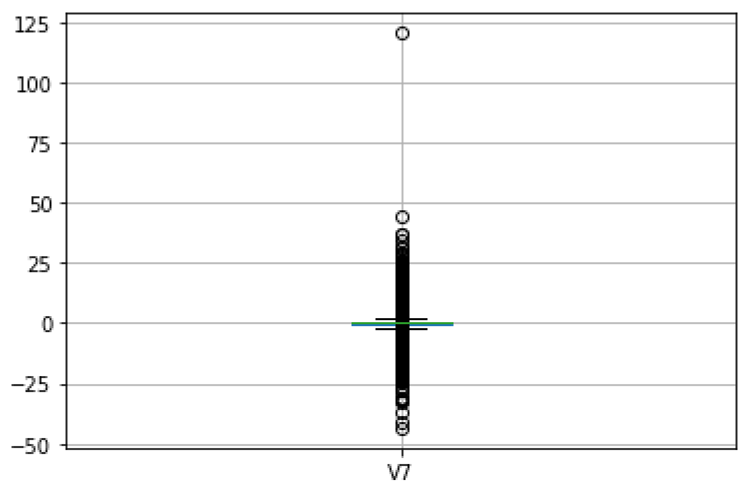

15
10
5
0
−5
V4



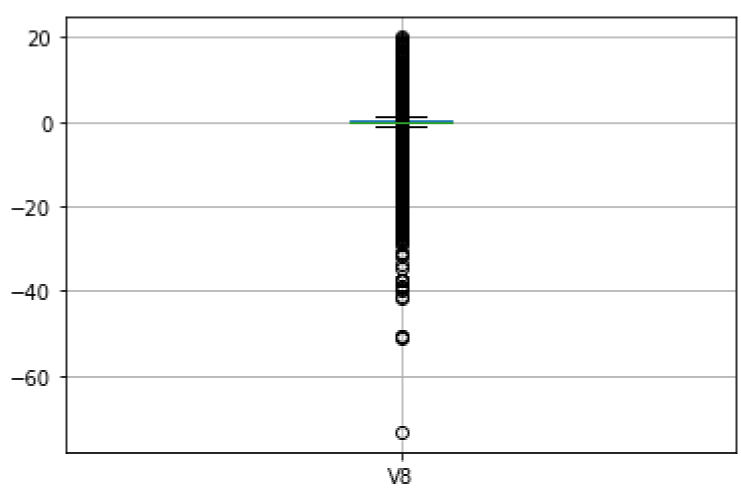
40
20
0
−20
−40
−60
−80
−100
−120
V5



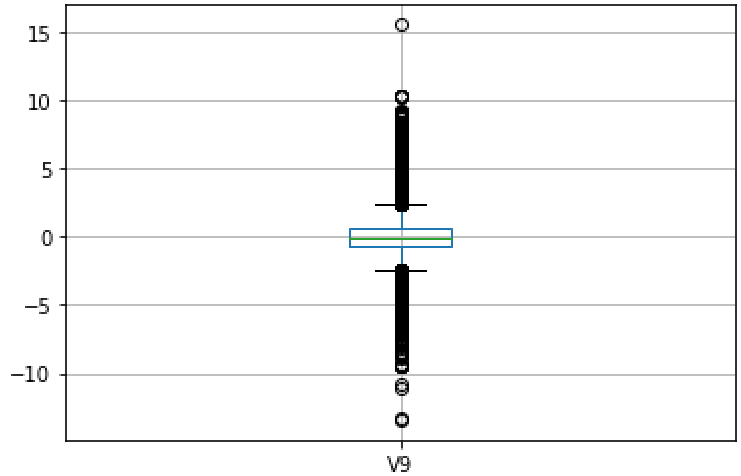
60
40
20
0
−20
V6

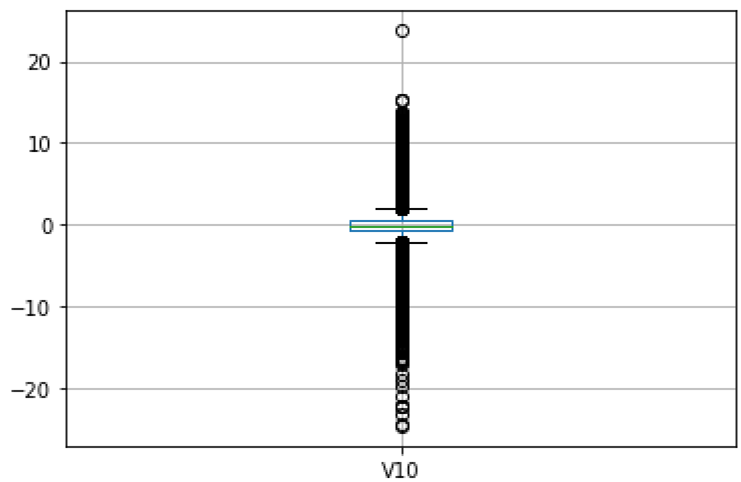


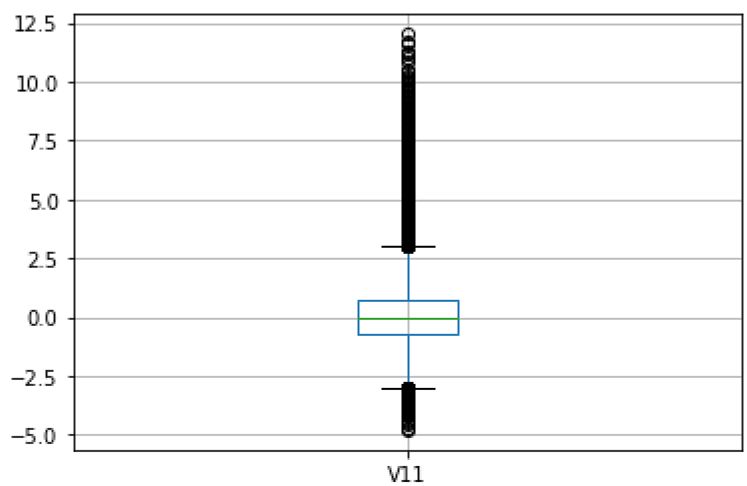
125
100
75
50
25
0
−25
−50
V7

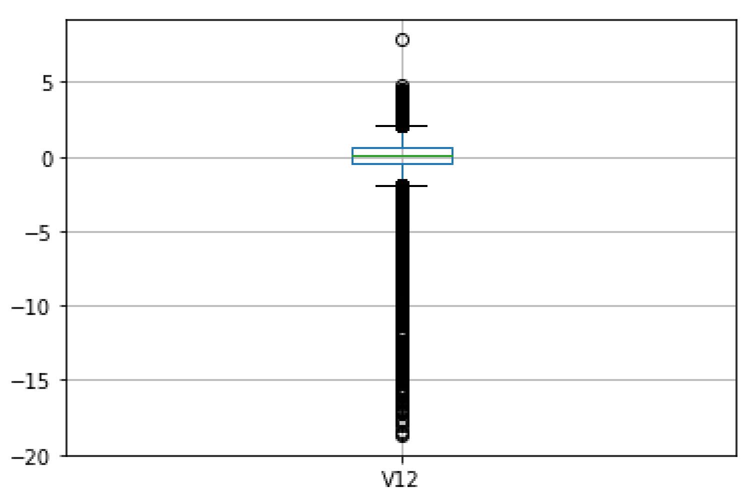


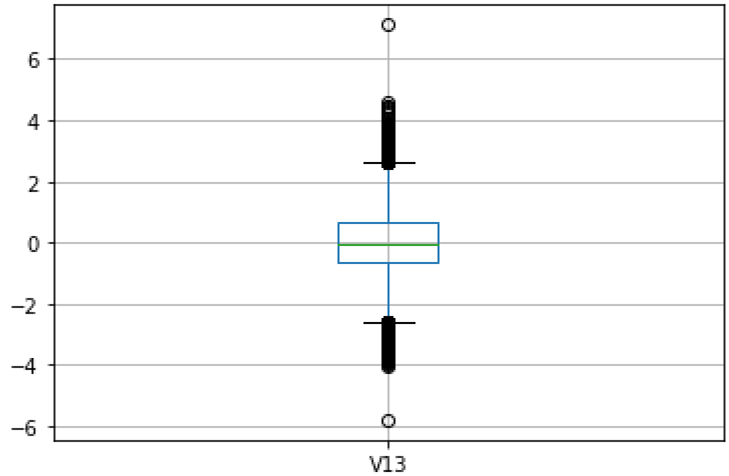
20
0
−20
−40
−60
V8

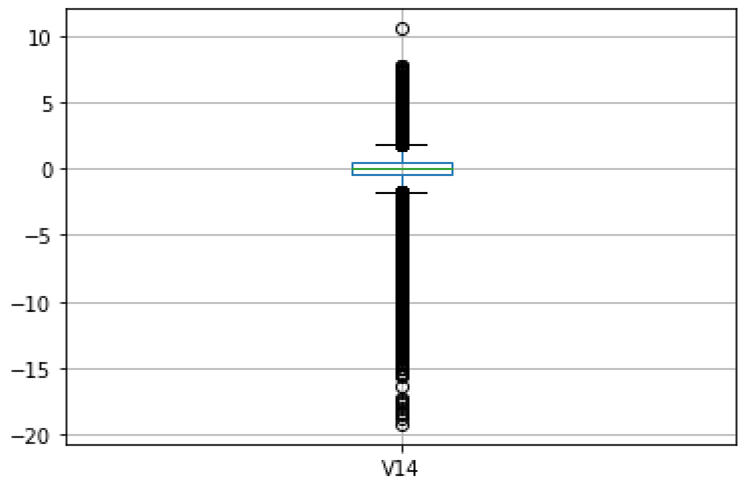

15
10
5
0
−5
−10
V9

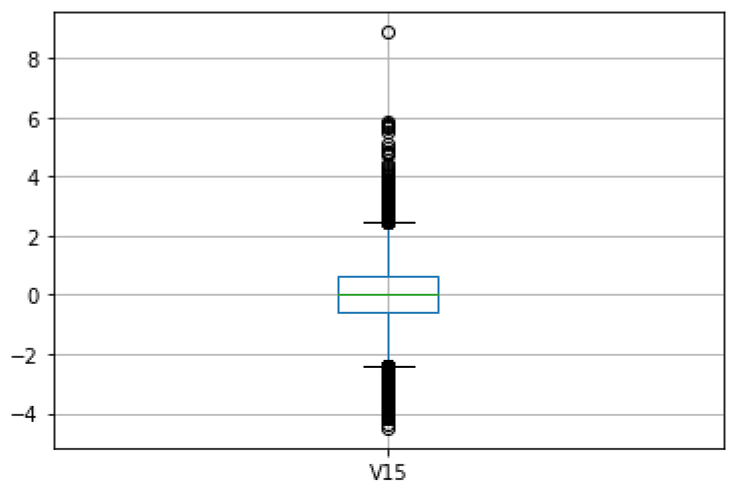

20
10
0
−10
−20
V10

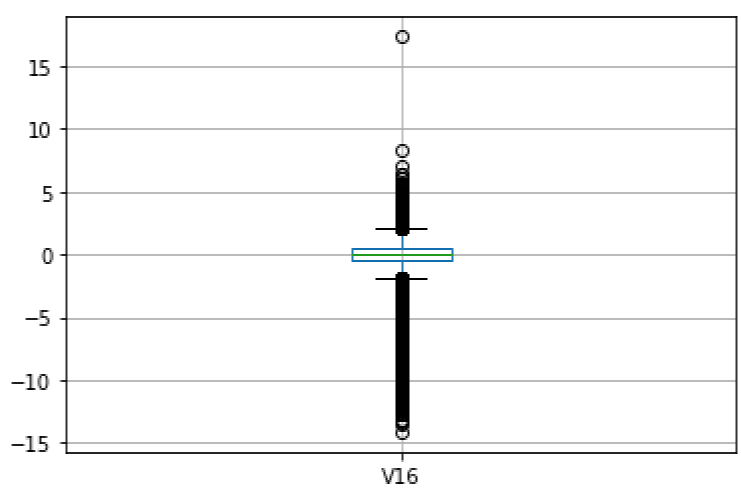

12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
V11

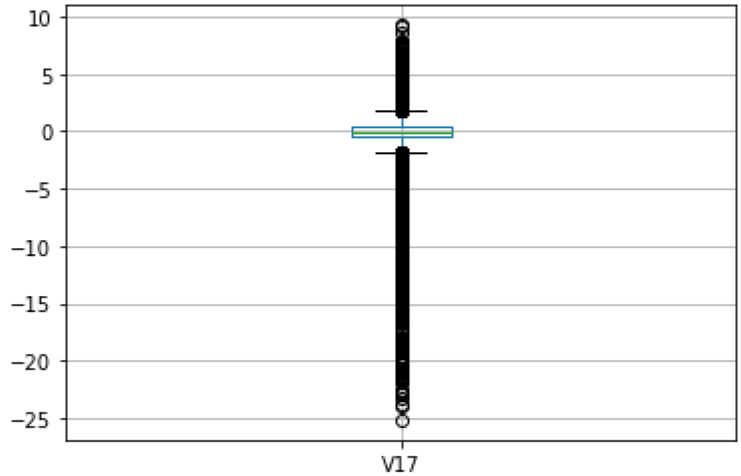

5
0
−5
−10
−15
−20
V12


6
4
2
0
−2
−4
−6
V13

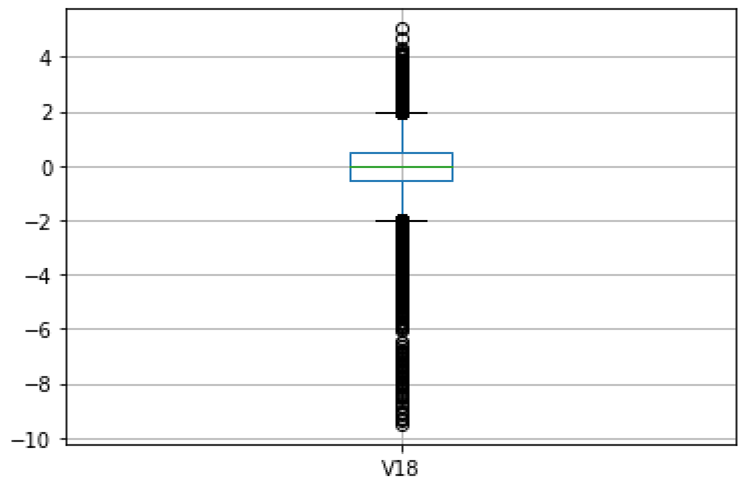

10
5
0
−5
−10
−15
−20
V14


8
6
4
2
0
−2
−4
V15

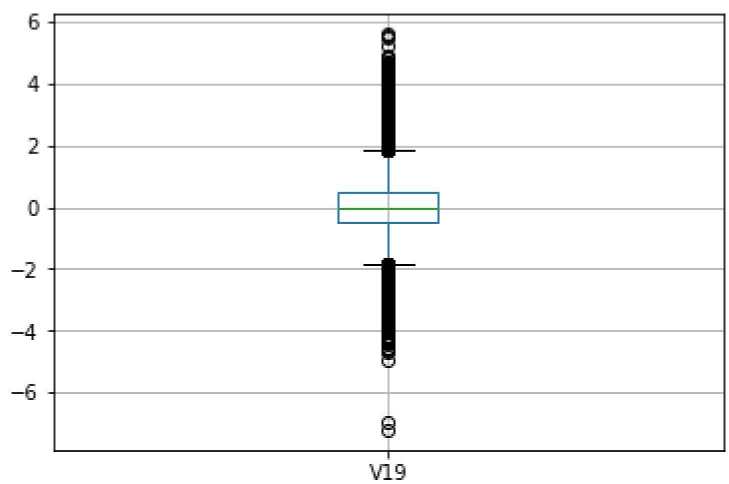

15
10
5
0
−5
−10
−15
V16



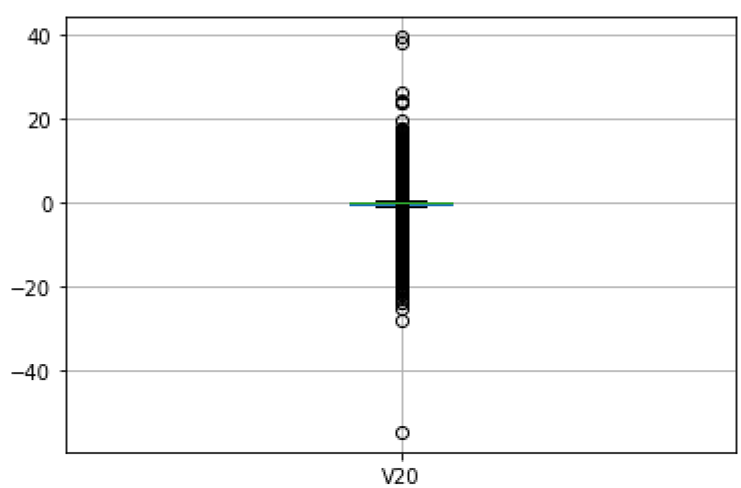
10
5
0
−5
−10
−15
−20
−25
V17



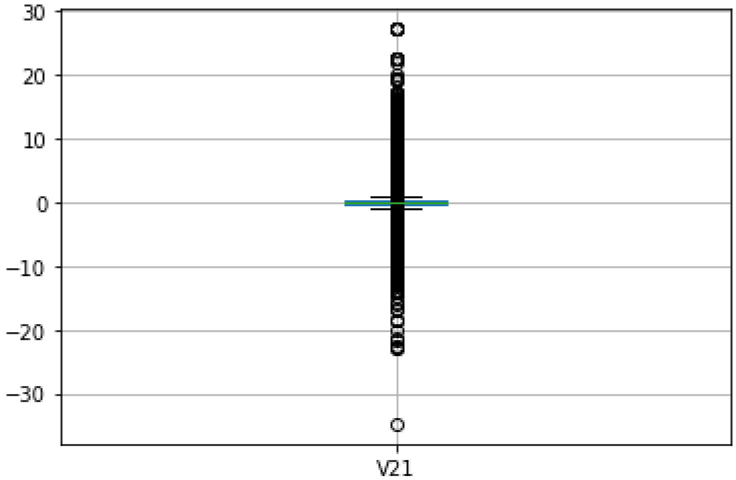
4
2
0
−2
−4
−6
−8
−10
V18



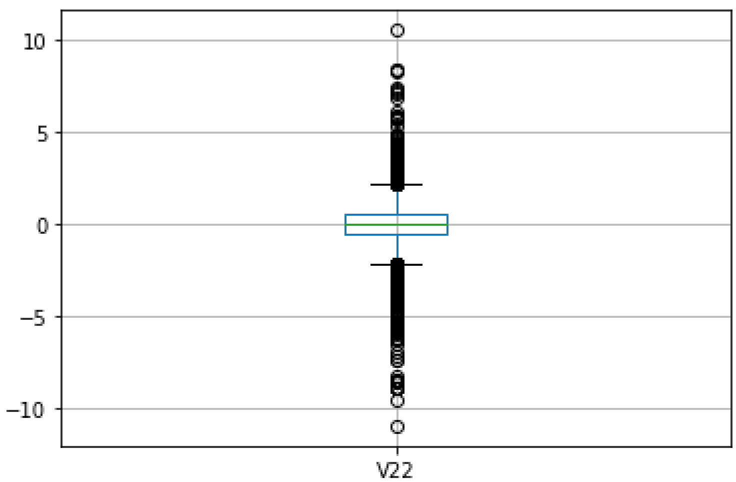
6
4
2
0
−2
−4
−6
V19



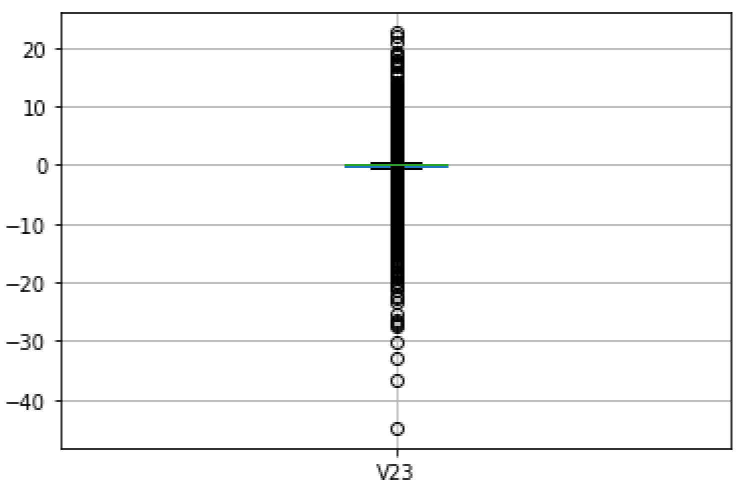
40
20
0
−20
−40
V20



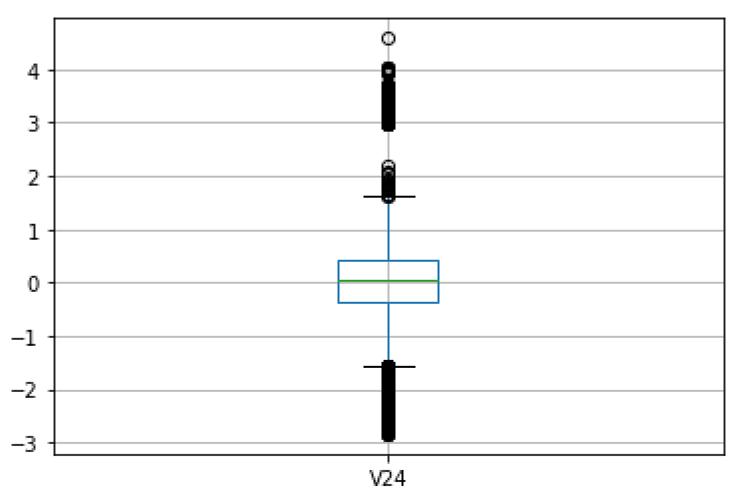
30
20
10
0
−10
−20
−30
V21



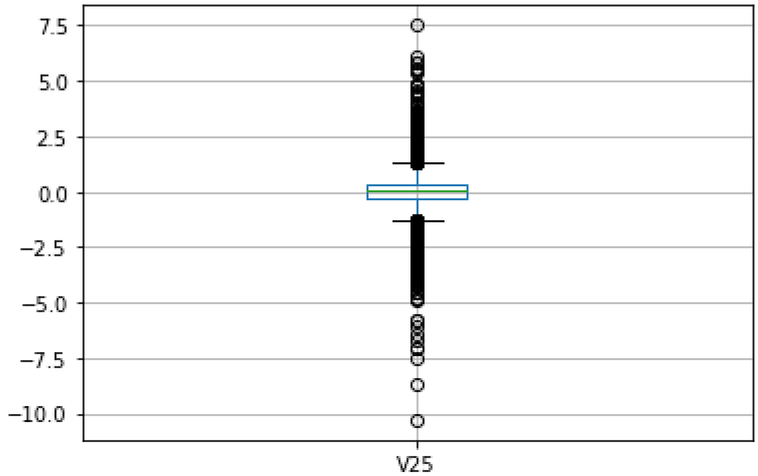
10
5
0
−5
−10
V22

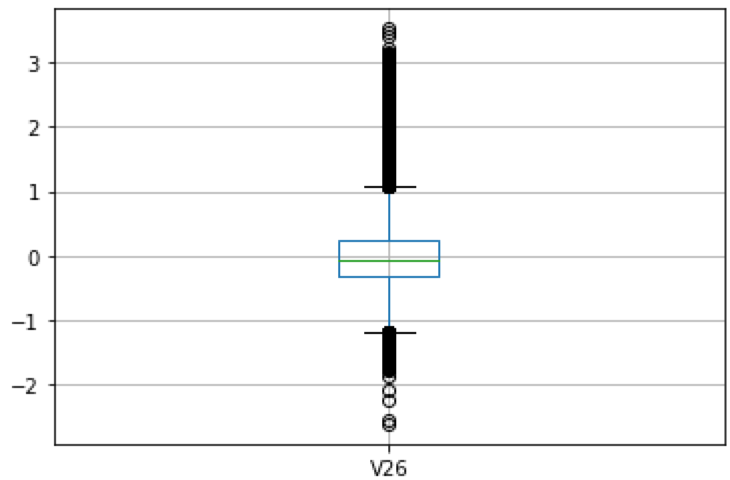


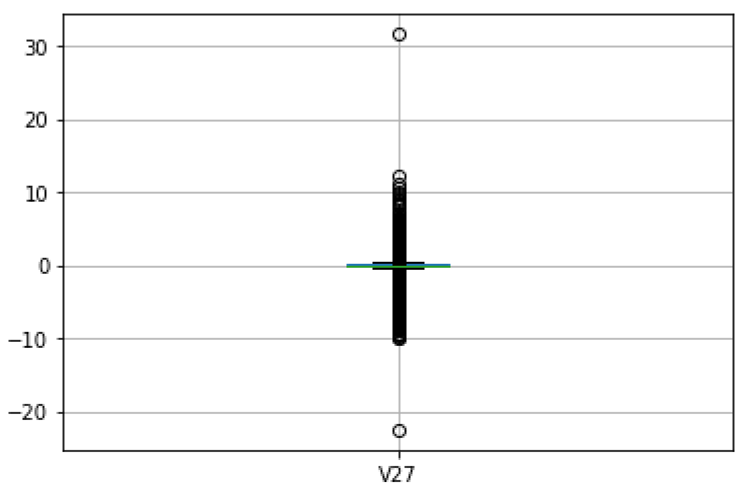
20
10
0
−10
−20
−30
−40
V23

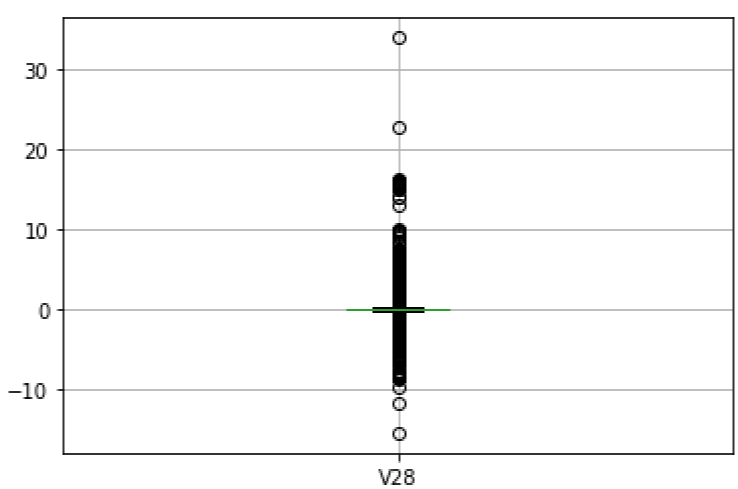


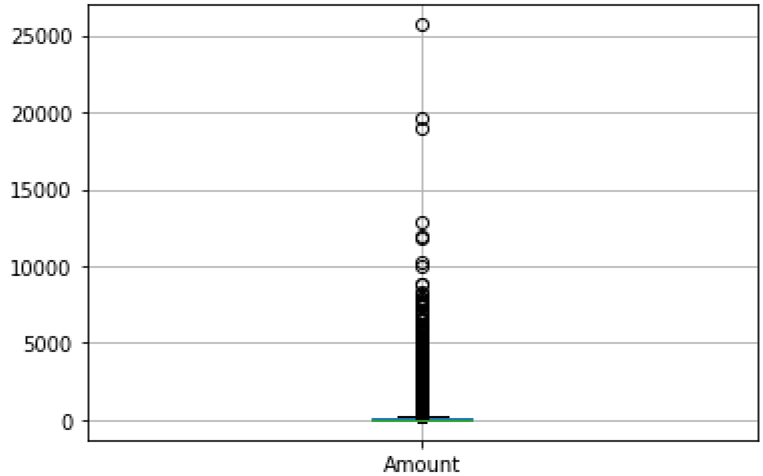
4
3
2
1
0
−1
−2
−3
V24


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


3
2
1
0
−1
−2
V26


30
20
10
0
−10
−20
V27


30
20
10
0
−10
V28



1.2 Data Quality Issues and Data Quality Plan

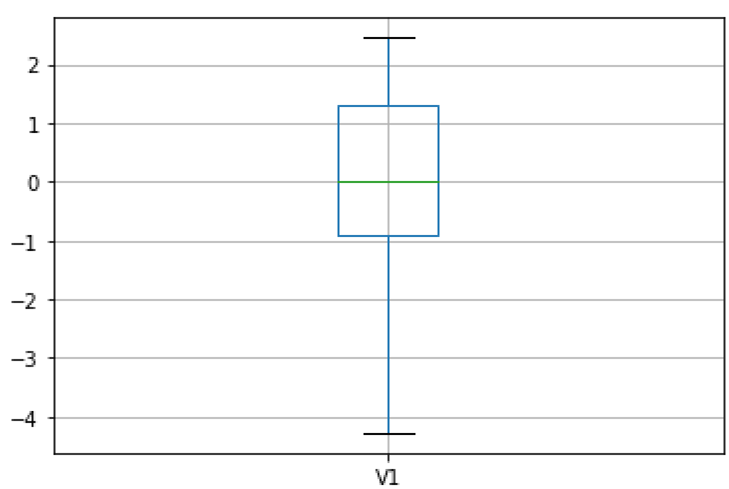
The dataset contains a large number of outliers and removing them would cause even greater class imbalance. Instead, the outlying values will be replaced with a capped min/max value.

1.3 Data Preprocessing

In [10]:

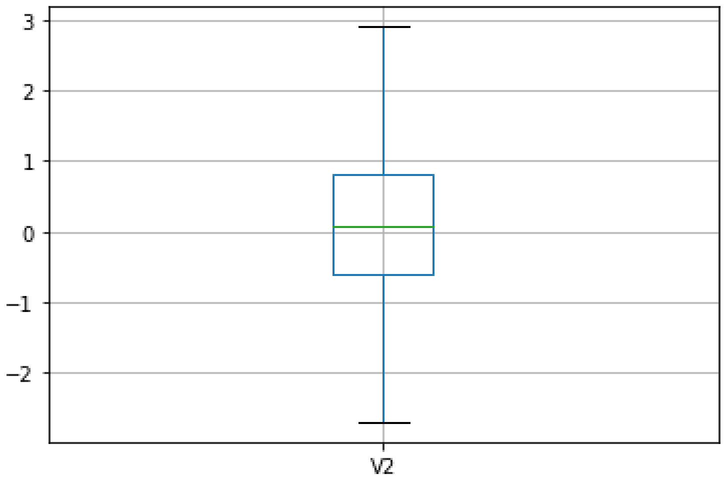
```
# Replace the outliers with capped values
# code reference: Tutorial [1]
for column in contin_feat_names:

  Q1 = df[column].quantile(0.25)
  Q3 = df[column].quantile(0.75)

  IQR = Q3-Q1
  print(f"IQR = {Q3} - {Q1} = {IQR}")
  outliers_df1 = df[(df[column] < (Q1 - 1.5 * IQR)) ]
  df[column][outliers_df1.index] = (Q1 - 1.5 * IQR)

  outliers_df2 = df[(df[column] > (Q3 + 1.5 * IQR)) ]
  df[column][outliers_df2.index] = (Q3 + 1.5 * IQR)

  tot_outliers = len(outliers_df1) + len(outliers_df2)
  print(column, "Num of outliers: ",  tot_outliers, "Percent outliers: ", tot_outliers/ler

  df.boxplot(column)
  plt.show()
```

```
IQR = 1.315641693877865 - -0.920373384390322 = 2.236015078268187
V1 Num of outliers:  7062 Percent outliers:  2.4795738868777804
```

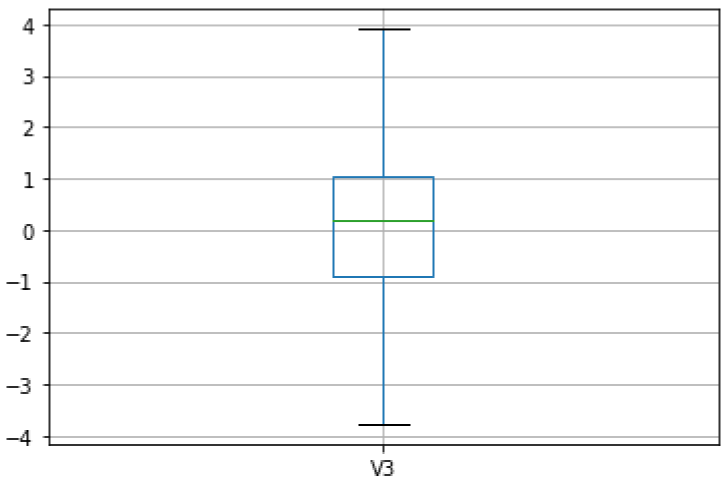



```
IQR = 0.8037238712400945 - -0.598549913464916 = 1.4022737847050104
```

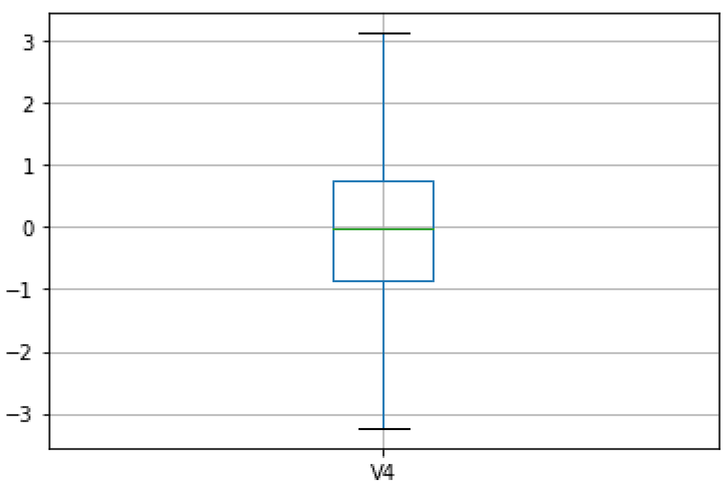
```
V2 Num of outliers:  13526 Percent outliers:  4.7491810243428
```



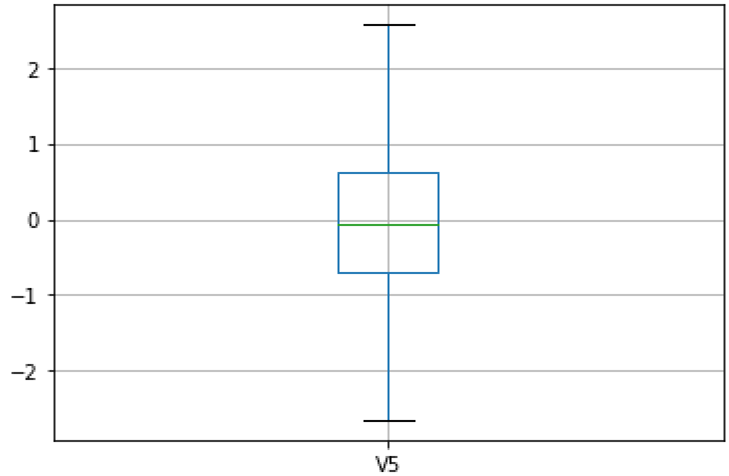

```
IQR = 1.027195542465555 - -0.8903648381551406 = 1.9175603806206956
V3 Num of outliers:  3363 Percent outliers:  1.1807996292225964
```



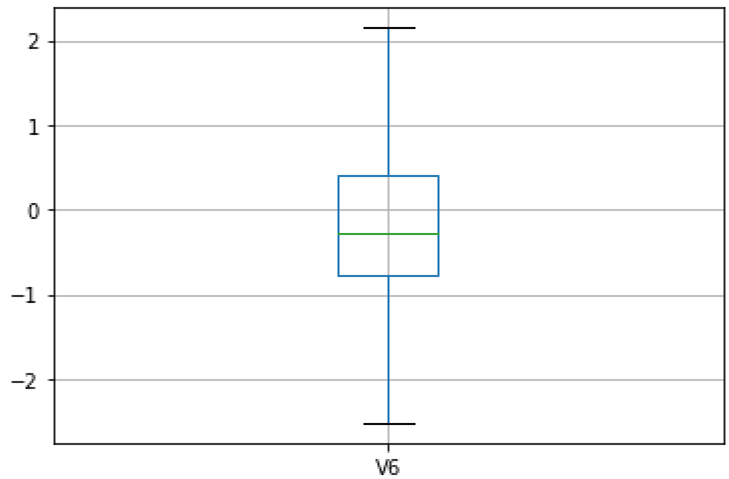

```
IQR = 0.7433412894685876 - -0.848640116331273 = 1.5919814057998605
V4 Num of outliers:  11148 Percent outliers:  3.9142296362097846
```



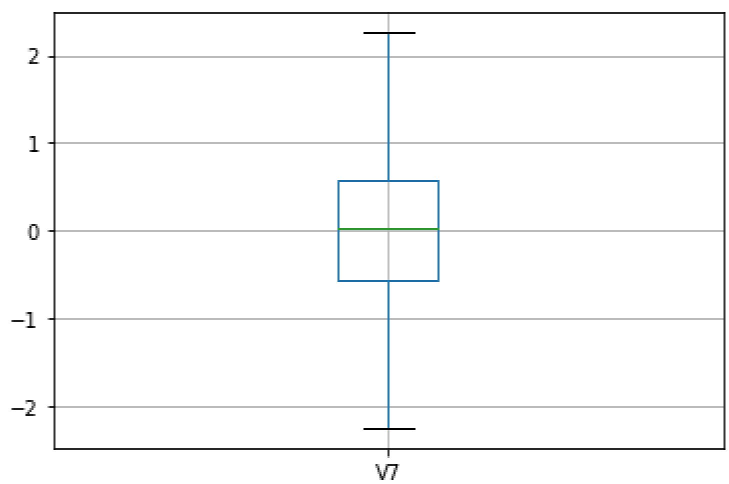

```
IQR = 0.611926439735193 - -0.6915970708876575 = 1.3035235106228504
V5 Num of outliers:  12295 Percent outliers:  4.316958501722219
```



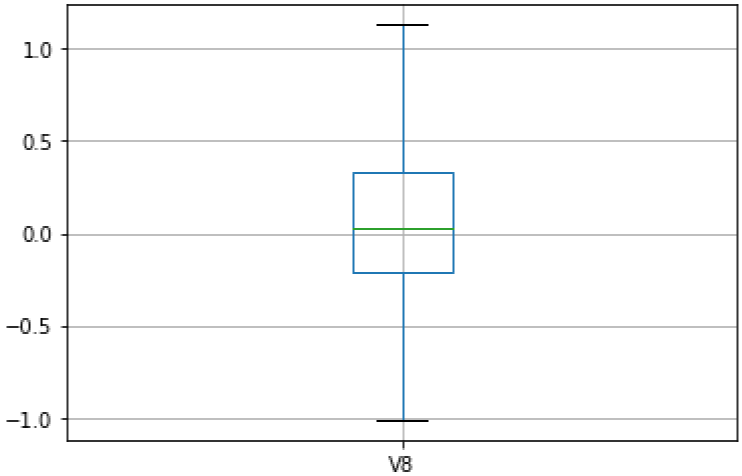

```
IQR = 0.39856489635610504 - -0.768295608460489 = 1.166860504816594
V6 Num of outliers:  22965 Percent outliers:  8.063355184388024
```



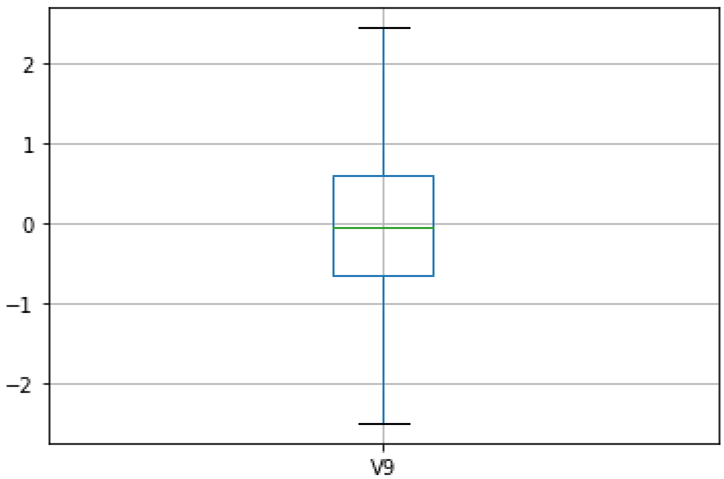

```
IQR = 0.5704360728775986 - -0.5540758790365226 = 1.1245119519141211
V7 Num of outliers:  8948 Percent outliers:  3.1417767119487934
```



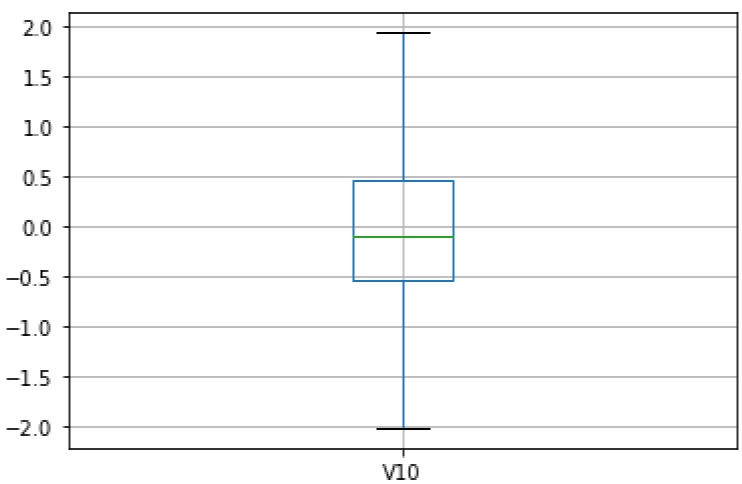

```
IQR = 0.327345861923449 - -0.2086297440394665 = 0.5359756059629155
V8 Num of outliers:  24134 Percent outliers:  8.47380857914307
```



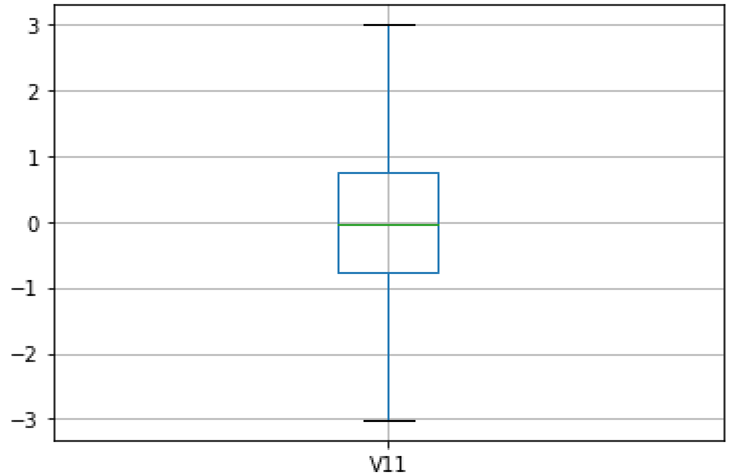

```
IQR = 0.5971390302822686 - -0.6430975702665915 = 1.24023660054886
V9 Num of outliers:  8283 Percent outliers:  2.9082852598426303
```



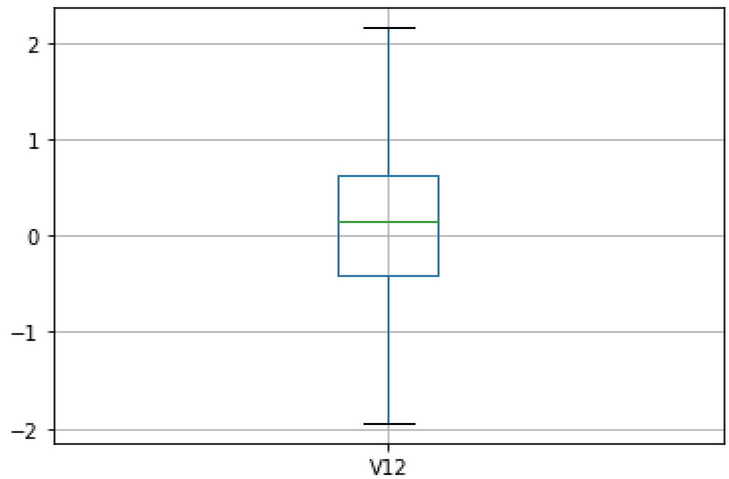

```
IQR = 0.453923445139507 - -0.5354257264933235 = 0.9893491716328305
V10 Num of outliers:  9496 Percent outliers:  3.334187713082895
```



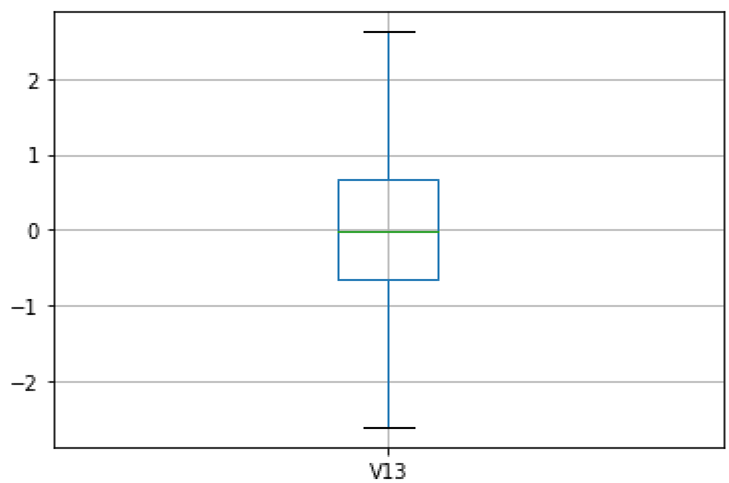

```
IQR = 0.739593407321606 - -0.7624941955129775 = 1.5020876028345835
V11 Num of outliers:  780 Percent outliers:  0.27386967314707855
```



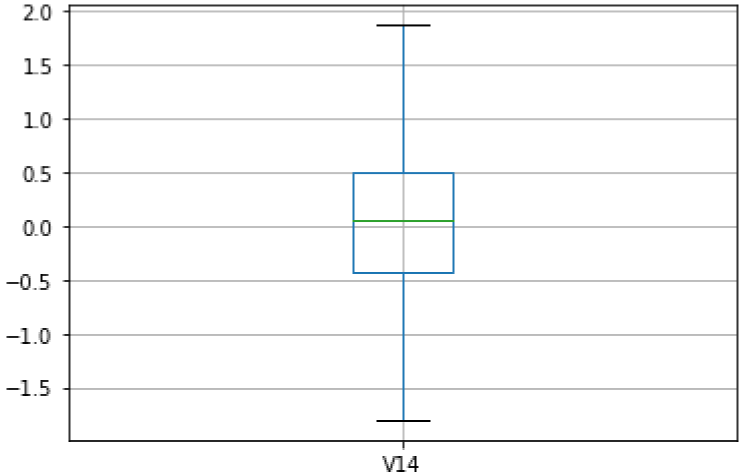

```
IQR = 0.618238032946136 - -0.40557148544041355 = 1.0238095183865497
V12 Num of outliers:  15348 Percent outliers:  5.3889124916171305
```



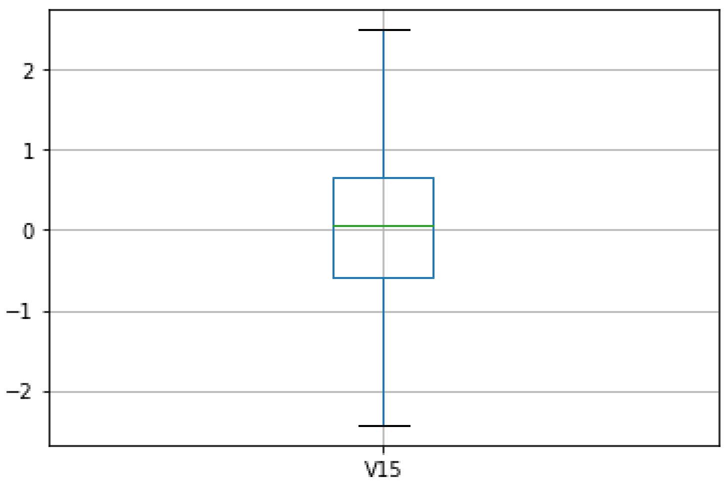

```
IQR = 0.662504959439974 - -0.6485392991145684 = 1.3110442585545425
V13 Num of outliers:  3368 Percent outliers:  1.182555204050462
```



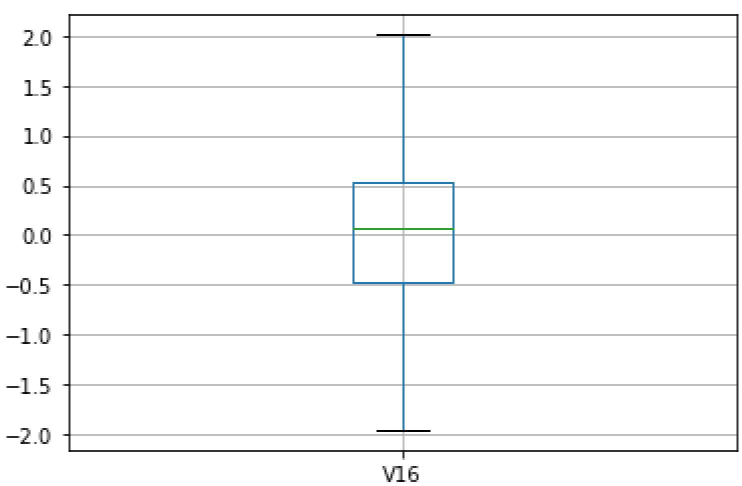

```
IQR = 0.493149849218149 - -0.4255740124549935 = 0.9187238616731425
V14 Num of outliers:  14149 Percent outliers:  4.96792564789489
```



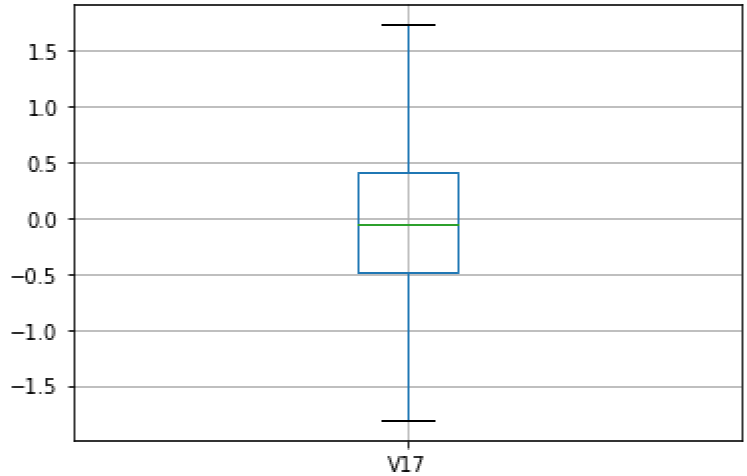

```
IQR = 0.648820806317158 - -0.582884279157456 = 1.2317050854746139
V15 Num of outliers:  2894 Percent outliers:  1.0161267103687759
```



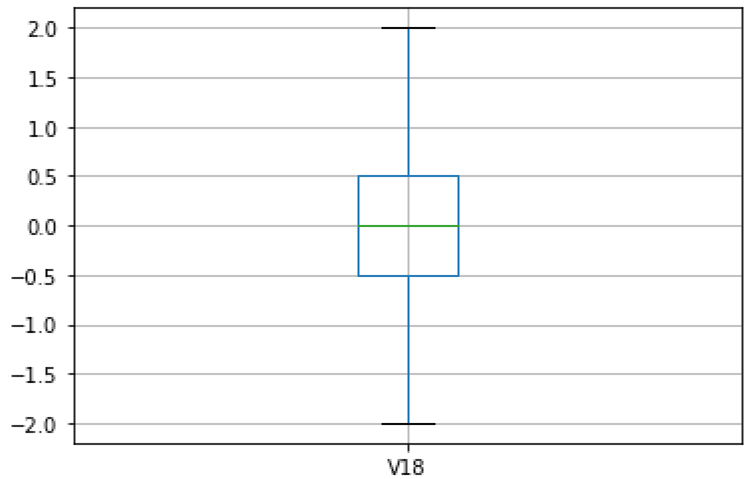

```
IQR = 0.523296312475344 - -0.46803676671289796 = 0.991333079188242
V16 Num of outliers:  8184 Percent outliers:  2.873524878250886
```



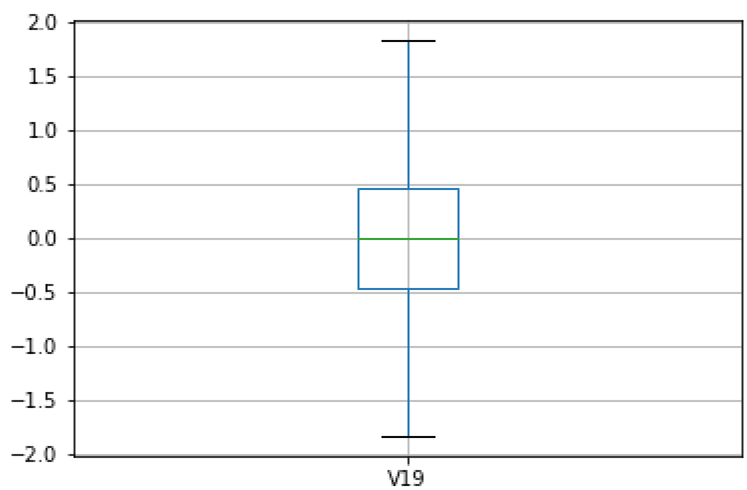

```
IQR = 0.3996749826503845 - -0.483748313707048 = 0.8834232963574324
V17 Num of outliers:  7420 Percent outliers:  2.605273044552978
```



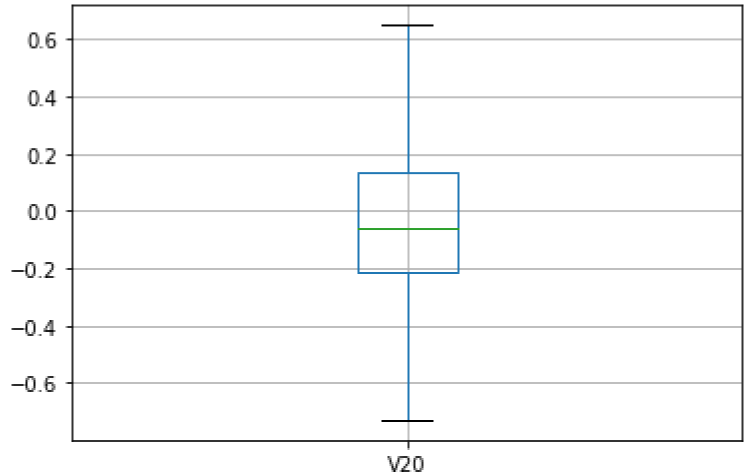
IQR = 0.5008067468872159 - -0.498849798665041 = 0.9996565455522569
V18 Num of outliers: 7533 Percent outliers: 2.644949035662747



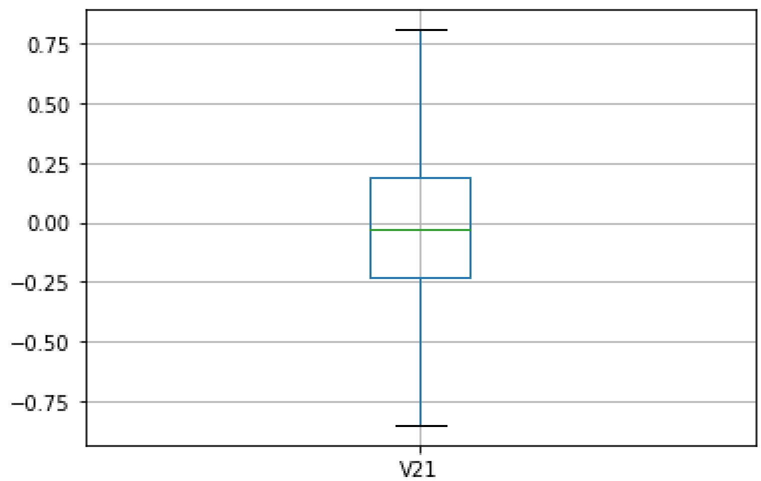
IQR = 0.458949355762679 - -0.4562989187444475 = 0.9152482745071264
V19 Num of outliers: 10205 Percent outliers: 3.5831282236742776



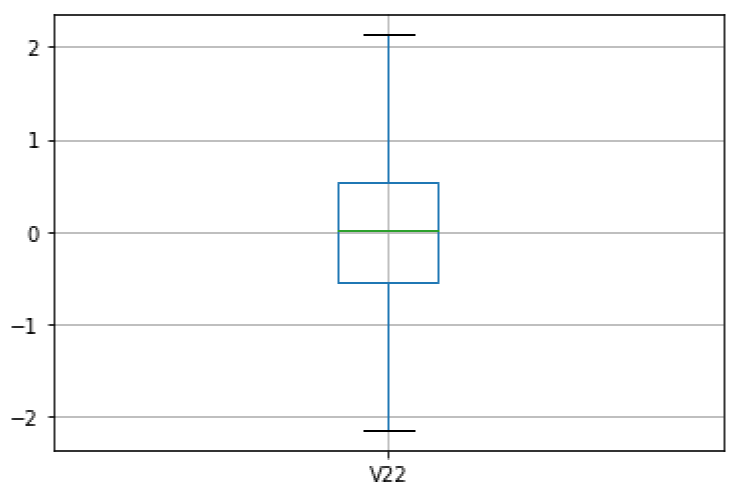
IQR = 0.1330408409942945 - -0.21172136467424701 = 0.34476220566854154
V20 Num of outliers: 27770 Percent outliers: 9.750462593967143



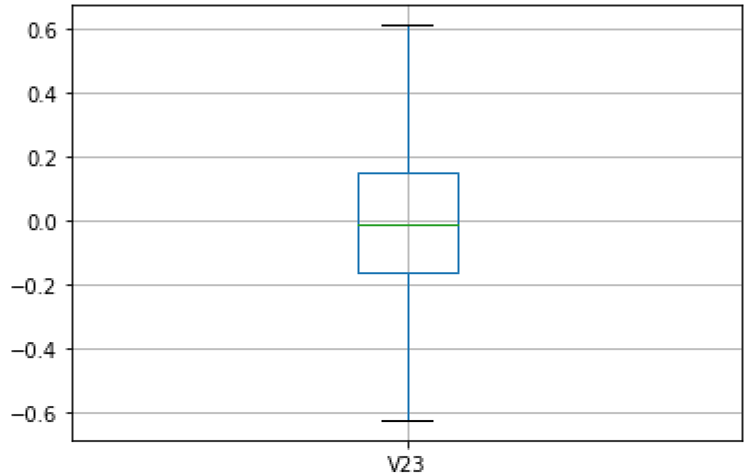
IQR = 0.1863772033785755 - -0.22839494677851702 = 0.4147721501570925
V21 Num of outliers: 14497 Percent outliers: 5.090113655914355



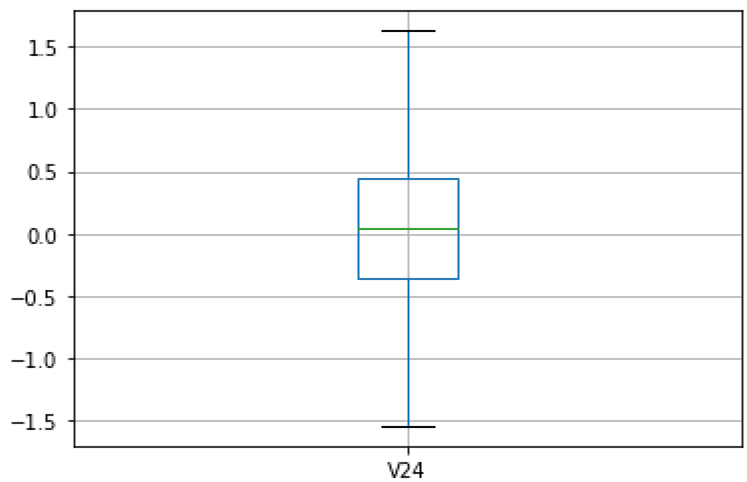
IQR = 0.5285536353339865 - -0.5423503726606616 = 1.0709040079946481
V22 Num of outliers: 1317 Percent outliers: 0.4624184096598749



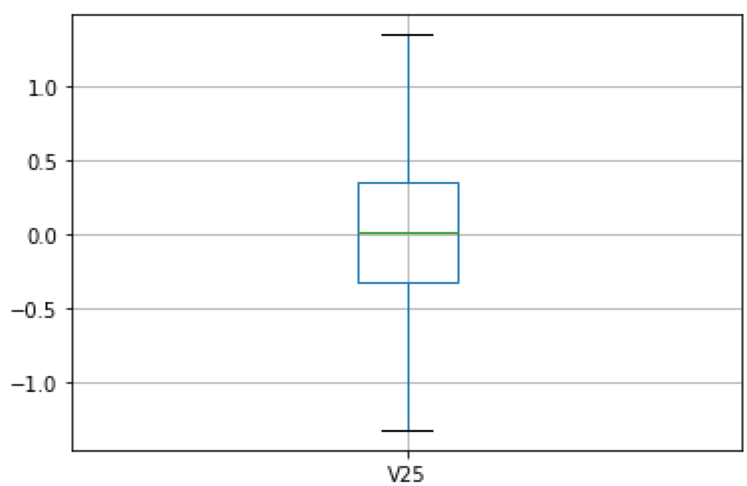
IQR = 0.14764206385605 - -0.16184634501488449 = 0.3094884088709345
V23 Num of outliers: 18541 Percent outliers: 6.510022576692287



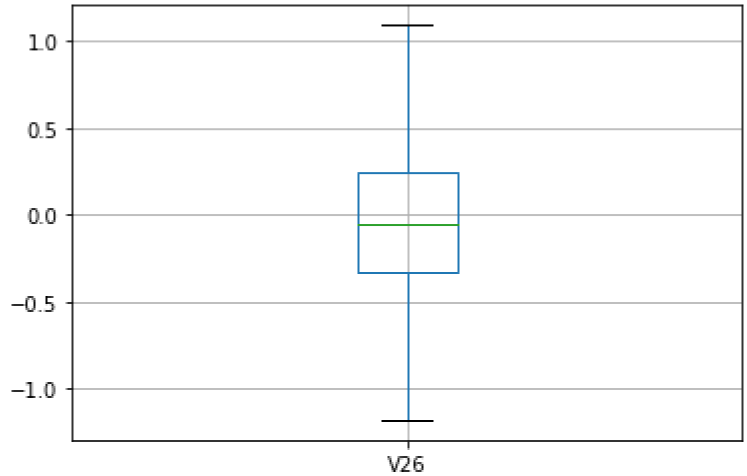
IQR = 0.439526600168186 - -0.3545861364094985 = 0.7941127365776846
V24 Num of outliers: 4774 Percent outliers: 1.67622284564635



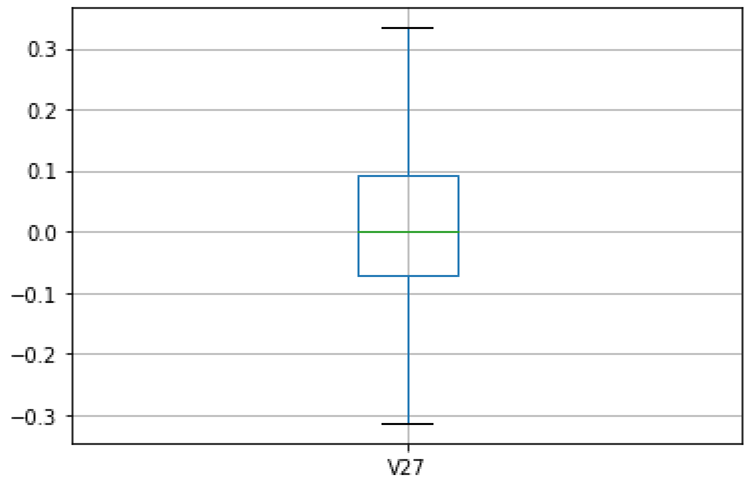
IQR = 0.350715562867386 - -0.31714505406527 = 0.667860616932656
V25 Num of outliers: 5367 Percent outliers: 1.8844340202312442



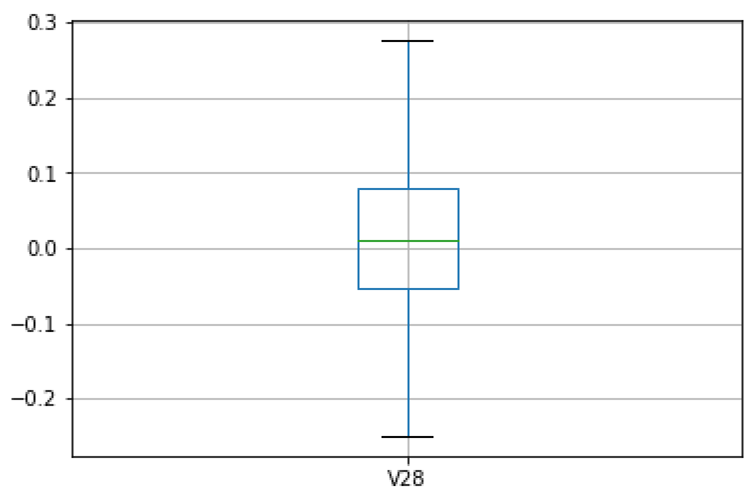
IQR = 0.2409521737147555 - -0.3269839258807195 = 0.567936099595475
V26 Num of outliers: 5596 Percent outliers: 1.9648393473475019



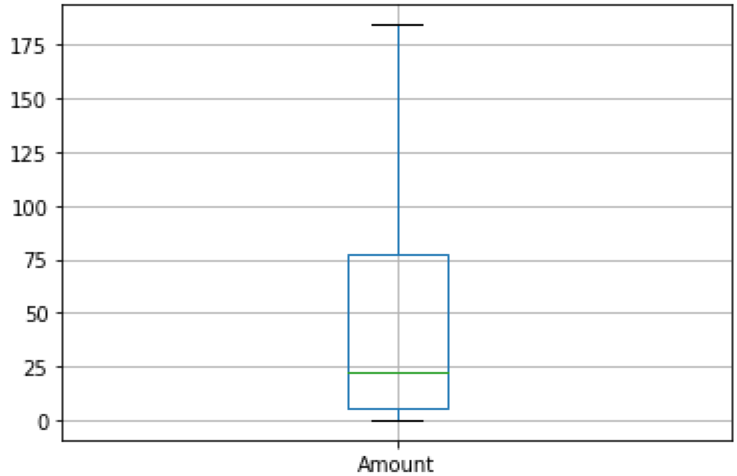
IQR = 0.09104511968580689 - -0.07083952930446921 = 0.1618846489902761
V27 Num of outliers: 39163 Percent outliers: 13.750715396742356



IQR = 0.07827995475782015 - -0.0529597930169809 = 0.13123974777480105
V28 Num of outliers: 30342 Percent outliers: 10.653530285421356

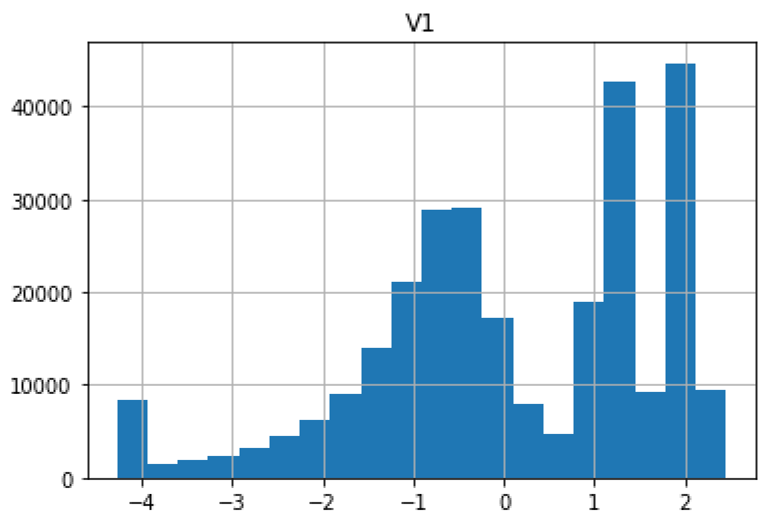


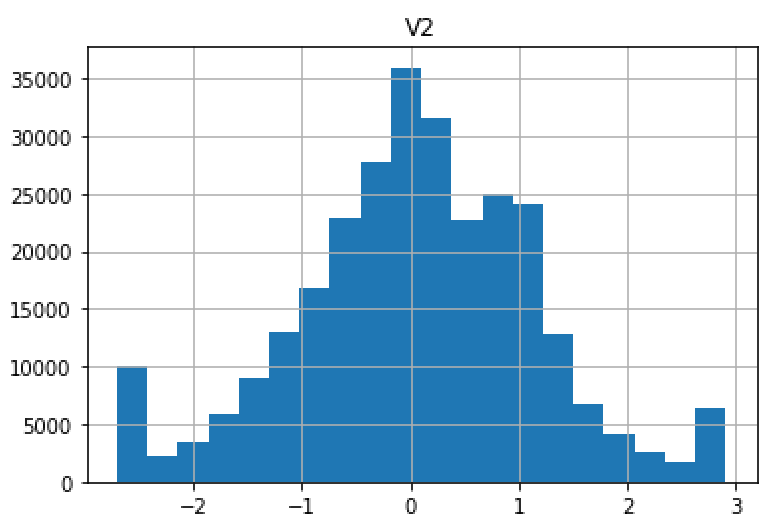
IQR = 77.16499999999999 - 5.6 = 71.565
Amount Num of outliers: 31904 Percent outliers: 11.201971861646658

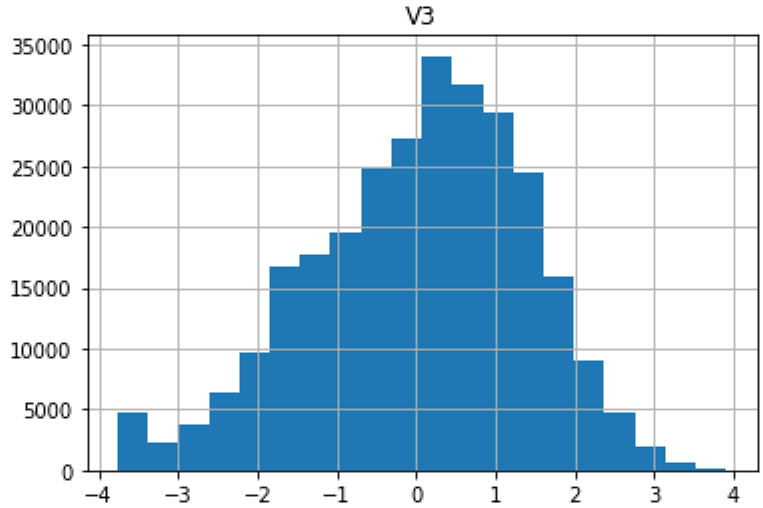


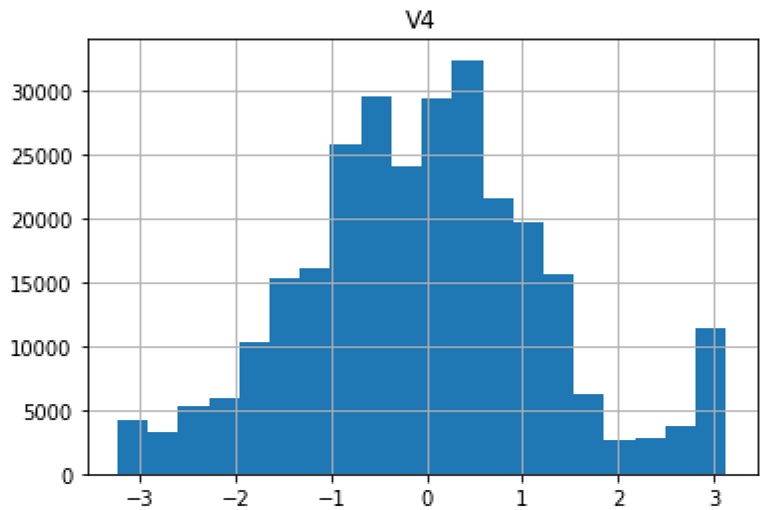
In [11]:

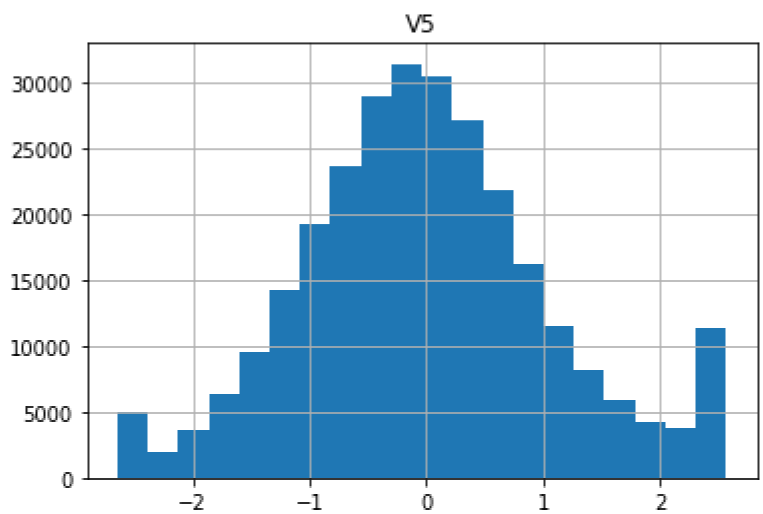
```
# replot the feature distributions
for column in contin_feat_names:
    range = int(df[column].max() - df[column].min())
    df.hist(column=column, bins=20)
    plt.show()
```

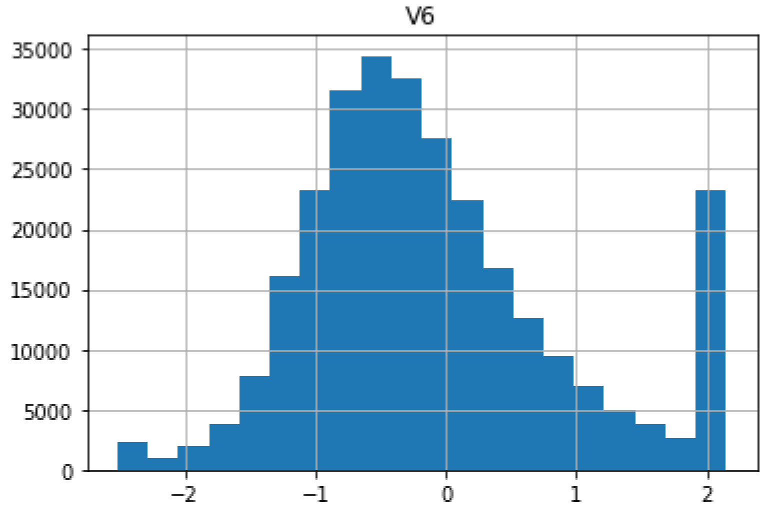


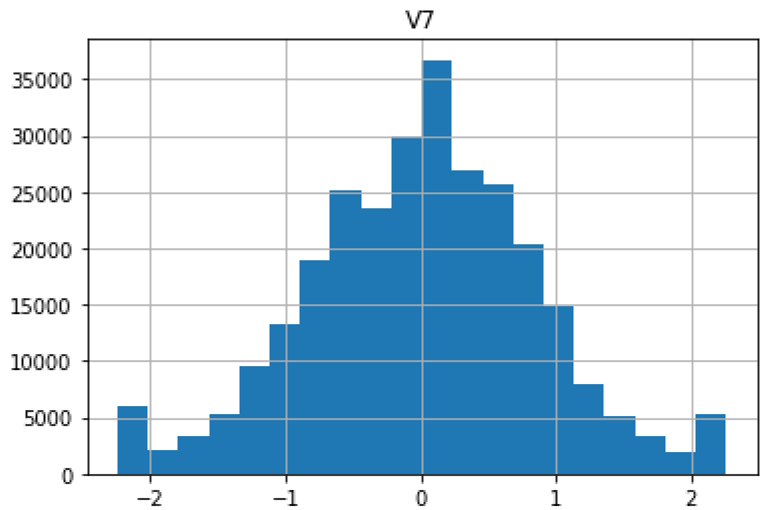


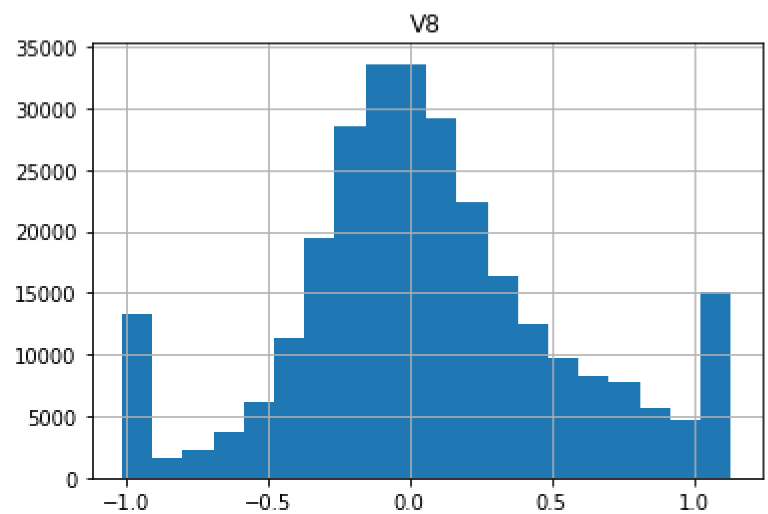

V3
35000
30000
25000
20000
15000
10000
5000
0
−4
−3
−2
−1
0
1
2
3
4

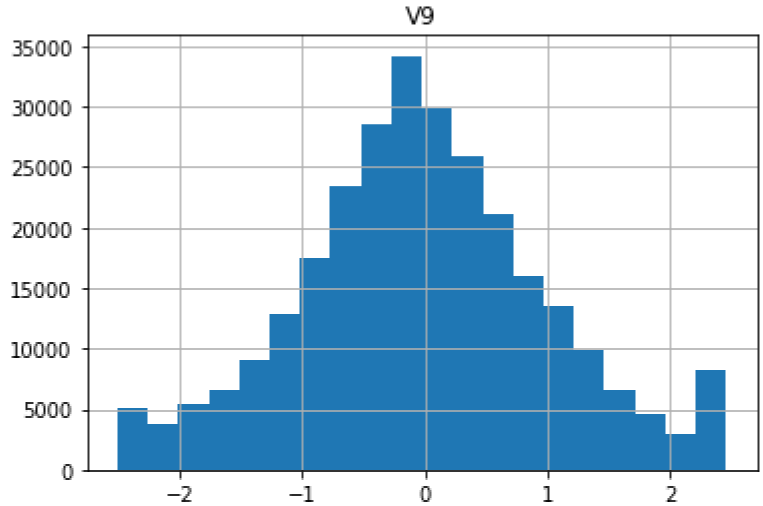

V4
30000
25000
20000
15000
10000
5000
0
−3
−2
−1
0
1
2
3

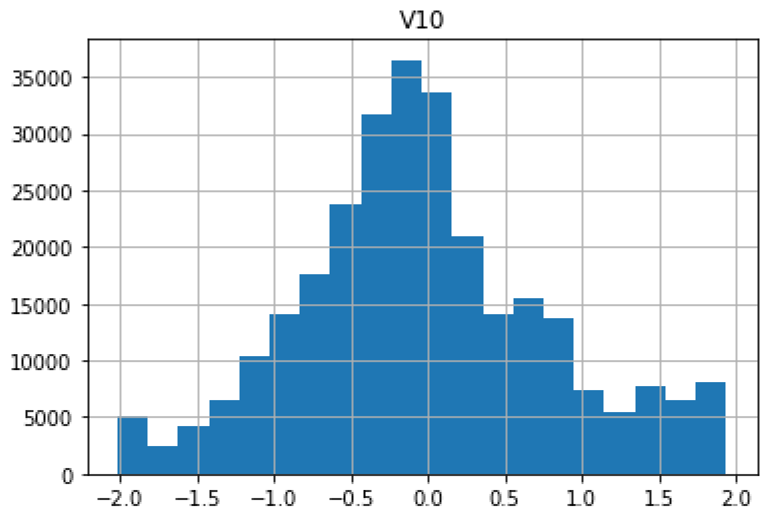

V5
30000
25000
20000
15000
10000
5000
0
−2
−1
0
1
2

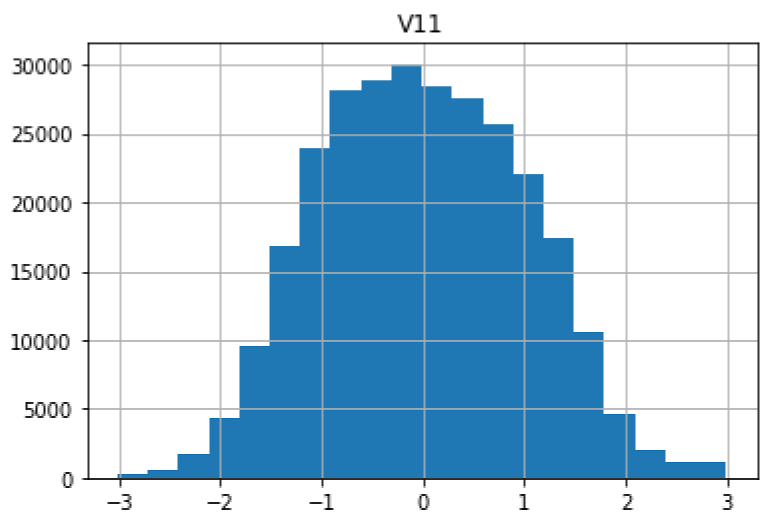

V6
35000
30000
25000
20000
15000
10000
5000
0
−2
−1
0
1
2

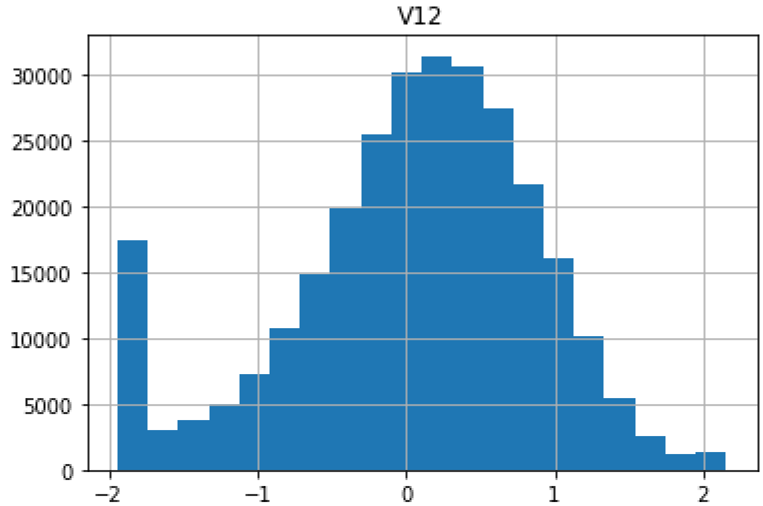

V7
35000
30000
25000
20000
15000
10000
5000
0
−2
−1
0
1
2

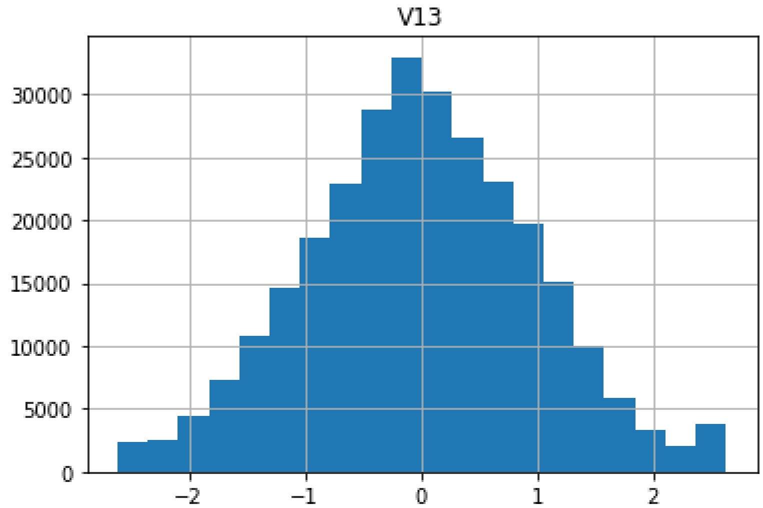

V8
35000
30000
25000
20000
15000
10000
5000
0
−1.0
−0.5
0.0
0.5
1.0

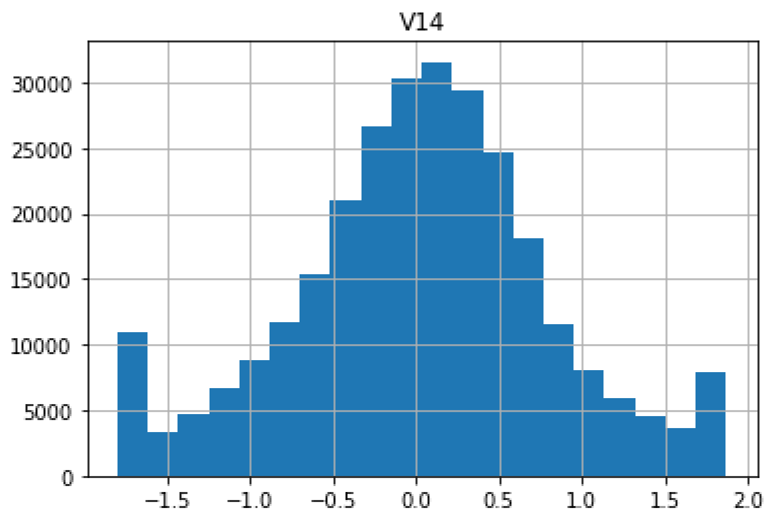

V9
35000
30000
25000
20000
15000
10000
5000
0
−2
−1
0
1
2

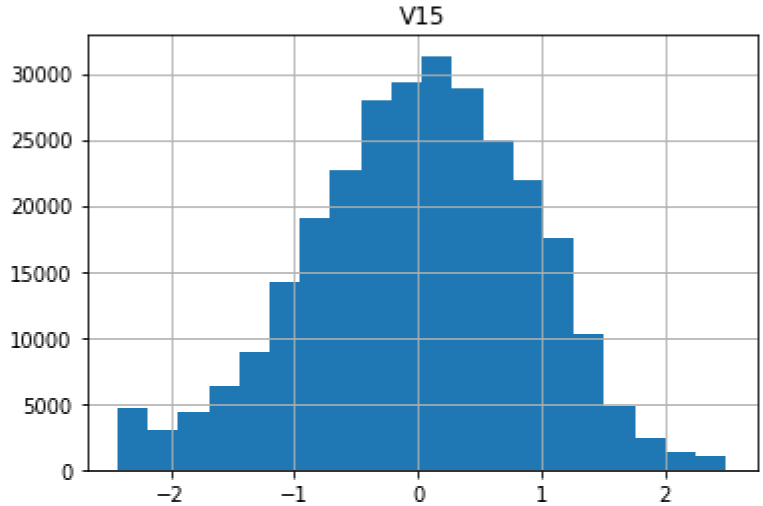

V10
35000
30000
25000
20000
15000
10000
5000
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

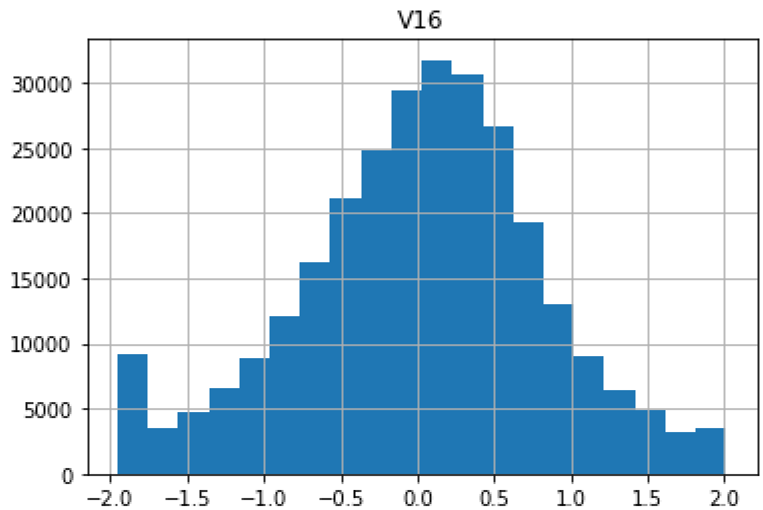

V11
30000
25000
20000
15000
10000
5000
0
−3
−2
−1
0
1
2
3

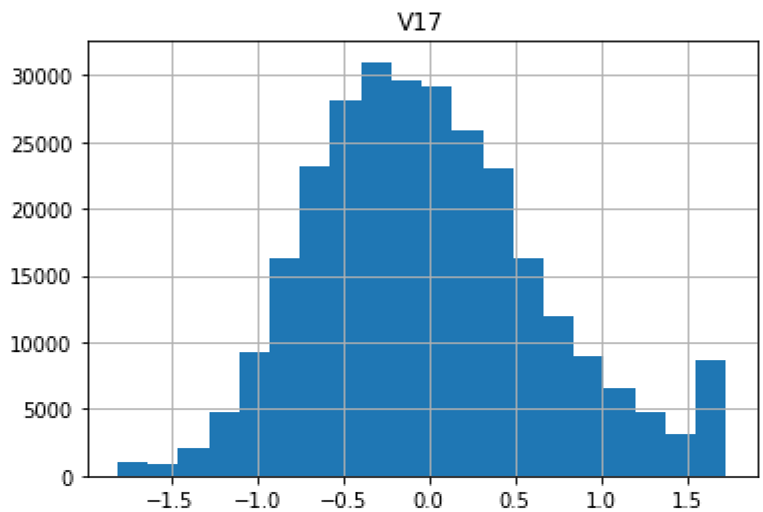

V12
30000
25000
20000
15000
10000
5000
0
−2
−1
0
1
2

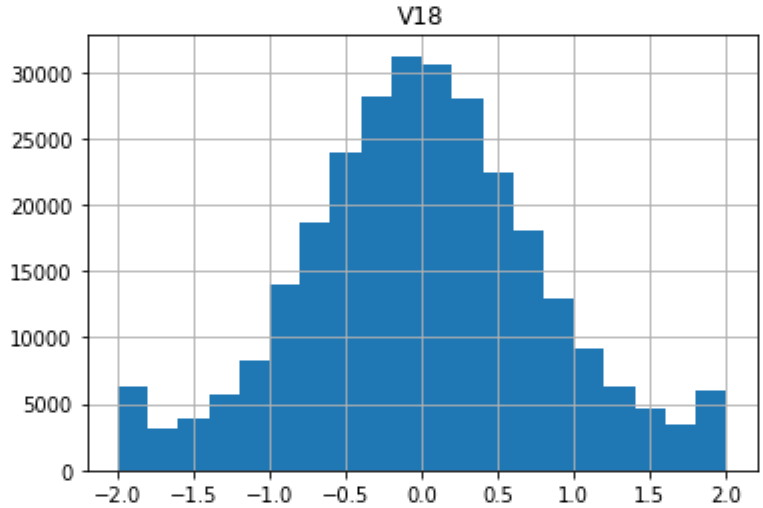

V13
30000
25000
20000
15000
10000
5000
0
−2
−1
0
1
2

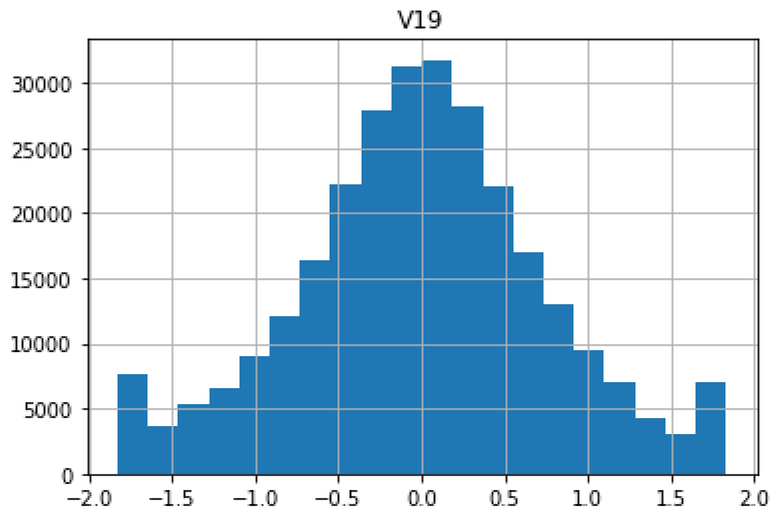

V14
30000
25000
20000
15000
10000
5000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

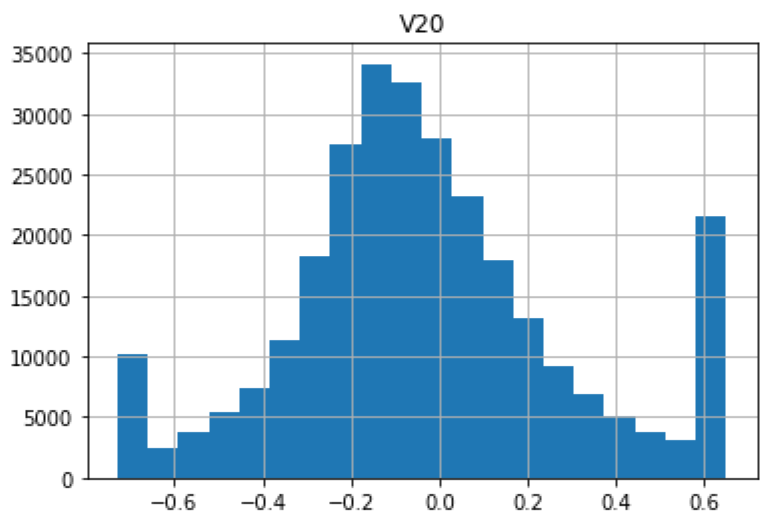

V15
0
5000
10000
15000
20000
25000
30000
−2
−1
0
1
2

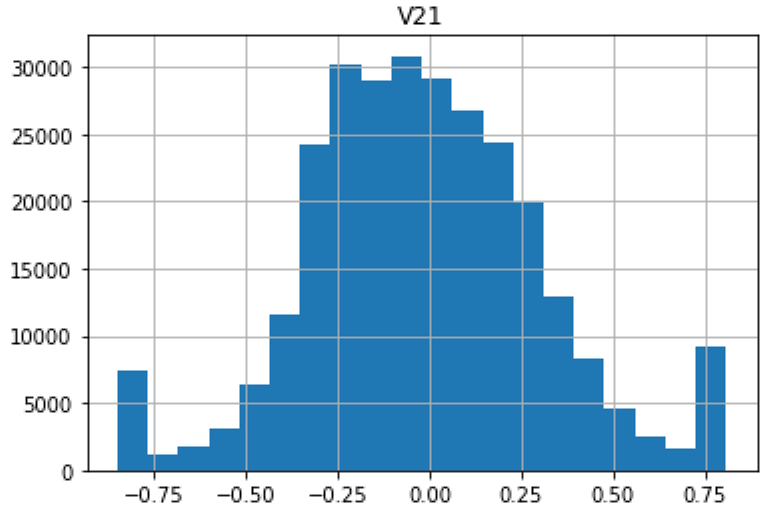

V16
0
5000
10000
15000
20000
25000
30000
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

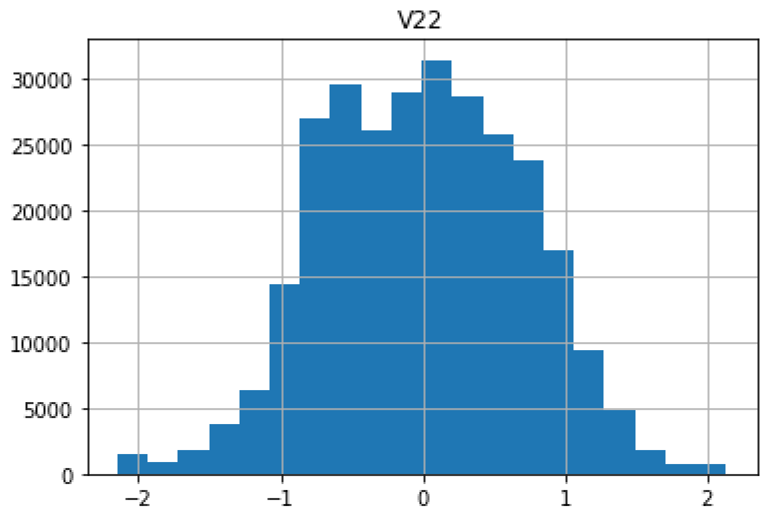

V17
0
5000
10000
15000
20000
25000
30000
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

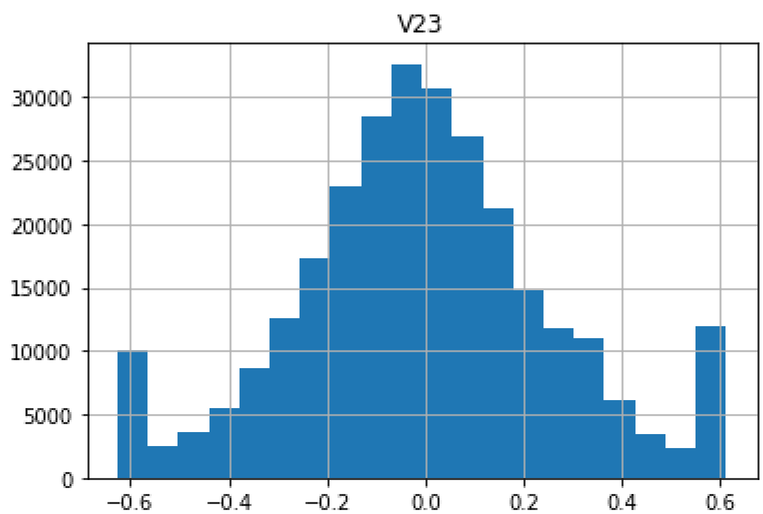

V18
0
5000
10000
15000
20000
25000
30000
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

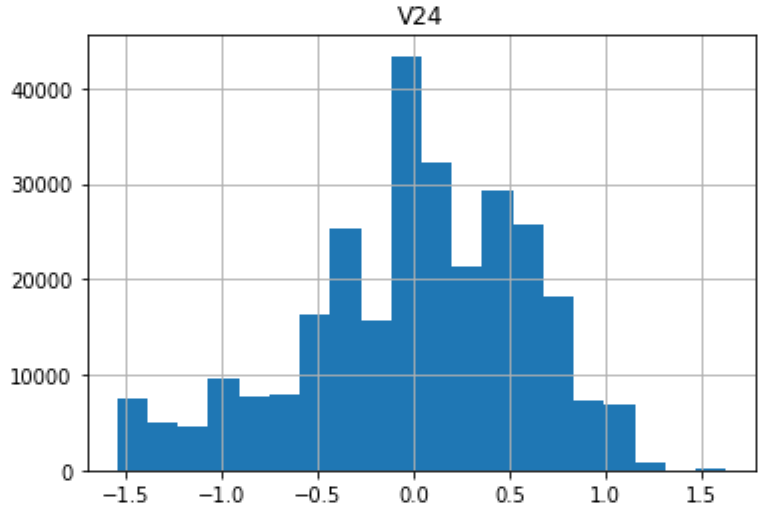

V19
0
5000
10000
15000
20000
25000
30000
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

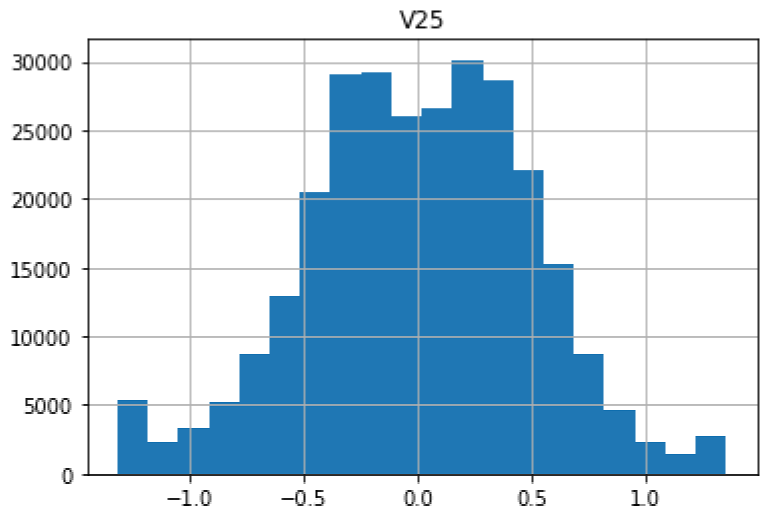

V20
0
5000
10000
15000
20000
25000
30000
35000
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

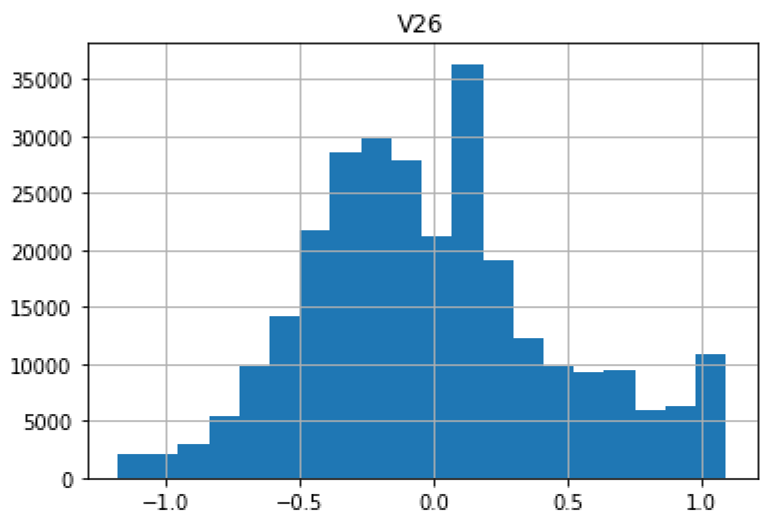

V21
30000
25000
20000
15000
10000
5000
0
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75

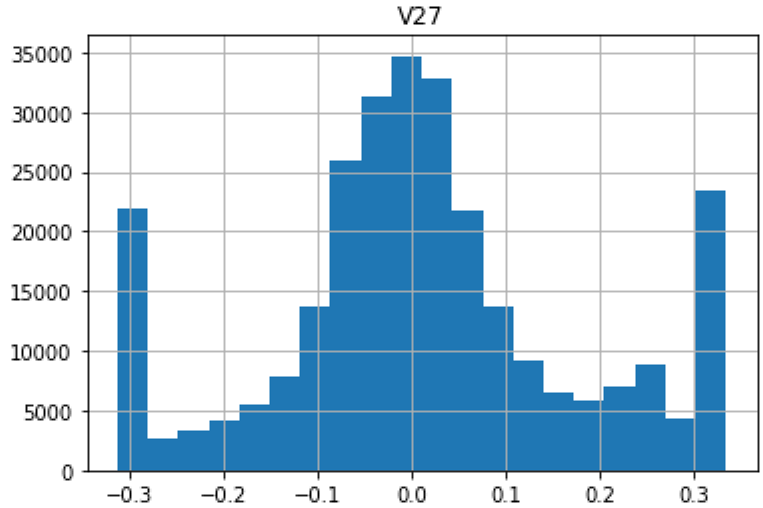

V22
30000
25000
20000
15000
10000
5000
0
−2
−1
0
1
2

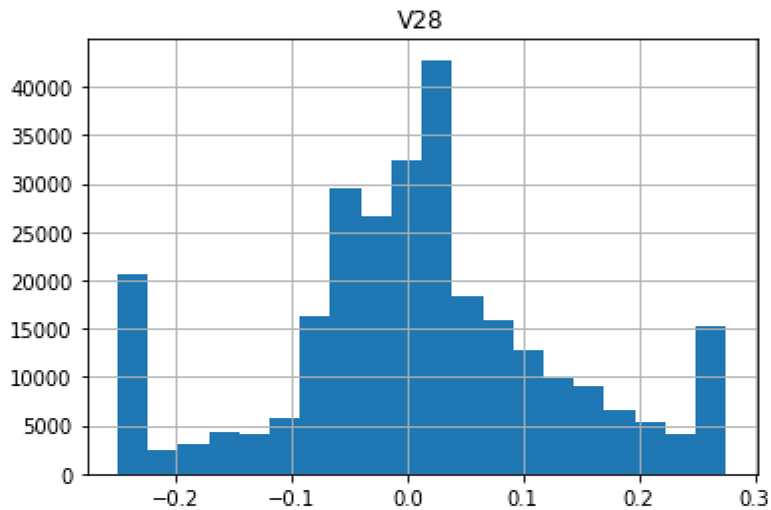

V23
30000
25000
20000
15000
10000
5000
0
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

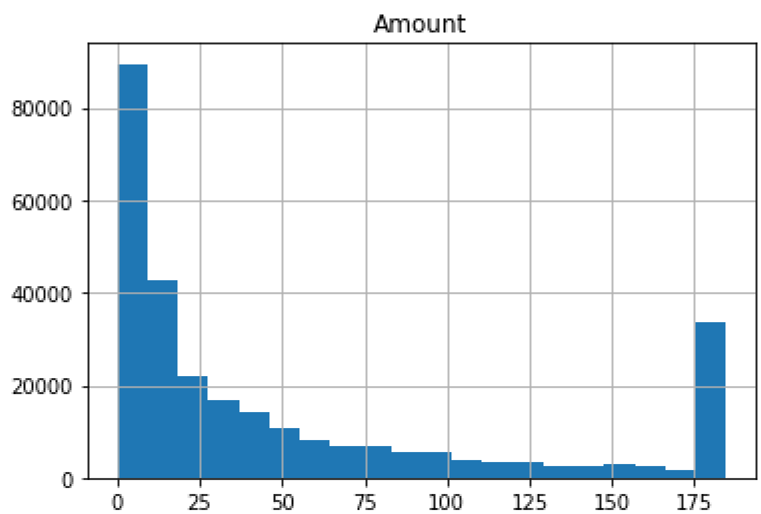

V24
40000
30000
20000
10000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


V25
30000
25000
20000
15000
10000
5000
0
−1.0
−0.5
0.0
0.5
1.0


V26
35000
30000
25000
20000
15000
10000
5000
0
−1.0
−0.5
0.0
0.5
1.0







2. Model Training

In [12]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
```

In [80]:

```
y = df['Class'].to_numpy(dtype=int)
X = df.drop('Class', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
cv = ShuffleSplit(n_splits=2, test_size=0.2, random_state=0)
```

In [58]:

```
# code source: Tutorial [1]
def plot_learning_curve(
    estimator,
    title,
    X,
    y,
    axes=None,
    ylim=None,
    cv=None,
    n_jobs=None,
    train_sizes=np.linspace(0.1, 1.0, 5),
):

    _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = learning_curve(
        estimator,
        X,
        y,
        cv=cv,
        n_jobs=n_jobs,
        train_sizes=train_sizes,
        return_times=True,
        scoring="accuracy",
    )
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(
        train_sizes,
        train_scores_mean - train_scores_std,
        train_scores_mean + train_scores_std,
        alpha=0.1,
        color="r",
    )
    axes[0].fill_between(
        train_sizes,
        test_scores_mean - test_scores_std,
        test_scores_mean + test_scores_std,
        alpha=0.1,
        color="g",
    )
    axes[0].plot(
        train_sizes, train_scores_mean, "o-", color="r", label="Training score"
    )
    axes[0].plot(
        train_sizes, test_scores_mean, "o-", color="g", label="Cross-validation score"
    )
    axes[0].legend(loc="best")
```

```
    # Plot n_samples vs fit_times
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, "o-")
    axes[1].fill_between(
        train_sizes,
        fit_times_mean - fit_times_std,
        fit_times_mean + fit_times_std,
        alpha=0.1,
    )
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    fit_time_argsort = fit_times_mean.argsort()
    fit_time_sorted = fit_times_mean[fit_time_argsort]
    test_scores_mean_sorted = test_scores_mean[fit_time_argsort]
    test_scores_std_sorted = test_scores_std[fit_time_argsort]
    axes[2].grid()
    axes[2].plot(fit_time_sorted, test_scores_mean_sorted, "o-")
    axes[2].fill_between(
        fit_time_sorted,
        test_scores_mean_sorted - test_scores_std_sorted,
        test_scores_mean_sorted + test_scores_std_sorted,
        alpha=0.1,
    )
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")
    return plt
```

2.1 Strong Learner: Gaussian Naive Bayes

source: [7]

Gaussian Naive Bayes is used when the data is continuous because it computes the probalities of the likelihoods. The data should have Gaussian distribution ideally, our visiualization plots show Gaussian-like distribution. Also, can be accurate with just a few data points, which is ideal as we have limited number of Class 1 data. The data requires strong indepenence between the features. Our dataset is majority transformed to hide the original feature information but feature indepenence is a reasonable assumption in this case as previous data scientists processed the features into a usable set.

In [14]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
```

Hyperparameter tuning

In [81]:

```
# code source : [2]

import warnings
warnings.filterwarnings('ignore')

model = GaussianNB()

params_NB = {'var_smoothing': np.logspace(0,-9, num=10)}
gs_NB = GridSearchCV(estimator=model,
                 param_grid=params_NB,
```

```
                      cv=cv,
                      verbose=1,
                      scoring='accuracy')
gs_NB.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
```

Out[81]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=2, random_state=0, test_size=0.2, train_size=None),
             estimator=GaussianNB(),
             param_grid={'var_smoothing': array([1.e+00, 1.e-01, 1.e-02, 1.e-03, 1.e-04,
1.e-05, 1.e-06, 1.e-07,
       1.e-08, 1.e-09])},
             scoring='accuracy', verbose=1)
```

In [82]:
```
gs_NB.best_params_
```

Out[82]:
```
{'var_smoothing': 0.0001}
```

Build and train model

In [83]:
```
model = GaussianNB(var_smoothing=gs_NB.best_params_['var_smoothing'])
model.fit(X_train, y_train);
```

In [84]:
```
y_pred = model.predict(X_test)
```

In [85]:
```
confusion_matrix(y_test, y_pred)
```

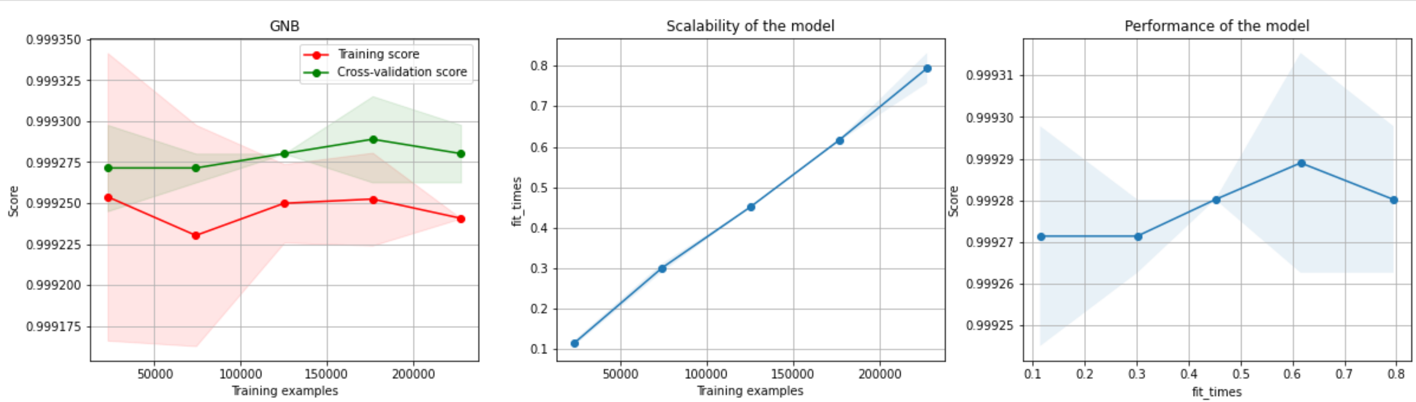
Out[85]:
```
array([[93792,    46],
       [   25,   124]])
```

In [86]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     93838
           1       0.73      0.83      0.78       149

    accuracy                           1.00     93987
   macro avg       0.86      0.92      0.89     93987
weighted avg       1.00      1.00      1.00     93987
```

Plot and analyze learning curve

In [87]:
```
estimator = GaussianNB(var_smoothing=gs_NB.best_params_['var_smoothing'])
p = plot_learning_curve(estimator,  "GNB", X, y, cv=cv)
plt.show()
```



The learning curve shows that model achieves good trainig score as the increase in the training examples. The model achieves best score with approx. 150000 training examples and after that the model is overfitting with decrease in cross-validation score.

The model fit_time increases with training examples and also the performance of the model.

2.2 Bagging Model

Sklearn's enembled BaggingClassifier uses DecisionTreeClassifier by default as the base estimator and ensembles multiple estimators. The DecisionTreeClassifier is a good option for our data as classification is our goal and this model is low cost. Both numerical and categorical data is supported without requiring preprocessing so our data meets the requirements. Ensembling these models with bagging may provide better results than a single decision tree.

Source: [3]

In [22]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
```

Hyperparameter tuning

In [89]:
```
param_grid = {
    'n_estimators' : [2, 5, 7 ]
}

clf = GridSearchCV(BaggingClassifier(),
                   param_grid,
                   cv=cv,
                   scoring = 'accuracy')

clf.fit(X_train, y_train)

print(clf.best_params_)
```

```
{'n_estimators': 7}
```

Build and train model

In [94]:
```
model = BaggingClassifier(n_estimators=7)

# weight the class 1 samples higher to compensate for the sample inbalance
#sample_weight = np.ones(shape=(len(y_train),))
#sample_weight[y_train == 1] = 100
# Note: the weighting value was selected with trial and error

model = model.fit(X_train, y_train, )
```

In [95]:
```
y_pred = model.predict(X_test)
```

In [96]:
```
confusion_matrix(y_test, y_pred)
```

Out[96]:
```
array([[93823,    15],
       [   28,   121]])
```

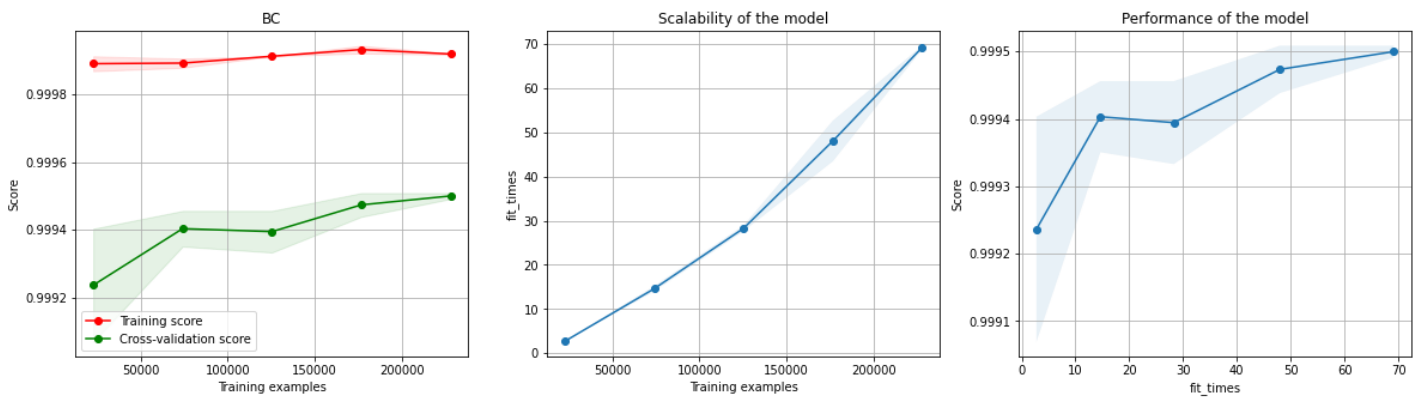
In [97]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     93838
           1       0.89      0.81      0.85       149

    accuracy                           1.00     93987
   macro avg       0.94      0.91      0.92     93987
weighted avg       1.00      1.00      1.00     93987
```

Learning curve

In [98]:

```
estimator = BaggingClassifier(n_estimators=7)
p = plot_learning_curve(estimator, "BC", X, y, cv=cv)
plt.show()
```



The learning curve for bagging classifier shows that the traning score is good throughout thr increment of training examples on the other side, the cross-validation score increases. The fit timing of the model also increases with training examples. The performance of model never decreases with increase of training examples.

2.3 Boost Model

AdaBoostClassifier was chosen as it also uses DecisionTreeClassifier as the base estimator, but implements a Boost algorithm for ensemble. This will be interesting to compare with the bagging model. The same data requirements as above.

Source: [4]

In [31]:

```
from sklearn.ensemble import AdaBoostClassifier
```

Tune hyperparameters

In [32]:

```
param_grid = {
    'n_estimators' : [5, 10, 15 ]
}

clf = GridSearchCV(AdaBoostClassifier(),
                   param_grid, scoring = 'accuracy')

clf.fit(X_train, y_train)

clf.best_params_
```

Out[32]:

```
{'n_estimators': 10}
```

Build and train model

In [53]:
```
model_ADA = AdaBoostClassifier(n_estimators=10)

model_ADA = model_ADA.fit(X_train, y_train, )
```

In [54]:
```
y_pred = model_ADA.predict(X_test)
```

In [55]:
```
confusion_matrix(y_test, y_pred)
```

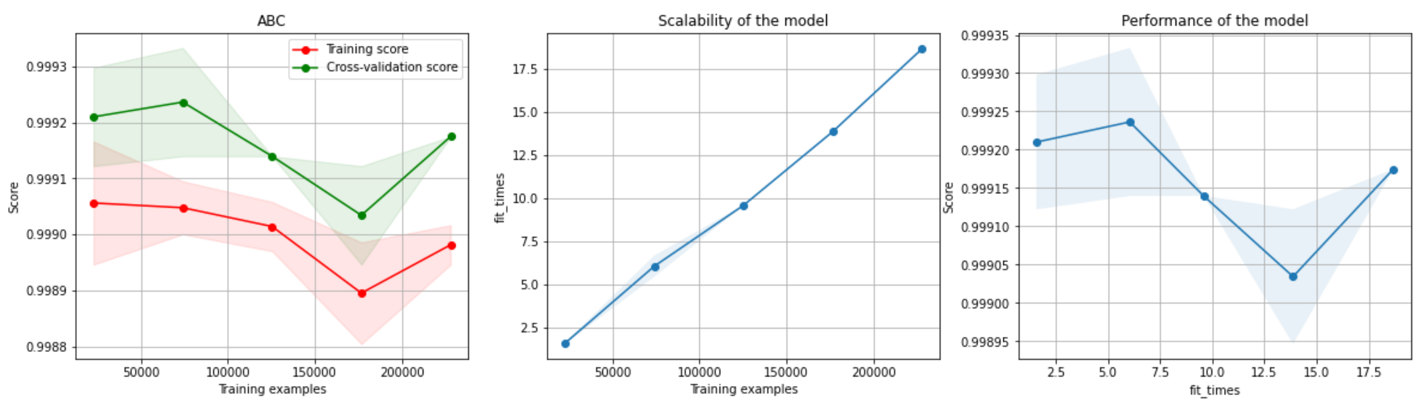
Out[55]:
```
array([[93806,    32],
       [   40,   109]])
```

In [56]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     93838
           1       0.77      0.73      0.75       149

    accuracy                           1.00     93987
   macro avg       0.89      0.87      0.88     93987
weighted avg       1.00      1.00      1.00     93987
```

Learning Curve

In [99]:
```
estimator = AdaBoostClassifier(n_estimators=10)
p = plot_learning_curve(estimator, "ABC", X, y, cv=cv)
plt.show()
```



The training score has been seen decreasing with increase of training examples. The cross validation score is first increasing with some training examples but after that it also has been seen decreasing.

The performance of the model has been seen gradually decreasing after a certain amount of time model has taken to train.

3. Model Comparison

3.1 Variance and Bias Analysis

The Strong learner model has overfitting in the data which might arised from variance and bias trade-off tension between the errors.

The variance has been decreased by the bagging classifier model, therefore throughout the increase in training examples, the score has been highest and same.

In the boosting model, it is seen that the training score decreases but the cross validation score increases with increase with training examples upto a point. Then both becomes stable, that is, the model is baised to a particular kind of solution.

The bagging classifier model is suitable for fraud detection because it has the best accuracy and marco-avg f1 score compared to the rest of models. Also, it has good training score throughout and the model is trained better with more training examples as the cross-validation score, scalability and performance of the model also increased.

Source: [5]

Macro average f1 calculates the F1 separated by class but not using weights for the aggregation. This results in a bigger penalisation when the model does not perform well with the minority classes which is exactly what we want when there is imbalance.

3.2 Imbalance Classification Analysis

Source: [6] The problem with the fraud detection dataset is that the majority of the data is non-fraudulent transactions (Class 0) and contains very few fraudulent (Class 1). The overall accuracy may appear to be high if the model tries to classify most or all data points as Class 0 but fails to complete the goal of identifing Class 1. Imbalanced data like this must be handled appropriately to ensure the business problem is solved.

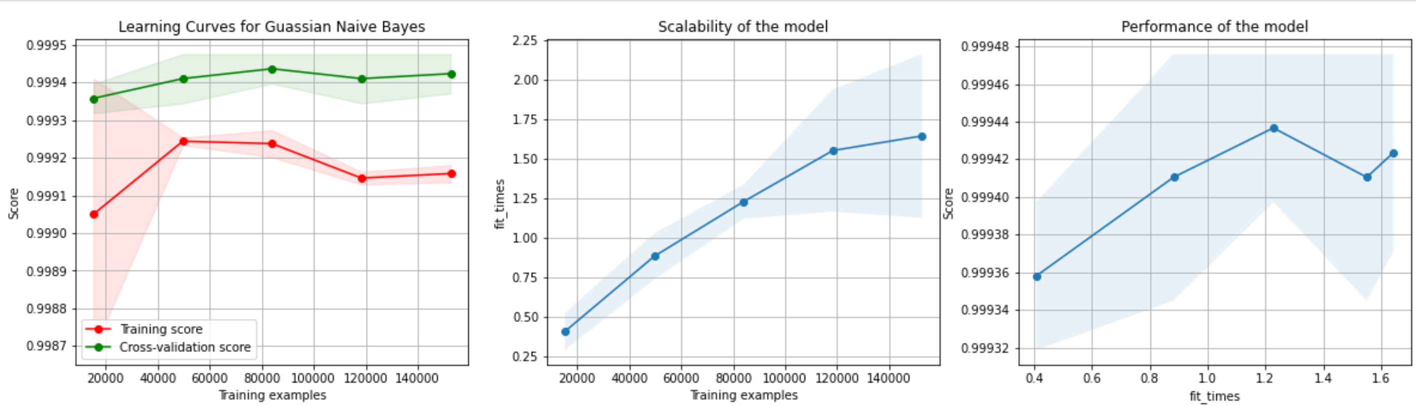
The model which can handle the imbalanced data best is the ADA boosting model. As ADA boost builds ensemble of weak learners by adjusting weight of missclassified data during each iteration, higher weight is given to the minority class as they will be misclassified more often if the model tries to become biased towards Class 0. To achieve similar accuracy with the other emsemble model, sample weights were required to be used, otherwise Class 1 accuracy was very poor. The strong learner model

3.3 Run Time Analysis

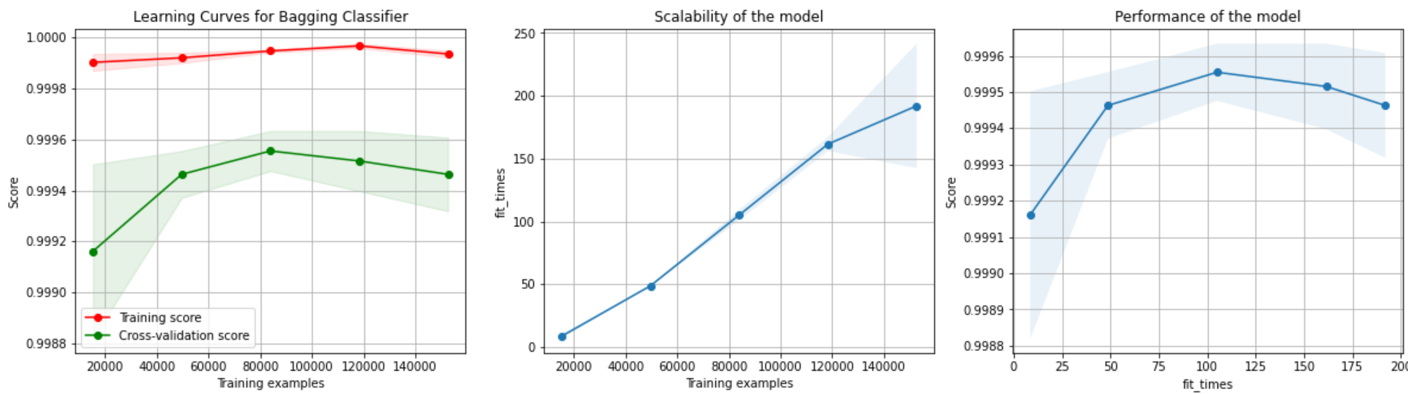
The run time analysis curves with 1 worker thread are plotted in section 2 along with the learning curves. The fastest model in this case is gaussian naive bayes model.

In [100…

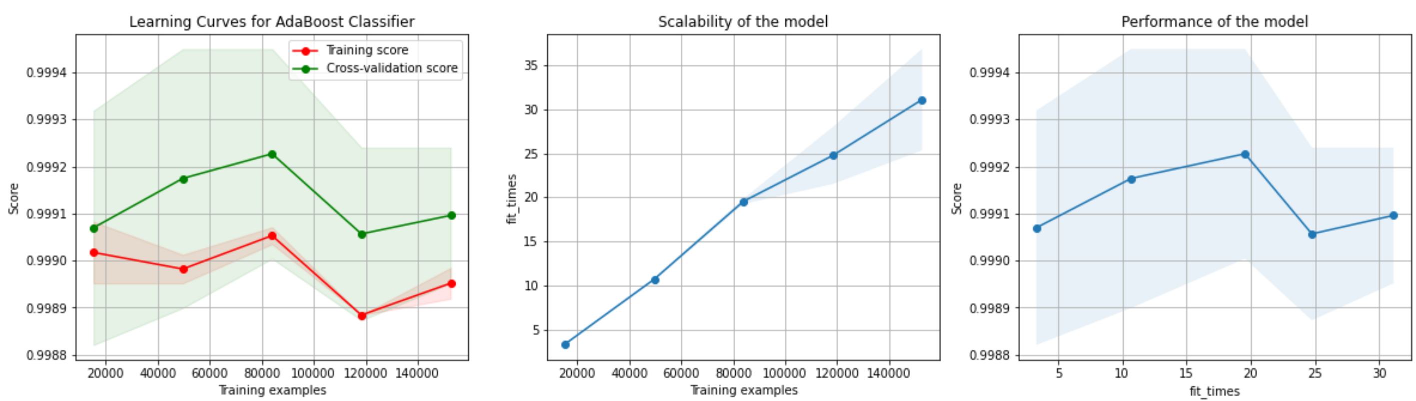
```
from sklearn.model_selection import ShuffleSplit

title = "Learning Curves for Guassian Naive Bayes"
n_jobs = 4

estimator = GaussianNB(var_smoothing=gs_NB.best_params_['var_smoothing'])
plot_learning_curve(estimator, title, X_train, y_train, cv=cv, n_jobs=n_jobs)
plt.show()
```



```
In [101…
title = "Learning Curves for Bagging Classifier"

estimator = BaggingClassifier(n_estimators=15)
plot_learning_curve(estimator, title, X_train, y_train, cv=cv, n_jobs=n_jobs)
plt.show()
```




```
In [102…
title = "Learning Curves for AdaBoost Classifier"

estimator = AdaBoostClassifier(n_estimators=10)
plot_learning_curve(estimator, title, X_train, y_train, cv=cv, n_jobs=n_jobs)
plt.show()
```



The fastest model in the case of 4 worker threads is the Gaussian Bayes classifier.

References

[1] In-class Tutorial \ [2] https://stackoverflow.com/questions/39828535/how-to-tune-gaussiannb \ [3] https://scikit-learn.org/stable/modules/tree.html#tree \ [4] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html \ [5] https://datascience.stackexchange.com/questions/65839/macro-average-and-weighted-average-meaning-in-classification-report \ [6] https://towardsdatascience.com/https-medium-com-abrown004-how-to-ease-the-pain-of-working-with-imbalanced-data-a7f7601f18ba \ [7] https://iq.opengenus.org/gaussian-naive-bayes/