In [22]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings("ignore")
```

In [23]:
```
geo = pd.read_csv('Geo_scores.csv')
instance = pd.read_csv("instance_scores.csv")
lambdawts = pd.read_csv("Lambda_wts.csv")
qset = pd.read_csv("Qset_tats.csv")
test_data = pd.read_csv("test_share.csv")
train_data = pd.read_csv('train.csv')
```

In [24]:
```
train_data['data'] = 'train'
test_data['data'] = 'test'
```

In [25]:
```
train_data.columns
```

Out[25]:
```
Index(['id', 'Group', 'Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7',
       'Per8', 'Per9', 'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7',
       'Dem8', 'Dem9', 'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6',
       'Normalised_FNT', 'Target', 'data'],
      dtype='object')
```

In [26]:
```
test_data.columns
```

Out[26]:
```
Index(['id', 'Group', 'Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7',
       'Per8', 'Per9', 'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7',
       'Dem8', 'Dem9', 'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6',
       'Normalised_FNT', 'data'],
      dtype='object')
```

In [27]:
```
all_data = pd.concat([train_data, test_data], axis=0)
```

In [28]:
```
print("all_data id", all_data['id'].nunique())
print()
print("all_data group", all_data['Group'].nunique())
```

```
all_data id 284807

all_data group 1400
```

In [29]:

```
print(geo.isnull().sum())
print()
print(instance.isnull().sum())
print()
print(lambdawts.isnull().sum())
print()
print(qset.isnull().sum())
print()
print(all_data.isnull().sum())
```

```
id               0
geo_score    71543
dtype: int64

id                 0
instance_scores    0
dtype: int64

Group        0
lambda_wt    0
dtype: int64

id                            0
qsets_normalized_tat    103201
dtype: int64

id                    0
Group                 0
Per1                  0
Per2                  0
Per3                  0
Per4                  0
Per5                  0
Per6                  0
Per7                  0
Per8                  0
Per9                  0
Dem1                  0
Dem2                  0
Dem3                  0
Dem4                  0
Dem5                  0
Dem6                  0
Dem7                  0
Dem8                  0
Dem9                  0
Cred1                 0
Cred2                 0
Cred3                 0
Cred4                 0
Cred5                 0
Cred6                 0
Normalised_FNT        0
Target            56962
data                  0
dtype: int64
```

In [30]:

```
print(geo.describe())
print()
print(qset.describe())
```

```
                 id     geo_score
count  1.424035e+06  1.352492e+06
mean   1.424030e+05 -9.279168e-06
std    8.221673e+04  7.827199e+00
min    0.000000e+00 -1.093900e+02
25%    7.120100e+04 -5.860000e+00
50%    1.424030e+05  1.800000e-01
75%    2.136050e+05  5.860000e+00
max    2.848060e+05  4.581000e+01

                 id  qsets_normalized_tat
count  1.424035e+06          1.320834e+06
mean   1.424030e+05          1.094006e-05
std    8.221673e+04          7.731794e+00
min    0.000000e+00         -1.404400e+02
25%    7.120100e+04         -5.860000e+00
50%    1.424030e+05          2.000000e-02
75%    2.136050e+05          5.860000e+00
max    2.848060e+05          6.110000e+01
```

In [31]:

```
geo['geo_score'] = geo['geo_score'].fillna(geo['geo_score'].median())
qset['qsets_normalized_tat'] = 
qset['qsets_normalized_tat'].fillna(qset['qsets_normalized_tat'].median())
```

In [32]:

```
geo = geo.groupby('id').mean()
```

In [33]:

```
geo.shape
```

Out[33]:

```
(284807, 1)
```

In [34]:

```
qset = qset.groupby('id').mean()
```

In [35]:

```
qset.shape
```

Out[35]:

```
(284807, 1)
```

In [36]:

```
instance.shape
```

Out[36]:

```
(1424035, 2)
```

In [37]:

```
instance = instance.groupby('id').mean()
```

In [38]:

```
lambdawts.shape
```

Out[38]:

```
(1400, 2)
```

In [39]:
```
all_data.head()
```

Out[39]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	...
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	...
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	...
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	...
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	...

5 rows × 29 columns

In [40]:
```
all_data = pd.merge(all_data, geo, on='id', how='left')
```

In [41]:
```
all_data.head(2)
```

Out[41]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.34	1.010000	...	0.6
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.81	0.783333	...	0.6

2 rows × 30 columns

In [42]:
```
all_data = pd.merge(all_data,instance , on='id', how='left')
```

In [43]:
```
all_data.head(1)
```

Out[43]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	Cred2	Cred3
0	112751	Grp169	1.07	0.58	0.48	0.766667	1.233333	1.993333	0.34	1.01	...	1.01	0.933333

1 rows × 31 columns

In [44]:
```
all_data['Group'].nunique()
```

Out[44]:
```
1400
```

In [45]:
```
all_data = pd.merge(all_data,qset , on='id', how='left')
```

In [46]:
```
lambdawts.head(2)
```

Out[46]:

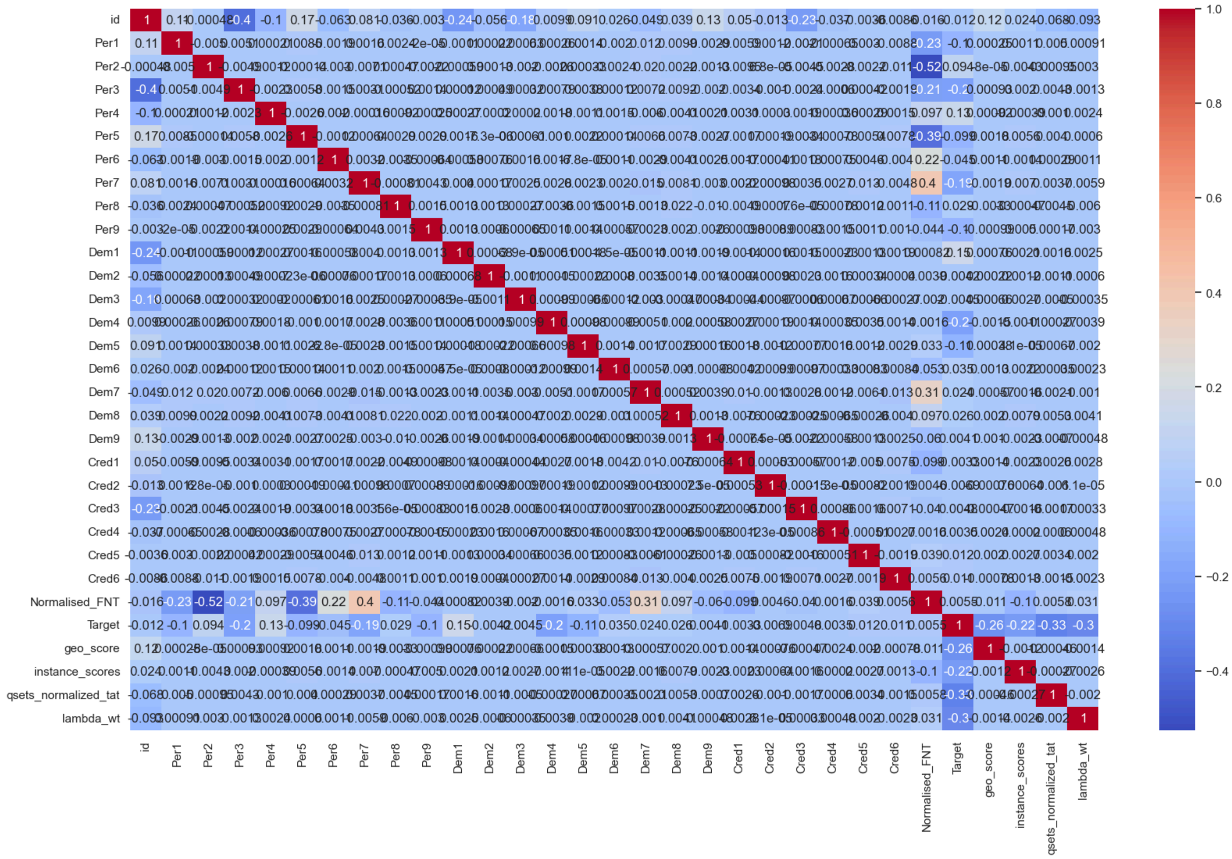
	Group	lambda_wt
0	Grp936	3.41
1	Grp347	-2.88

In [47]:
```
lambdawts['Group'].nunique()
```
Out[47]:
```
1400
```
In [48]:
```
all_data = pd.merge(all_data,lambdawts , on='Group', how='left')
```
In [49]:
```
all_data.head()
```
Out[49]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	...
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	...
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	...
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	...
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	...

5 rows × 33 columns

In [50]:
```
all_data['lambda_wt'].count()
```
Out[50]:
```
284807
```
In [51]:
```
train_data = all_data[all_data['data']=='train']
test_data = all_data[all_data['data']=='test']
```
In [52]:
```
test_data.shape
```
Out[52]:
```
(56962, 33)
```
In [53]:
```
train_data.shape
```
Out[53]:
```
(227845, 33)
```
In [54]:
```
plt.figure(figsize=(20,12))
sns.heatmap(train_data.corr(), annot=True, cmap='coolwarm')
plt.show()
```



In [55]:

```
# splitting the data into independent and dependent variable
x = train_data.drop(['id', 'Group','Target', 'data'], axis=1) # ind
variable
y = train_data['Target'] # dependent
```

In [56]:

```
x.head(2)
```

Out[56]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	...	Cr
0	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.34	1.010000	0.863333	0.46	...	
1	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.81	0.783333	0.190000	0.47	...	

2 rows × 29 columns

In [57]:

```
x.columns
```

Out[57]:

```
Index(['Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7', 'Per8', 'Per9',
       'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7', 'Dem8', 'Dem9',
       'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6', 'Normalised_FNT',
       'geo_score', 'instance_scores', 'qsets_normalized_tat', 'lambda_wt'],
      dtype='object')
```

In [58]:

```
y.head()
```

Out[58]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: Target, dtype: float64
```

In [59]:
```
test_data.columns
```

Out[59]:
```
Index(['id', 'Group', 'Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7',
       'Per8', 'Per9', 'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7',
       'Dem8', 'Dem9', 'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6',
       'Normalised_FNT', 'Target', 'data', 'geo_score', 'instance_scores',
       'qsets_normalized_tat', 'lambda_wt'],
      dtype='object')
```

In [60]:
```
test_data.isnull().sum()/len(test_data)*100
```

Out[60]:
```
id                        0.0
Group                     0.0
Per1                      0.0
Per2                      0.0
Per3                      0.0
Per4                      0.0
Per5                      0.0
Per6                      0.0
Per7                      0.0
Per8                      0.0
Per9                      0.0
Dem1                      0.0
Dem2                      0.0
Dem3                      0.0
Dem4                      0.0
Dem5                      0.0
Dem6                      0.0
Dem7                      0.0
Dem8                      0.0
Dem9                      0.0
Cred1                     0.0
Cred2                     0.0
Cred3                     0.0
Cred4                     0.0
Cred5                     0.0
Cred6                     0.0
Normalised_FNT            0.0
Target                  100.0
data                      0.0
geo_score                 0.0
instance_scores           0.0
qsets_normalized_tat      0.0
lambda_wt                 0.0
dtype: float64
```

In [61]:
```
test_data = test_data.drop(['id','Group','Target','data'], axis=1)
```

In [62]:
```
# Task :
# This data is for prediction whether listed customer will do fraudulent or not
test_data.head()
```

Out[62]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	
227845	-0.300000	1.540000	0.220000	-0.280000	0.570000	0.260000	0.700000	1.076667	0.930000	0.
227846	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0.550000	0.543333	0.
227847	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0.686667	0.593333	1.
227848	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0.663333	1.156667	1.
227849	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0.686667	1.476667	1.

5 rows × 29 columns

Actual Data

In [63]:
```
x.head()
```

Out[63]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1
0	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.460000
1	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000
2	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000
3	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667
4	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667

5 rows × 29 columns

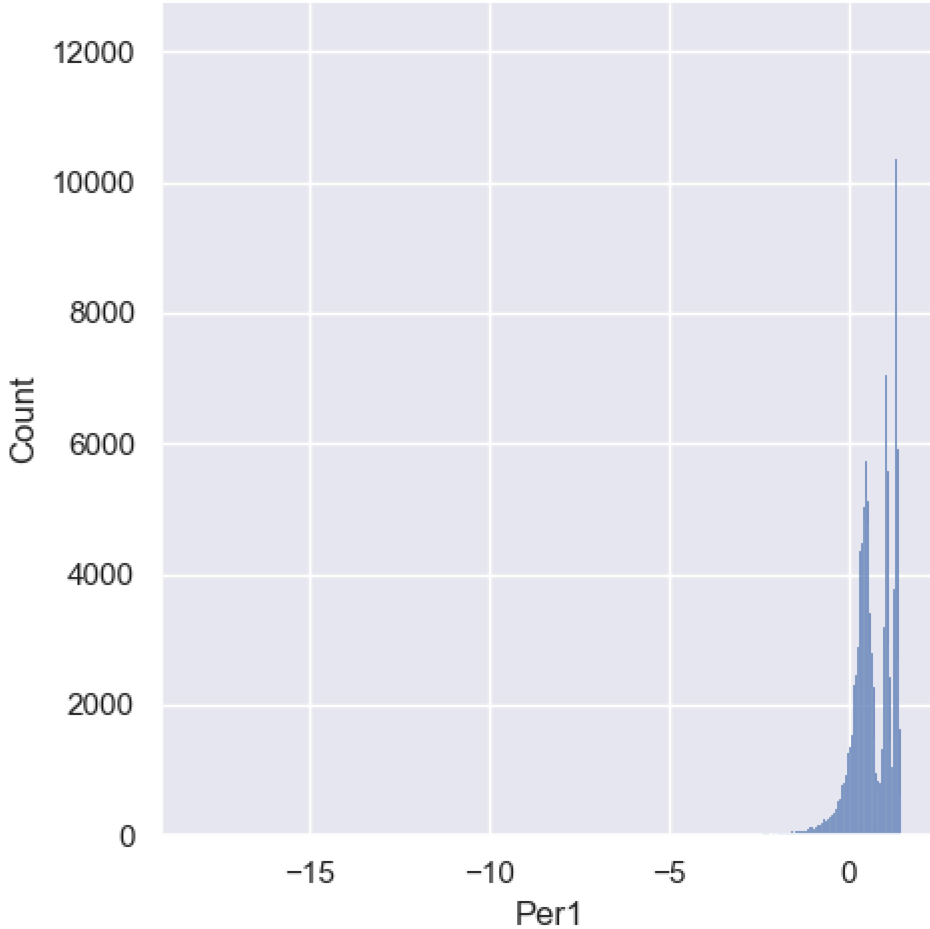
In [64]:
```
x.isnull().any()
```

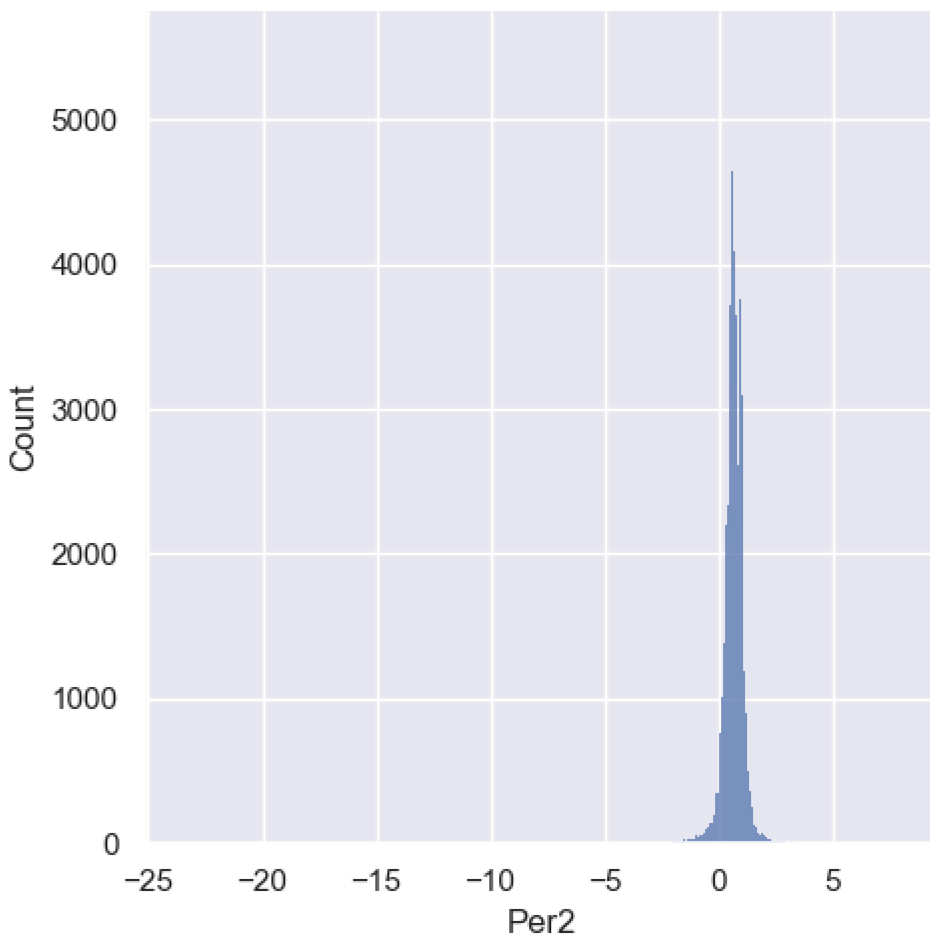
Out[64]:

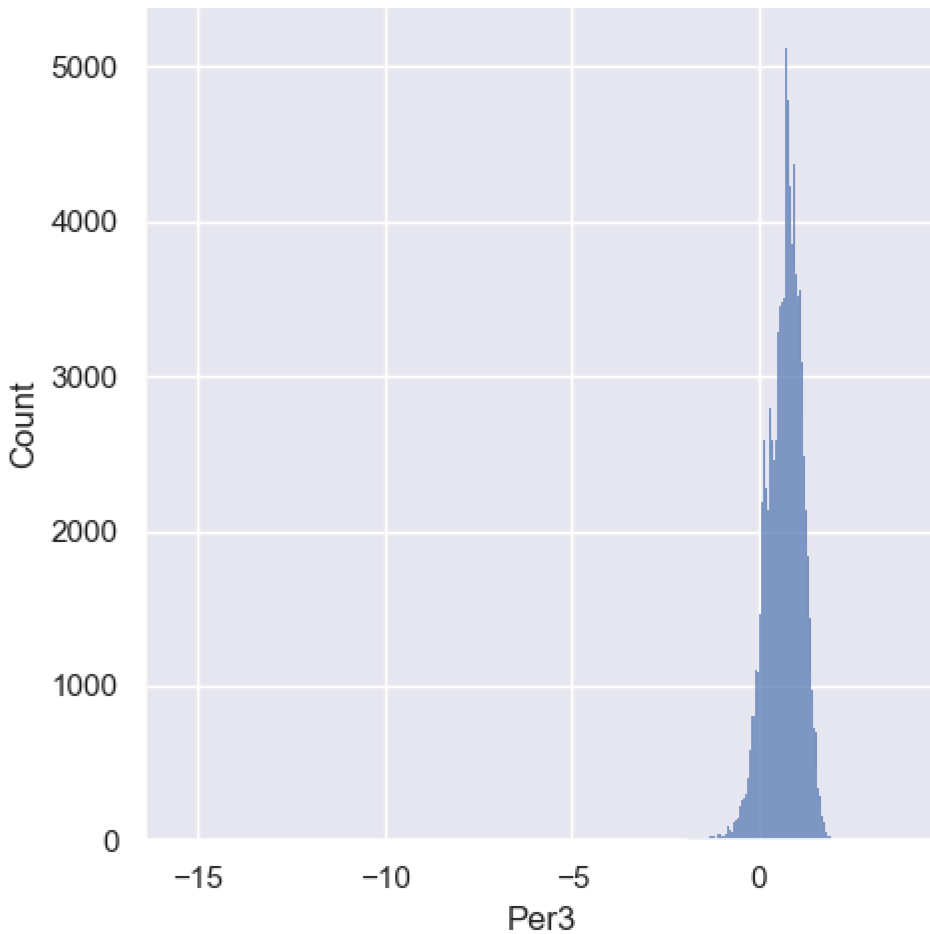
```
Per1                     False
Per2                     False
Per3                     False
Per4                     False
Per5                     False
Per6                     False
Per7                     False
Per8                     False
Per9                     False
Dem1                     False
Dem2                     False
Dem3                     False
Dem4                     False
Dem5                     False
Dem6                     False
Dem7                     False
Dem8                     False
Dem9                     False
Cred1                    False
Cred2                    False
Cred3                    False
Cred4                    False
Cred5                    False
Cred6                    False
Normalised_FNT           False
geo_score                False
instance_scores          False
qsets_normalized_tat     False
lambda_wt                False
dtype: bool
```

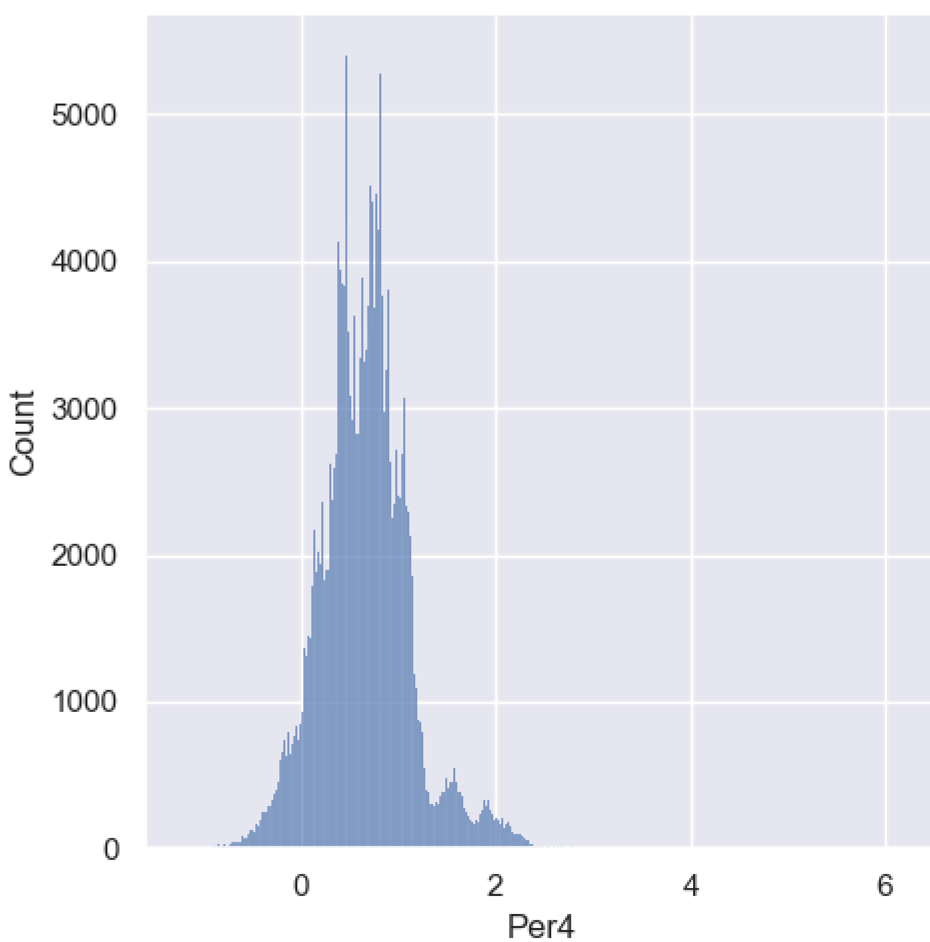
In [65]:

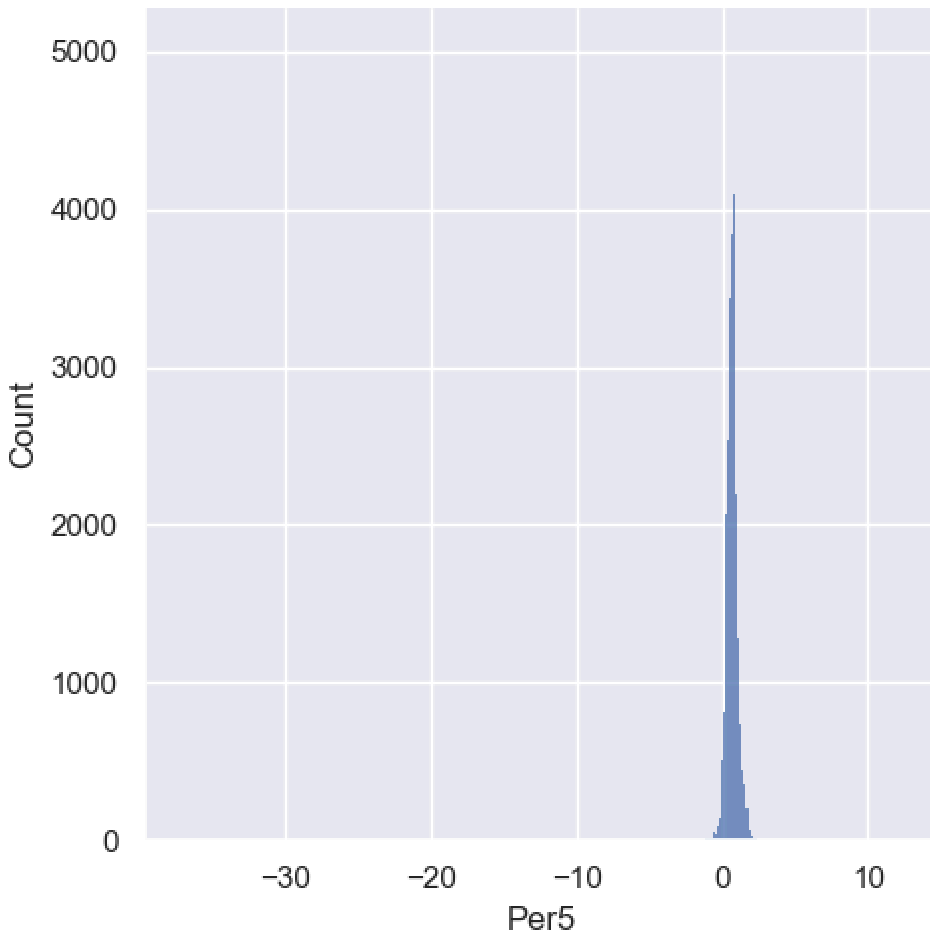
```
def distplots(col):
    sns.displot(x[col])
    plt.show()

for i in list(x.columns)[0:]:
    distplots(i)
```

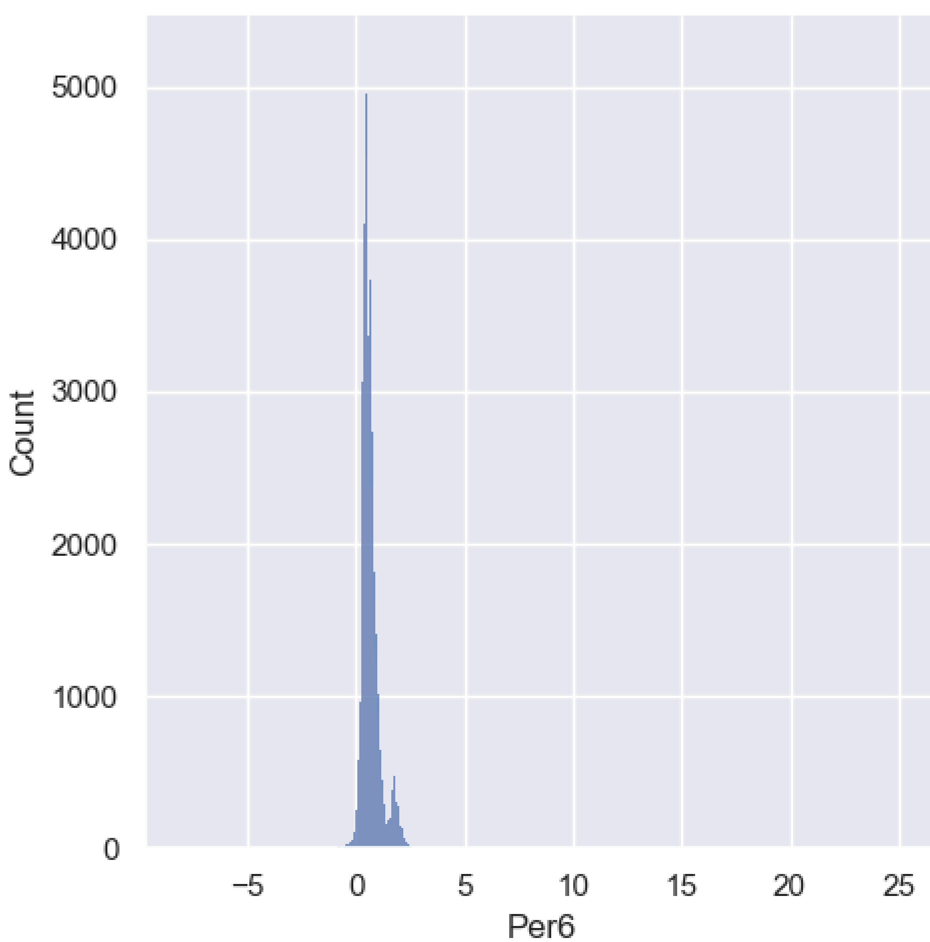

12000
10000
8000
6000
4000
2000
0
Count
−15
−10
−5
0
Per1

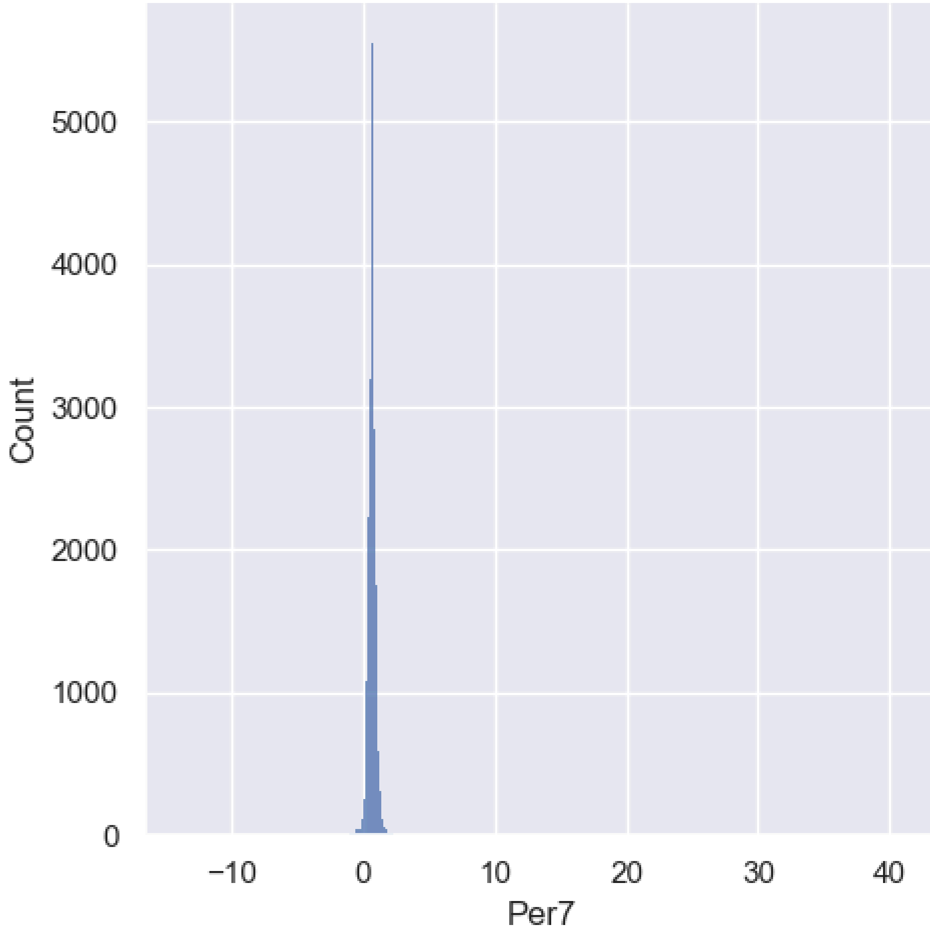

5000
4000
3000
2000
1000
0
Count
−25
−20
−15
−10
−5
0
5
Per2

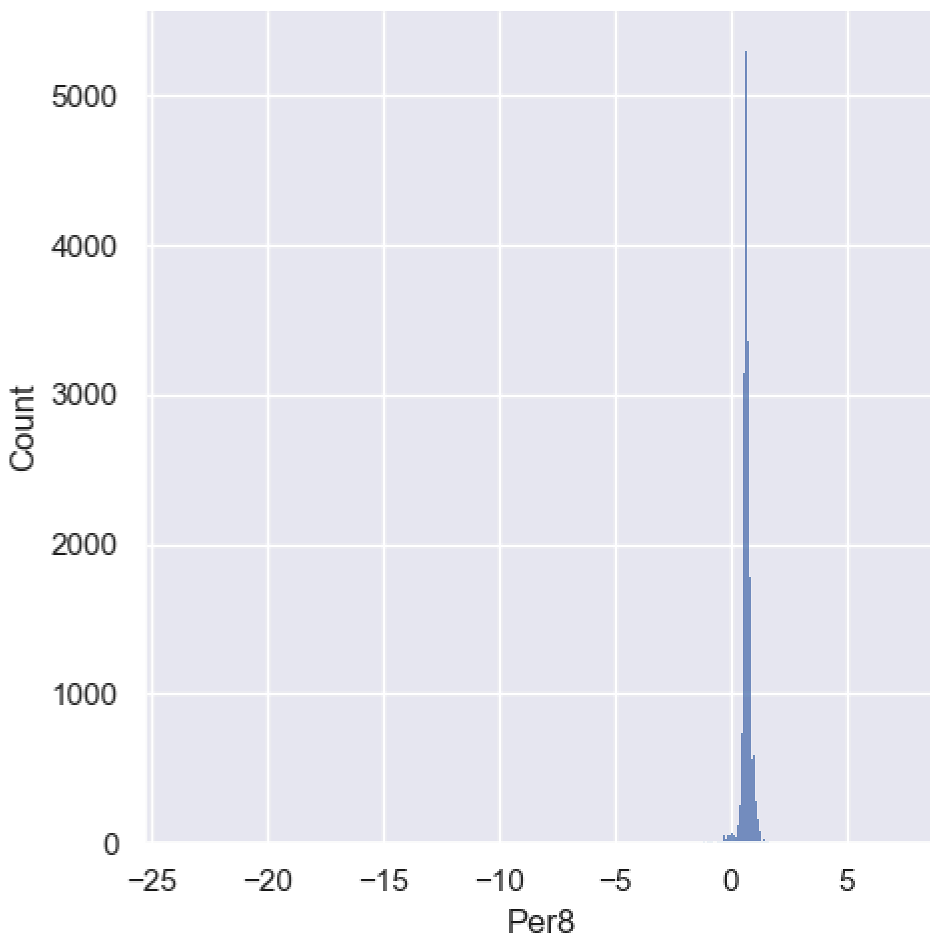

Count
5000
4000
3000
2000
1000
0
−15
−10
−5
0
Per3

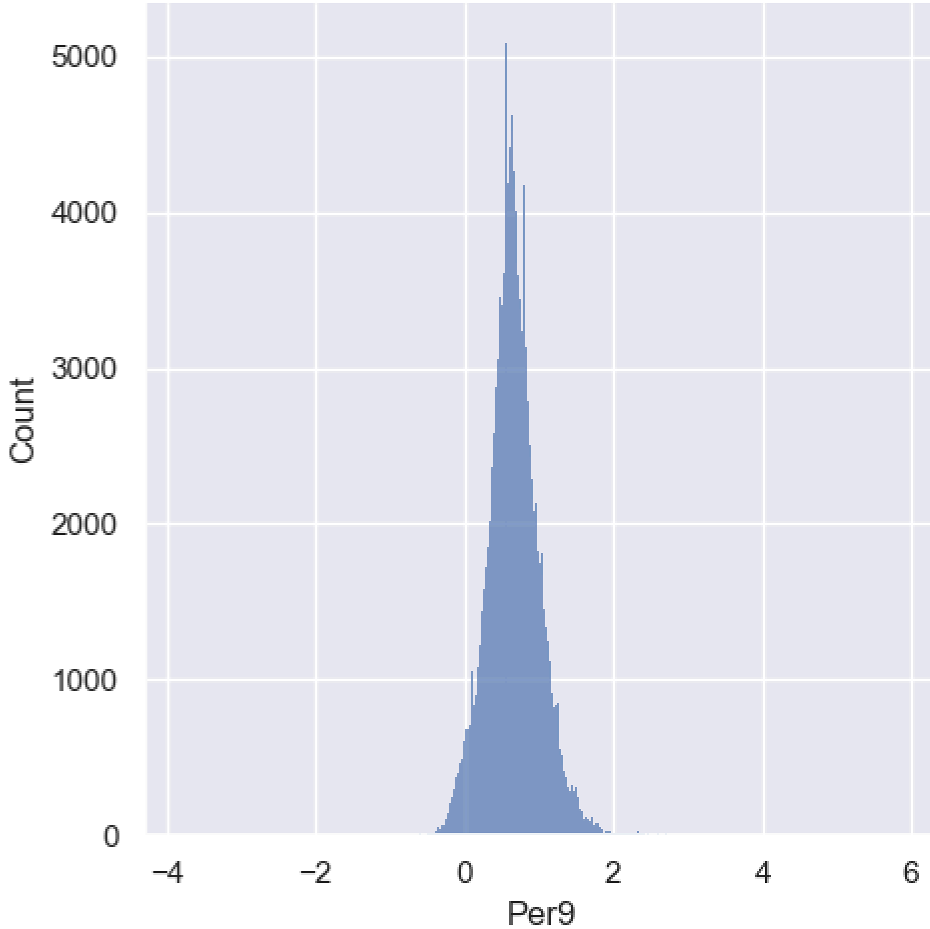

Count
5000
4000
3000
2000
1000
0
0
2
4
6
Per4

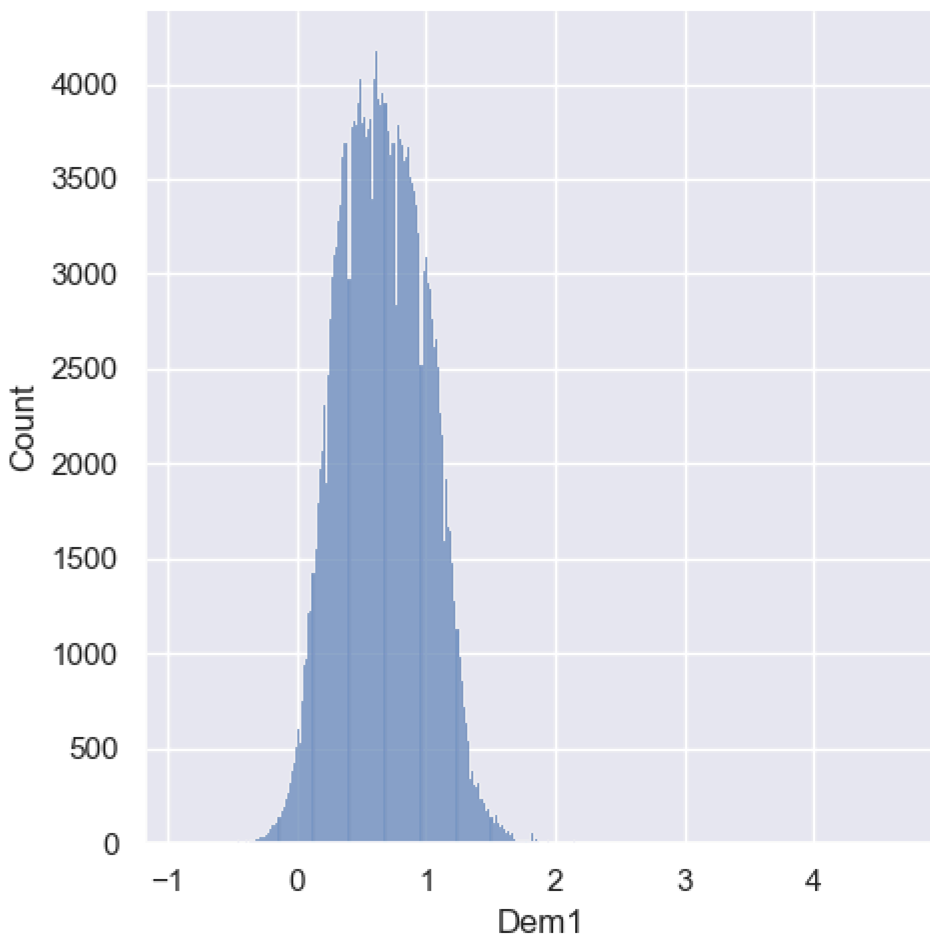

Count
5000
4000
3000
2000
1000
0
−30
−20
−10
0
10
Per5

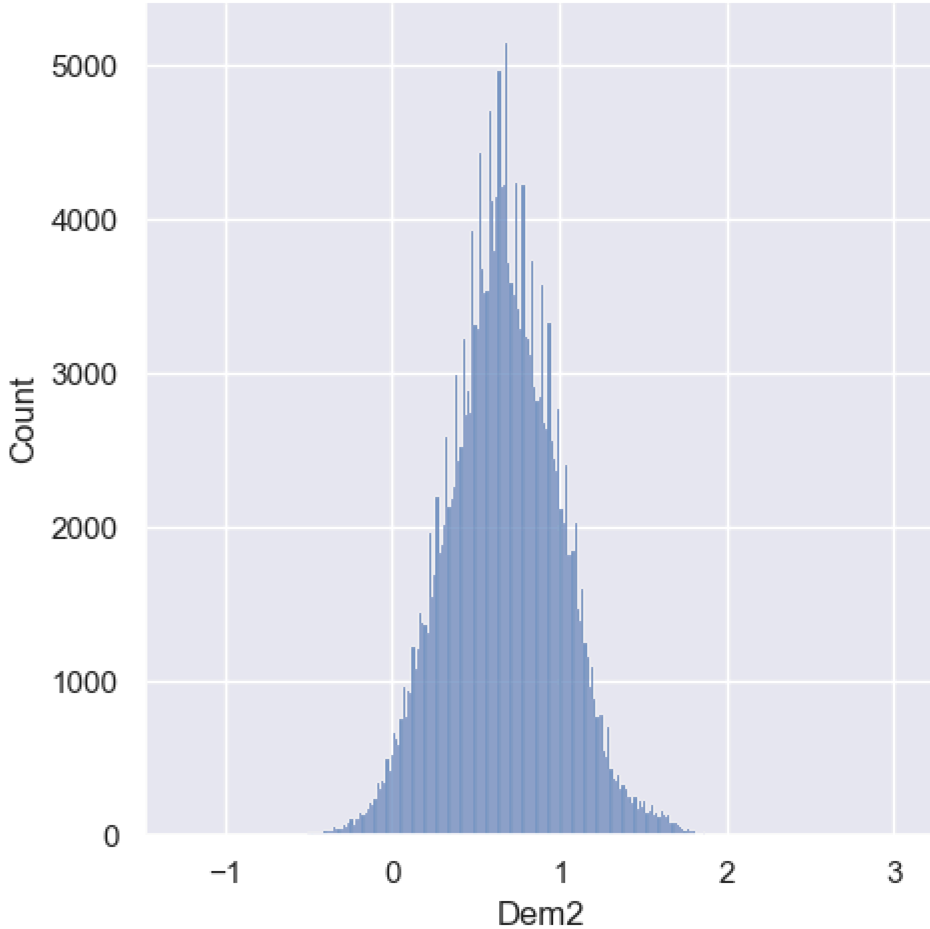

Count
5000
4000
3000
2000
1000
0
−5
0
5
10
15
20
25
Per6


Count
0
1000
2000
3000
4000
5000
−10
0
10
20
30
40
Per7

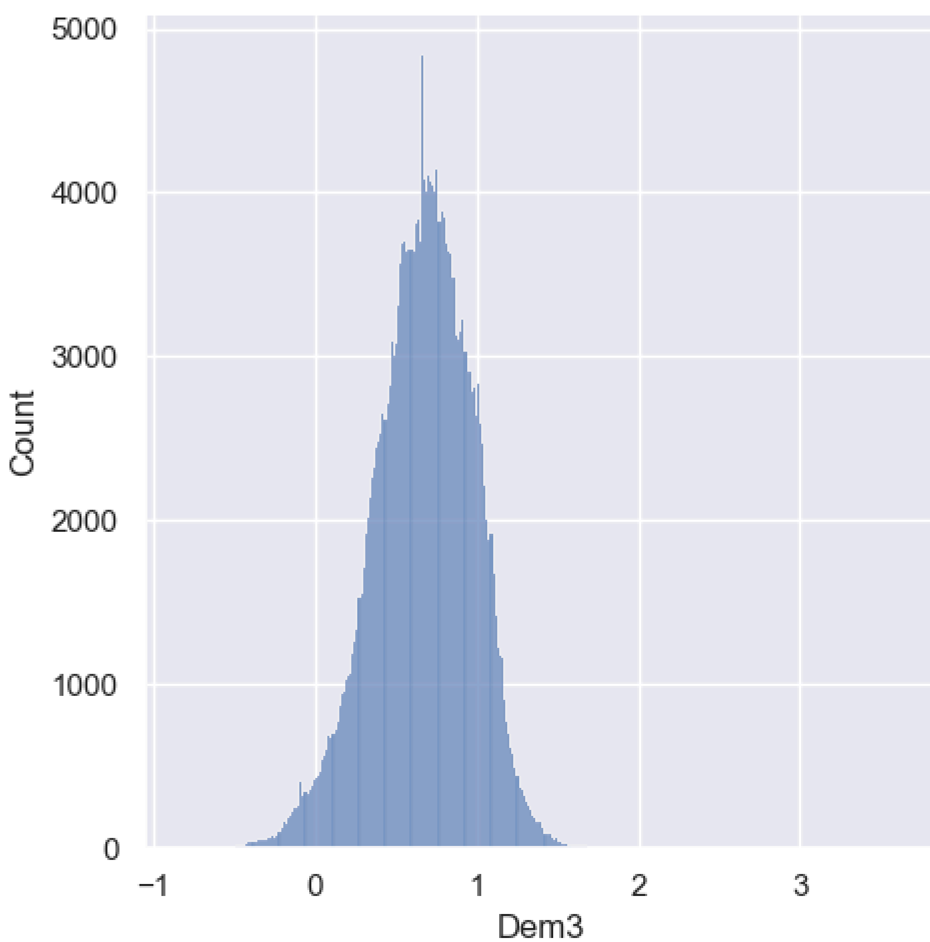

Count
0
1000
2000
3000
4000
5000
−25
−20
−15
−10
−5
0
5
Per8


Count
0
1000
2000
3000
4000
5000
−4
−2
0
2
4
6
Per9

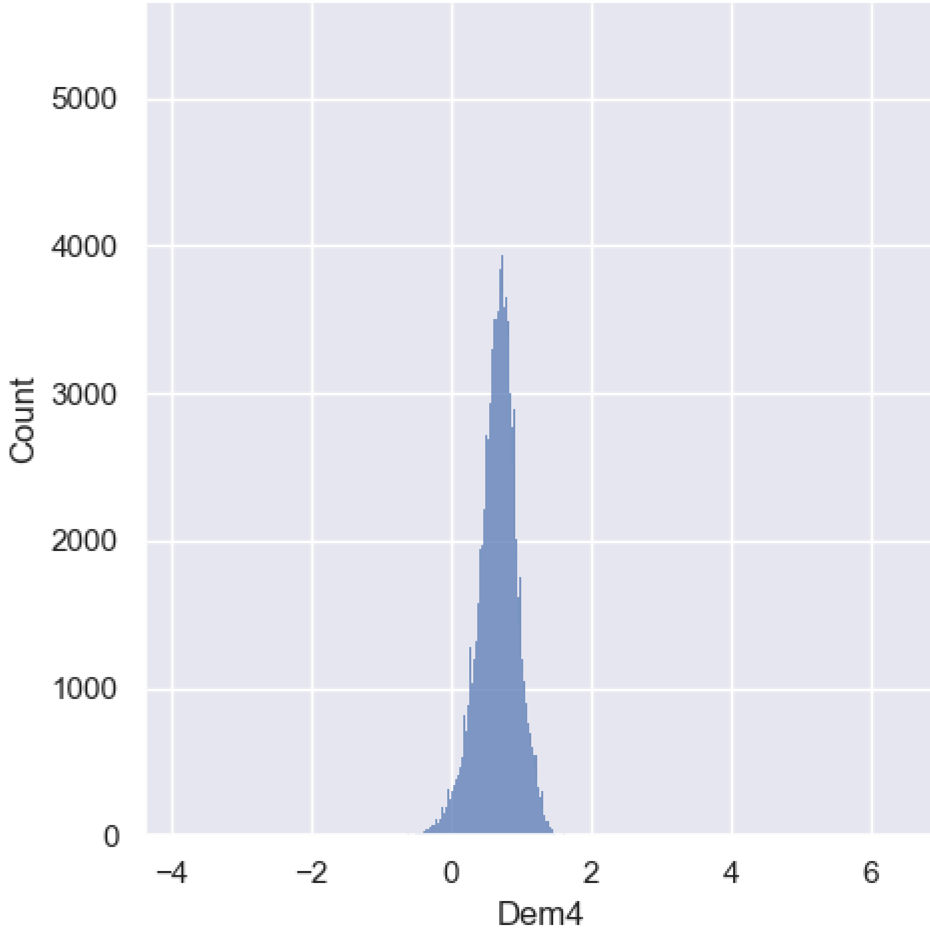

Count
0
500
1000
1500
2000
2500
3000
3500
4000
−1
0
1
2
3
4
Dem1

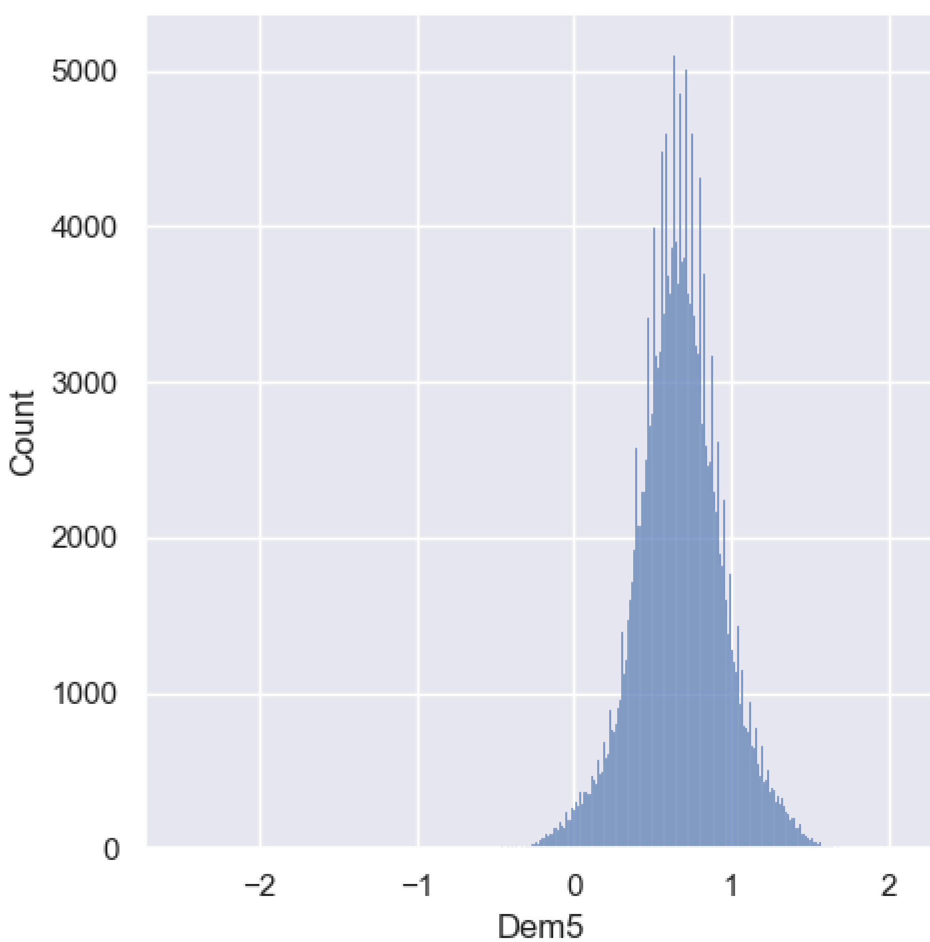


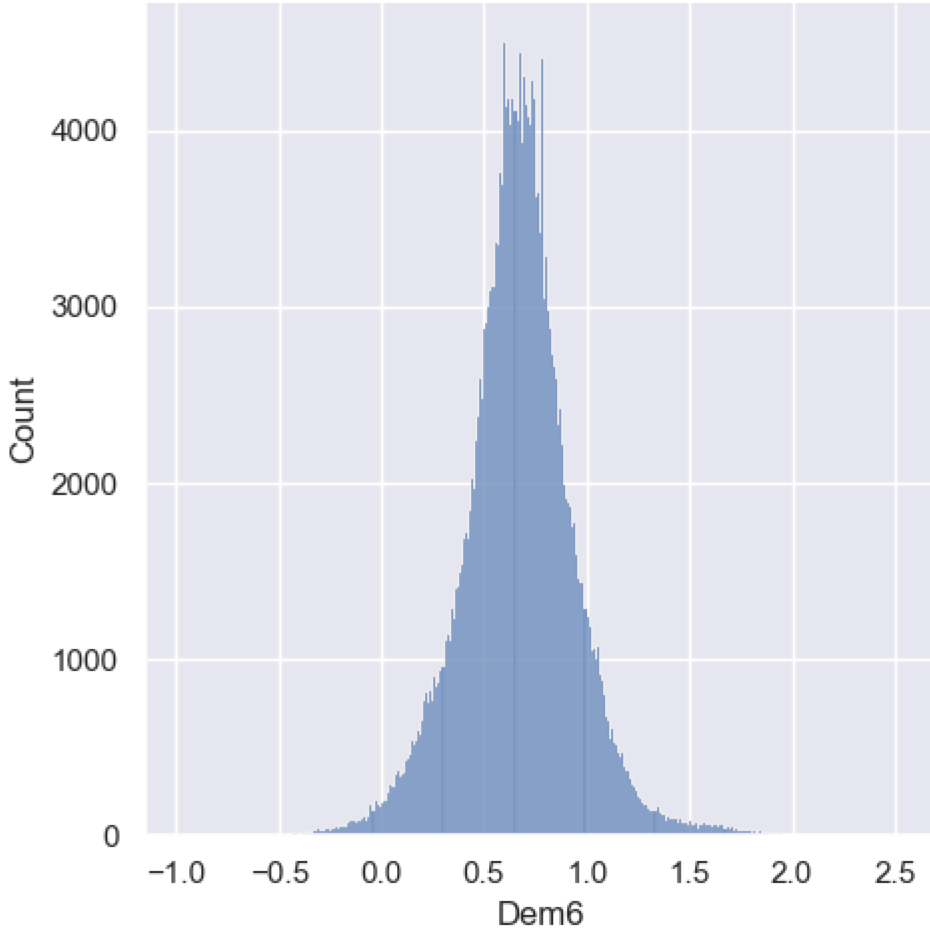
Count
Dem2

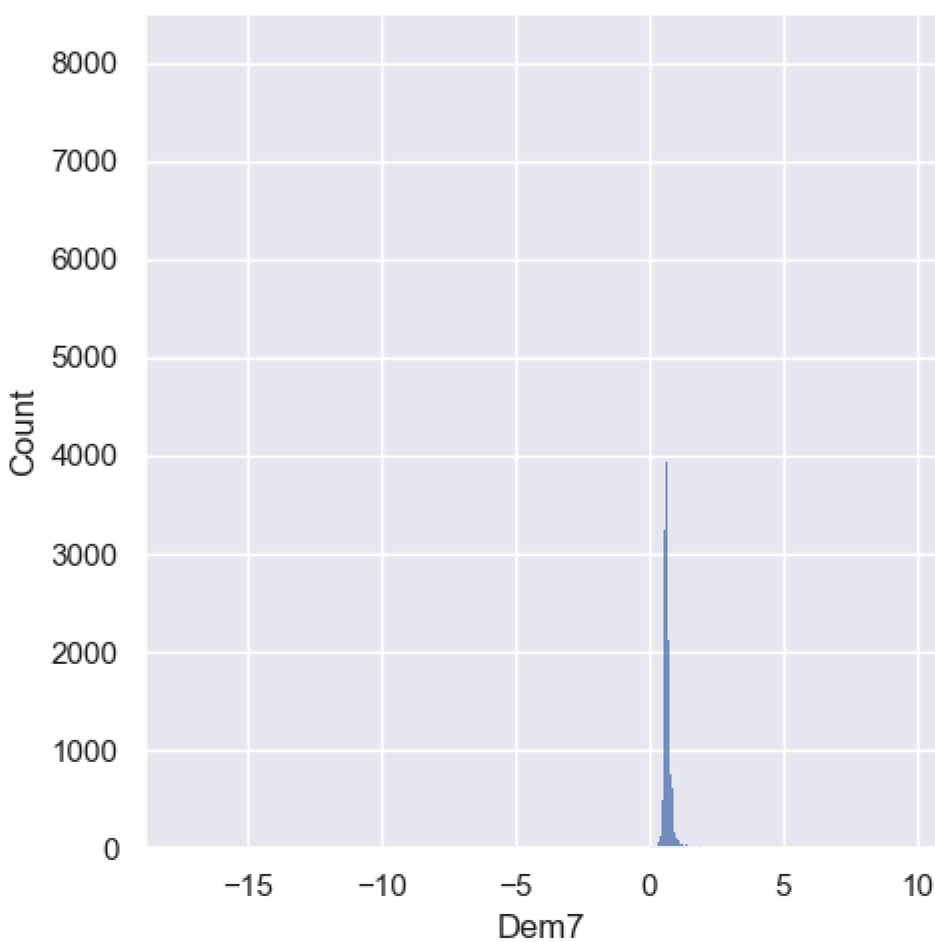


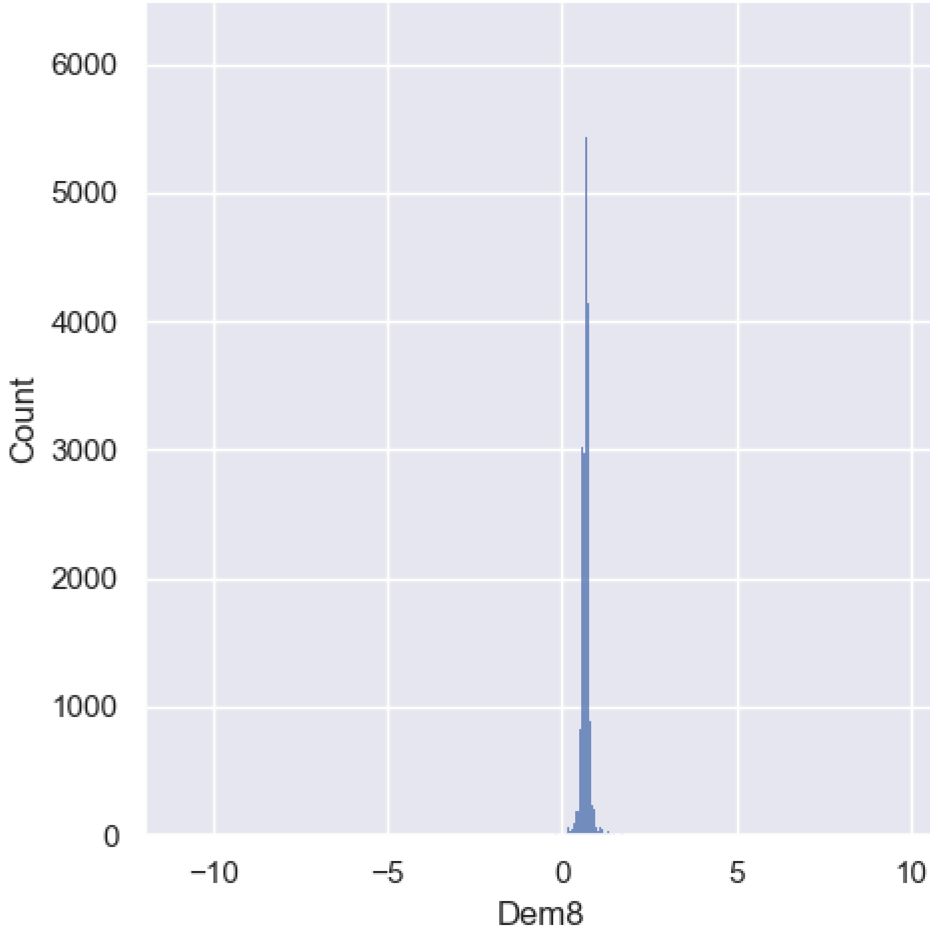
Count
Dem3

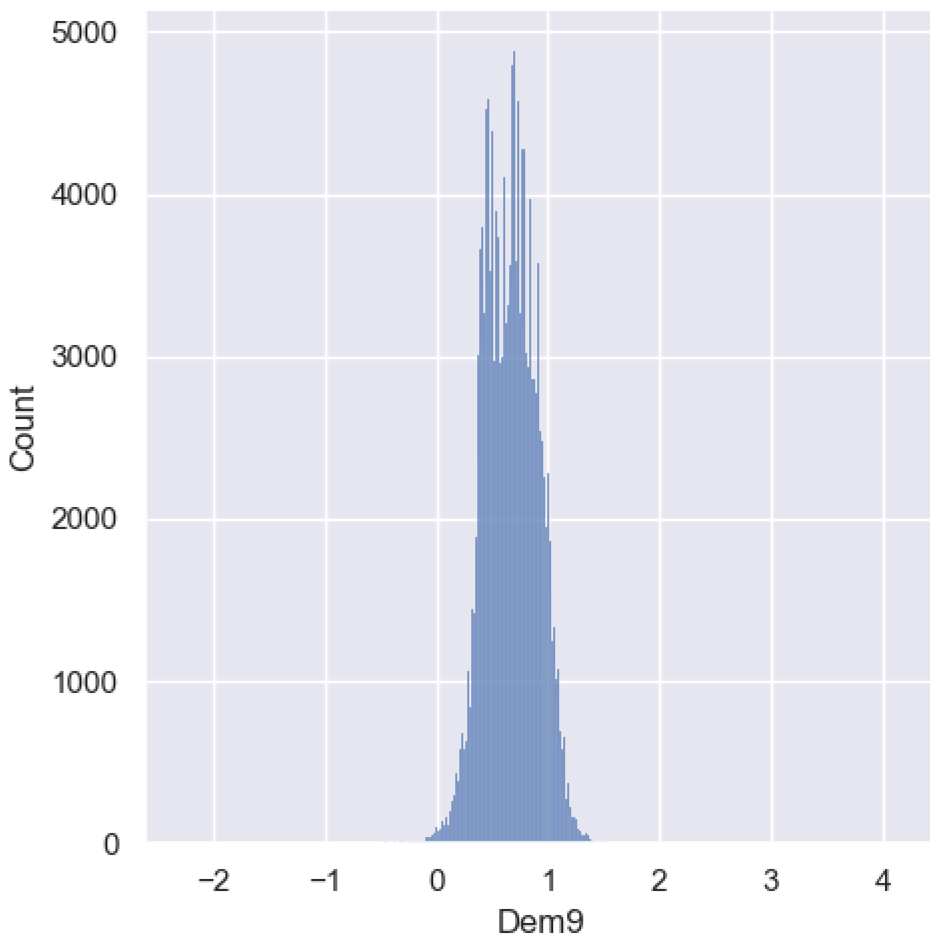

Count
Dem4
0
1000
2000
3000
4000
5000
−4
−2
0
2
4
6

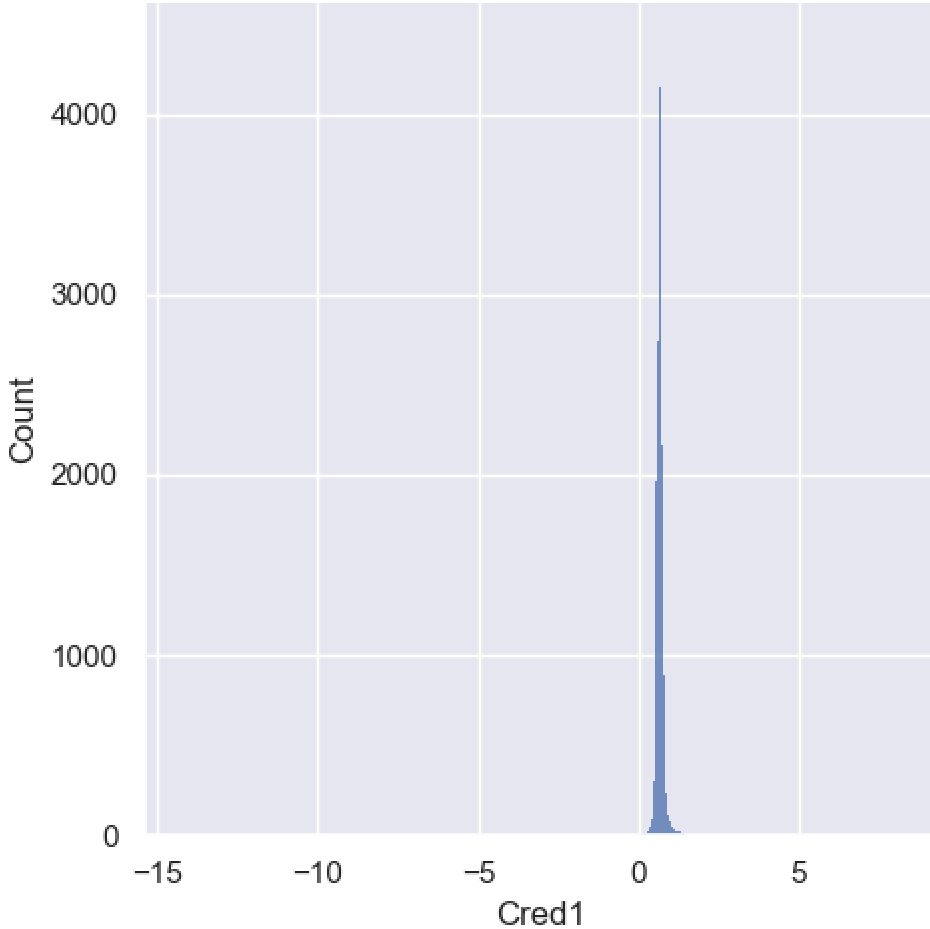

Count
Dem5
0
1000
2000
3000
4000
5000
−2
−1
0
1
2

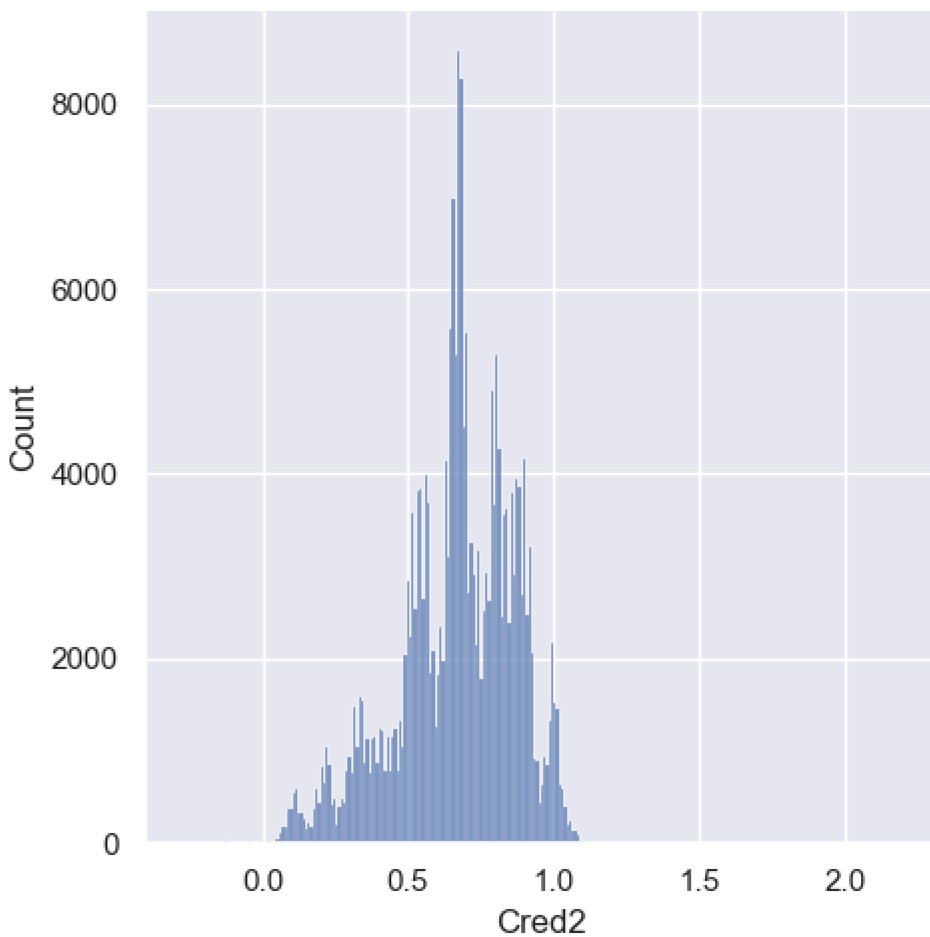

4000
3000
2000
1000
0
Count
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Dem6

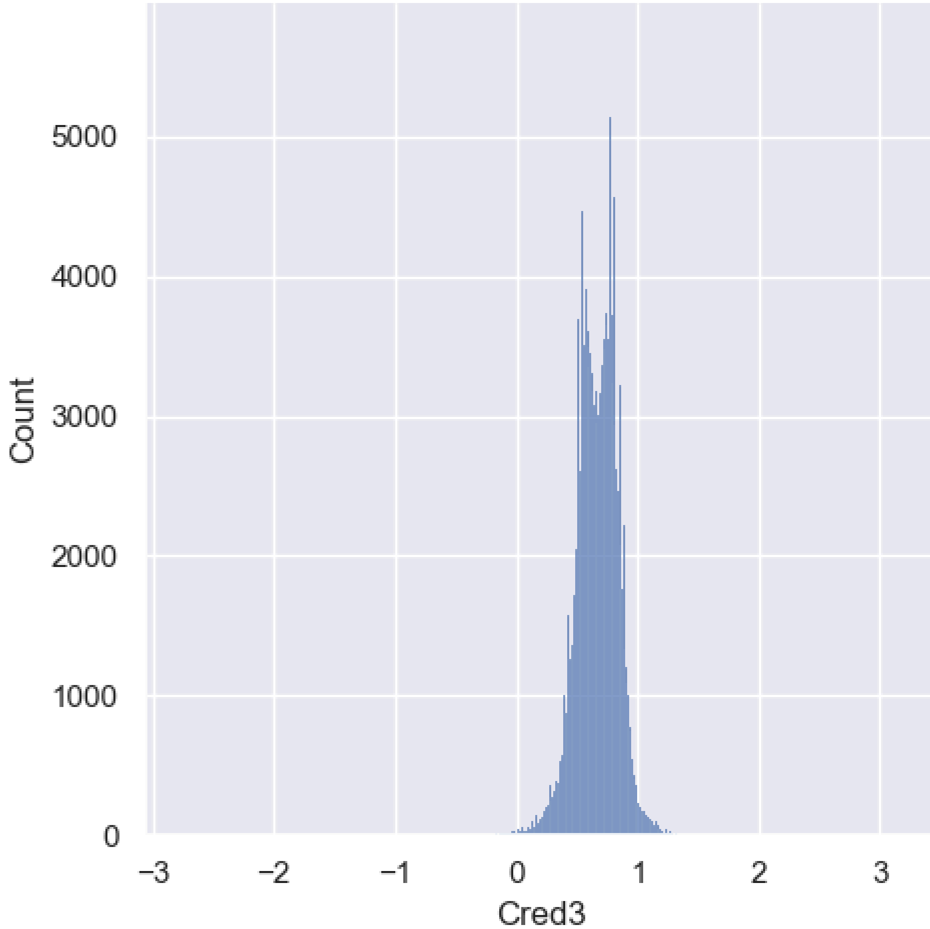

8000
7000
6000
5000
4000
3000
2000
1000
0
Count
−15
−10
−5
0
5
10
Dem7

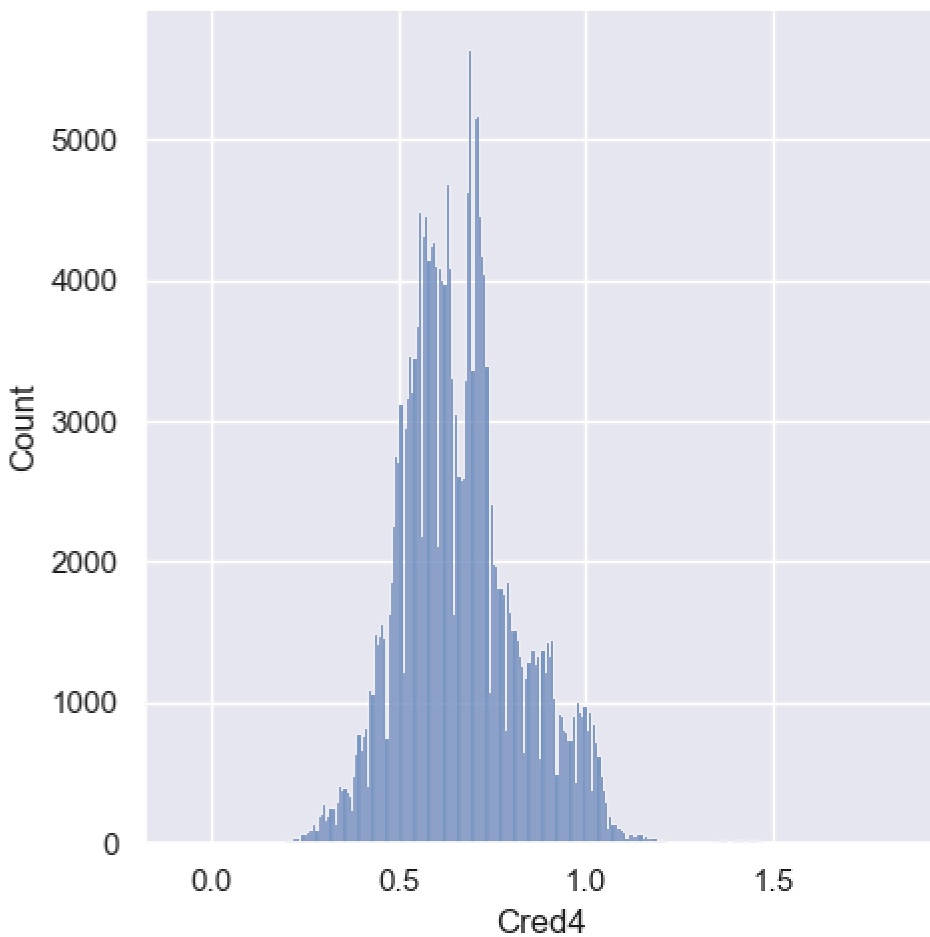

6000
5000
4000
3000
2000
1000
0
Count
−10
−5
0
5
10
Dem8

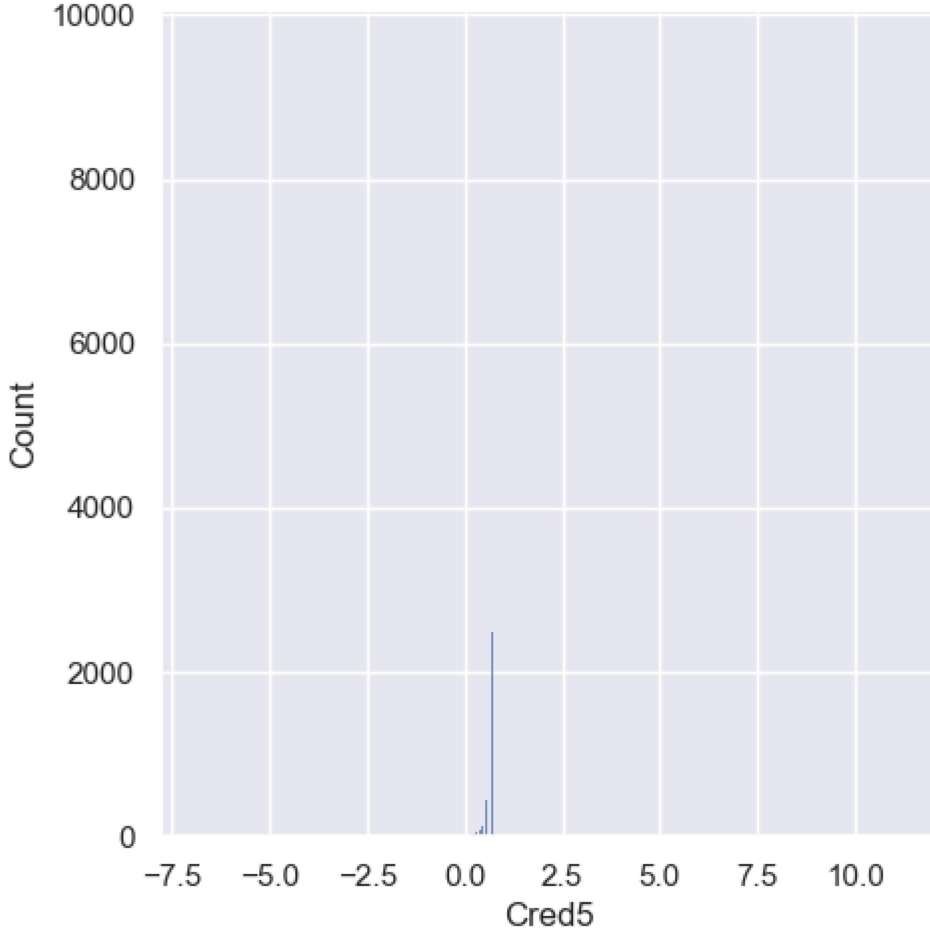

5000
4000
3000
2000
1000
0
Count
−2
−1
0
1
2
3
4
Dem9

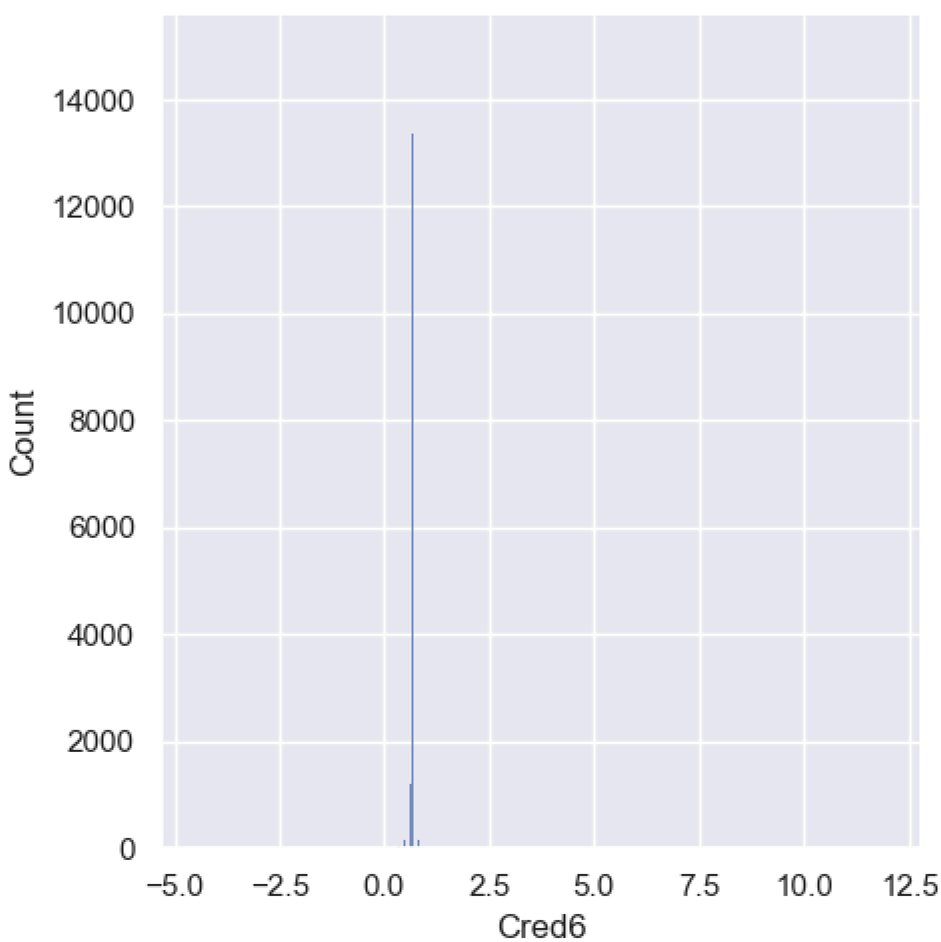

4000
3000
2000
1000
0
Count
−15
−10
−5
0
5
Cred1

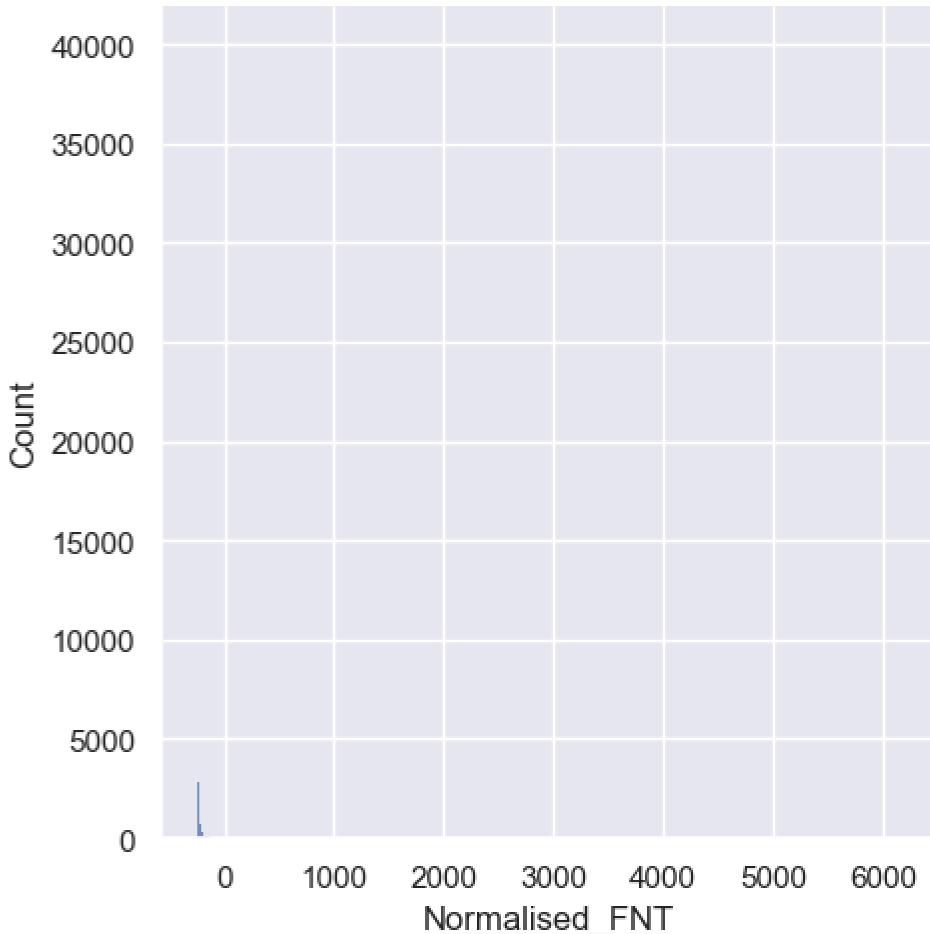

8000
6000
4000
2000
0
Count
0.0
0.5
1.0
1.5
2.0
Cred2

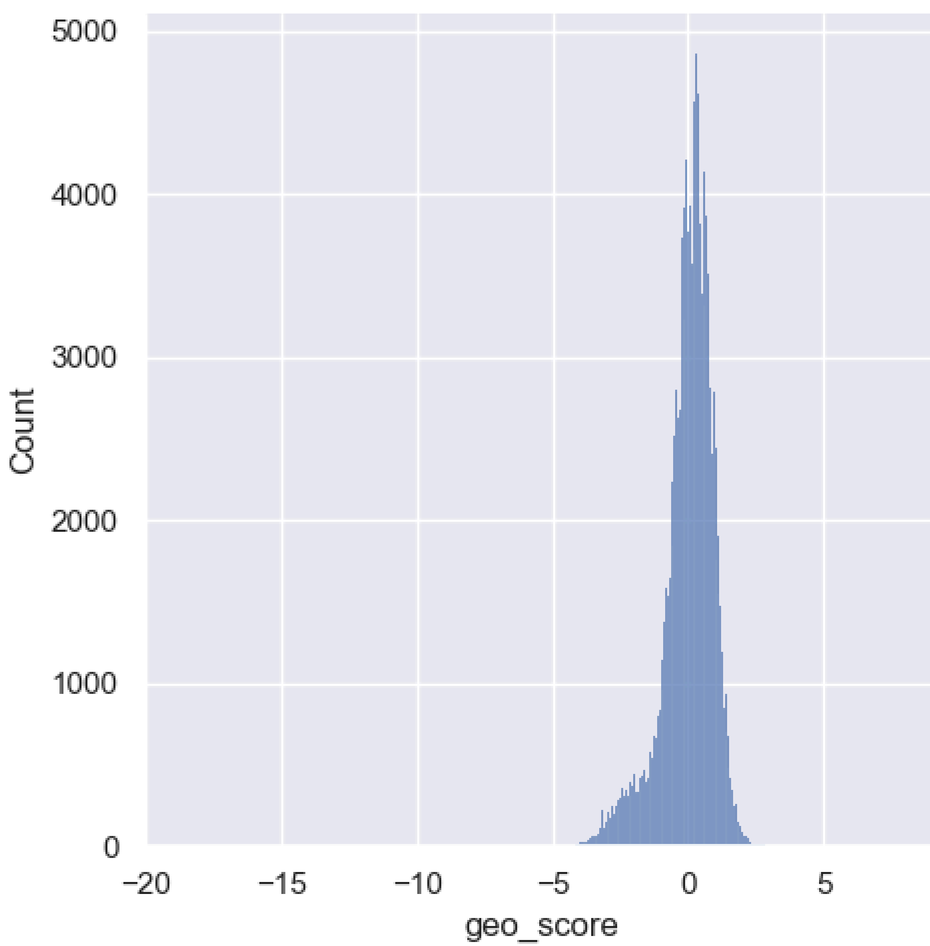

5000
4000
3000
2000
1000
0
Count
−3
−2
−1
0
1
2
3
Cred3

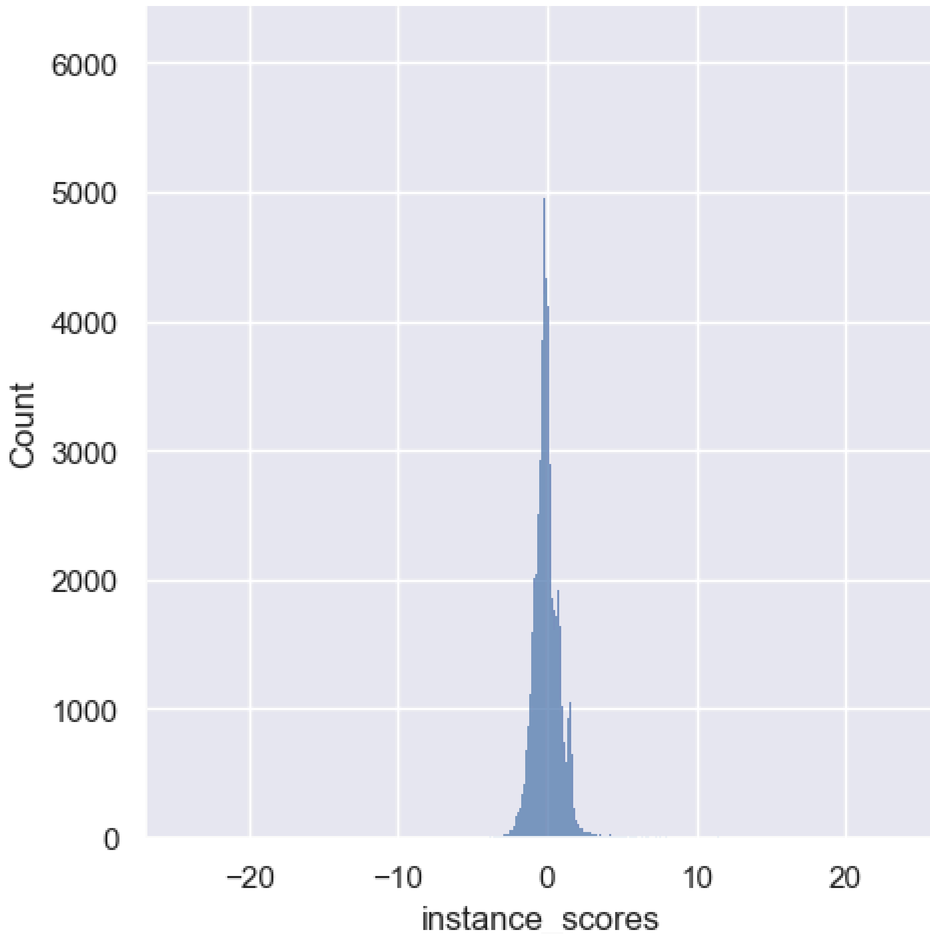

5000
4000
3000
2000
1000
0
Count
0.0
0.5
1.0
1.5
Cred4

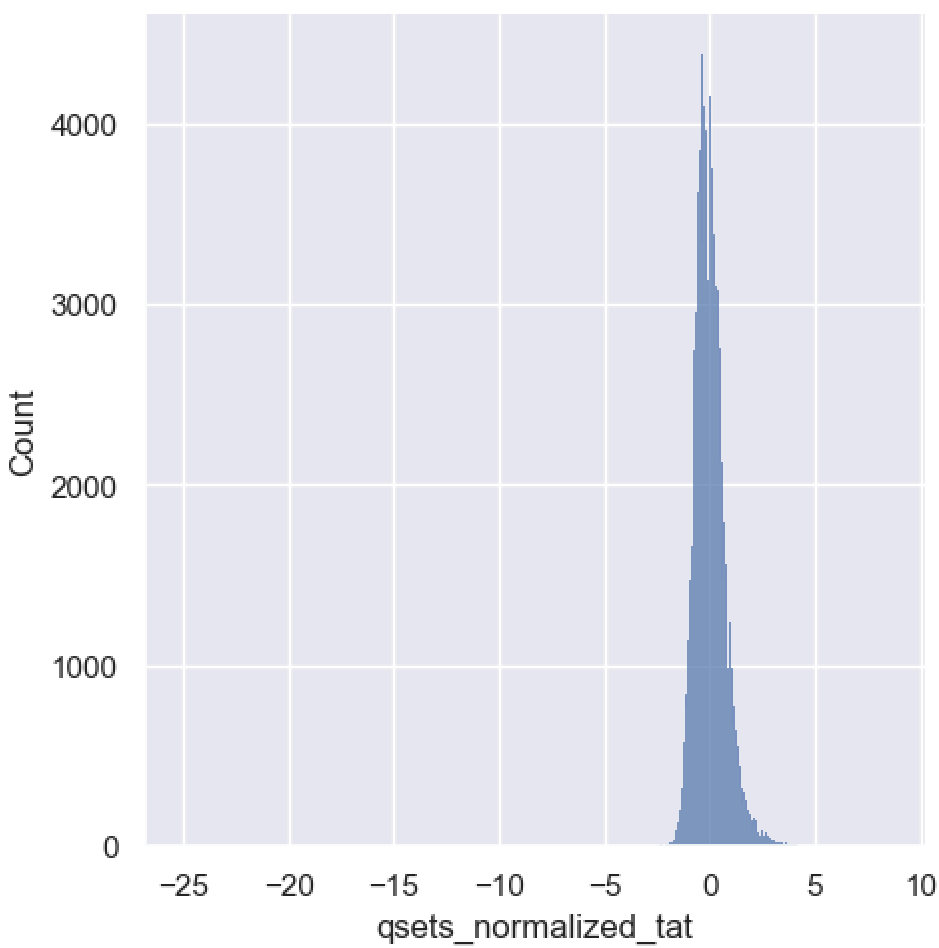

10000
8000
6000
4000
2000
0
Count
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Cred5

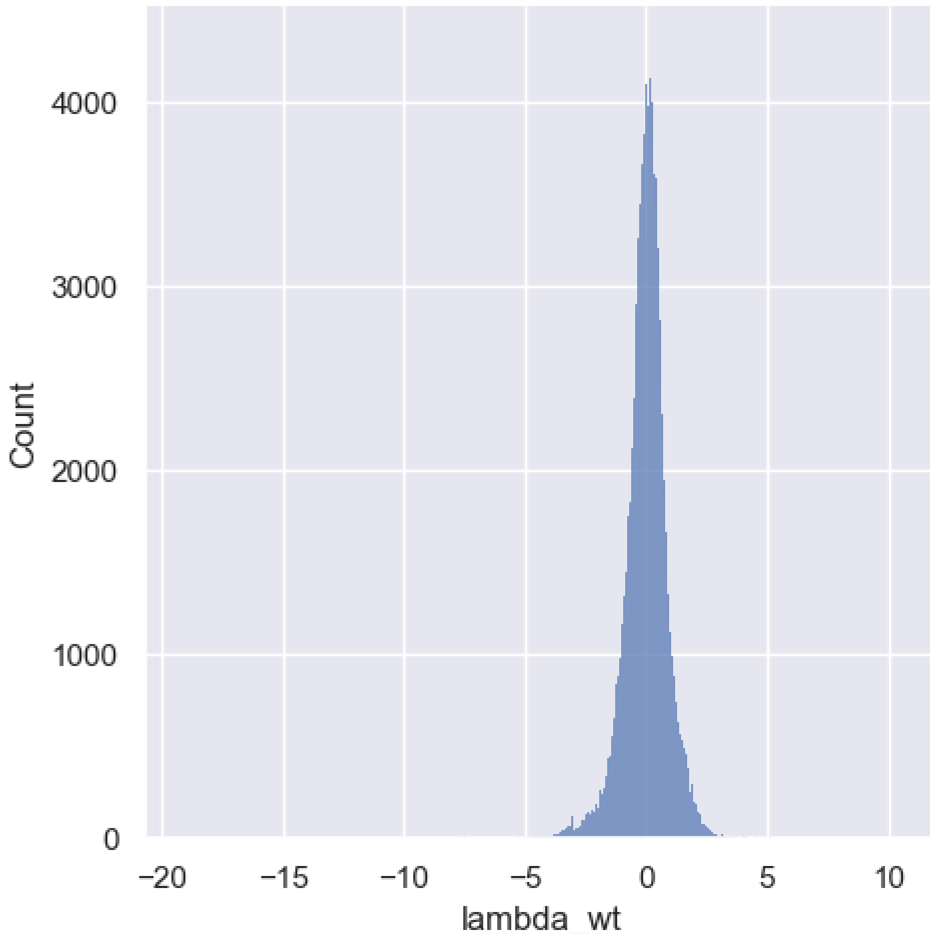

14000
12000
10000
8000
6000
4000
2000
0
Count
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
Cred6


40000
35000
30000
25000
20000
15000
10000
5000
0
Count
0
1000
2000
3000
4000
5000
6000
Normalised_FNT


5000
4000
3000
2000
1000
0
Count
−20
−15
−10
−5
0
5
geo_score

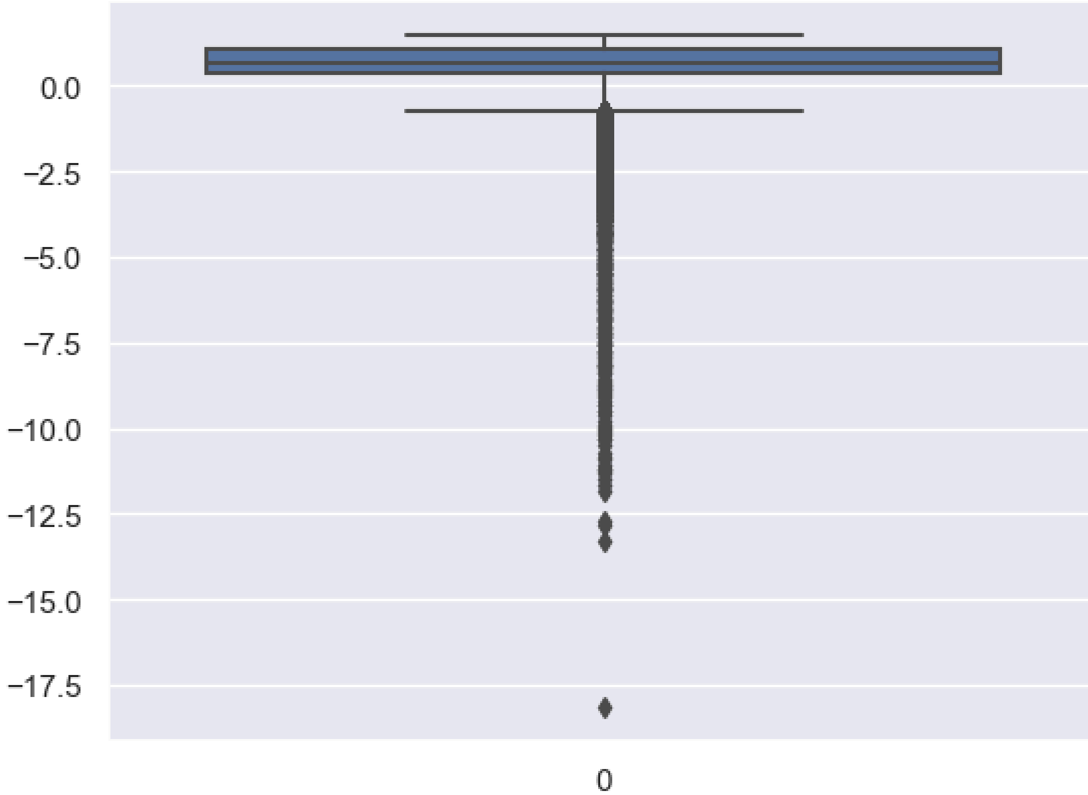

6000
5000
4000
3000
2000
1000
0
Count
−20
−10
0
10
20
instance_scores

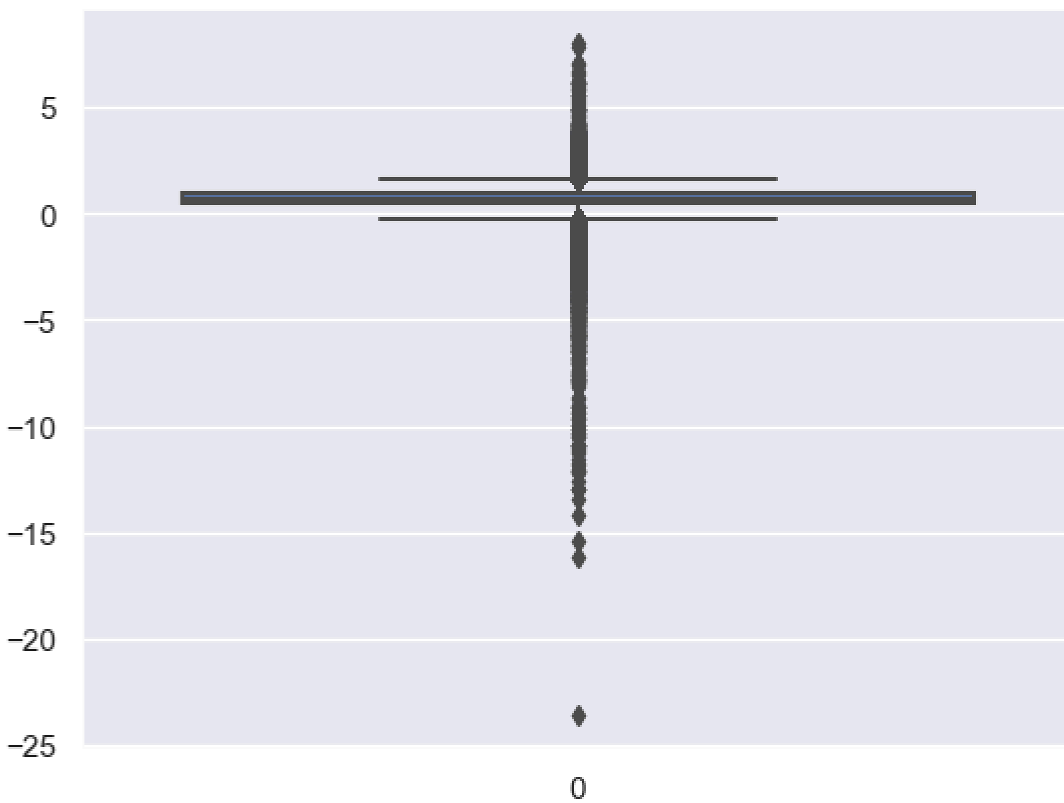

4000
3000
2000
1000
0
Count
−25
−20
−15
−10
−5
0
5
10
qsets_normalized_tat

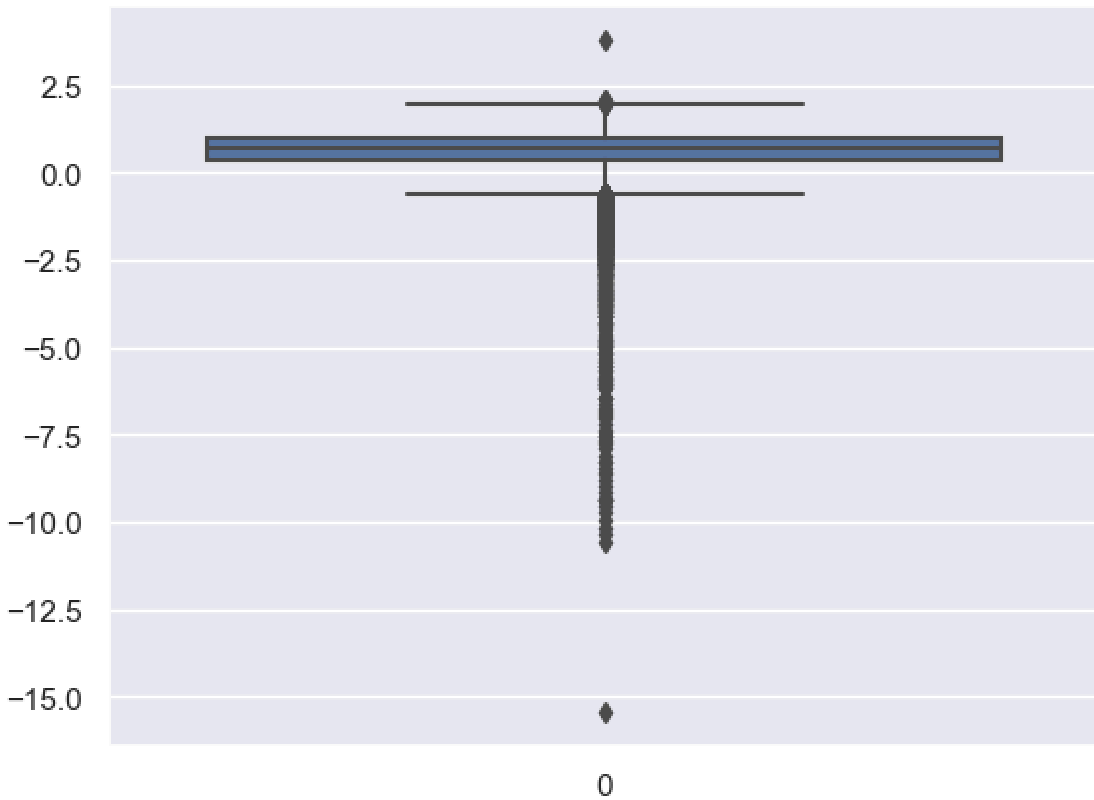


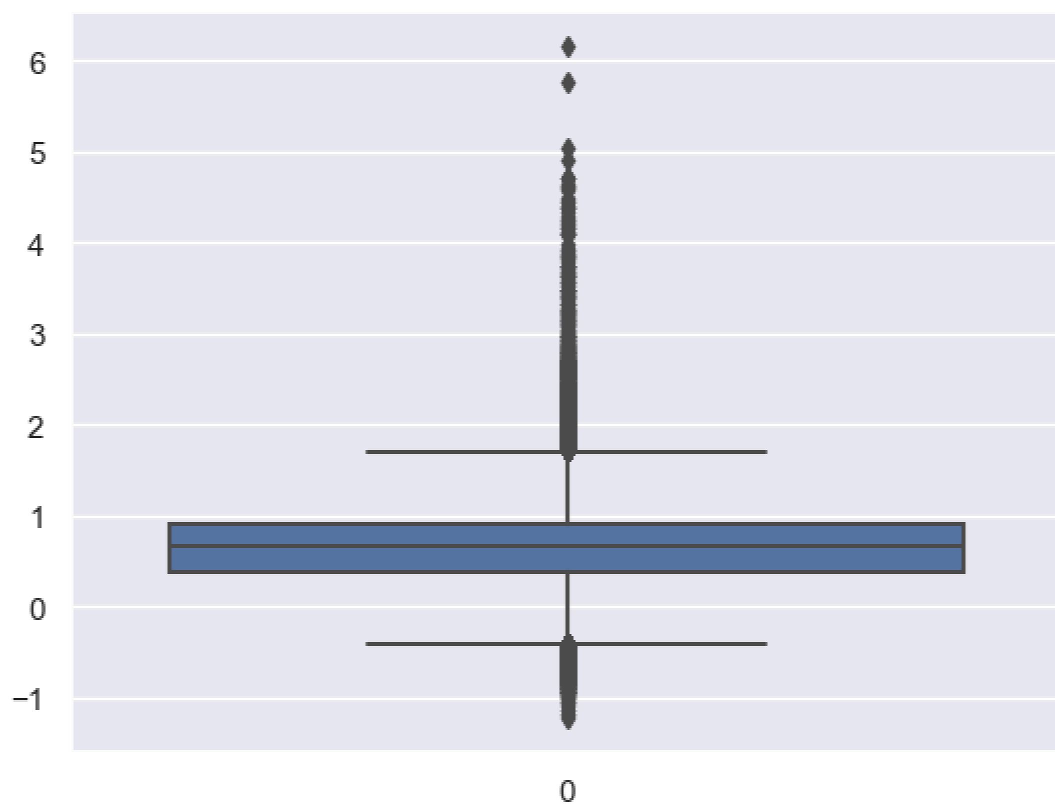
In [66]:

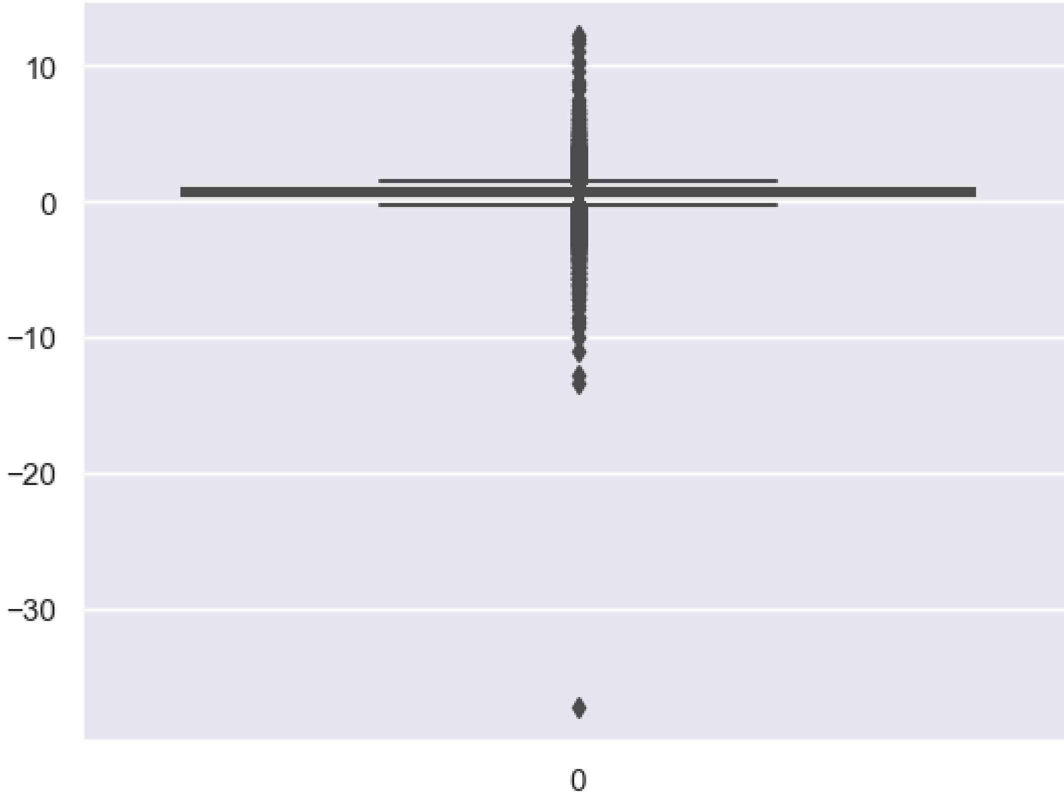
```
def boxplots(col):
    sns.boxplot(x[col])
    plt.show()

for i in list(x.select_dtypes(exclude=['object']).columns)[0:]:
    boxplots(i)
```

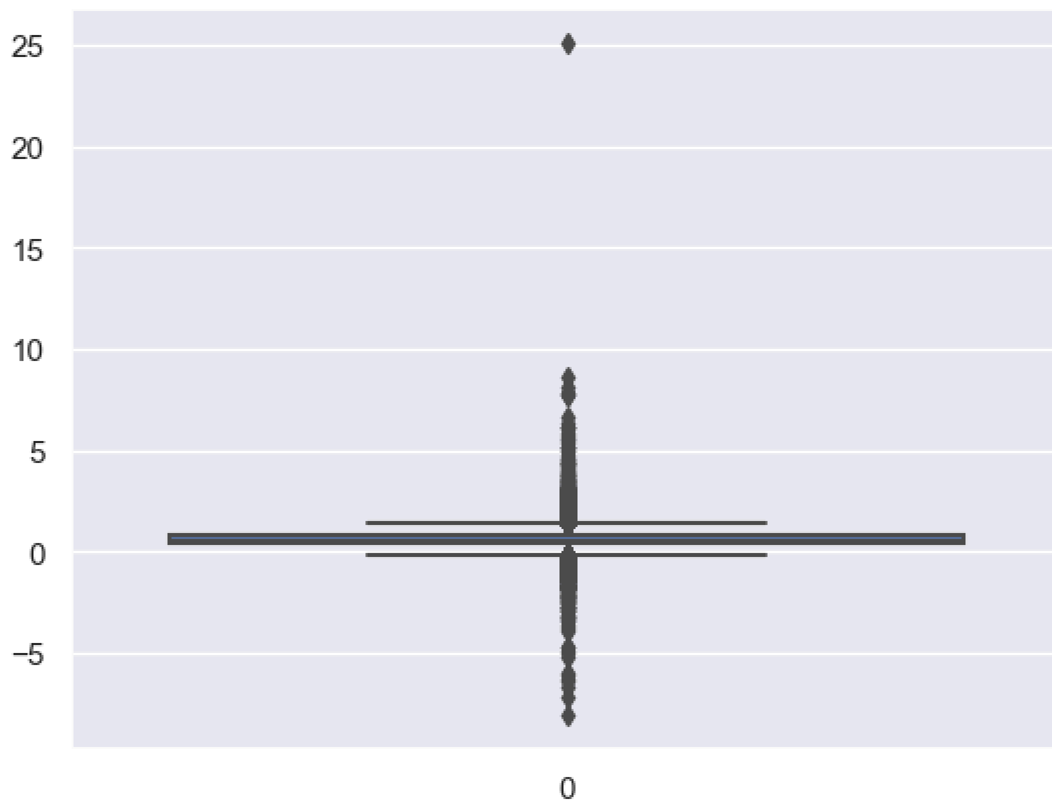

0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
0

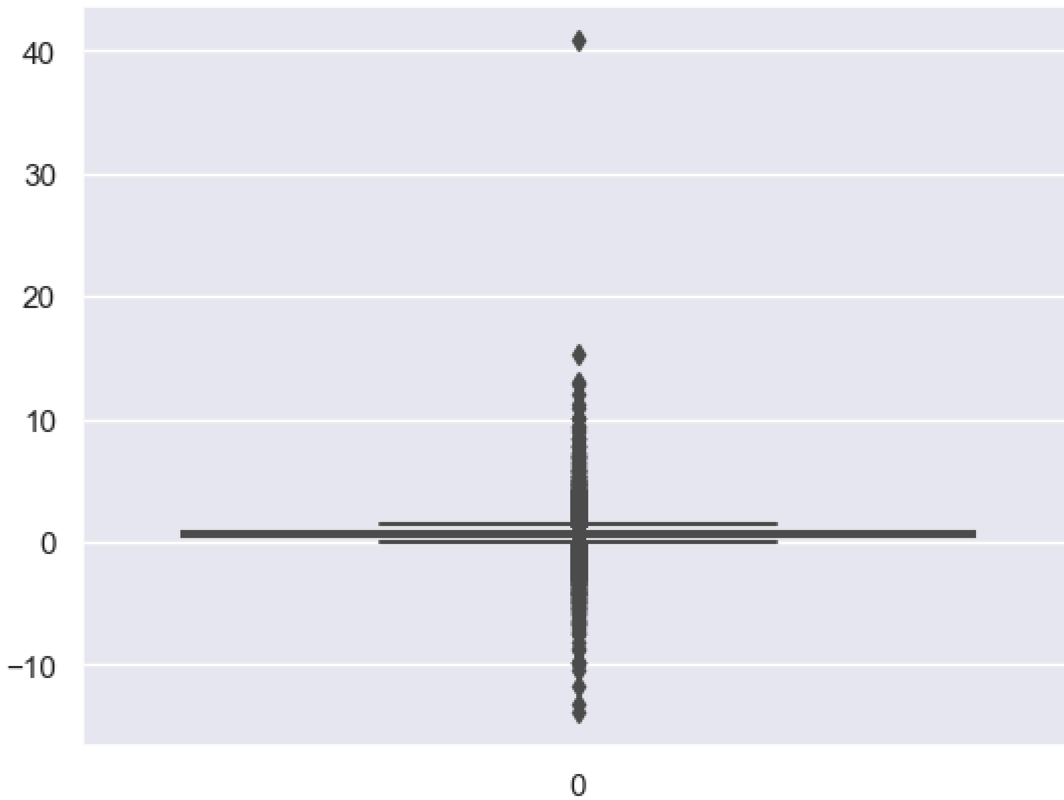

5
0
−5
−10
−15
−20
−25
0

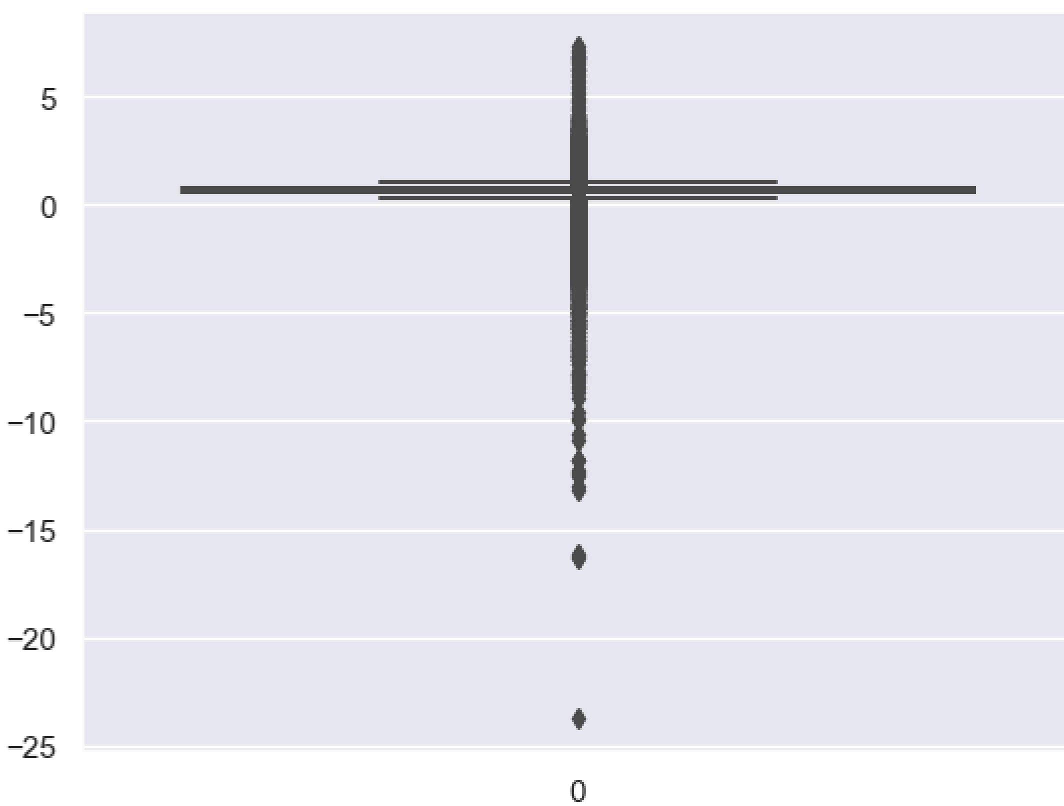

2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
0

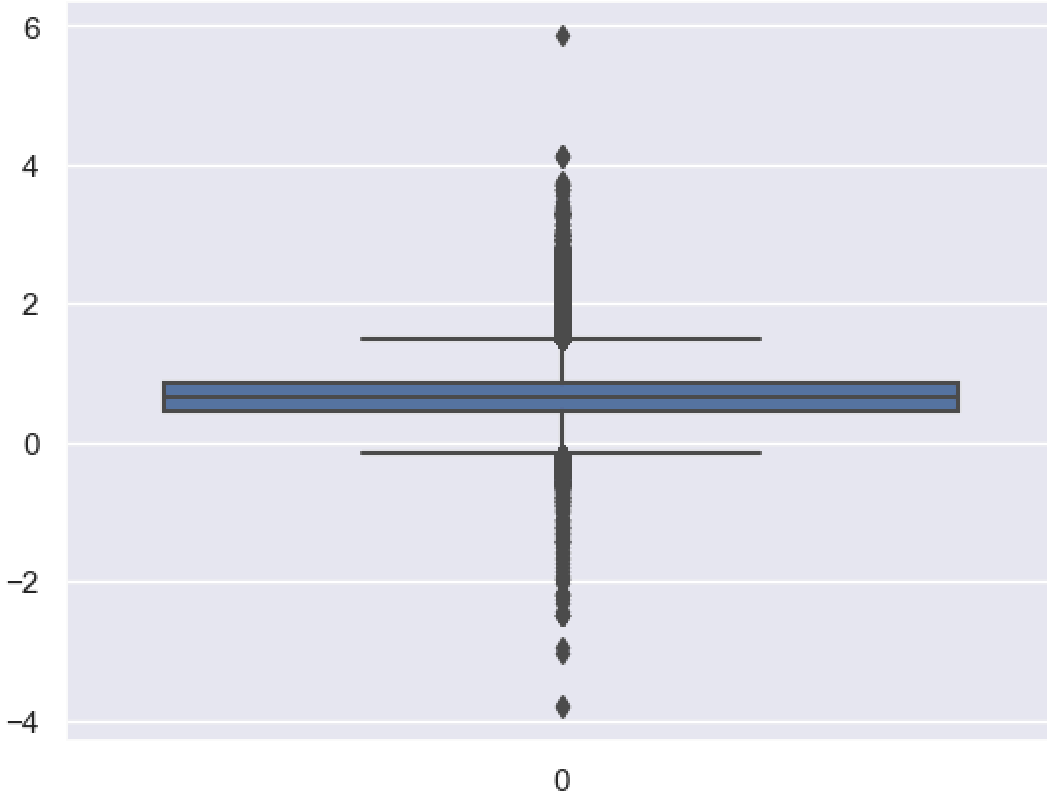

6
5
4
3
2
1
0
−1
0

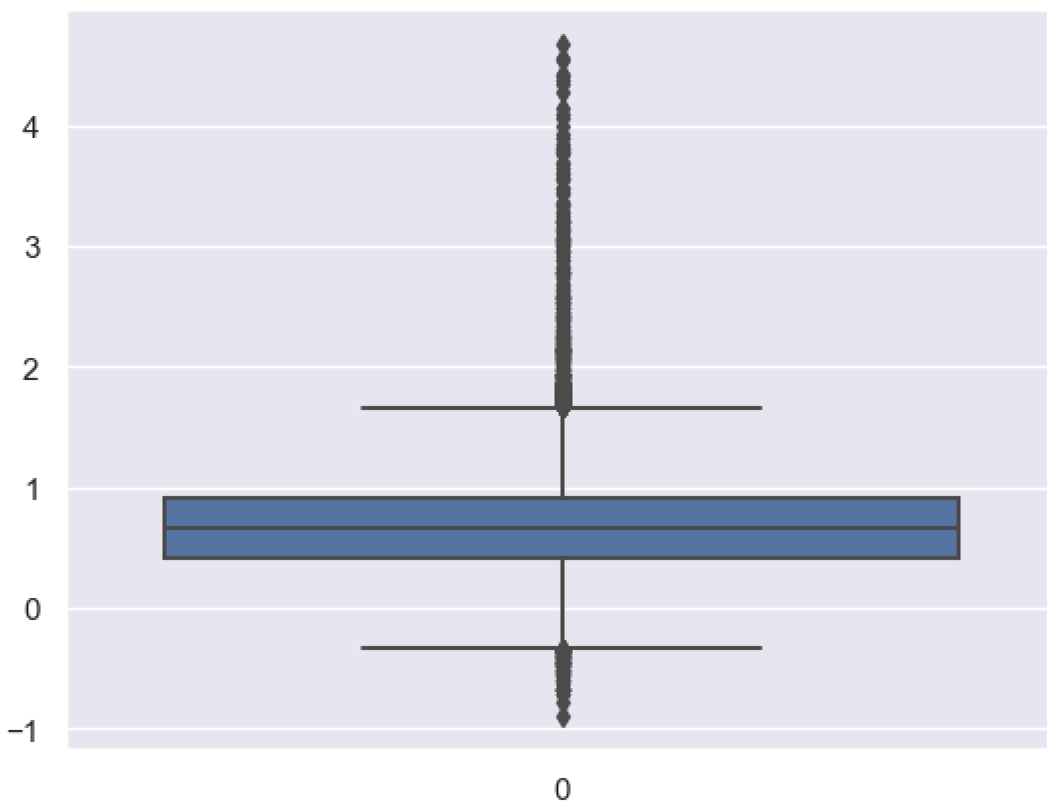

10
0
−10
−20
−30
0

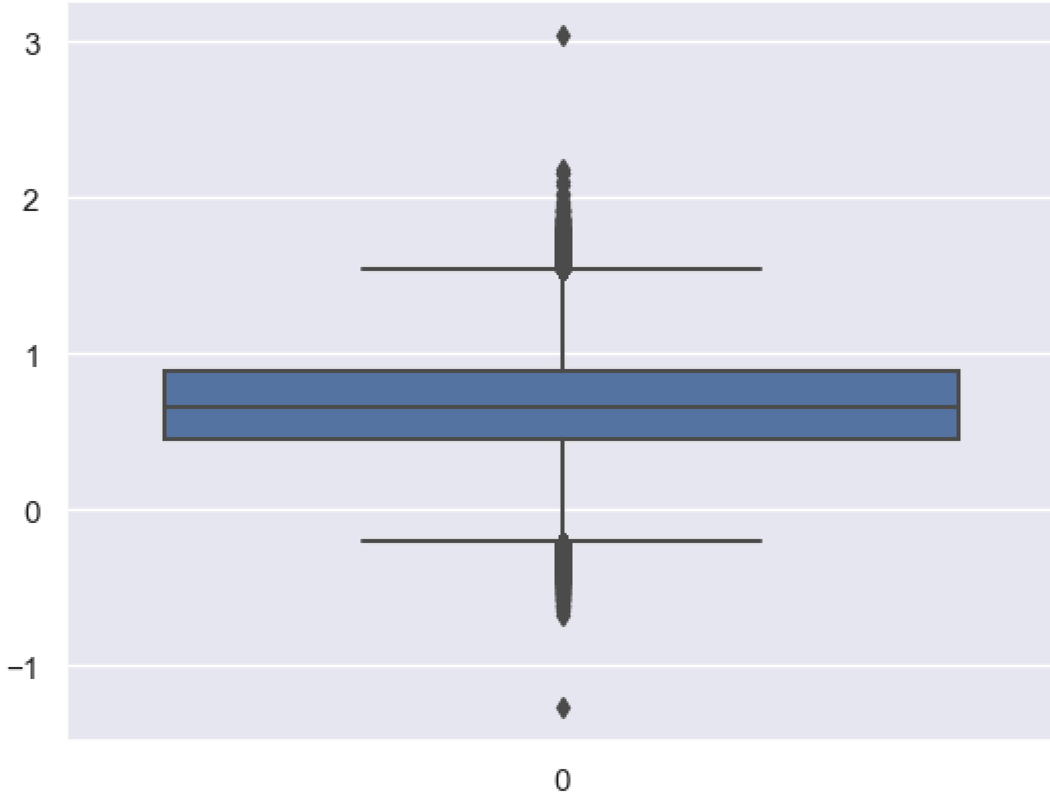

25
20
15
10
5
0
−5
0

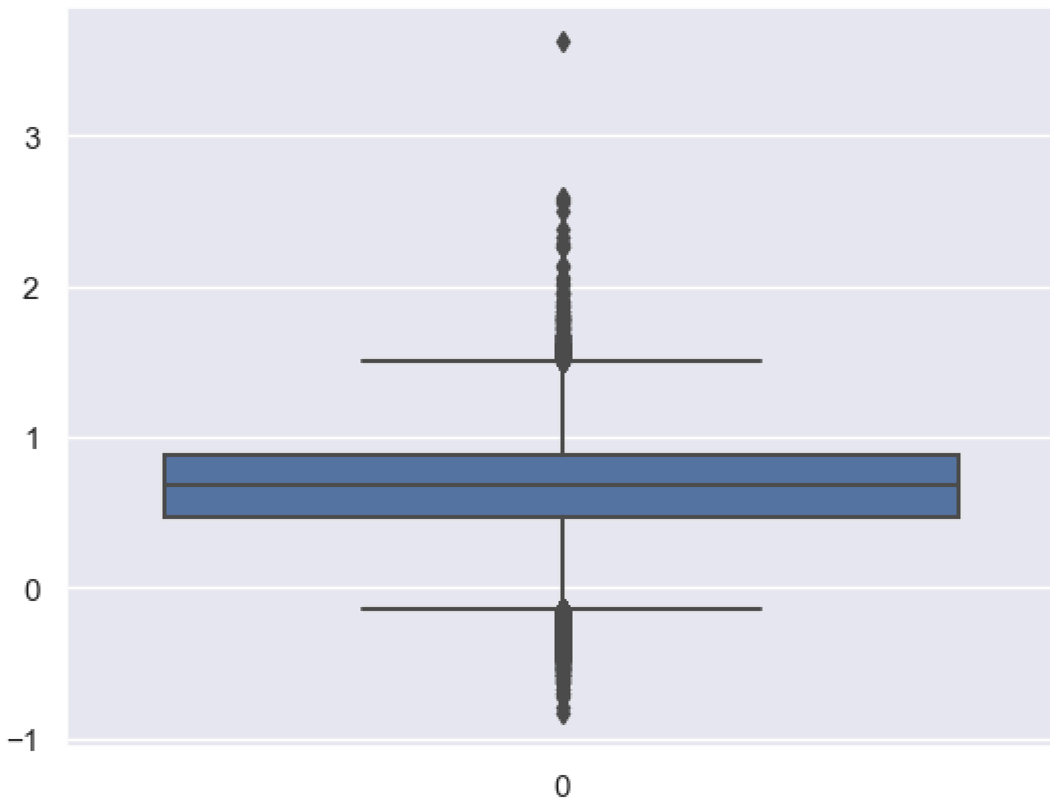

40
30
20
10
0
−10
0

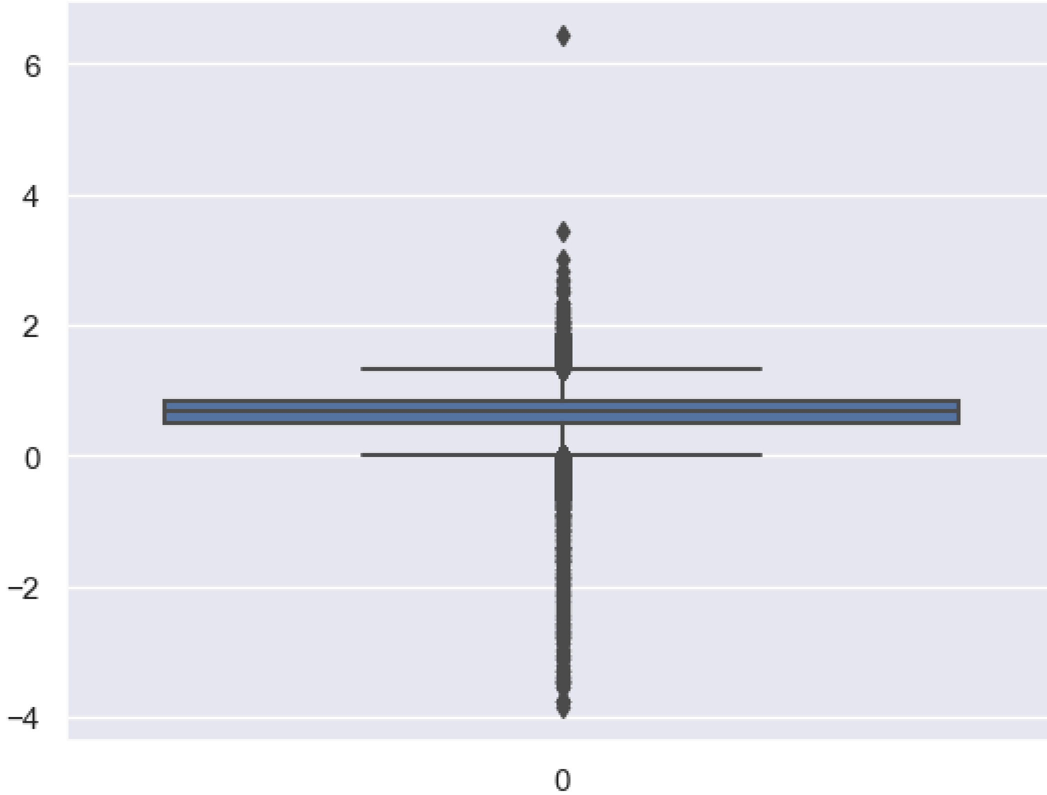

5
0
−5
−10
−15
−20
−25
0

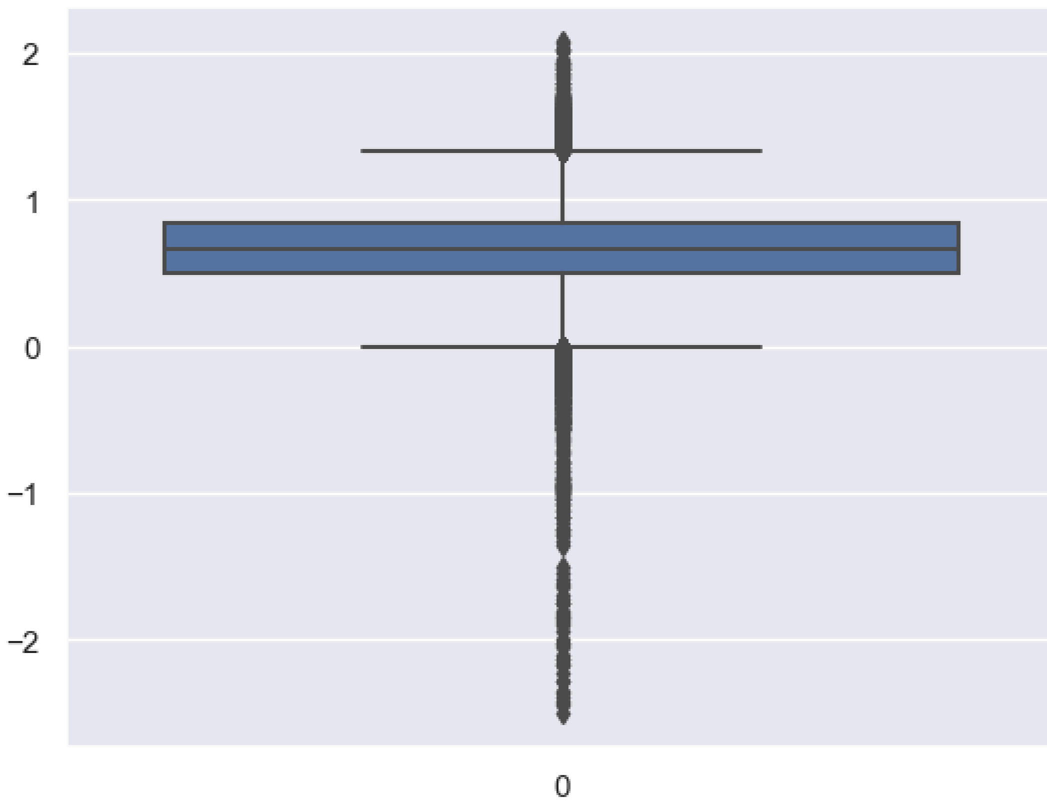

6
4
2
0
−2
−4
0

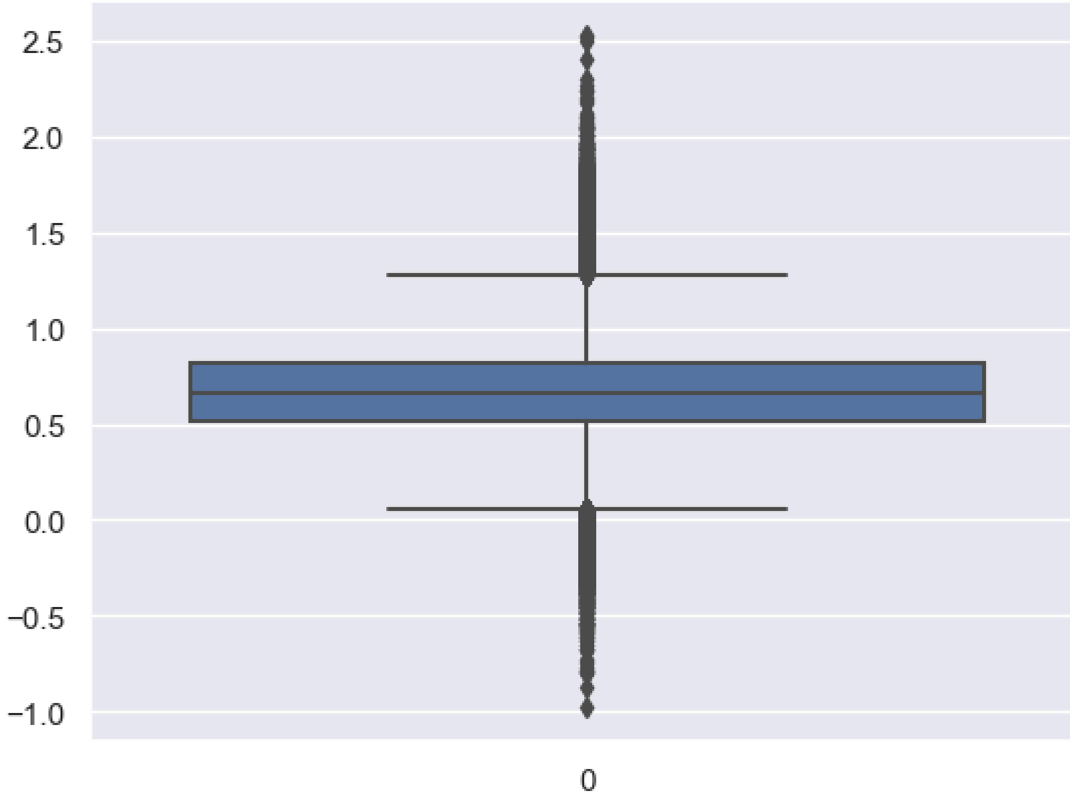

4
3
2
1
0
−1
0

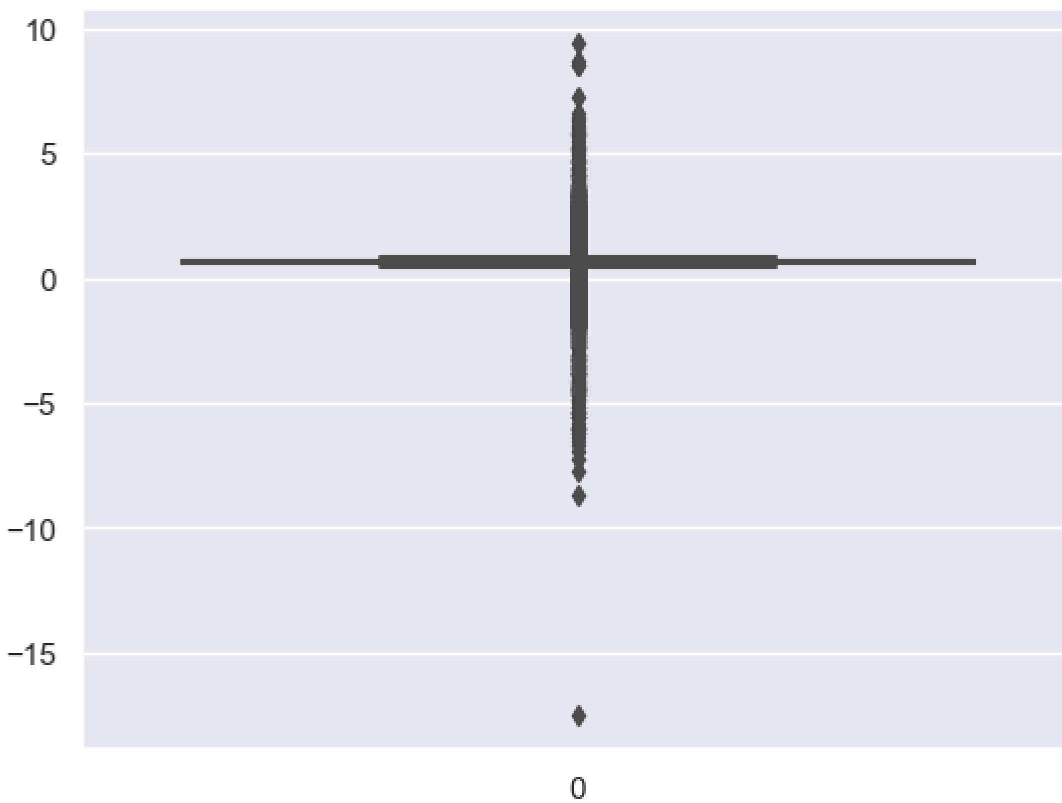

3
2
1
0
−1
0

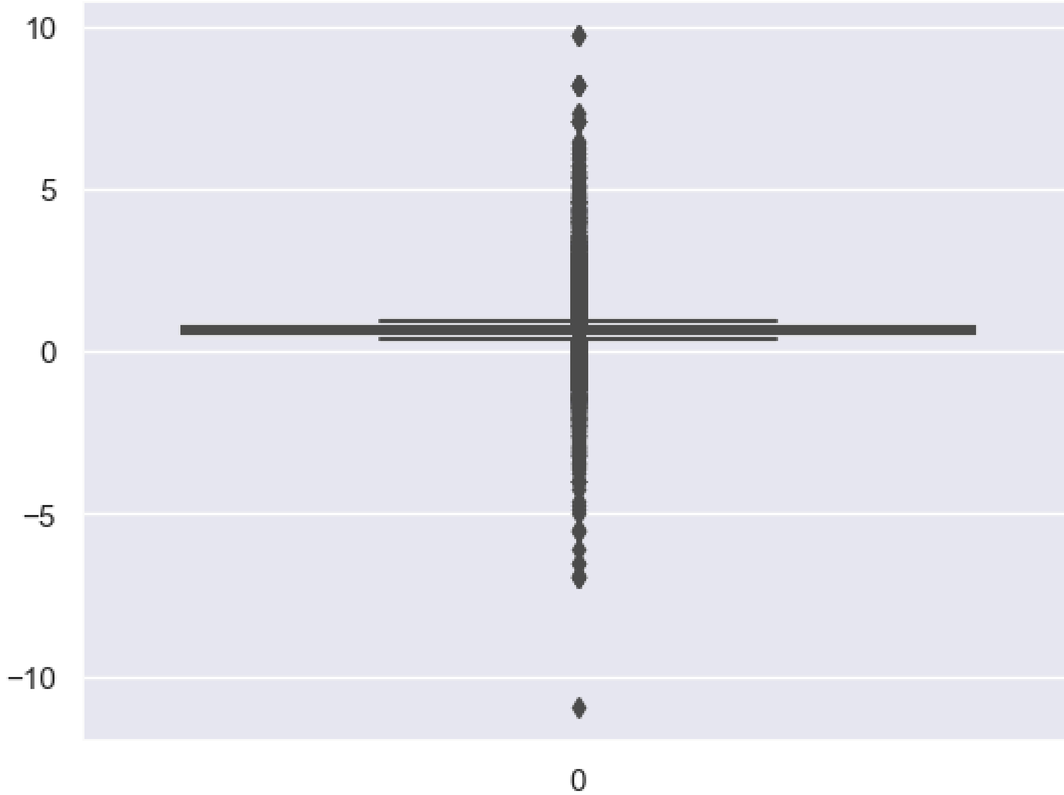

3
2
1
0
−1
0

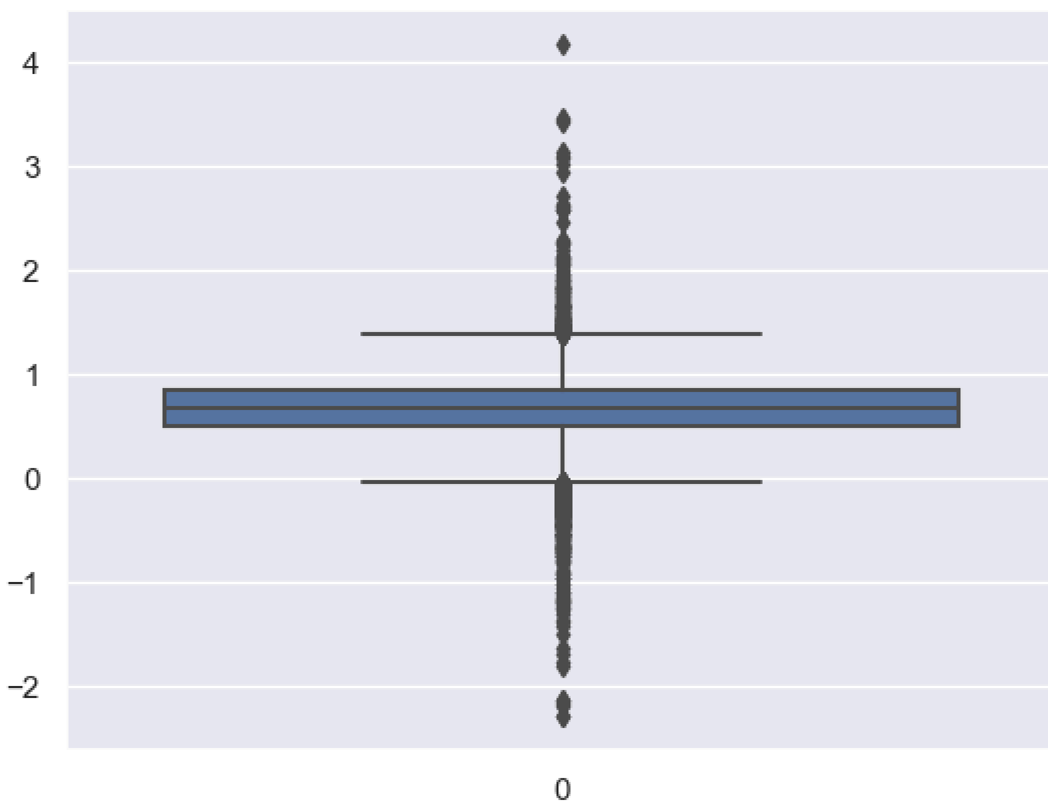

6
4
2
0
−2
−4
0

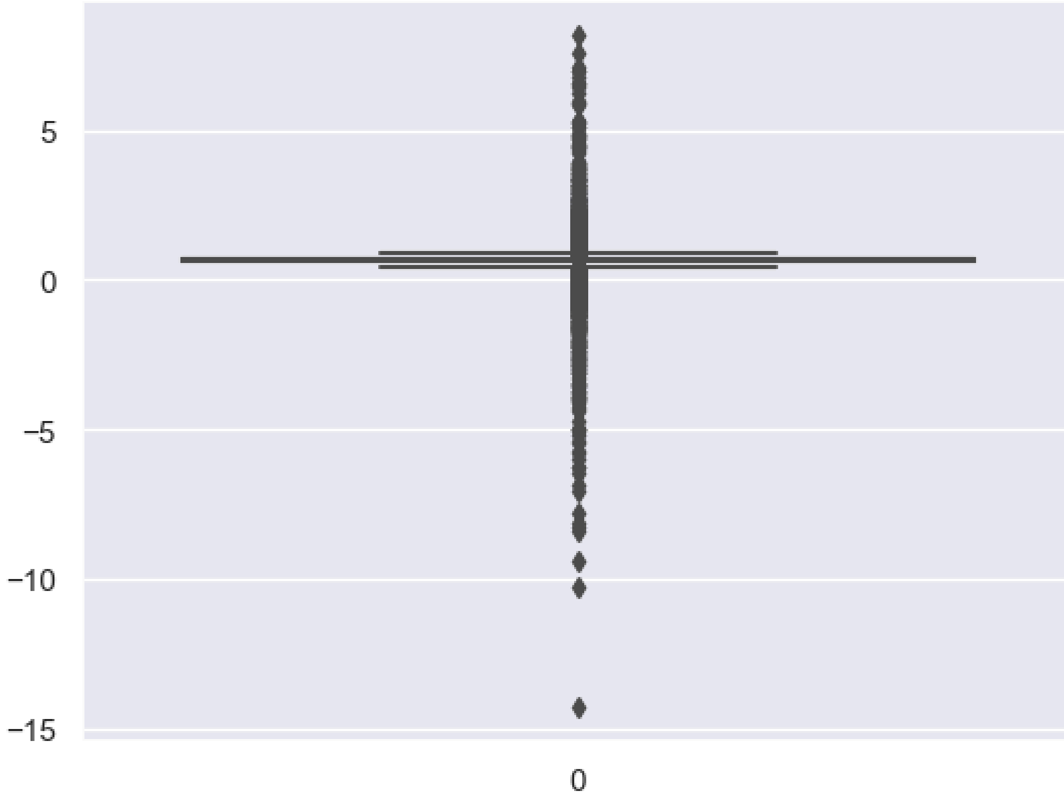

2
1
0
−1
−2
0

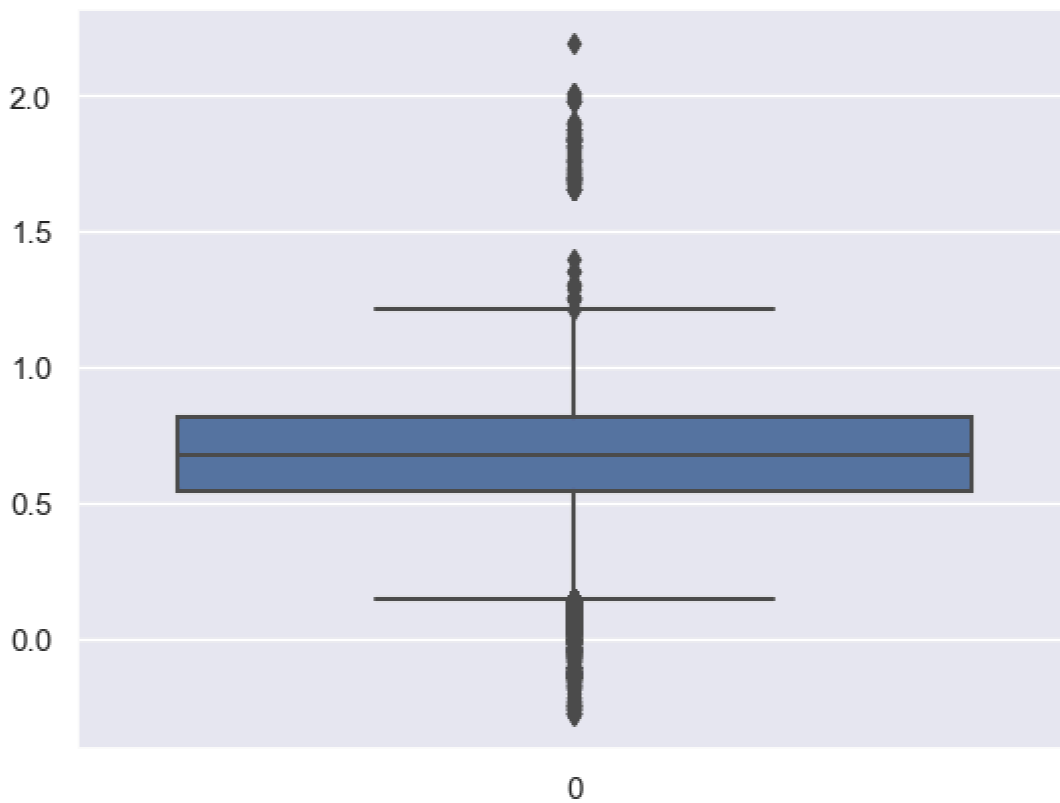

2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0

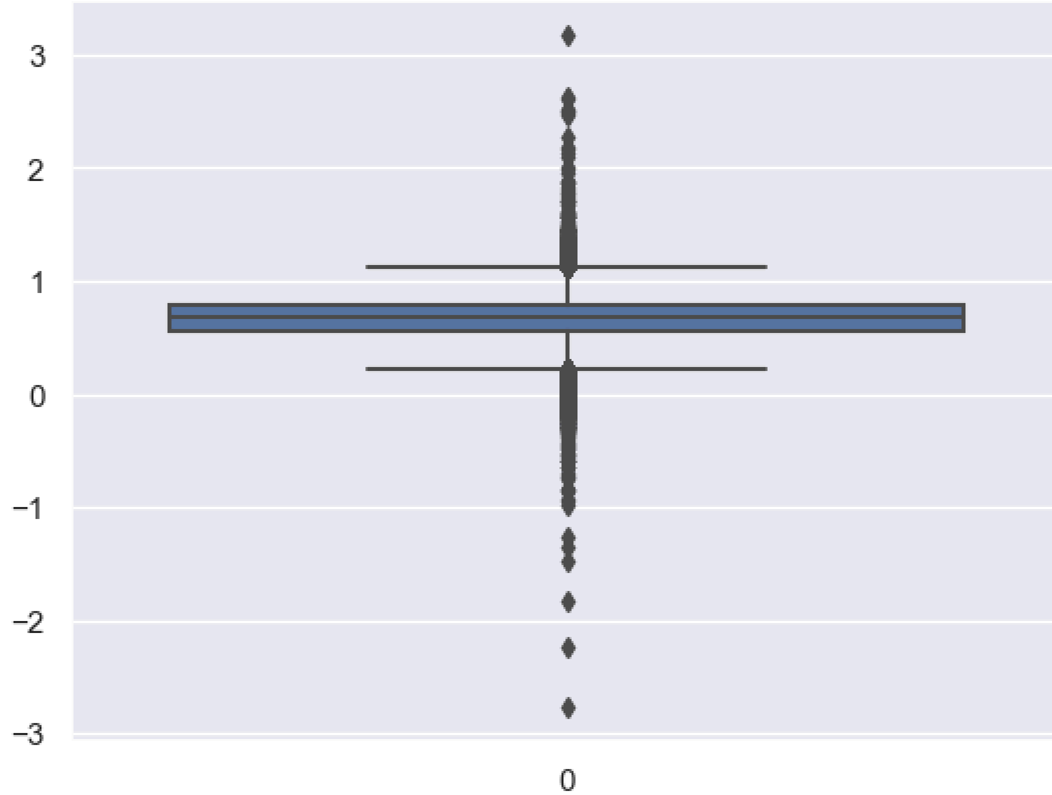

10
5
0
−5
−10
−15
0

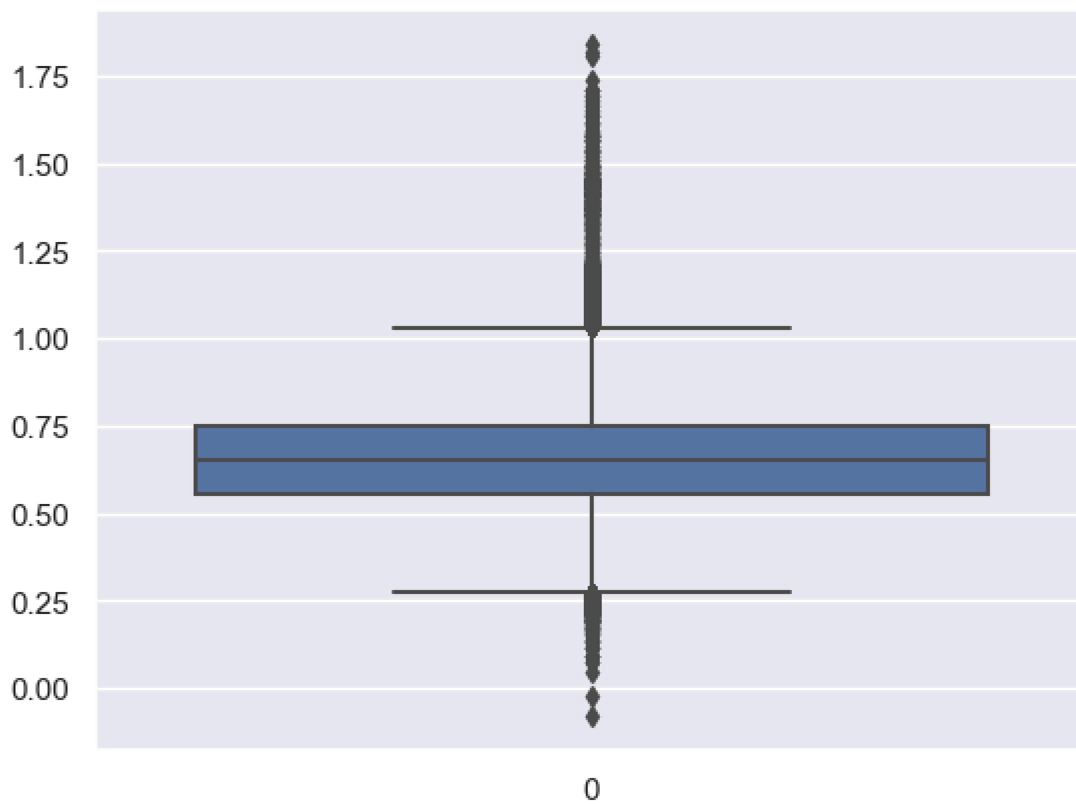

10
5
0
−5
−10
0

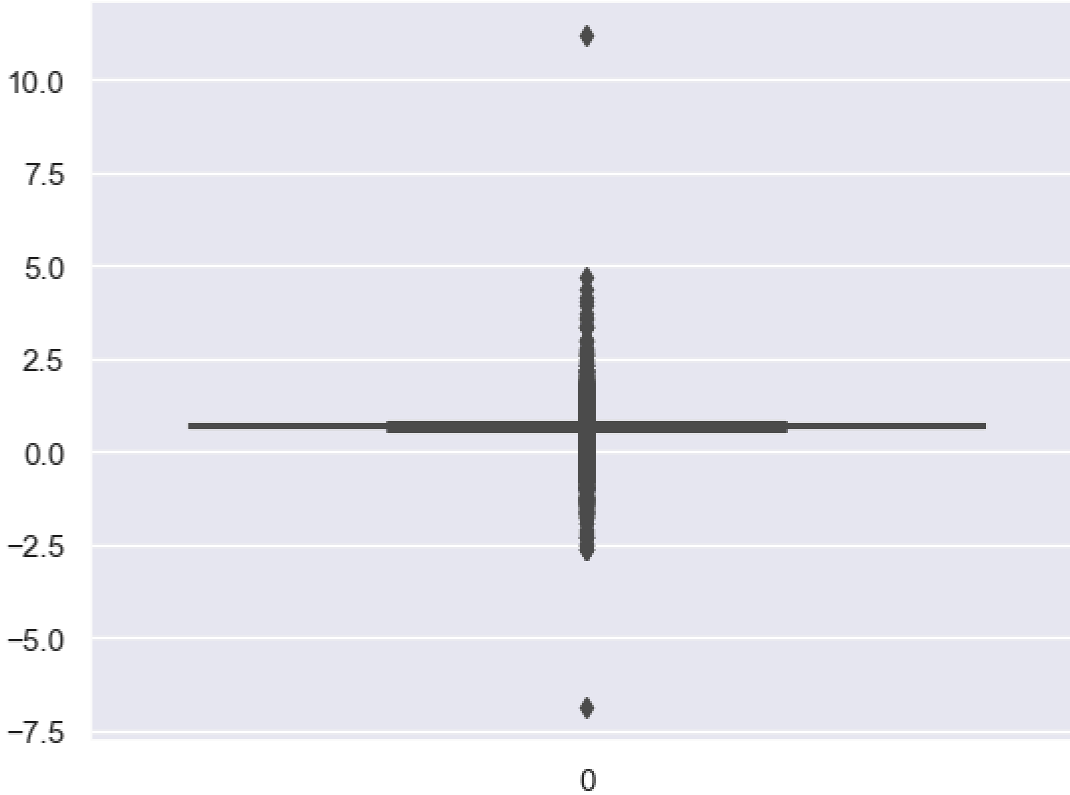

4
3
2
1
0
−1
−2
0

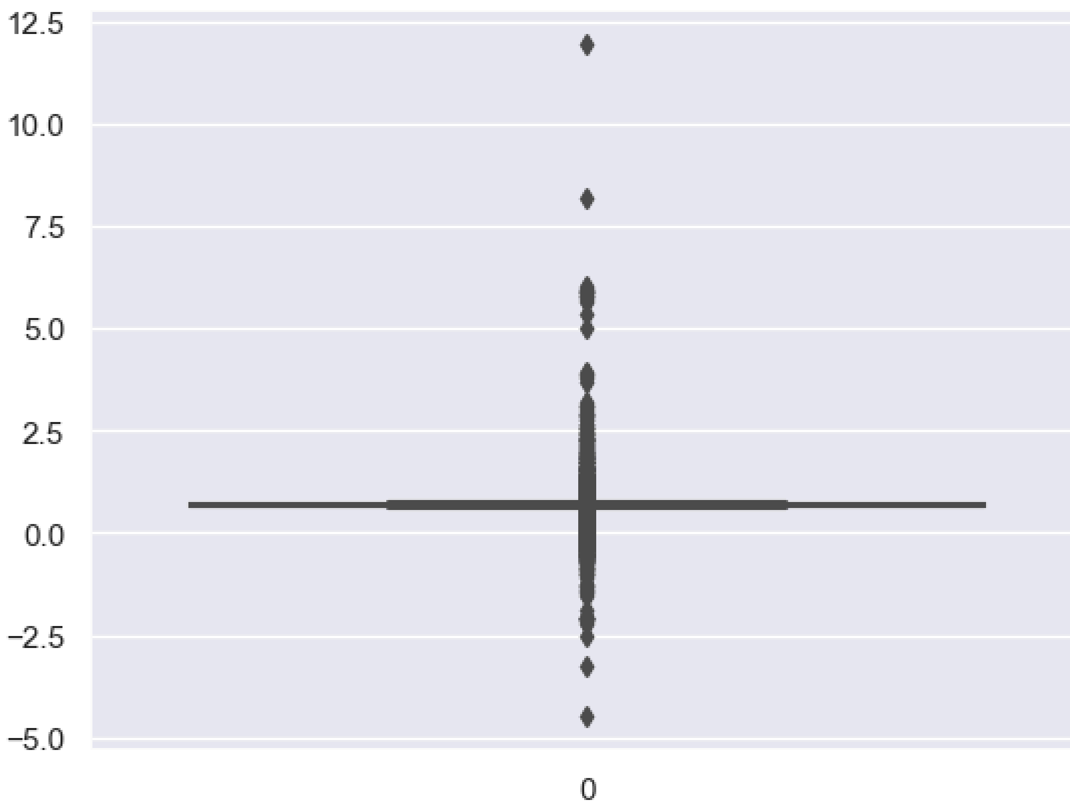

5
0
−5
−10
−15
0

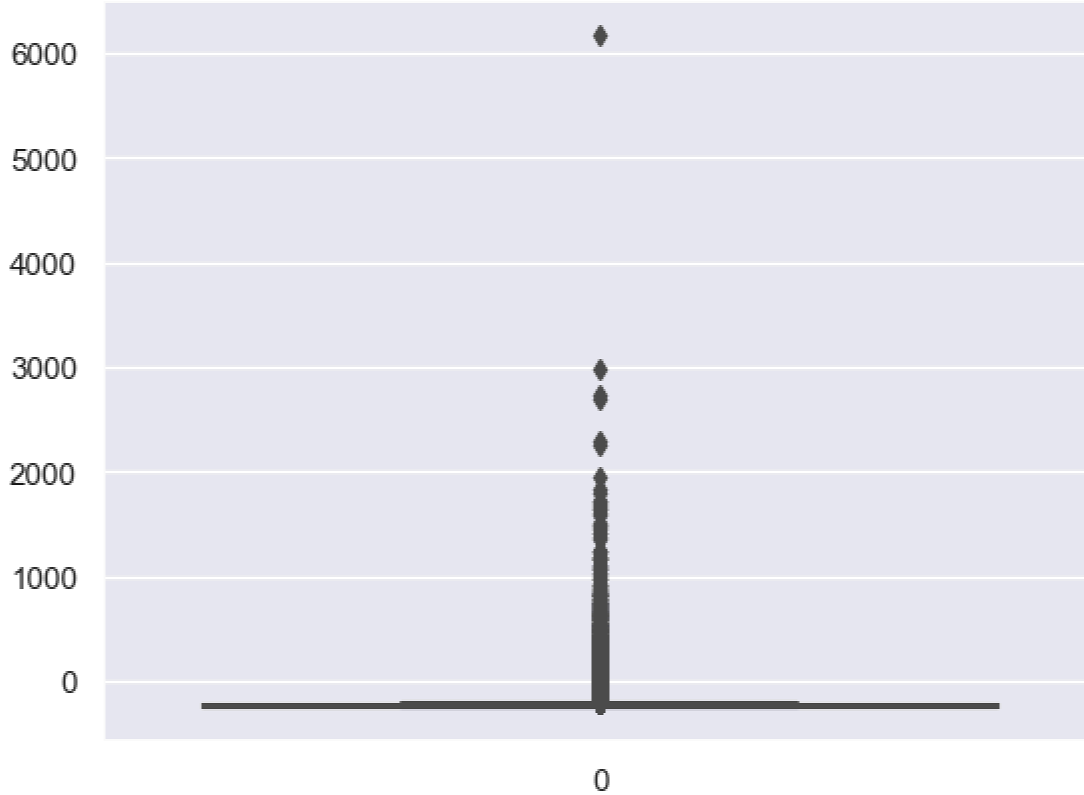

2.0
1.5
1.0
0.5
0.0
0

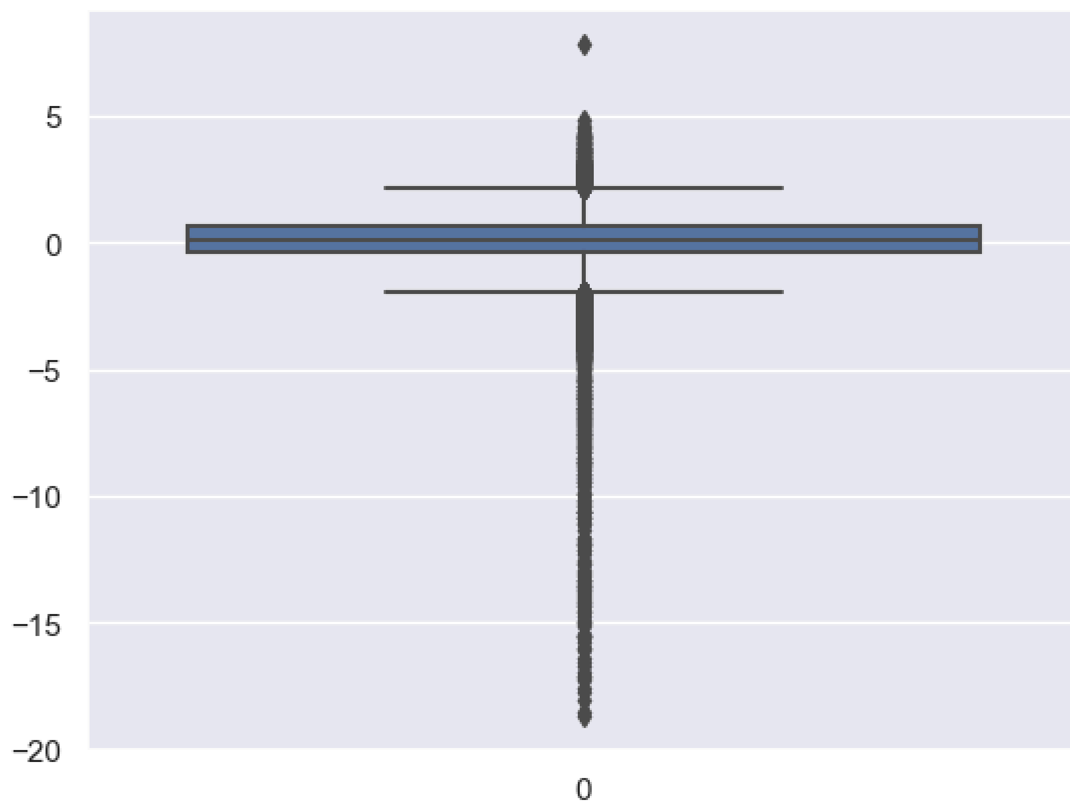

3
2
1
0
−1
−2
−3
0

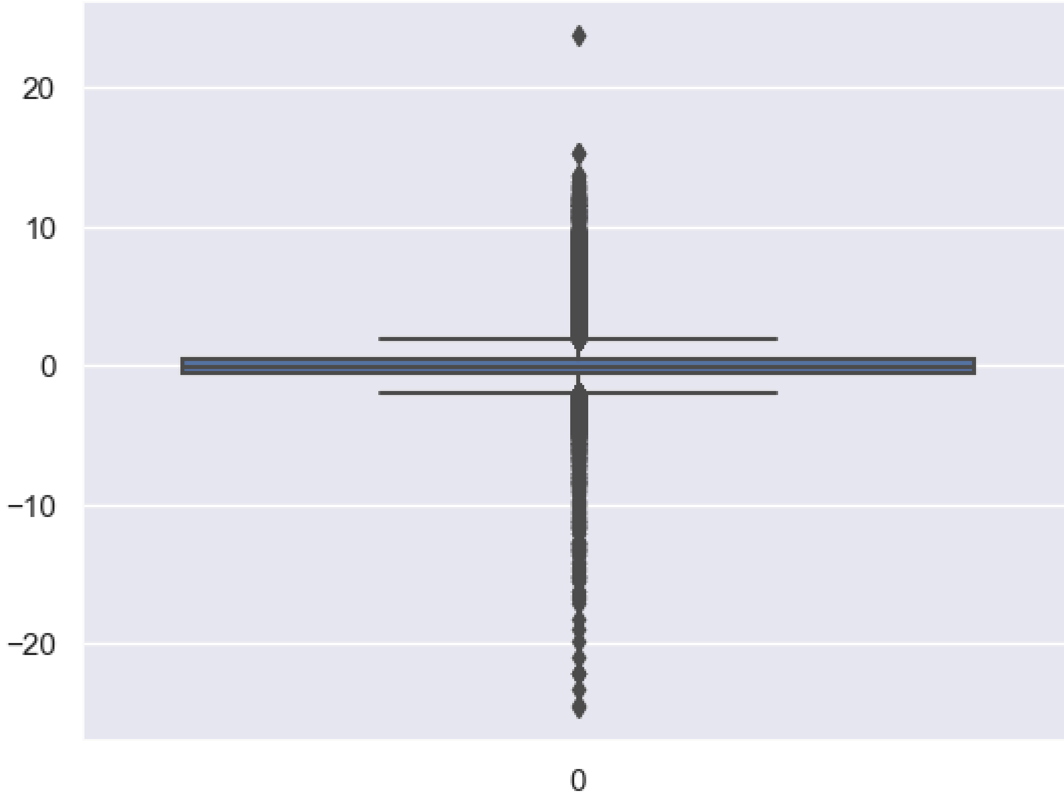

1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0

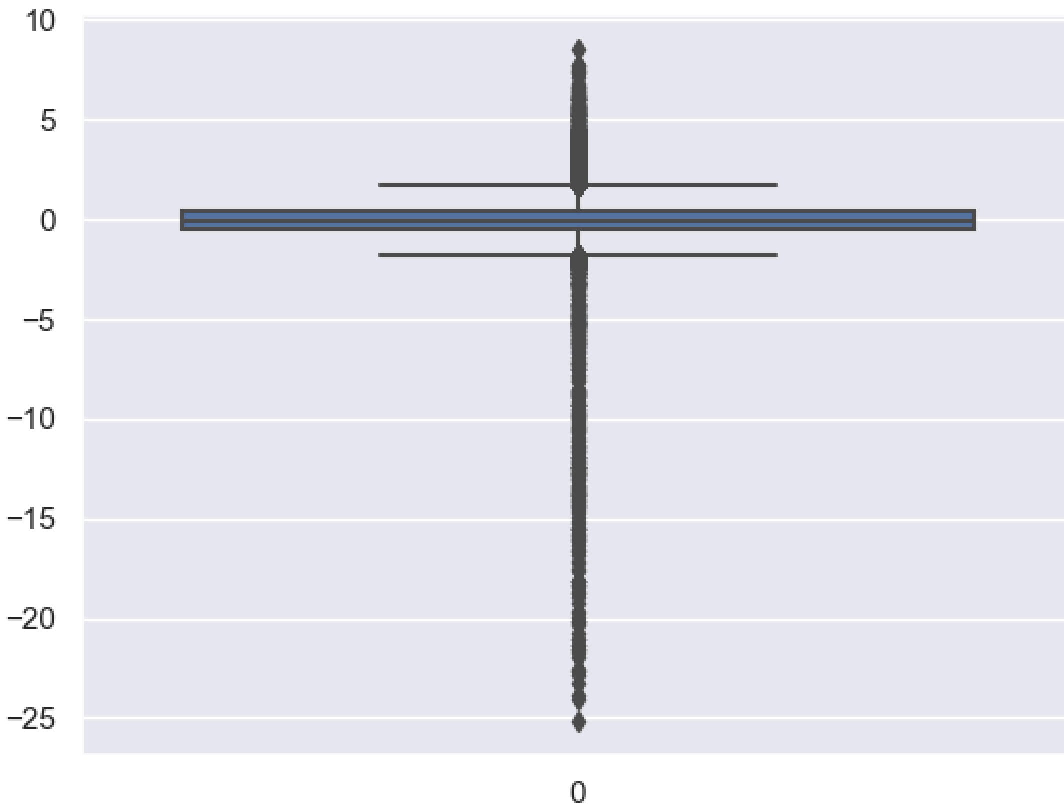

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0

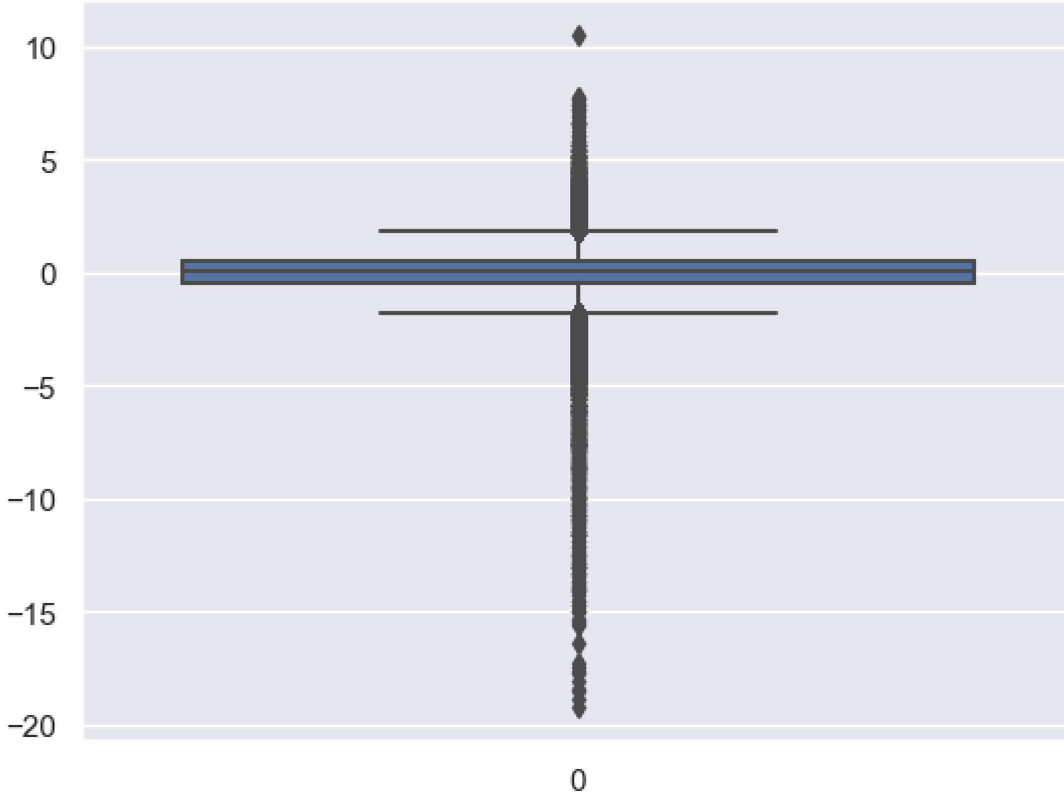

12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0


6000
5000
4000
3000
2000
1000
0
0


5
0
−5
−10
−15
−20
0


20
10
0
−10
−20
0


10
5
0
−5
−10
−15
−20
−25
0



In [70]:
```
# Outlier treament


def check_outlier(col):
    sorted(col)
    Q1,Q3 = col.quantile([.25, .75])
    IQR = Q3 -Q1
    lower_range = Q1 - 1.5 * IQR
    upper_range = Q3 + 1.5 * IQR
    return lower_range, upper_range
```

In [71]:
```
check_outlier(x['lambda_wt'])
```

Out[71]:
```
(-1.8099999999999998, 1.8699999999999999)
```

Capping the outliers treament

In [74]:
```
"""
def treat_outlier(x):
    # taking 5,25, 75,95
    q5 = np.percentile(x,5)
    q25 = np.percentile(x,25)
```

```
    q75 = np.percentile(x,75)
    q95 = np.percentile(x,95)
    # calculating IQR
    IQR = q75 - q25
    # calculating minimum and max threshold value
    lower_bound = q25 - 1.5 * IQR
    upper_bound = q75 + 1.5 * IQR
    print(q5, q25, q75, q95, min, max)
    # apply capping method
    return x.apply(lambda y: q95 if y > upper_bound else y).apply(lambda y
:q5 if y < lower_bound else y )

"""
```

Out[74]:

```
'\ndef treat_outlier(x):\n    # taking 5,25, 75,95\n    q5 = np.percentile(x,5)\n    q25 = np.percentile(x,25)\n    q75 = np.percentile(x,75)\n    q95 = np.percentile(x,95)\n    # calculating IQR \n    IQR = q75 - q25\n    # calculating minimum and max threshold value\n    lower_bound = q25 - 1.5 * IQR\n    upper_bound = q75 + 1.5 * IQR\n    print(q5, q25, q75, q95, min, max)\n    # apply capping method\n    return x.apply(lambda y: q95 if y > upper_bound else y).apply(lambda y :q5 if y < lower_bound else y )\n\n    \n'
```

In [83]:

```
def treat_outlier(x):
    # taking 5,25, 75,95
    q5 = np.percentile(x,5)
    q25 = np.percentile(x,25)
    q75 = np.percentile(x,75)
    q95 = np.percentile(x,95)
    # calculating IQR
    IQR = q75 - q25
    # calculating minimum and max threshold value
    lower_bound = q25 - 1.5 * IQR
    upper_bound = q75 + 1.5 * IQR
    print(q5, q25, q75, q95, min, max)
    # apply capping method
    return x.apply(lambda y: q95 if y > upper_bound else y).apply(lambda y
:q5 if y < lower_bound else y )
```

In [81]:

```
#x is a Series-like object that contains numerical data. Each element in x
represents a data point.
#x.apply(lambda y: q95 if y > upper_bound else y):
```

```
#This applies a lambda function to each element y in x.
#If y is greater than upper_bound, it replaces y with q95.
#Otherwise, it keeps y as is.
```

In [82]:
```
# q5 = np.percentile(x, 5) uses the NumPy library to calculate the
# 5th percentile of the data in the array x. This can be helpful for
understanding the distribution of data,
# especially for identifying the lower end of the data range.
#Lambda to the Rescue: Instead of writing a full function, you use a
lambda function. It's like saying, "For each number, if it's bigger than a
certain value, replace it with another value; otherwise, leave it as it
is."
```

In [75]:
```
x
```

Out[75]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	D
0	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.46
1	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.47
2	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.66
3	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.09
4	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.75
...	...	...	...	...	...	...	...	...	...	
227840	0.476667	1.013333	0.536667	0.576667	1.406667	1.846667	0.600000	1.103333	0.356667	0.53
227841	1.363333	0.730000	0.060000	0.776667	0.883333	0.466667	0.733333	0.590000	0.806667	0.43
227842	1.060000	0.756667	0.906667	0.896667	0.503333	0.396667	0.683333	0.620000	0.630000	0.87
227843	0.433333	1.013333	1.163333	0.940000	0.930000	0.900000	0.813333	0.720000	1.020000	0.41
227844	1.006667	0.553333	0.946667	1.206667	0.406667	0.750000	0.520000	0.756667	1.053333	0.27

227845 rows × 29 columns

In [84]:
```
x.columns
```

Out[84]:
```
Index(['Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7', 'Per8', 'Per9',
       'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7', 'Dem8', 'Dem9',
       'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6', 'Normalised_FNT',
       'geo_score', 'instance_scores', 'qsets_normalized_tat', 'lambda_wt'],
      dtype='object')
```

In [85]:
```
for i in x:
```

```
    x[i] = treat_outlier(x[i])
```

```
-0.3 0.36 1.1033333333333333 1.36 <built-in function min> <built-in function max>
0.01 0.47 0.9333333333333332 1.27 <built-in function min> <built-in function max>
-0.13 0.3699999999999999 1.01 1.3533333333333335 <built-in function min> <built-in fu
nction max>
-0.06333333333333333 0.38333333333333333 0.9133333333333334 1.5233333333333334 <built-i
n function min> <built-in function max>
0.1 0.4366666666666667 0.87 1.3666666666666665 <built-in function min> <built-in func
tion max>
0.1966666666666666 0.41 0.7999999999999999 1.72 <built-in function min> <built-in fun
ction max>
0.19 0.4833333333333333 0.8566666666666666 1.1366666666666667 <built-in function min>
<built-in function max>
0.3833333333333333 0.5966666666666667 0.7766666666666667 1.0166666666666666 <built-in
function min> <built-in function max>
0.08 0.4533333333333333 0.8666666666666667 1.26 <built-in function min> <built-in fun
ction max>
0.1433333333333333 0.4133333333333333 0.9133333333333334 1.2033333333333334 <built-in
function min> <built-in function max>
0.12 0.45 0.8866666666666667 1.2033333333333334 <built-in function min> <built-in fun
ction max>
0.1333333333333333 0.4733333333333333 0.8833333333333333 1.1233333333333333 <built-in
function min> <built-in function max>
0.17 0.51 0.84 1.11 <built-in function min> <built-in function max>
0.2133333333333333 0.5 0.8333333333333334 1.13 <built-in function min> <built-in func
tion max>
0.2133333333333333 0.5133333333333333 0.82 1.0966666666666667 <built-in function min>
<built-in function max>
0.48 0.5966666666666667 0.71 0.9466666666666668 <built-in function min> <built-in fun
ction max>
0.4966666666666666 0.59 0.73 0.8466666666666667 <built-in function min> <built-in fun
ction max>
0.3066666666666666 0.4866666666666666 0.8433333333333334 1.0433333333333332 <built-in
function min> <built-in function max>
0.51 0.6133333333333334 0.7166666666666667 0.8300000000000001 <built-in function min>
<built-in function max>
0.2833333333333334 0.5466666666666667 0.8133333333333334 0.9566666666666668 <built-in
function min> <built-in function max>
0.3899999999999999 0.5599999999999999 0.7833333333333333 0.92 <built-in function min>
<built-in function max>
0.4333333333333333 0.5566666666666666 0.7466666666666667 0.9733333333333332 <built-in
function min> <built-in function max>
0.5266666666666667 0.6433333333333333 0.6966666666666667 0.7966666666666667 <built-in
function min> <built-in function max>
0.5599999999999999 0.65 0.6933333333333334 0.7533333333333333 <built-in function min>
<built-in function max>
-249.77 -248.6175 -230.75 -158.91750000000016 <built-in function min> <built-in funct
ion max>
-1.9599999999999997 -0.39999999999999913 0.6299999999999997 1.2500000000000002 <built
-in function min> <built-in function max>
-1.34 -0.5399999999999999 0.45 1.55 <built-in function min> <built-in function max>
-0.9800000000000001 -0.4800000000000001 0.4000000000000002 1.2740000000000005 <built-
```

```
in function min> <built-in function max>
-1.44 -0.43 0.49 1.39 <built-in function min> <built-in function max>
```

In [86]:
```
# for every value it is check for the outlier is available to be modified or not
```

In [87]:
```
x.describe()
```

Out[87]:

	Per1	Per2	Per3	Per4	Per5	Per6	
count	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	22
mean	0.702647	0.684707	0.680757	0.654235	0.665889	0.664244	
std	0.504588	0.343378	0.443981	0.417920	0.339695	0.392628	
min	-0.753333	-0.223333	-0.590000	-0.410000	-0.210000	-0.173333	
25%	0.360000	0.470000	0.370000	0.383333	0.436667	0.410000	
50%	0.670000	0.690000	0.726667	0.660000	0.650000	0.576667	
75%	1.103333	0.933333	1.010000	0.913333	0.870000	0.800000	
max	1.483333	1.626667	1.963333	1.706667	1.516667	1.720000	

8 rows × 29 columns

In [88]:
```
# Feature Scaling - please check
# Imbalance check
y.value_counts()
```

Out[88]:
```
0.0    227451
1.0       394
Name: Target, dtype: int64
```

In [89]:
```
394/(394+227451)*100
```

Out[89]:
```
0.17292457591783889
```

In [ ]:
```
# percentage of 1 value
```

In [90]:
```
227451/(394+227451)*100
```

Out[90]:
```
99.82707542408215
```

In [ ]:
```
# percentage of 1 value
```

In [93]:
```
# split the dataset into train and test
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=101, stratify=y)
# stratify=y - handling imbalance dataset
```

In [91]:
```
# Without stratify=y, there's a chance that the training or testing set
could end up with an unequal representation of classes, especially for
imbalanced datasets.
# With stratify=y, each set will contain approximately the same proportion
of samples from each class as the original dataset, minimizing bias in
subsequent analyses or model training.
# x and y are the features and target variable, respectively, of the
dataset.
```

In [92]:
```
# train_test_split is a standard function provided by scikit-learn, a
popular machine learning library in Python. It's one of the most commonly
used functions for splitting datasets into training and testing sets,
making it essential in the machine learning workflow.
# train_test_split is a standard function in scikit-learn for splitting
datasets.
# It's essential for preparing data for machine learning model training
and evaluation.
# The function ensures reproducibility and can handle stratified splitting
to preserve class distributions.
```

In [94]:
```
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(182276, 29) (45569, 29) (182276,) (45569,)
```

In [95]:
```
y_train.value_counts()
```

Out[95]:
```
0.0    181961
1.0       315
Name: Target, dtype: int64
```

In [98]:
```
y_test.value_counts()
```

Out[98]:
```
0.0    45490
1.0       79
Name: Target, dtype: int64
```

In [99]:
```
315/394*100
```

Out[99]:
```
79.94923857868021
```

Model Building

In [100…

```
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score

from sklearn.linear_model import LogisticRegression
```

LogisticRegression

In [101…

```
logit = LogisticRegression()
lr = logit.fit(x_train, y_train)
y_pred_train = logit.predict(x_train)
y_pred_test = logit.predict(x_test)
# Confusion Matrix
print(confusion_matrix(y_train, y_pred_train))
print()
print(confusion_matrix(y_test, y_pred_test))
print()
# classification_report
print(classification_report(y_train, y_pred_train))
print()
print(classification_report(y_test, y_pred_test))
print()
# accuracy_score
print("Train Accuracy", accuracy_score(y_train, y_pred_train))
print()
print("Test Accuracy", accuracy_score(y_test, y_pred_test))
```

```
[[181920     41]
 [    63    252]]

[[45480    10]
 [   25    54]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       0.86      0.80      0.83       315

    accuracy                           1.00    182276
   macro avg       0.93      0.90      0.91    182276
weighted avg       1.00      1.00      1.00    182276


              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     45490
         1.0       0.84      0.68      0.76        79

    accuracy                           1.00     45569
   macro avg       0.92      0.84      0.88     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 0.9994294366784436

Test Accuracy 0.9992319339902126
```

In [102…

```
y.value_counts()
```

Out[102]:

```
0.0    227451
1.0       394
Name: Target, dtype: int64
```

In [103…

```
outlier_fraction = 394 / (394+227451)
outlier_fraction
```

Out[103]:

```
0.0017292457591783894
```

DecisionTree Classifier

In [104…

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(criterion='entropy')
dt = dtree.fit(x_train, y_train)
y_pred_train_dt = dtree.predict(x_train)
y_pred_test_dt = dtree.predict(x_test)
```

```python
# Confusion Matrix
print(confusion_matrix(y_train, y_pred_train_dt))
print()
print(confusion_matrix(y_test, y_pred_test_dt))
print()
# classification_report
print(classification_report(y_train, y_pred_train_dt))
print()
print(classification_report(y_test, y_pred_test_dt))
print()
# accuracy_score
print("Train Accuracy", accuracy_score(y_train, y_pred_train_dt))
print()
print("Test Accuracy", accuracy_score(y_test, y_pred_test_dt))
```

```
[[181961      0]
 [     0    315]]

[[45471    19]
 [   28    51]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       1.00      1.00      1.00       315

    accuracy                           1.00    182276
   macro avg       1.00      1.00      1.00    182276
weighted avg       1.00      1.00      1.00    182276


              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     45490
         1.0       0.73      0.65      0.68        79

    accuracy                           1.00     45569
   macro avg       0.86      0.82      0.84     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 1.0

Test Accuracy 0.9989685970725712
```

RandomForestClassifier

In [105…

```
from sklearn.ensemble import RandomForestClassifier
rforest = RandomForestClassifier()
rf = rforest.fit(x_train, y_train)
y_pred_train_rf = rforest.predict(x_train)
y_pred_test_rf = rforest.predict(x_test)
# Confusion Matrix
print(confusion_matrix(y_train, y_pred_train_rf))
print()
print(confusion_matrix(y_test, y_pred_test_rf))
print()
# classification_report
print(classification_report(y_train, y_pred_train_rf))
print()
print(classification_report(y_test, y_pred_test_rf))
print()
# accuracy_score
print("Train Accuracy", accuracy_score(y_train, y_pred_train_rf))
print()
print("Test Accuracy", accuracy_score(y_test, y_pred_test_rf))
```

```
[[181961      0]
 [     0    315]]

[[45486     4]
 [   26    53]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       1.00      1.00      1.00       315

    accuracy                           1.00    182276
   macro avg       1.00      1.00      1.00    182276
weighted avg       1.00      1.00      1.00    182276


              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     45490
         1.0       0.93      0.67      0.78        79

    accuracy                           1.00     45569
   macro avg       0.96      0.84      0.89     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 1.0

Test Accuracy 0.9993416577058966
```

XGBoost Classifier

In [106…

```
from xgboost import XGBClassifier
xgboost = XGBClassifier()
xgb = xgboost.fit(x_train, y_train)
y_pred_train_xgb = xgboost.predict(x_train)
y_pred_test_xgb = xgboost.predict(x_test)
# Confusion Matrix
print(confusion_matrix(y_train, y_pred_train_xgb))
print()
print(confusion_matrix(y_test, y_pred_test_xgb))
print()
# classification_report
print(classification_report(y_train, y_pred_train_xgb))
print()
print(classification_report(y_test, y_pred_test_xgb))
```

```
print()
# accuracy_score
print("Train Accuracy", accuracy_score(y_train, y_pred_train_xgb))
print()
print("Test Accuracy", accuracy_score(y_test, y_pred_test_xgb))
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[106], line 1
----> 1 from xgboost import XGBClassifier
      2 xgboost = XGBClassifier()
      3 xgb = xgboost.fit(x_train, y_train)

ModuleNotFoundError: No module named 'xgboost'
```

Support Vector Maching

In [107…

```
from sklearn.svm import SVC
SVClass = SVC()
svm = SVClass.fit(x_train, y_train)
y_pred_train_svm = SVClass.predict(x_train)
y_pred_test_svm = SVClass.predict(x_test)
# Confusion Matrix
print(confusion_matrix(y_train, y_pred_train_svm))
print()
print(confusion_matrix(y_test, y_pred_test_svm))
print()
# classification_report
print(classification_report(y_train, y_pred_train_svm))
print()
print(classification_report(y_test, y_pred_test_svm))
print()
# accuracy_score
print("Train Accuracy", accuracy_score(y_train, y_pred_train_svm))
print()
print("Test Accuracy", accuracy_score(y_test, y_pred_test_svm))
```

```
[[181961      0]
 [   315      0]]

[[45490     0]
 [   79     0]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       0.00      0.00      0.00       315

    accuracy                           1.00    182276
   macro avg       0.50      0.50      0.50    182276
weighted avg       1.00      1.00      1.00    182276


              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     45490
         1.0       0.00      0.00      0.00        79

    accuracy                           1.00     45569
   macro avg       0.50      0.50      0.50     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 0.9982718514779785

Test Accuracy 0.9982663652921943
```

K Nearest Neighbors

Unsupported Cell Type. Double-Click to inspect/edit the content

Naive Bayes Theorem

In [109…

```
from sklearn.naive_bayes import BernoulliNB
bernb = BernoulliNB()
bnb = bernb.fit(x_train, y_train)
y_pred_train_bnb = bernb.predict(x_train)
y_pred_test_bnb = bernb.predict(x_test)
# Confusion Matrix
print(confusion_matrix(y_train, y_pred_train_bnb))
print()
print(confusion_matrix(y_test, y_pred_test_bnb))
print()
```

```
# classification_report
print(classification_report(y_train, y_pred_train_bnb))
print()
print(classification_report(y_test, y_pred_test_bnb))
print()
# accuracy_score
print("Train Accuracy", accuracy_score(y_train, y_pred_train_bnb))
print()
print("Test Accuracy", accuracy_score(y_test, y_pred_test_bnb))
```

```
[[181934     27]
 [   258     57]]

[[45480    10]
 [   66    13]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       0.68      0.18      0.29       315

    accuracy                           1.00    182276
   macro avg       0.84      0.59      0.64    182276
weighted avg       1.00      1.00      1.00    182276


              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     45490
         1.0       0.57      0.16      0.25        79

    accuracy                           1.00     45569
   macro avg       0.78      0.58      0.63     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 0.9984364370515043

Test Accuracy 0.9983321995216046
```

Voting Classifier

In [110…

```
from sklearn.ensemble import VotingClassifier
```

In [111…

```
voting = VotingClassifier(estimators=[('logit', lr ),('dtree', dt),
('rforest', rf),('xgboost', xgb),
```

```
                                    ("svm", svm),("bnb",bnb)])
voting_evc = voting.fit(x_train, y_train)
y_pred_train_voting = voting.predict(x_train)
y_pred_test_voting = voting.predict(x_test)
# Confusion Matrix
print(confusion_matrix(y_train, y_pred_train_voting))
print()
print(confusion_matrix(y_test, y_pred_test_voting))
print()
# classification_report
print(classification_report(y_train, y_pred_train_voting))
print()
print(classification_report(y_test, y_pred_test_voting))
print()
# accuracy_score
print("Train Accuracy", accuracy_score(y_train, y_pred_train_voting))
print()
print("Test Accuracy", accuracy_score(y_test, y_pred_test_voting))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[111], line 1
----> 1 voting = VotingClassifier(estimators=[('logit', lr ),('dtree', dt),('rfores
t', rf),('xgboost', xgb),
      2                                     ("svm", svm),("bnb",bnb)])
      3 voting_evc = voting.fit(x_train, y_train)
      4 y_pred_train_voting = voting.predict(x_train)

NameError: name 'xgb' is not defined
```

In [112…

```
accuracy_logit = accuracy_score(y_test, y_pred_test)
accuracy_dtree = accuracy_score(y_test, y_pred_test_dt)
accuracy_rf = accuracy_score(y_test, y_pred_test_rf)
accuracy_xgb = accuracy_score(y_test, y_pred_test_xgb)
accuracy_svm = accuracy_score(y_test, y_pred_test_svm)
accuracy_bnb = accuracy_score(y_test, y_pred_test_bnb)
accuracy_voting = accuracy_score(y_test, y_pred_test_voting)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[112], line 4
      2 accuracy_dtree = accuracy_score(y_test, y_pred_test_dt)
      3 accuracy_rf = accuracy_score(y_test, y_pred_test_rf)
----> 4 accuracy_xgb = accuracy_score(y_test, y_pred_test_xgb)
      5 accuracy_svm = accuracy_score(y_test, y_pred_test_svm)
      6 accuracy_bnb = accuracy_score(y_test, y_pred_test_bnb)

NameError: name 'y_pred_test_xgb' is not defined
```

In [113…

```
point1 = ["Logistic", 'Dtree','RForest','XGBoost','SVM','BNB','Voting']
point2 =
[accuracy_logit,accuracy_dtree,accuracy_rf,accuracy_xgb,accuracy_svm,accurac

final_output = pd.DataFrame({"Method Used": point1, "Accuracy":point2})
print(final_output)

# visualization

chart = sns.barplot(x="Method Used", y="Accuracy", data=final_output)
chart.set_xticklabels(chart.get_xticklabels(), rotation=90)
print(chart)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[113], line 2
      1 point1 = ["Logistic", 'Dtree','RForest','XGBoost','SVM','BNB','Voting']
----> 2 point2 = [accuracy_logit,accuracy_dtree,accuracy_rf,accuracy_xgb,accuracy_sv
m,accuracy_bnb,accuracy_voting]
      4 final_output = pd.DataFrame({"Method Used": point1, "Accuracy":point2})
      5 print(final_output)

NameError: name 'accuracy_xgb' is not defined
```

In [114…

```
# Tomorrow Target
# Anomaly Detection model
## Stacking method
## Isolation Forest
## Local Outlier Factor
## OneSVM


## MLP - MultiLayerPerceptron
```