Problem Statement: 'PredCatch Analytics' Australian banking client's profitability and reputation are being hit by fraudulent ATM transactions. They want PredCatch to help them in reducing and if possible completely eliminating such fraudulent transactions. PredCatch believes it can do the same by building a predictive model to catch such fraudulent transactions in real time and decline them. Your job as PredCatch's Data Scientist is to build this fraud detection & prevention predictive model in the first step. If successful, in the 2nd step you will have to present your solutions and explain how it works to the client. The data has been made available to you.

The challenging part of the problem is that the data contains very few fraud instances in comparison to the overall population. To give more edge to the solution they have also collected data regarding location [geo_scores] of the transactions, their own proprietary index [Lambda_wts], on network turn around times [Qset_tats] and vulnerability qualification score [instance_scores]. As of now you don't need to understand what they mean.

Training data contains masked variables pertaining to each transaction id . Your prediction target here is 'Target'.

In [143…

```
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 [144…

```
geo = pd.read_csv('Geo_scores.csv')
instance = pd.read_csv('instance_scores.csv')
lamdbawts = 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 (1).csv')
```

In [145…

```
print (geo.shape)
print()
print(instance.shape)
print()
print(lamdbawts.shape)
print()
print(qset.shape)
print()
print(test_data.shape)
print()
print(train_data.shape)
```

```
(1424035, 2)

(1424035, 2)

(1400, 2)

(1424035, 2)

(56962, 27)

(227845, 28)
```

In [146…

```
print (geo.columns)
print()
print(instance.columns)
print()
print(lamdbawts.columns)
print()
print(qset.columns)
print()
print(test_data.columns)
print()
print(train_data.columns)
```

```
Index(['id', 'geo_score'], dtype='object')

Index(['id', 'instance_scores'], dtype='object')

Index(['Group', 'lambda_wt'], dtype='object')

Index(['id', 'qsets_normalized_tat'], dtype='object')

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'],
      dtype='object')

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'],
      dtype='object')
```

In [147…

```
print ("geo ID", geo['id'].nunique())
print()
print("instance id", instance['id'].nunique())
print()
print("lamdbawts group", lamdbawts['Group'].nunique())
print()
print("qset id", qset['id'].nunique())
print()
print("test_data id ", test_data['id'].nunique())
print()
print("train_data id", train_data['id'].nunique())
print()
print("test_data group ", test_data['Group'].nunique())
print()
print("train_data group", train_data['Group'].nunique())
```

```
geo ID 284807

instance id 284807

lamdbawts group 1400

qset id 284807

test_data id  56962

train_data id 227845

test_data group  915

train_data group 1301
```

In [148…

```
train_data['data'] = 'train' # adding the data column with train
test_data['data'] = 'test'   # adding the data column with test
```

In [149…

```
train_data.columns
```

Out[149]:

```
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 [150…

```
test_data.columns
```

Out[150]:

```
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 [151…

```
train_data.tail()
```

Out[151]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	Dem9	Cred1	Cred2
227840	97346	Grp232	0.476667	1.013333	0.536667	0.576667	1.406667	1.846667	0.600000	1.103333	...	0.630000	0.633333	0.996667
227841	147361	Grp199	1.363333	0.730000	0.060000	0.776667	0.883333	0.466667	0.733333	0.590000	...	0.356667	0.766667	0.730000
227842	50989	Grp36	1.060000	0.756667	0.906667	0.896667	0.503333	0.396667	0.683333	0.620000	...	0.510000	0.740000	0.873333
227843	149780	Grp445	0.433333	1.013333	1.163333	0.940000	0.930000	0.900000	0.813333	0.720000	...	0.606667	0.540000	0.643333
227844	22175	Grp143	1.006667	0.553333	0.946667	1.206667	0.406667	0.750000	0.520000	0.756667	...	0.646667	0.636667	0.683333

5 rows × 29 columns

In [152…

```
test_data.tail()
```

Out[152]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	Dem8	Dem9	Cred1
56957	18333	Grp102	0.553333	1.043333	1.096667	0.686667	0.673333	0.340000	0.900000	0.643333	...	0.576667	0.433333	0.660000
56958	244207	Grp504	1.353333	0.616667	0.276667	0.783333	0.690000	0.650000	0.473333	0.670000	...	0.713333	0.870000	0.683333
56959	103277	Grp78	1.083333	0.433333	0.806667	0.490000	0.243333	0.316667	0.533333	0.606667	...	0.433333	0.063333	0.753333
56960	273294	Grp134	0.566667	1.153333	0.370000	0.616667	0.793333	0.226667	0.910000	0.696667	...	0.776667	1.026667	0.626667
56961	223337	Grp18	1.426667	0.110000	-0.006667	-0.200000	0.983333	1.870000	0.033333	0.963333	...	0.616667	0.670000	0.770000

5 rows × 28 columns

In [153…

```
# adding train and test data

all_data = pd.concat([train_data, test_data], axis = 0) # concatinting in row wise
```

In [154…

```
all_data.head()
```

Out[154]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	Dem9	Cred1	Cred2	Cre
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	...	0.726667	0.606667	1.010000	0.9333
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	...	0.743333	0.680000	0.690000	0.5600
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	...	0.820000	0.600000	0.383333	0.7633
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	...	0.900000	0.680000	0.846667	0.4233
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	...	0.486667	0.693333	0.526667	0.5200

5 rows × 29 columns

In [155…

```
all_data.tail()
```

Out[155]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	Dem9	Cred1	Cred2
56957	18333	Grp102	0.553333	1.043333	1.096667	0.686667	0.673333	0.340000	0.900000	0.643333	...	0.433333	0.660000	0.776667
56958	244207	Grp504	1.353333	0.616667	0.276667	0.783333	0.690000	0.650000	0.473333	0.670000	...	0.870000	0.683333	0.690000
56959	103277	Grp78	1.083333	0.433333	0.806667	0.490000	0.243333	0.316667	0.533333	0.606667	...	0.063333	0.753333	0.780000
56960	273294	Grp134	0.566667	1.153333	0.370000	0.616667	0.793333	0.226667	0.910000	0.696667	...	1.026667	0.626667	0.646667
56961	223337	Grp18	1.426667	0.110000	-0.006667	-0.200000	0.983333	1.870000	0.033333	0.963333	...	0.670000	0.770000	0.893333

5 rows × 29 columns

In [156…

```
all_data.columns
```

Out[156]:

```
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 [157…

```
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 [158…

```
# we can merge all the data because same number of unique id's are present in geo ID 284807 instance id 284807
# qset id 284807 and same with group
```

In [159…

```
# checking missing values

print (geo.isnull().sum())
print()
print(instance.isnull().sum())
print()
print(lamdbawts.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 [160…

```
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 [161…

```
# Handling  Missing vales

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 [162…

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

```
id           0
geo_score    0
dtype: int64

id                 0
instance_scores    0
dtype: int64

Group        0
lambda_wt    0
dtype: int64

id                      0
qsets_normalized_tat    0
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 [163…
```
geo.shape
```

Out[163]:
```
(1424035, 2)
```

In [164…
```
geo
```

Out[164]:

	id	geo_score
0	26674	4.48
1	204314	4.48
2	176521	5.17
3	48812	-2.41
4	126870	6.55
...	...	...
1424030	107880	1.03
1424031	282410	8.62
1424032	209634	-1.72
1424033	211652	-10.00
1424034	73455	5.86

1424035 rows × 2 columns

In [165…
```
geo = geo.groupby('id').mean() # taking the unique id of all tranascations and their mean value
```

In [166…
```
geo.shape
```

Out[166]:
```
(284807, 1)
```

In [167…
```
qset.shape
```

Out[167]:
```
(1424035, 2)
```

In [168…
```
qset = qset.groupby('id').mean()
```

```
In [168]: qset = qset.groupby('id').mean()
```

```
In [169]: qset.shape
```

Out[169]: (284807, 1)

```
In [170]: instance.shape
```

Out[170]: (1424035, 2)

```
In [171]: instance = instance.groupby('id').mean()
```

```
In [172]: instance.shape
```

Out[172]: (284807, 1)

```
In [173]: lamdbawts.shape
```

Out[173]: (1400, 2)

```
In [174]: # its already 1400 and even in all_data its 1400 so no need to do groupby
```

```
In [175]: lamdbawts.head()
```

Out[175]:

	Group	lambda_wt
0	Grp936	3.41
1	Grp347	-2.88
2	Grp188	0.39
3	Grp1053	-2.75
4	Grp56	-0.83

```
In [176]: print (geo.shape)
print()
print(instance.shape)
print()
print(lamdbawts.shape)
print()
print(qset.shape)
print()
print(all_data.shape)
```

```
(284807, 1)

(284807, 1)

(1400, 2)

(284807, 1)

(284807, 29)
```

```
In [177]: all_data = pd.merge(all_data, geo, on = 'id' , how = 'left') # merging all data with geo b id and using left jo
```

```
In [178]: all_data.head()
```

Out[178]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	Cred1	Cred2	Cred3	Cre
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	...	0.606667	1.010000	0.933333	0.6033
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	...	0.680000	0.690000	0.560000	0.6700
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	...	0.600000	0.383333	0.763333	0.6700
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	...	0.680000	0.846667	0.423333	0.5200
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	...	0.693333	0.526667	0.520000	0.7166

5 rows × 30 columns

```
In [179]: all_data = pd.merge(all_data, instance, on = 'id' , how = 'left')
```

```
In [180]: all_data.head(2)
```

Out[180]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	Cred2	Cred3	Cred4	Cred5	
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.34	1.010000	...	1.01	0.933333	0.603333	0.686667	0.6
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.81	0.783333	...	0.69	0.560000	0.670000	0.553333	0.6

2 rows × 31 columns

In [181…
```
all_data = pd.merge(all_data, qset, on = 'id' , how = 'left')
```

In [182…
```
all_data.head(2)
```

Out[182]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	Cred3	Cred4	Cred5	Cred6
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.34	1.010000	...	0.933333	0.603333	0.686667	0.673333
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.81	0.783333	...	0.560000	0.670000	0.553333	0.653333

2 rows × 32 columns

In [183…
```
all_data.shape
```

Out[183]:
```
(284807, 32)
```

In [184…
```
all_data['Group'].nunique()
```

Out[184]:
```
1400
```

In [185…
```
lamdbawts.shape
```

Out[185]:
```
(1400, 2)
```

In [186…
```
lamdbawts['Group'].nunique()
```

Out[186]:
```
1400
```

In [187…
```
all_data = pd.merge(all_data, lamdbawts, on = 'Group' , how = 'left')
```

In [188…
```
all_data.head(2)
```

Out[188]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	...	Cred4	Cred5	Cred6	Normalise
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.34	1.010000	...	0.603333	0.686667	0.673333	-
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.81	0.783333	...	0.670000	0.553333	0.653333	-

2 rows × 33 columns

In [189…
```
all_data['lambda_wt'].count()
```

Out[189]:
```
284807
```

In [190…
```
all_data['lambda_wt'].nunique()
```

Out[190]:
```
1400
```

In [191…
```
# Now need to split train and test data before we merged because for merging all tables
```

In [192…
```
train_data = all_data[all_data['data'] == 'train']
test_data = all_data[all_data['data'] == 'test']
```

In [193…
```
train_data.shape
```

Out[193]:
```
(227845, 33)
```

In [194…
```
test_data.shape
```

Out[194]:
```
(56962, 33)
```

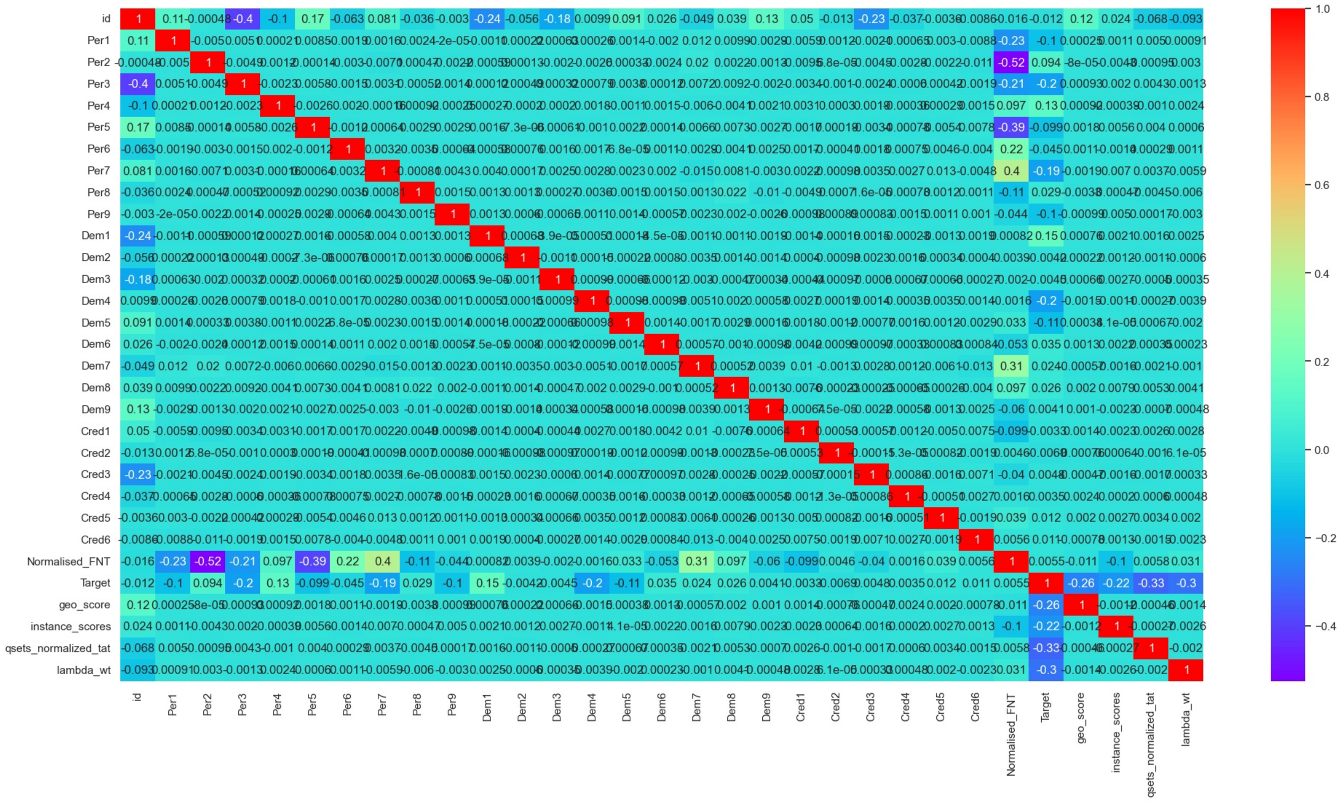
In [195…
```
train_data.columns
```

Out[195]:
```
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 [196…
```
test_data.columns
```

Out[196]:
```
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 [55]:
```python
plt.figure(figsize = (24,12))
sns.heatmap(train_data.corr(), annot = True, cmap = 'rainbow')
plt.show()
```



In [138…
```python
# splitting the data into dependent and independent variables

x= train_data.drop(['id','Group', 'Target', 'data'], axis = 1)

y = train_data['Target']
```

In [57]:
```python
x.head()
```

Out[57]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	...	Cred2	Cred3	Cred4	C
0	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.460000	...	1.010000	0.933333	0.603333	0.68
1	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000	...	0.690000	0.560000	0.670000	0.55
2	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000	...	0.383333	0.763333	0.670000	0.68
3	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667	...	0.846667	0.423333	0.520000	0.84
4	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667	...	0.526667	0.520000	0.716667	0.70

5 rows × 29 columns

In [58]:
```python
x.columns
```

Out[58]:
```
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 [59]:
```python
y.head()
```

Out[59]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: Target, dtype: float64
```

In [60]:
```python
test_data.columns
```

Out[60]:
```
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 [197…
```python
test_data.isnull().sum()/len(test_data)*100
```

```
Out[197]: 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 [198…
```
test_data = test_data.drop(['id','Group','Target','data'], axis=1) # target we need to predict
```

In [199…
```
# Task :
# This data is for prediction whether listed customer will do fraudulent or not
test_data.head()
```

Out[199]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	...	Cred2	Cred3	Cre
227845	-0.300000	1.540000	0.220000	-0.280000	0.570000	0.260000	0.700000	1.076667	0.930000	0.156667	...	0.813333	0.776667	0.7966
227846	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0.550000	0.543333	0.433333	...	0.703333	0.806667	0.6300
227847	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0.686667	0.593333	1.250000	...	0.753333	0.870000	0.5966
227848	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0.663333	1.156667	1.186667	...	0.606667	0.456667	0.3200
227849	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0.686667	1.476667	1.213333	...	0.896667	0.566667	0.5466

5 rows × 29 columns

Actual Data

In [64]:
```
x.head()
```

Out[64]:

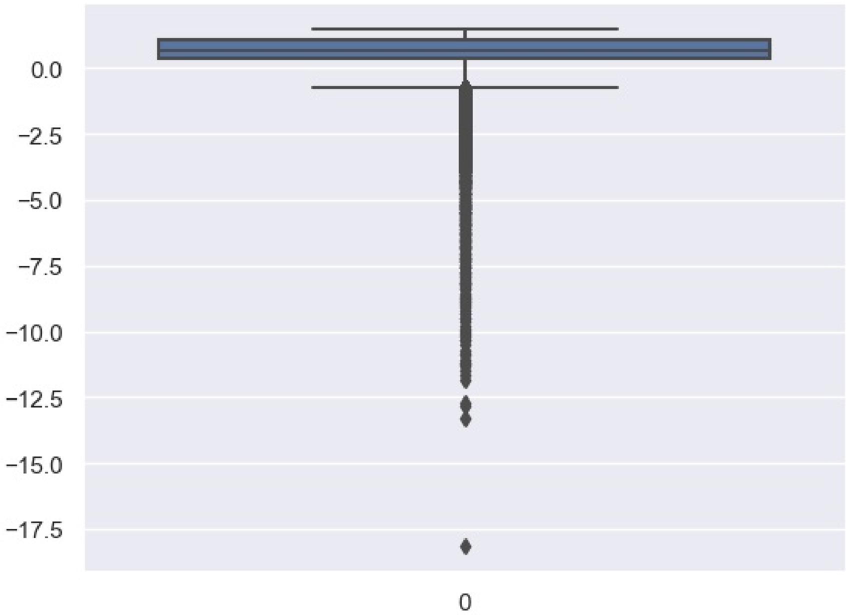
	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	...	Cred2	Cred3	Cred4	C
0	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.460000	...	1.010000	0.933333	0.603333	0.68
1	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000	...	0.690000	0.560000	0.670000	0.55
2	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000	...	0.383333	0.763333	0.670000	0.68
3	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667	...	0.846667	0.423333	0.520000	0.84
4	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667	...	0.526667	0.520000	0.716667	0.70

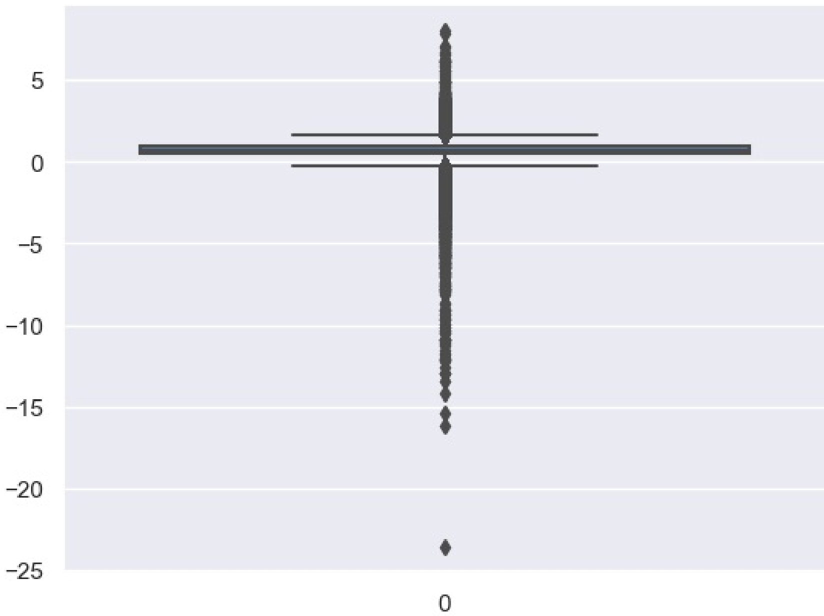
5 rows × 29 columns

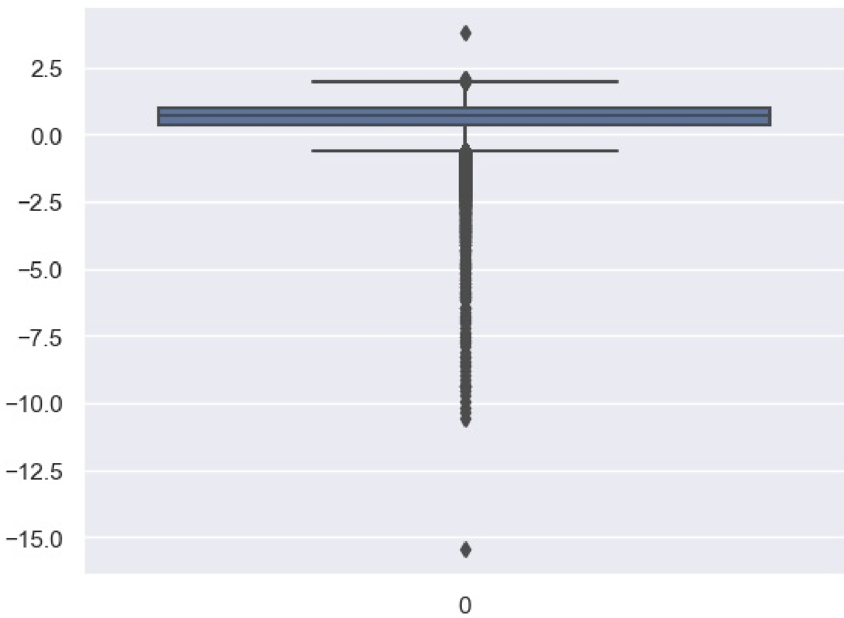
In [65]:
```
x.isnull().any()
```

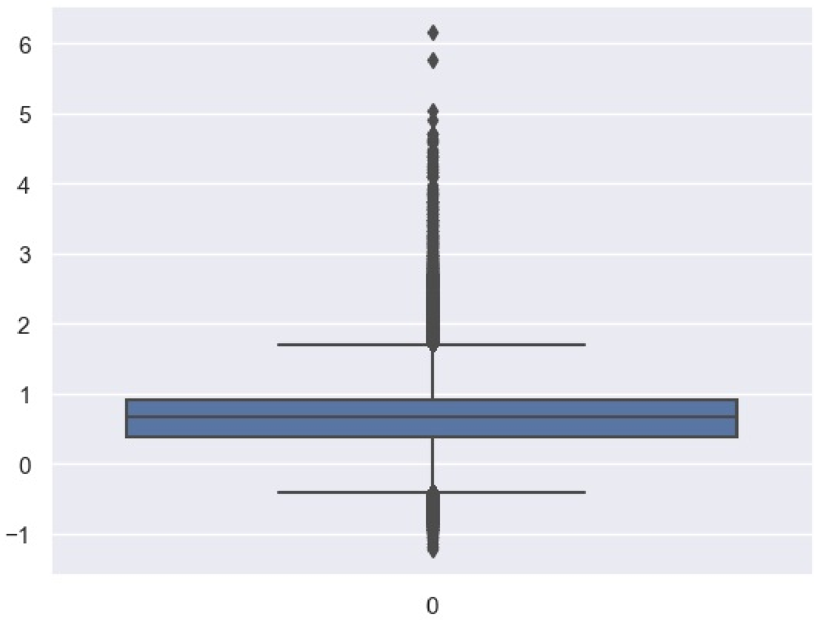
Out[65]:
```
Per1                        False
Per2                        False
Per3                        False
Per4                        False
Per5                        False
Per6                        False
Per7                        False
Per8                        False
Per9                        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
```

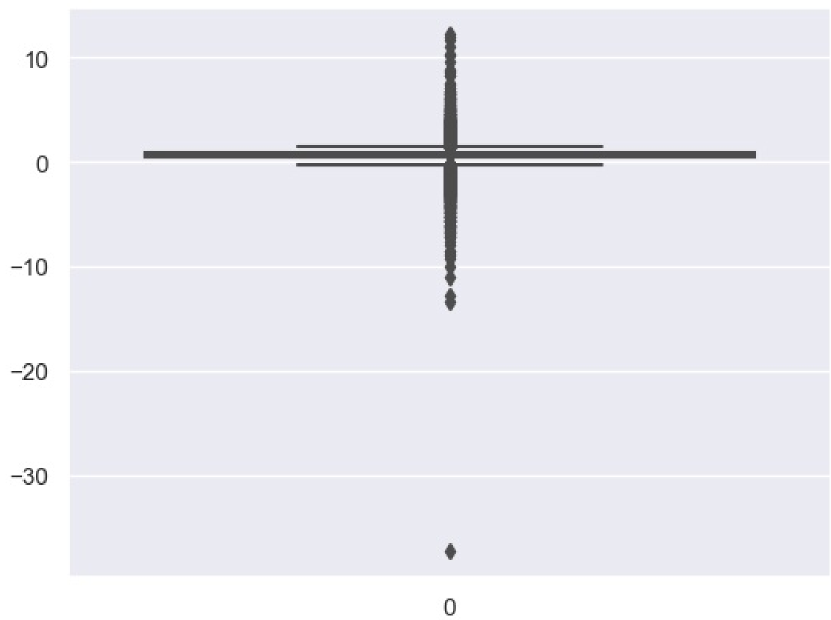
In [66]:
```python
def boxplots(col):
    sns.boxplot(x[col])
    plt.show()

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

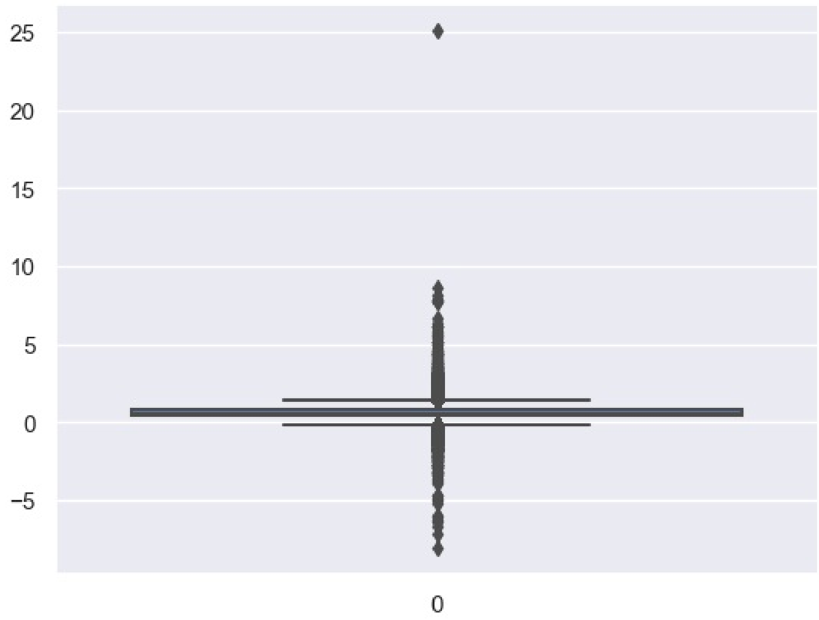


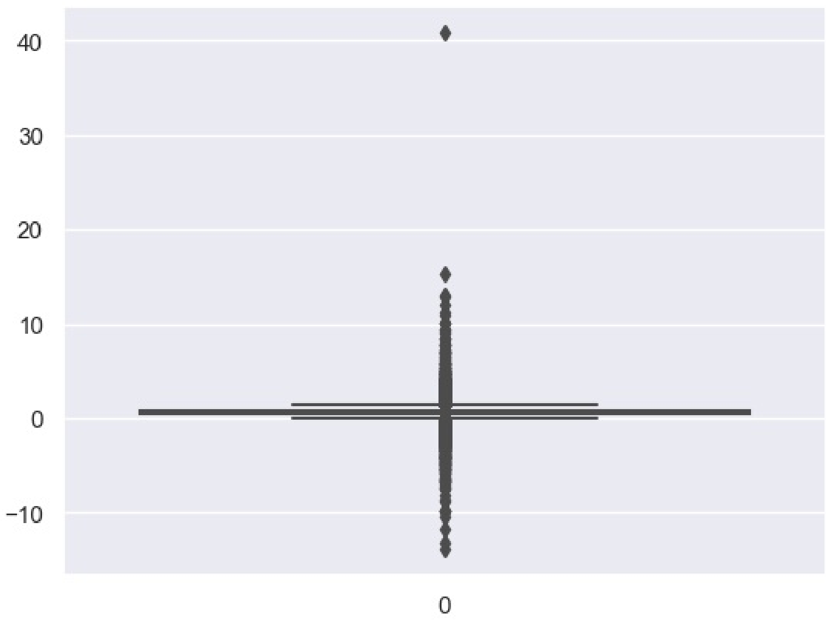

5
0
−5
−10
−15
−20
−25
0

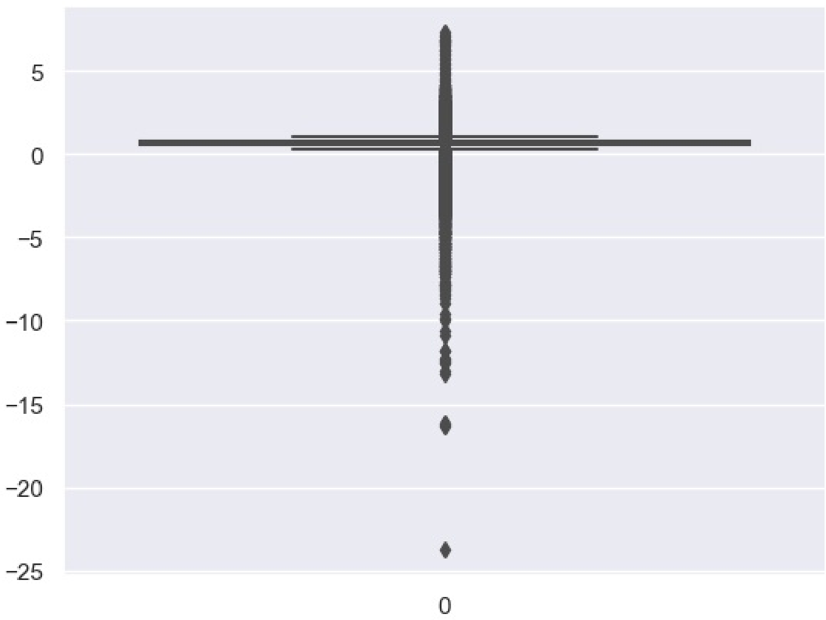

2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
0

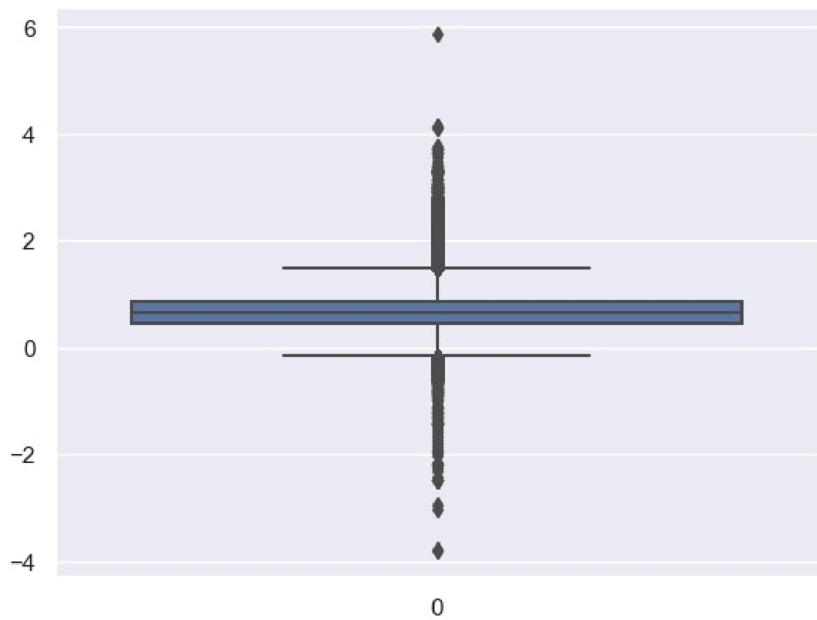

6
5
4
3
2
1
0
−1
0

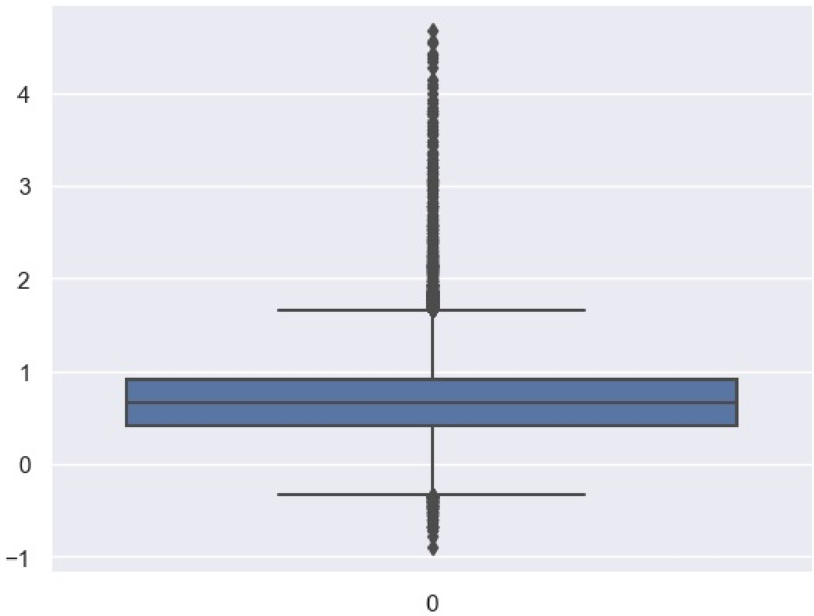

10
0
−10
−20
−30
0

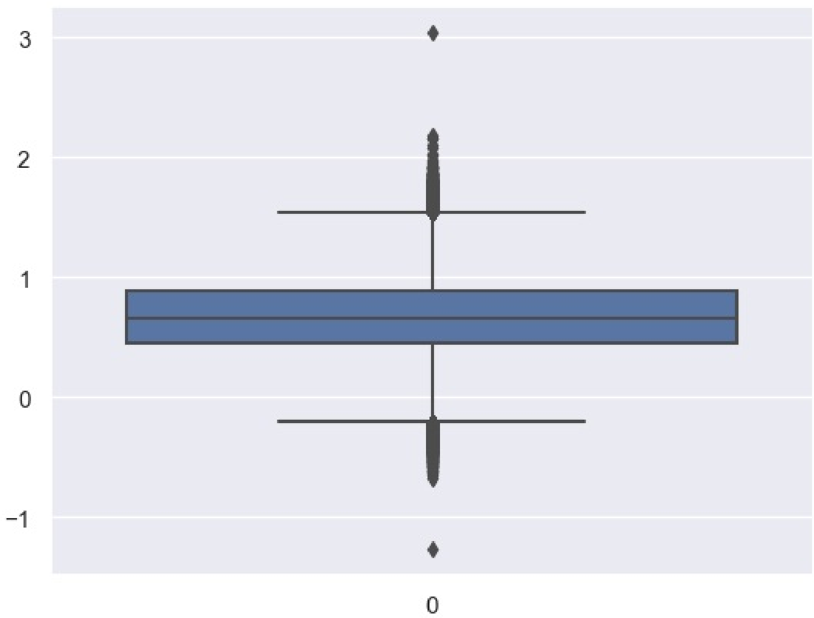

25
20
15
10
5
0
−5
0

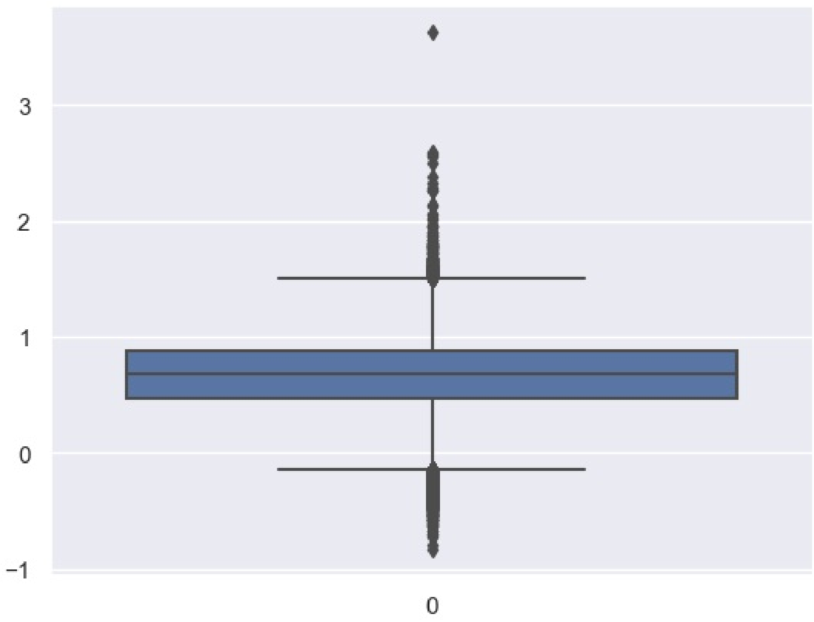

40
30
20
10
0
−10
0

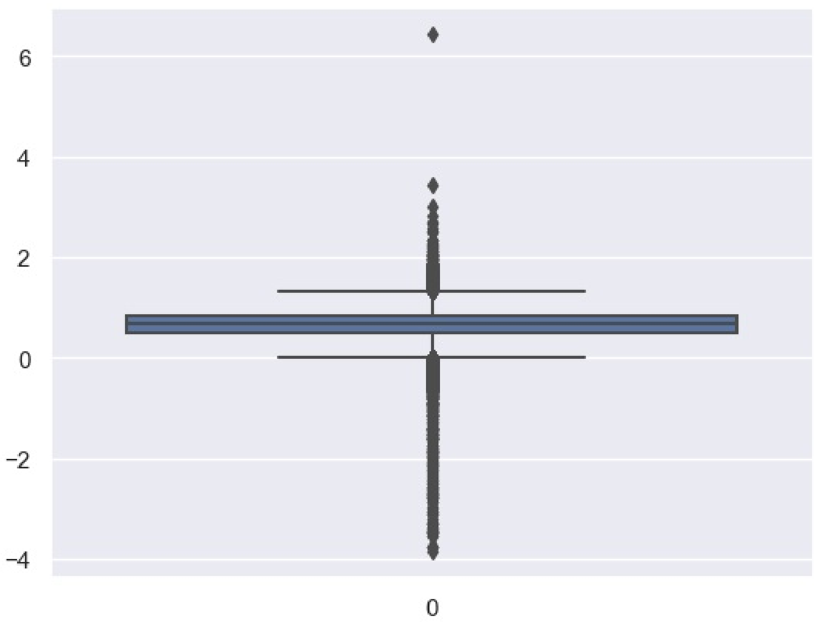

5
0
−5
−10
−15
−20
−25
0

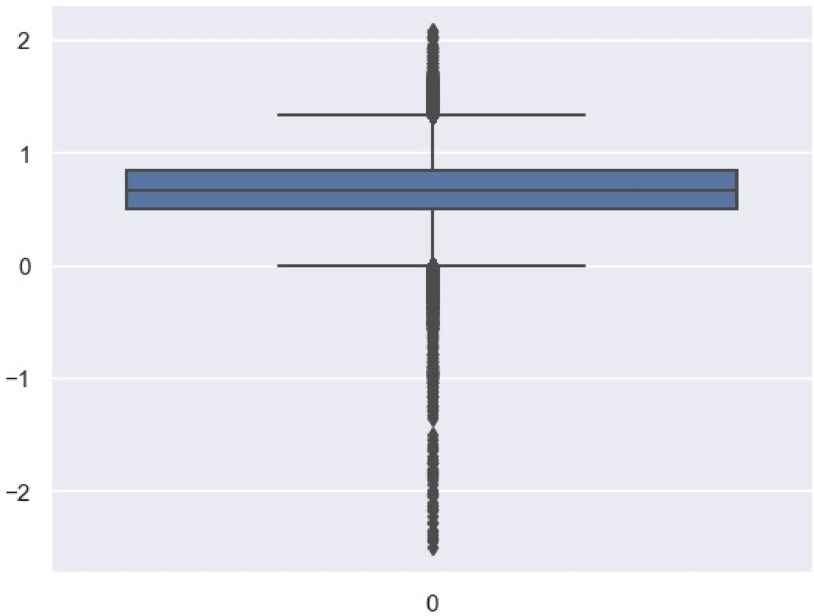

6
4
2
0
−2
−4
0

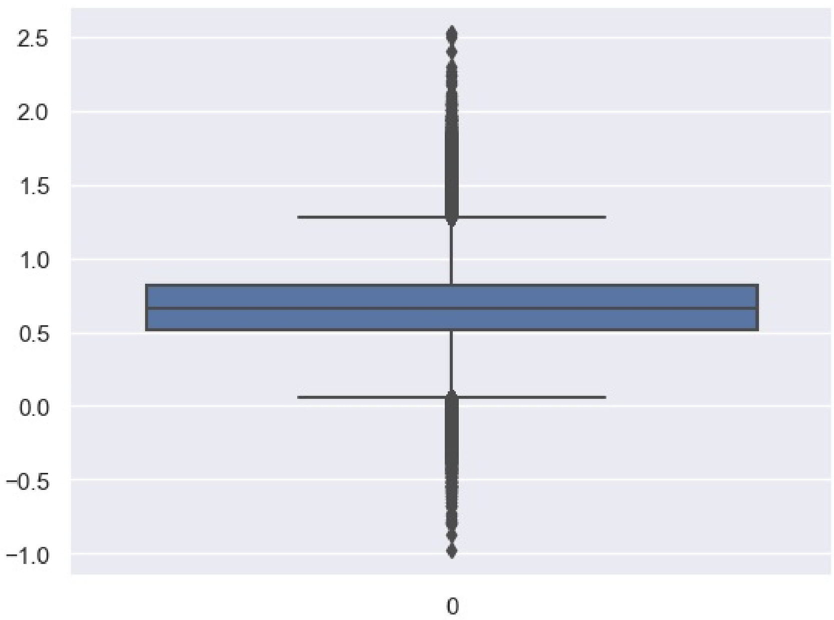

4
3
2
1
0
−1
0

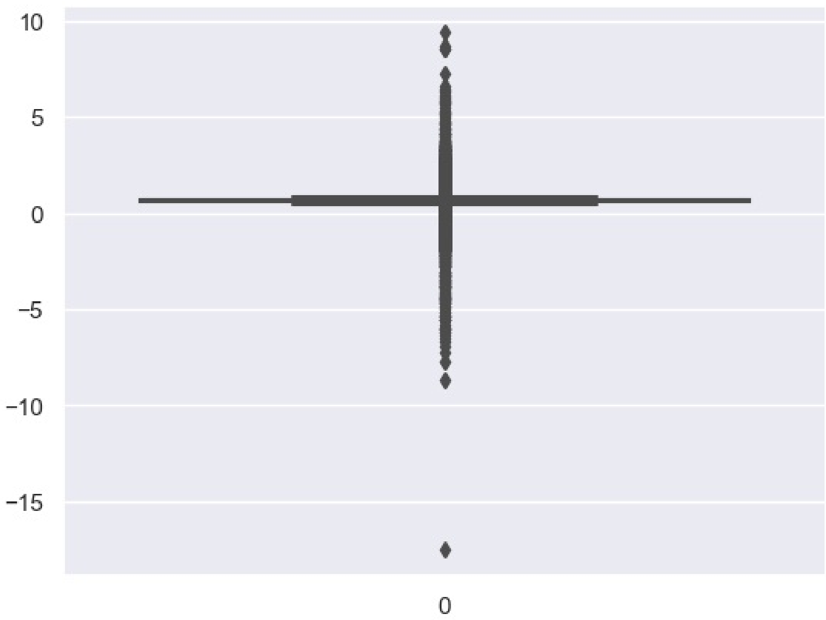

3
2
1
0
−1
0

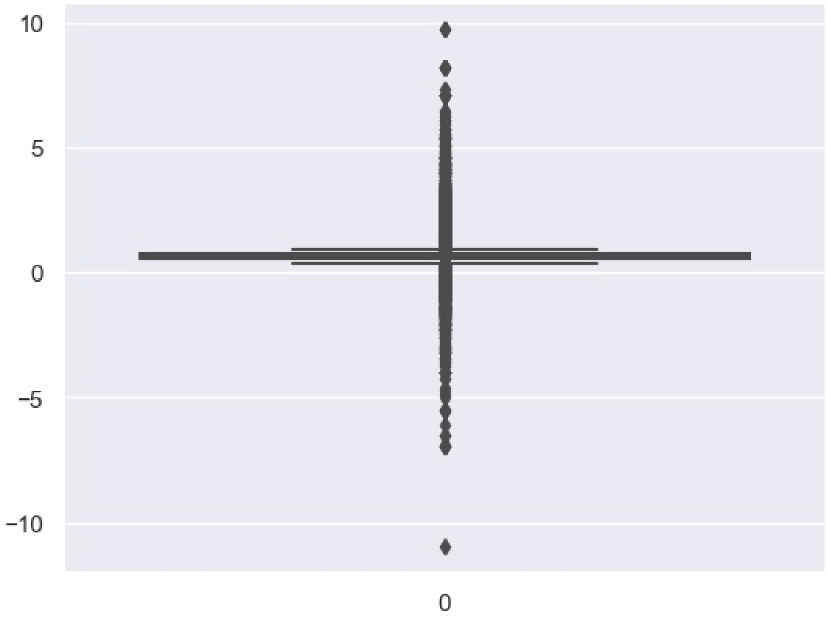

3
2
1
0
−1
0

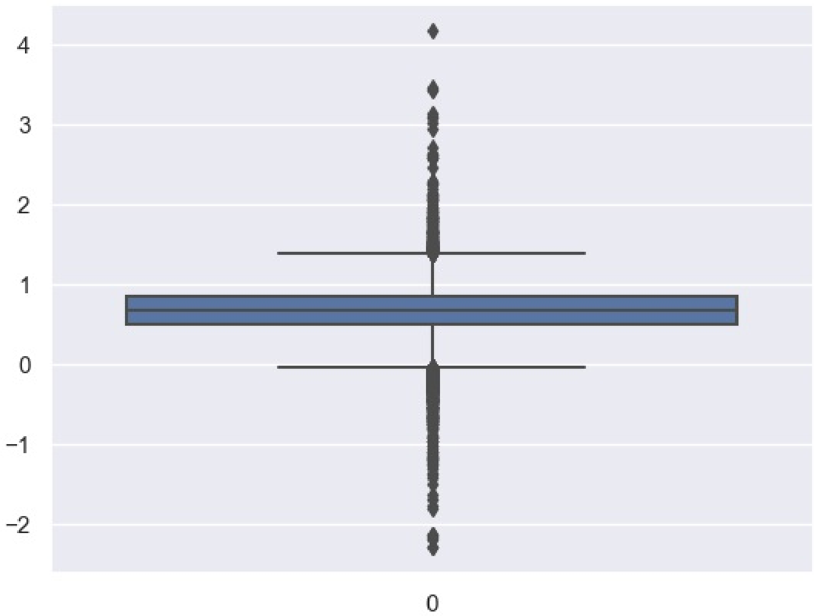

6
4
2
0
−2
−4
0

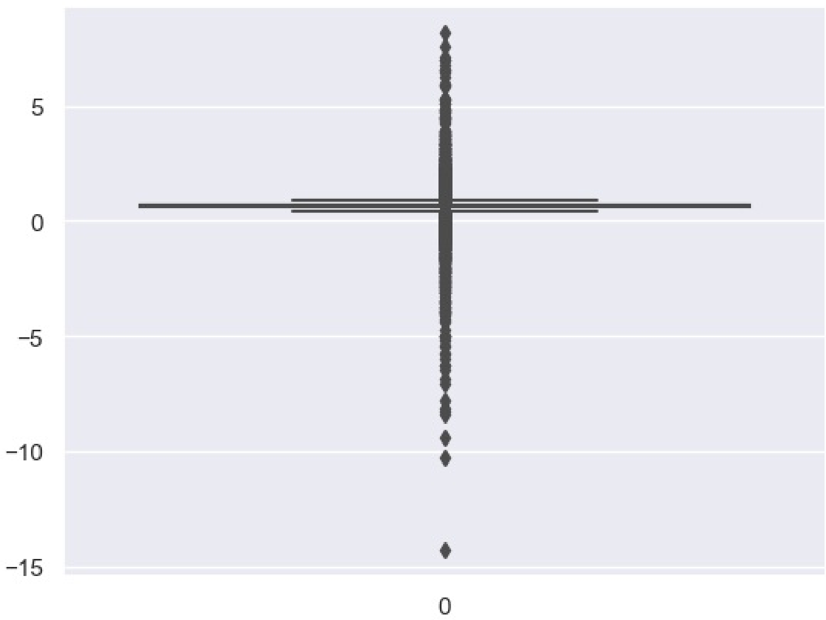

2
1
0
−1
−2
0

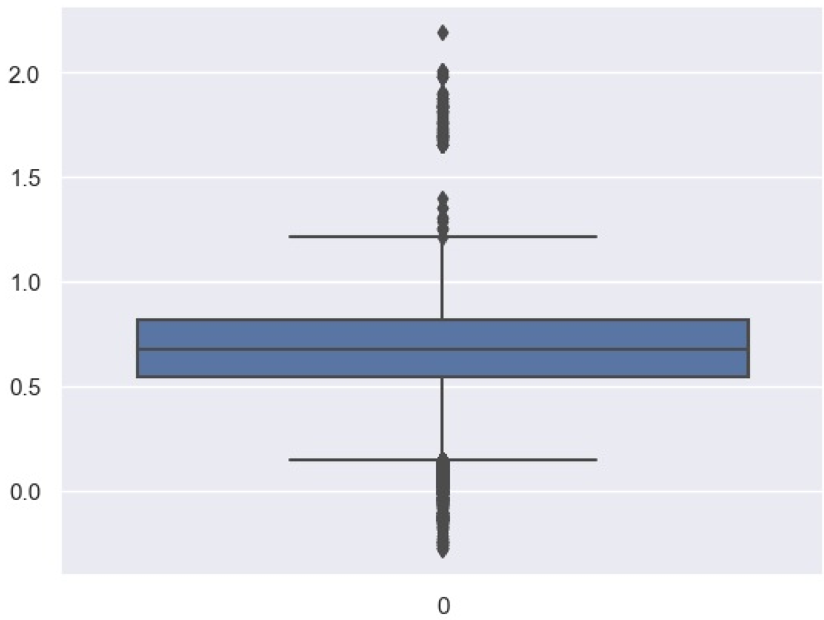

2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0

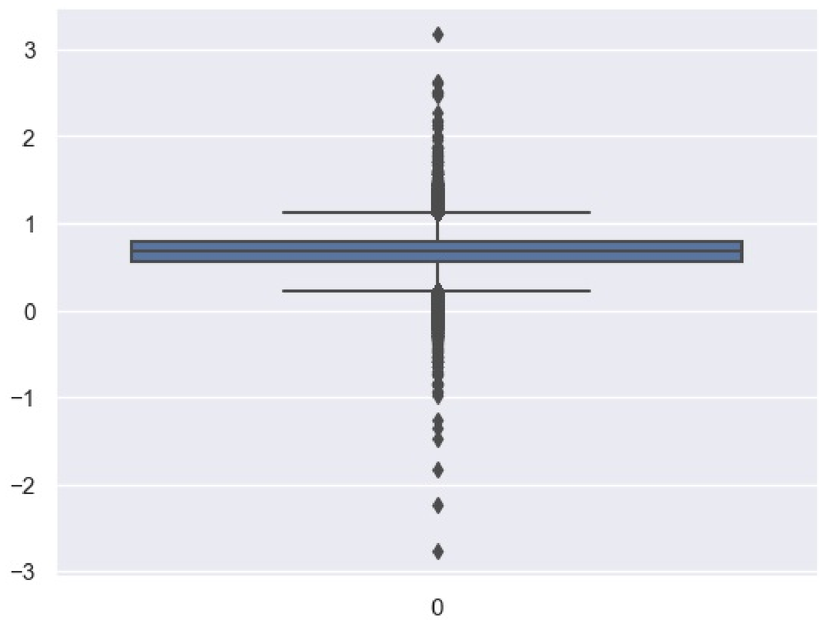

10
5
0
−5
−10
−15
0

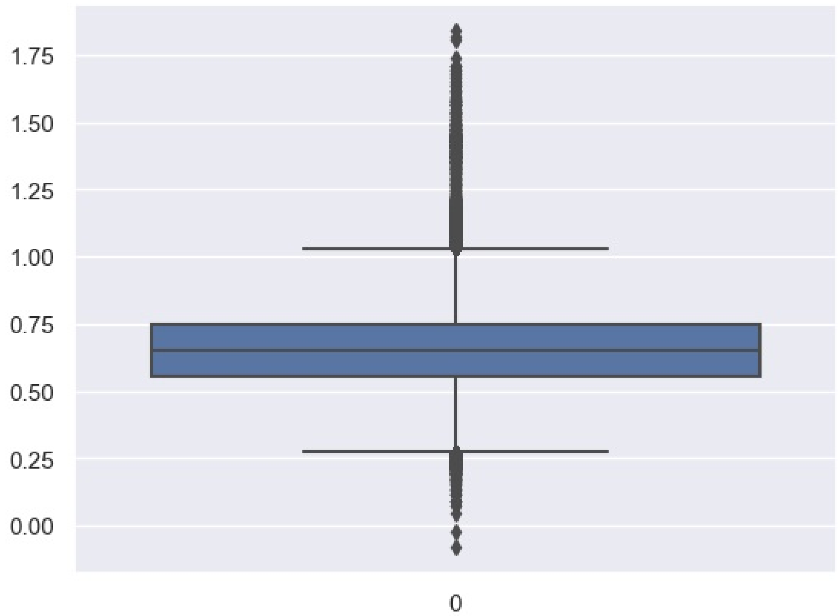

10
5
0
−5
−10
0

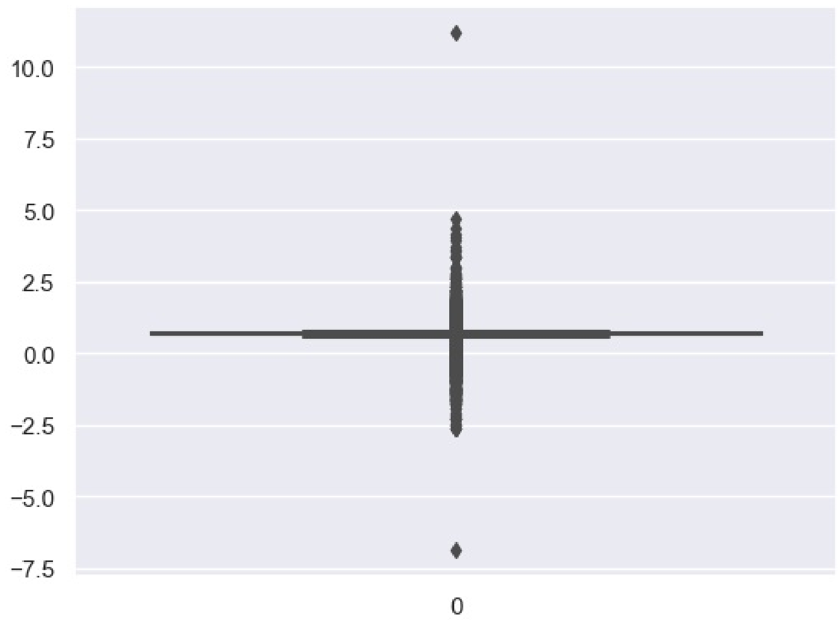

4
3
2
1
0
−1
−2
0

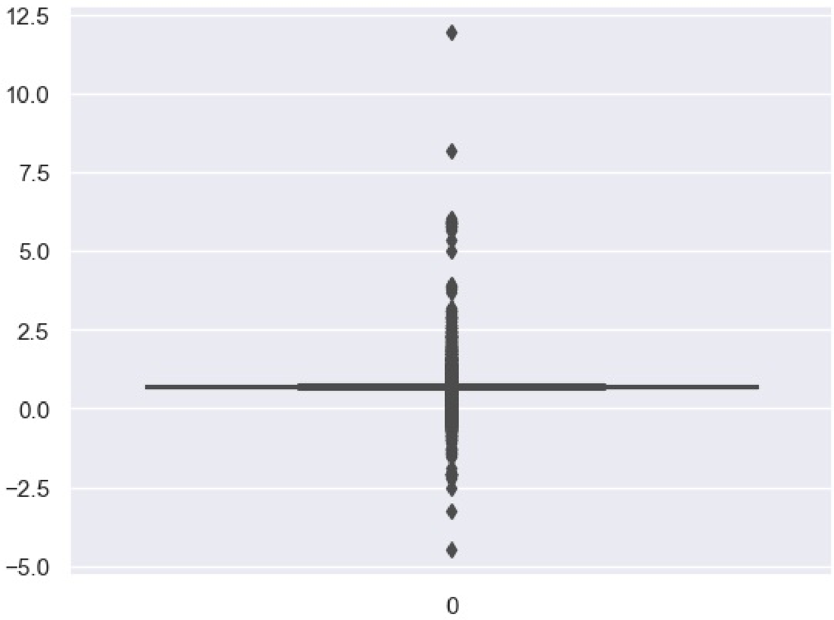

5
0
−5
−10
−15
0


2.0
1.5
1.0
0.5
0.0
0

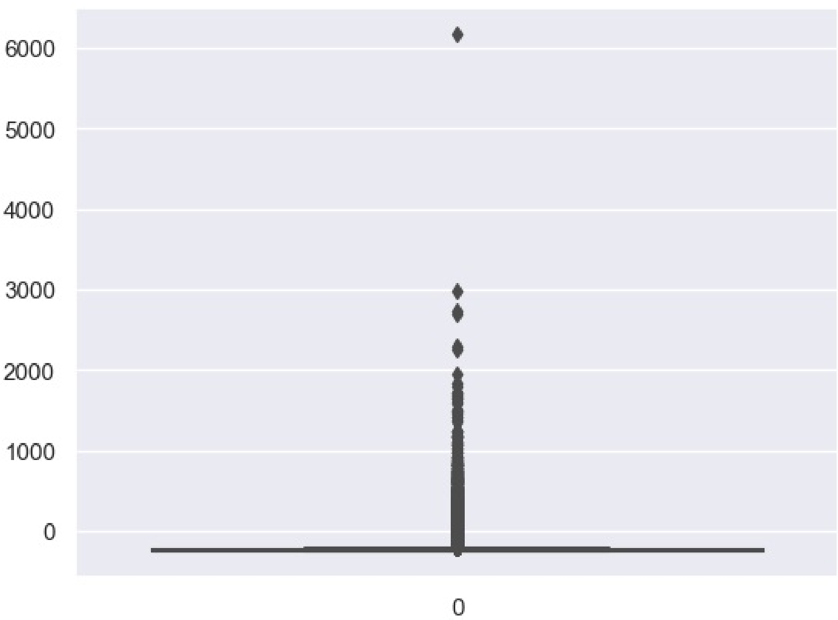

3
2
1
0
−1
−2
−3
0


1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0

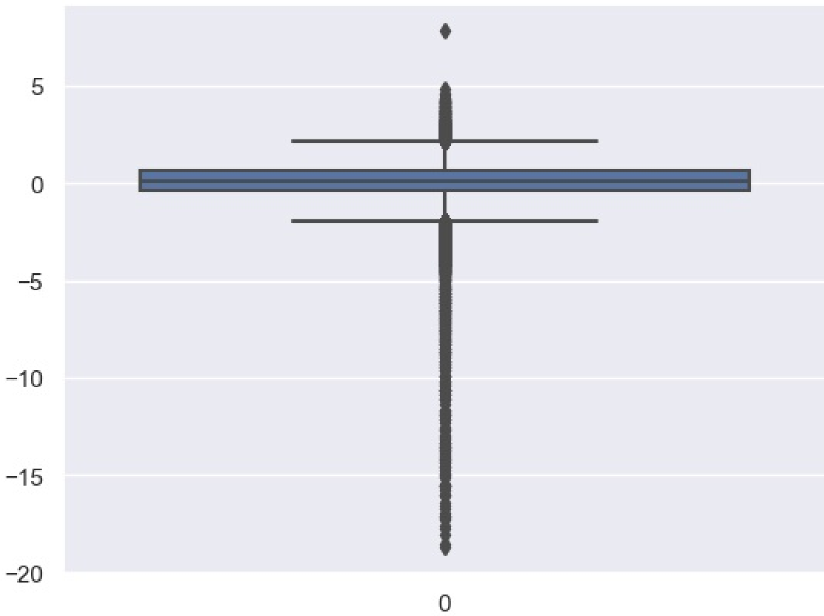

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0

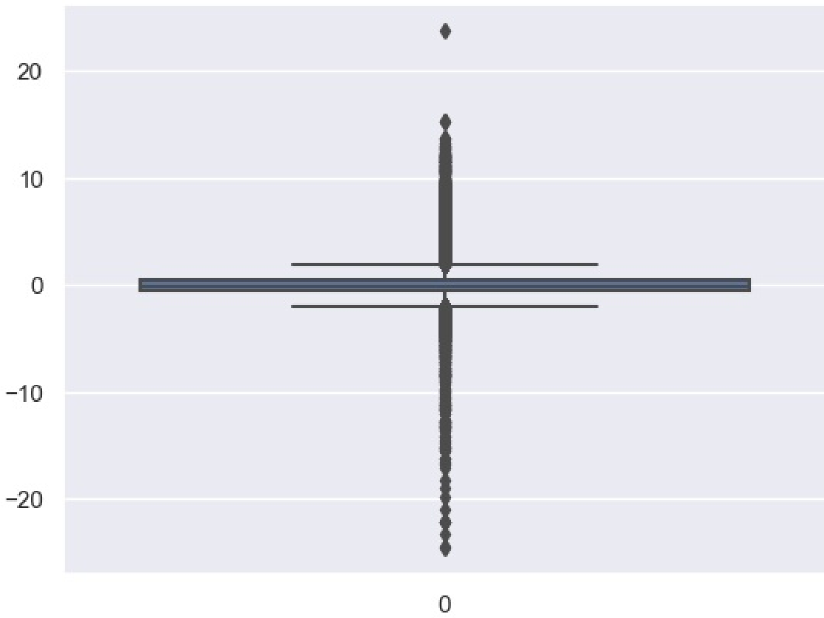


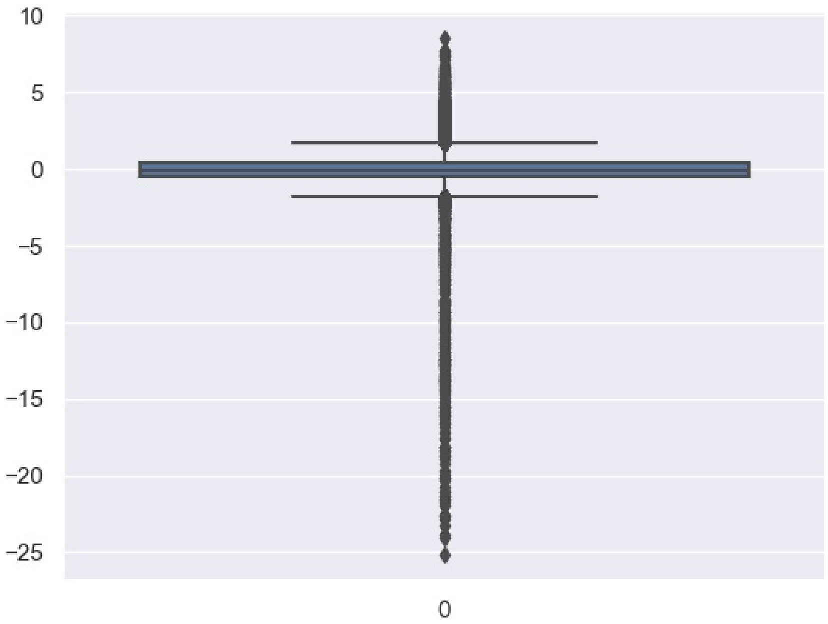
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0

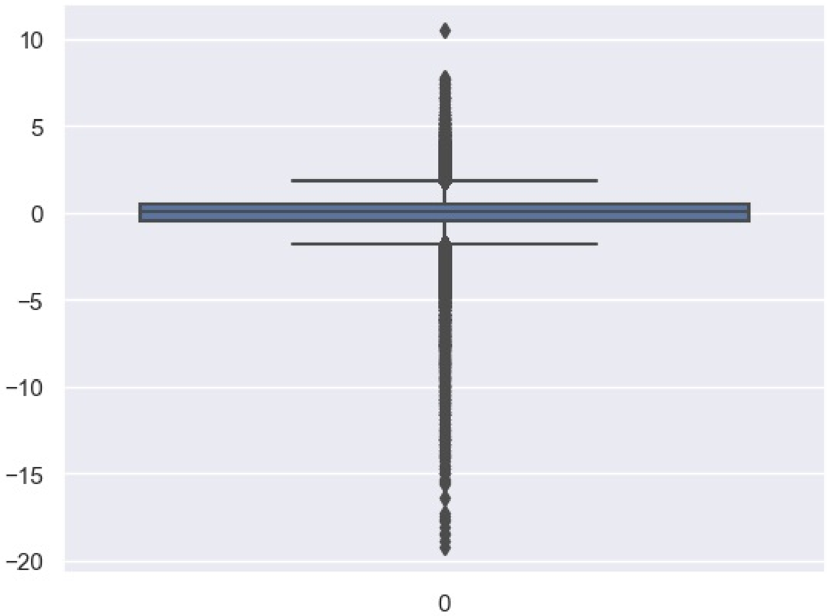


6000
5000
4000
3000
2000
1000
0
0


5
0
−5
−10
−15
−20
0


20
10
0
−10
−20
0





In [67]:
```
x.describe()
```

Out[67]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	F
count	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.00C
mean	0.666006	0.667701	0.666315	0.666687	0.666723	0.667378	0.666934	0.666279	0.666
std	0.654133	0.548305	0.506357	0.471956	0.461393	0.444573	0.415657	0.401546	0.366
min	-18.136667	-23.573333	-15.443333	-1.226667	-37.246667	-8.053333	-13.853333	-23.740000	-3.810
25%	0.360000	0.470000	0.370000	0.383333	0.436667	0.410000	0.483333	0.596667	0.453
50%	0.670000	0.690000	0.726667	0.660000	0.650000	0.576667	0.680000	0.673333	0.650
75%	1.103333	0.933333	1.010000	0.913333	0.870000	0.800000	0.856667	0.776667	0.866
max	1.483333	8.020000	3.793333	6.163333	12.266667	25.100000	40.863333	7.336667	5.863

8 rows × 29 columns

In [68]:
```
# feature scaling
```

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
sc_x = scaler.fit_transform(x)
pd.DataFrame(sc_x)
```

Out[68]:

	0	1	2	3	4	5	6	7	8	9	...	19	20	
0	0.617603	-0.159949	-0.367953	0.211843	1.228044	2.982542	-0.786549	0.855995	0.536498	-0.606800	...	1.699950	1.530275	-
1	-0.294547	0.982970	0.428588	1.617345	0.129918	-0.091573	0.344195	0.291510	-1.300520	-0.577426	...	0.117386	-0.612816	
2	0.709328	-0.956345	0.553665	-1.151282	-1.271637	0.380791	-1.468203	0.225099	-1.200485	-0.019317	...	-1.399237	0.554403	
3	-0.044852	0.770192	0.165269	0.579109	0.166041	-0.721392	0.697051	-0.082047	-0.491141	1.263355	...	0.892183	-1.397341	-
4	-0.162056	0.630367	-0.144132	-0.529754	0.231062	-0.466465	0.448448	0.324715	-0.409294	0.264633	...	-0.690380	-0.842433	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
227840	-0.289451	0.630367	-0.256042	-0.190738	1.603719	2.652637	-0.161031	1.088431	-0.845813	-0.401181	...	1.634010	-0.115313	-
227841	1.066035	0.113622	-1.197409	0.233031	0.469470	-0.451469	0.159747	-0.189964	0.381897	-0.675340	...	0.315207	-0.402334	
227842	0.602316	0.162257	0.474669	0.487293	-0.354124	-0.608924	0.039455	-0.115252	-0.100093	0.597541	...	1.024063	0.190843	
227843	-0.355697	0.630367	0.981559	0.579109	0.570614	0.523250	0.352214	0.133785	0.963923	-0.743880	...	-0.113404	1.377197	-
227844	0.520783	-0.208584	0.553665	1.144135	-0.563635	0.185847	-0.353498	0.225099	1.054864	-1.155118	...	0.084416	1.013637	-

227845 rows × 29 columns

In [69]:

```python
# Imbalance check

y.value_counts()
```

Out[69]:

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

In [70]:

```python
394/(394+227451)*100 # fraud data
```

Out[70]:

```
0.17292457591783889
```

In [71]:

```python
227451/(394+227451)*100
```

Out[71]:

```
99.82707542408215
```

In [200…

```python
# 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)
# used for imbalance data
```

In [73]:

```python
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

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

In [74]:

```python
y_train.value_counts()
```

Out[74]:

```
0.0    181961
1.0       315
Name: Target, dtype: int64
```

In [75]:

```python
y_test.value_counts()
```

Out[75]:

```
0.0    45490
1.0       79
Name: Target, dtype: int64
```

Model Building

In [76]:

```python
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.linear_model import LogisticRegression
```

Logistic Regression

In [77]:

```python
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))
```

```
[[181939     22]
 [   119    196]]

[[45484     6]
 [   31    48]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       0.90      0.62      0.74       315

    accuracy                           1.00    182276
   macro avg       0.95      0.81      0.87    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       0.89      0.61      0.72        79

    accuracy                           1.00     45569
   macro avg       0.94      0.80      0.86     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 0.9992264478044285

Test Accuracy 0.999188044503939
```

In [78]:

```
y.value_counts()
```

Out[78]:

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

In [201…

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

Out[201]:

```
0.001729245759178389
```

DecisionTree Classifier

In [80]:

```
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)
# 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]
 [   29    50]]

              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       0.72      0.63      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 0.9989466523294345
```

RandomForestClassifier

In [81]:

```python
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]]

[[45488     2]
 [   23    56]]

              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       0.97      0.71      0.82        79

    accuracy                           1.00     45569
   macro avg       0.98      0.85      0.91     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 1.0

Test Accuracy 0.9994513814215804
```

XGBoost Classifier

In [82]:

```python
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))
```

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

[[45489     1]
 [   21    58]]

              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       0.98      0.73      0.84        79

    accuracy                           1.00     45569
   macro avg       0.99      0.87      0.92     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 1.0

Test Accuracy 0.9995172156509908
```

Support Vector Maching

In [83]:

```
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))
```

```
[[181936     25]
 [   207    108]]

[[45484     6]
 [   57    22]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       0.81      0.34      0.48       315

    accuracy                           1.00    182276
   macro avg       0.91      0.67      0.74    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       0.79      0.28      0.41        79

    accuracy                           1.00     45569
   macro avg       0.89      0.64      0.71     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 0.9987272048980667

Test Accuracy 0.9986174811823828
```

K Nearest Neighbors

In [84]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier()
knn = knn_model.fit(x_train, y_train)
y_pred_train_knn = knn_model.predict(x_train)
y_pred_test_knn = knn_model.predict(x_test)
# Confusion Matrix
print(confusion_matrix(y_train, y_pred_train_knn))
print()
print(confusion_matrix(y_test, y_pred_test_knn))
print()
# classification_report
print(classification_report(y_train, y_pred_train_knn))
print()
print(classification_report(y_test, y_pred_test_knn))
print()
# accuracy_score
print("Train Accuracy", accuracy_score(y_train, y_pred_train_knn))
print()
print("Test Accuracy", accuracy_score(y_test, y_pred_test_knn))
```

```
[[181943     18]
 [    68    247]]

[[45488     2]
 [   27    52]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       0.93      0.78      0.85       315

    accuracy                           1.00    182276
   macro avg       0.97      0.89      0.93    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       0.96      0.66      0.78        79

    accuracy                           1.00     45569
   macro avg       0.98      0.83      0.89     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 0.9995281880225592

Test Accuracy 0.9993636024490333
```

Naive Bayes Theorem

In [85]:
```
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))
```

```
[[181595    366]
 [    95    220]]

[[45387   103]
 [   29    50]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       0.38      0.70      0.49       315

    accuracy                           1.00    182276
   macro avg       0.69      0.85      0.74    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       0.33      0.63      0.43        79

    accuracy                           1.00     45569
   macro avg       0.66      0.82      0.71     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 0.997470868353486

Test Accuracy 0.9971032939059449
```

Voting Classifier

In [86]:

```
from sklearn.ensemble import VotingClassifier
 # voting classifier combines all the algorithms and gives the best accuracy
```

In [87]:

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

```
[[181956      5]
 [    54    261]]

[[45488     2]
 [   24    55]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       0.98      0.83      0.90       315

    accuracy                           1.00    182276
   macro avg       0.99      0.91      0.95    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       0.96      0.70      0.81        79

    accuracy                           1.00     45569
   macro avg       0.98      0.85      0.90     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 0.9996763150387324

Test Accuracy 0.9994294366784436
```

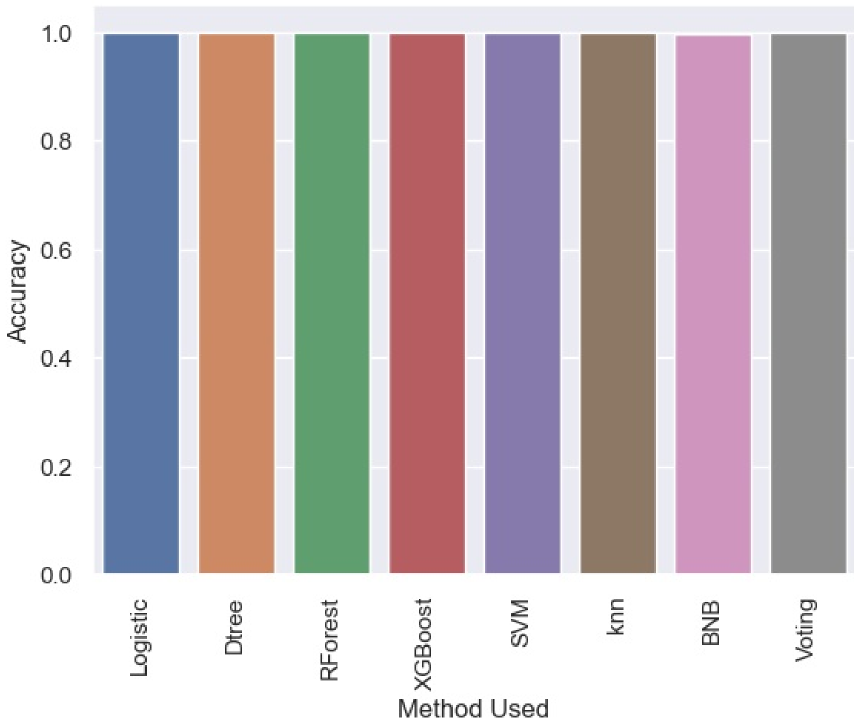
In [88]:

```
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_knn = accuracy_score(y_test, y_pred_test_knn)
accuracy_bnb = accuracy_score(y_test, y_pred_test_bnb)
accuracy_voting = accuracy_score(y_test, y_pred_test_voting)
```

In [89]:

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

```
  Method Used  Accuracy
0    Logistic  0.999188
1       Dtree  0.998947
2     RForest  0.999451
3     XGBoost  0.999517
4         SVM  0.998617
5         knn  0.999364
6         BNB  0.997103
7      Voting  0.999429
Axes(0.125,0.11;0.775x0.77)
```



Stacking method

In [90]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import BernoulliNB
```

In [91]:
```
estimators = [('dt', DecisionTreeClassifier()),
             ('rf', RandomForestClassifier()),
             ('gdbt',GradientBoostingClassifier()),
             ('svm', SVC()),
             ('knn',KNeighborsClassifier()),
             ('nbt',BernoulliNB())]
```

In [92]:
```
from sklearn.ensemble import StackingClassifier
```

In [93]:
```
classifier = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression(),
                                cv=10)
```

In [94]:
```
classifier.fit(x_train, y_train)
```

Out[94]:

```
StackingClassifier
dt                      rf                      gdbt                        svm   knn
DecisionTreeClassifier  RandomForestClassifier  GradientBoostingClassifier  SVC   KNeighborsClassifier
final_estimator
LogisticRegression
```

In [95]:

```python
y_pred_train = classifier.predict(x_train)
y_pred_test = classifier.predict(x_test)
```

In [96]:

```python
# 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))
```

```
[[181961      0]
 [    52    263]]

[[45488     2]
 [   28    51]]

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181961
         1.0       1.00      0.83      0.91       315

    accuracy                           1.00    182276
   macro avg       1.00      0.92      0.95    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       0.96      0.65      0.77        79

    accuracy                           1.00     45569
   macro avg       0.98      0.82      0.89     45569
weighted avg       1.00      1.00      1.00     45569


Train Accuracy 0.9997147183392219

Test Accuracy 0.9993416577058966
```

Anomaly Detection - Isolation Forest Classifier

In [97]:

```python
from sklearn.ensemble import IsolationForest

# domain specific algorithim extensively for banking domain to identify the fraud data
```

In [98]:

```python
isolation = IsolationForest(contamination=fraud_data)
isolation.fit(x_train, y_train)
```

Out[98]:

```
IsolationForest
IsolationForest(contamination=0.001729245759178389)
```

In [99]:

```python
anomaly_pred_train = isolation.predict(x_train)
anomaly_pred_test = isolation.predict(x_test)
```

In [100…

```python
pd.DataFrame(anomaly_pred_test).value_counts()
```

Out[100]:

```
 1    45497
-1       72
dtype: int64
```

In [101…

```python
# Confusion Matrix
```

```
print(confusion_matrix(y_train, anomaly_pred_train))
print()
print(confusion_matrix(y_test, anomaly_pred_test))
print()
# classification_report
print(classification_report(y_train, anomaly_pred_train))
print()
print(classification_report(y_test, anomaly_pred_test))
print()
# accuracy_score
print("Train Accuracy", accuracy_score(y_train, anomaly_pred_train))
print()
print("Test Accuracy", accuracy_score(y_test, anomaly_pred_test))
```

```
[[     0      0      0]
 [   229      0 181732]
 [    87      0    228]]

[[    0     0     0]
 [   52     0 45438]
 [   20     0    59]]

              precision    recall  f1-score   support

        -1.0       0.00      0.00      0.00         0
         0.0       0.00      0.00      0.00    181961
         1.0       0.00      0.72      0.00       315

    accuracy                           0.00    182276
   macro avg       0.00      0.24      0.00    182276
weighted avg       0.00      0.00      0.00    182276


              precision    recall  f1-score   support

        -1.0       0.00      0.00      0.00         0
         0.0       0.00      0.00      0.00     45490
         1.0       0.00      0.75      0.00        79

    accuracy                           0.00     45569
   macro avg       0.00      0.25      0.00     45569
weighted avg       0.00      0.00      0.00     45569


Train Accuracy 0.0012508503587965504

Test Accuracy 0.0012947398450701135
```

In [102…

```
#iso_model = IsolationForest(contamination=0.001)
#iso_model.fit(x_train, y_train)
#anomaly_pred = iso_model.predict(x_test)

for i, x in enumerate(x_test):
    if anomaly_pred_test[i] == 1:
        print(f"Anomaly Detected:{x}")
    else:
        print(f"Normal Transaction:{x}")
```

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

In [103…

```
from sklearn.neighbors import LocalOutlierFactor
```

```python
from sklearn.svm import OneClassSVM
```

In [104…
```python
OneClassSVM()
```

Out[104]:

▾ OneClassSVM

OneClassSVM()

In [139…
```python
len(x)
```

Out[139]: 227845

In [140…
```python
final_classification_model = {"IsolationForest": IsolationForest(n_estimators=100,contamination=fraud_data, max
                              "LocalOutlierFactor" : LocalOutlierFactor(contamination=fraud_data),
                             "OneClassSVM" : OneClassSVM()}
```

In [204…
```python
train_data.columns
```

Out[204]:
```
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 [205…
```python
test_data.columns
```

Out[205]:
```
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 [206…
```python
fraud = train_data[train_data['Target']==1]
normal = train_data[train_data['Target']==0]
```

In [207…
```python
 len(fraud)
```

Out[207]: 394

In [208…
```python
total_outlier_found = len(fraud)

for i , (clf_name, clf) in enumerate(final_classification_model.items()):
    if clf_name =="LocalOutlierFactor" :
        y_pred = clf.fit_predict(x_test)
        score_prediction = clf.negative_outlier_factor_
    elif clf_name =="OneClassSVM":
        clf.fit(x_train)
        y_pred = clf.predict(x_test)

    else:
        clf.fit(x_train)
        score_prediction = clf.decision_function(x_train)
        y_pred = clf.predict(x_test)

    y_pred[y_pred == 1] = 0
    y_pred[y_pred == -1] = 1
    n_error = (y_pred !=y_test).sum()

    print("{} : {}".format(clf_name, n_error))
    print("Accuracy Score :")

    print(accuracy_score(y_test, y_pred))
```

```
IsolationForest : 123
Accuracy Score :
0.9973007965941759
LocalOutlierFactor : 144
Accuracy Score :
0.9968399569883034
OneClassSVM : 22759
Accuracy Score :
0.5005595909499879
```

In [ ]: