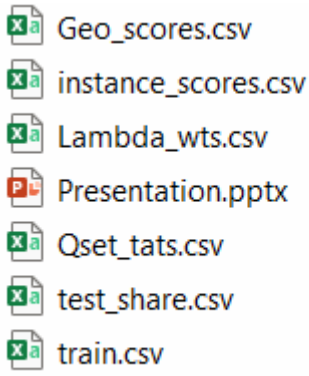
Bank ATM Fraud Detection

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' .□ 1: Fraudulent transactions□ 0: Clean transactions

In [1]:
```
import os, sys
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')
import datetime
```



In [2]:
```
geo = pd.read_csv('Geo_scores.csv')
instance = pd.read_csv('instance_scores.csv')
lambda_wts = pd.read_csv('Lambda_wts.csv')
qset = pd.read_csv('Qset_tats.csv')
train = pd.read_csv('train.csv')
test = pd.read_csv('test_share.csv')
```

geo

In [3]:
```
geo.head(2)
```

Out[3]:

	id	geo_score
0	26674	4.48
1	204314	4.48

In [4]:
```
geo.shape
```

Out[4]:
```
(1424035, 2)
```

In [5]:
```
geo['id'].nunique()
```

Out[5]:
```
284807
```

In [7]:
```
geo.isnull().sum()/len(geo)*100
```

Out[7]:
```
id           0.000000
geo_score    5.023964
dtype: float64
```

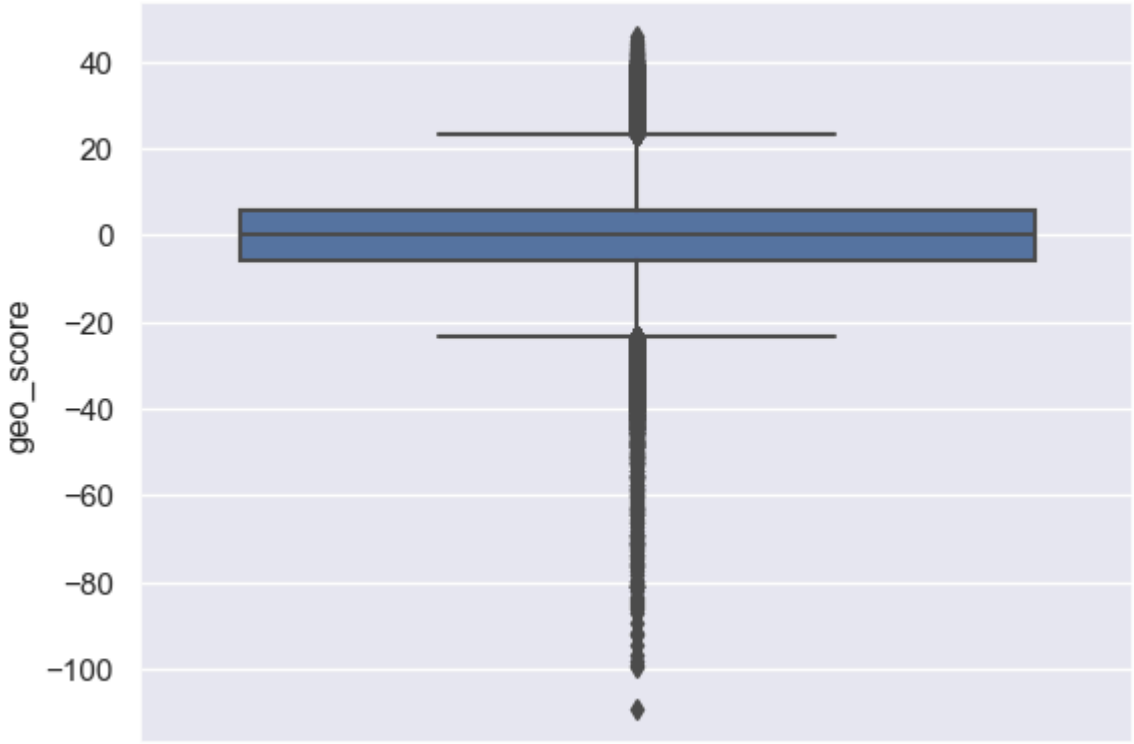
In [8]:
```
geo.describe()
```

Out[8]:

	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

In [9]:
```
sns.boxplot(y='geo_score', data=geo)
```

Out[9]:
```
<AxesSubplot:ylabel='geo_score'>
```



In [10]:
```
geo.fillna(geo['geo_score'].median(), inplace=True)
```

In [11]:
```
geo.isnull().sum()
```

Out[11]:
```
id           0
geo_score    0
dtype: int64
```

In [50]:
```
geo.shape
```

Out[50]:
```
(1424035, 2)
```

In [51]:
```
geo.nunique()
```

Out[51]:
```
id           284807
geo_score     25524
dtype: int64
```

In [52]:
```
geo = geo.groupby('id').mean()
```

In [53]:
```
geo.shape
```

Out[53]:
```
(284807, 1)
```

In [56]:
```
geo.head(2)
```

Out[56]:

	geo_score
id	
0	-0.620
1	1.106

In [ ]:
```

```

instance

In [12]:
```
instance.shape
```

Out[12]:
```
(1424035, 2)
```

In [13]:
```
instance.head(2)
```

Out[13]:

	id	instance_scores
0	173444	-0.88
1	259378	1.50

In [14]:
```
instance.nunique()
```

Out[14]:
```
id                284807
instance_scores    11158
dtype: int64
```

In [15]:
```
instance.isnull().sum()
```

Out[15]:
```
id                0
instance_scores   0
dtype: int64
```

In [54]:
```
instance.shape
```

Out[54]:
```
(1424035, 2)
```

In [57]:
```
instance = instance.groupby('id').mean()
```

In [58]:
```
instance.shape
```

Out[58]:
```
(284807, 1)
```

In [ ]:
```

```

lambda_wts

In [17]:
```
lambda_wts.head(2)
```

Out[17]:

	Group	lambda_wt
0	Grp936	3.41
1	Grp347	-2.88

In [18]:
```
lambda_wts.shape
```

Out[18]:
```
(1400, 2)
```

In [19]:
```
lambda_wts.isnull().sum()
```

Out[19]:
```
Group        0
lambda_wt    0
dtype: int64
```

In [ ]:
```
```

qset

In [20]:
```
qset.head(2)
```

Out[20]:

	id	qsets_normalized_tat
0	9983	2.41
1	266000	3.10

In [21]:
```
qset.shape
```

Out[21]:
```
(1424035, 2)
```

In [22]:
```
qset.nunique()
```

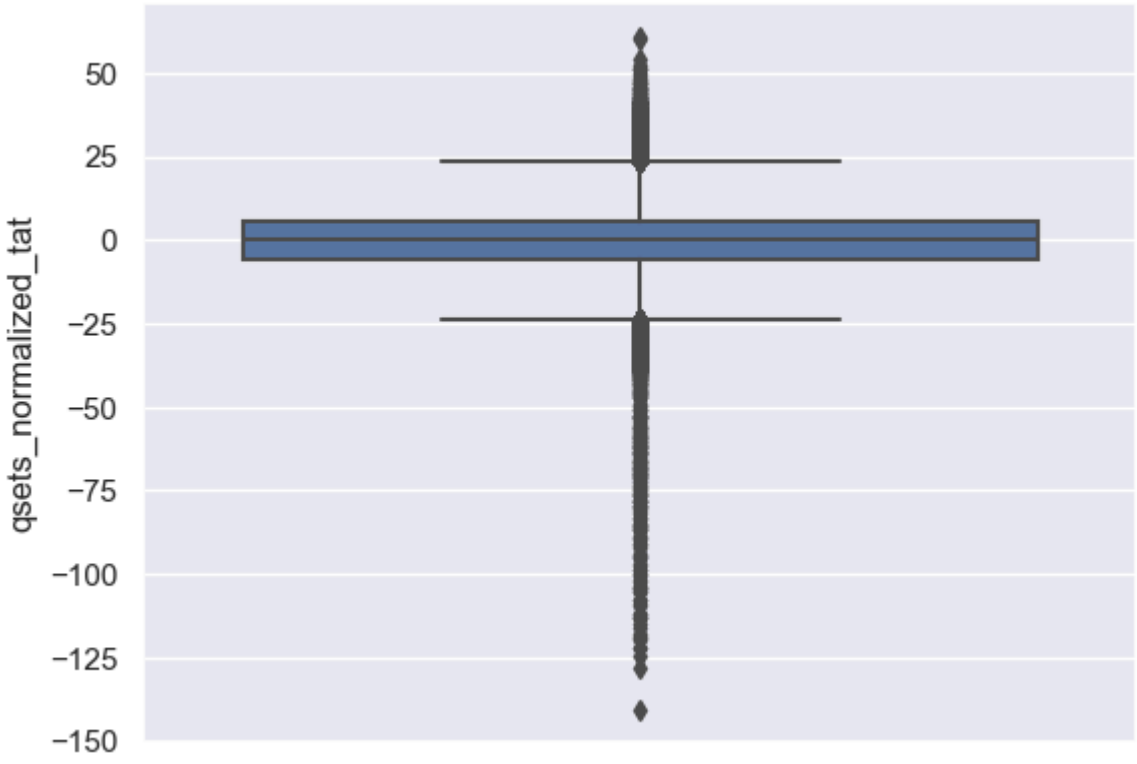
Out[22]:
```
id                     284807
qsets_normalized_tat    24832
dtype: int64
```

In [24]:
```
qset.isnull().sum()/len(qset)*100
```

Out[24]:
```
id                     0.000000
qsets_normalized_tat    7.247083
dtype: float64
```

In [25]:
```
sns.boxplot(y='qsets_normalized_tat', data=qset)
```

Out[25]:
```
<AxesSubplot:ylabel='qsets_normalized_tat'>
```



In [26]:
```
qset.fillna(qset['qsets_normalized_tat'].median(), inplace=True)
```

In [27]:
```
qset.isnull().sum()
```

Out[27]:
```
id                    0
qsets_normalized_tat    0
dtype: int64
```

In [59]:
```
qset.shape
```

Out[59]:
```
(1424035, 2)
```

In [60]:
```
qset = qset.groupby('id').mean()
```

In [61]:
```
qset.shape
```

Out[61]:
```
(284807, 1)
```

In [ ]:
```

```

train

In [28]:
```
train.head(2)
```

Out[28]:

	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	...
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.81	0.783333	...

2 rows × 28 columns

In [29]:
```
train.shape
```

Out[29]:
```
(227845, 28)
```

In [30]:
```
test.shape
```

Out[30]:
```
(56962, 27)
```

In [31]:
```
227845+56962
```

Out[31]:
```
284807
```

In [32]:
```
train.isnull().sum()
```

Out[32]:
```
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           0
dtype: int64
```

In [33]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227845 entries, 0 to 227844
Data columns (total 28 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   id              227845 non-null  int64
 1   Group           227845 non-null  object
 2   Per1            227845 non-null  float64
 3   Per2            227845 non-null  float64
 4   Per3            227845 non-null  float64
 5   Per4            227845 non-null  float64
 6   Per5            227845 non-null  float64
 7   Per6            227845 non-null  float64
 8   Per7            227845 non-null  float64
 9   Per8            227845 non-null  float64
 10  Per9            227845 non-null  float64
 11  Dem1            227845 non-null  float64
 12  Dem2            227845 non-null  float64
 13  Dem3            227845 non-null  float64
 14  Dem4            227845 non-null  float64
 15  Dem5            227845 non-null  float64
 16  Dem6            227845 non-null  float64
 17  Dem7            227845 non-null  float64
 18  Dem8            227845 non-null  float64
 19  Dem9            227845 non-null  float64
 20  Cred1           227845 non-null  float64
 21  Cred2           227845 non-null  float64
 22  Cred3           227845 non-null  float64
 23  Cred4           227845 non-null  float64
 24  Cred5           227845 non-null  float64
 25  Cred6           227845 non-null  float64
 26  Normalised_FNT  227845 non-null  float64
 27  Target          227845 non-null  int64
dtypes: float64(25), int64(2), object(1)
memory usage: 48.7+ MB
```

In [34]:
```
test.head()
```

Out[34]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per
0	146574	Grp229	-0.300000	1.540000	0.220000	-0.280000	0.570000	0.260000	0.700000	1.07666
1	268759	Grp141	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0.55000
2	59727	Grp188	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0.68666
3	151544	Grp426	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0.66333
4	155008	Grp443	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0.68666

5 rows × 27 columns

In [35]:
```
train['Group'].nunique()
```

Out[35]:
```
1301
```

In [36]:
```
test['Group'].nunique()
```

Out[36]:
```
915
```

In [38]:
```
train['data'] = 'train'
```

In [39]:
```
train.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]:
```
test['data']='test'
```

In [41]:
```
test.head()
```

Out[41]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per
0	146574	Grp229	-0.300000	1.540000	0.220000	-0.280000	0.570000	0.260000	0.700000	1.07666
1	268759	Grp141	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0.55000
2	59727	Grp188	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0.68666
3	151544	Grp426	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0.66333
4	155008	Grp443	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0.68666

5 rows × 28 columns

In [42]:
```
train.columns
```

Out[42]:
```
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 [43]:
```
test.columns
```

Out[43]:
```
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 [44]:
```
all_data = pd.concat([train, test], axis=0)
```

In [46]:
```
all_data.shape
```

Out[46]:
```
(284807, 29)
```

In [55]:
```
all_data['id'].nunique()
```

Out[55]:
```
284807
```

In [48]:
```
all_data.tail()
```

Out[48]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	
56957	18333	Grp102	0.553333	1.043333	1.096667	0.686667	0.673333	0.340000	0.900000	0.6
56958	244207	Grp504	1.353333	0.616667	0.276667	0.783333	0.690000	0.650000	0.473333	0.6
56959	103277	Grp78	1.083333	0.433333	0.806667	0.490000	0.243333	0.316667	0.533333	0.6
56960	273294	Grp134	0.566667	1.153333	0.370000	0.616667	0.793333	0.226667	0.910000	0.6
56961	223337	Grp18	1.426667	0.110000	-0.006667	-0.200000	0.983333	1.870000	0.033333	0.9

5 rows × 29 columns

In [49]:
```
all_data['Group'].nunique()
```

Out[49]:
```
1400
```

In [62]:
```
geo.head(1)
```

Out[62]:

	geo_score
id	
0	-0.62

merge all 4 data into all_dataset

In [63]:
```
all_data = pd.merge(all_data, geo, on='id', how='left')
```

In [64]:
```
all_data.shape
```

Out[64]:
```
(284807, 30)
```

In [65]:
```
all_data.head()
```

Out[65]:

	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 × 30 columns

In [66]:
```
# instance
instance.head(2)
```

Out[66]:

id	instance_scores
0	0.09
1	-0.17

In [67]:
```
all_data = pd.merge(all_data, instance, on='id', how='left')
```

In [68]:
```
all_data.head()
```

Out[68]:

	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 × 31 columns

In [69]:
```
lambda_wts.shape
```

Out[69]:
```
(1400, 2)
```

In [70]:
```
lambda_wts.head()
```

Out[70]:

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

In [71]:
```
all_data['Group'].nunique()
```

Out[71]:
```
1400
```

In [72]:
```
all_data = pd.merge(all_data, lambda_wts, on='Group', how='left')
```

In [73]:
```
all_data.head()
```

Out[73]:

	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 × 32 columns

In [74]:
```
qset.shape
```

Out[74]:
```
(284807, 1)
```

In [75]:
```
qset.head(2)
```

Out[75]:

id	qsets_normalized_tat
0	0.214
1	-0.110

In [76]:
```
all_data = pd.merge(all_data, qset, on='id', how='left')
```

In [77]:
```
all_data.head()
```

Out[77]:

	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 [78]:
```
all_data.isnull().sum()
```

Out[78]:
```
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
geo_score                   0
instance_scores             0
lambda_wt                   0
qsets_normalized_tat        0
dtype: int64
```

In [79]:
```
train = all_data[all_data['data']=='train']
test = all_data[all_data['data']=='test']
```

In [80]:
```
train.shape
```

Out[80]:
```
(227845, 33)
```

In [81]:
```
test.shape
```

Out[81]:
```
(56962, 33)
```

In [83]:
```
train.head(10)
```

Out[83]:

	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	144029	Grp45	0.873333	0.140000	0.836667	0.273333	0.190000	0.653333	0.503333	0.723333
6	127618	Grp69	0.980000	0.546667	0.820000	0.863333	0.680000	1.163333	0.486667	0.893333
7	116319	Grp198	-0.136667	0.360000	0.366667	0.996667	0.473333	0.706667	0.676667	0.933333
8	66485	Grp57	1.010000	0.606667	0.823333	1.066667	0.516667	0.653333	0.633333	0.703333
9	117942	Grp271	-0.133333	0.890000	0.210000	0.333333	0.743333	1.023333	0.416667	0.096667

10 rows × 33 columns

In [82]:
```
train.columns
```

Out[82]:
```
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',
       'lambda_wt', 'qsets_normalized_tat'],
      dtype='object')
```

In [84]:
```
# split the data into ind and dependent variable
x_train = train.drop(['id','Group','Target','data'], axis=1)
y_train = train['Target']
```

In [85]:
```
x_train.head()
```

Out[85]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Der
0	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.4600
1	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.4700
2	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.6600
3	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.0966
4	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.7566

5 rows × 29 columns

In [87]:
```
x_train.isnull().sum()
```

Out[87]:
```
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
geo_score             0
instance_scores       0
lambda_wt             0
qsets_normalized_tat  0
dtype: int64
```

In [86]:
```
y_train.head()
```

Out[86]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: Target, dtype: float64
```

In [104…
```
y_train.value_counts()
```

Out[104]:
```
0.0    227451
1.0       394
Name: Target, dtype: int64
```

In [126…
```
fraud = train[train['Target']==1]
```

In [128…
```
fraud
```

Out[128]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per
357	64460	Grp1075	-3.070000	3.303333	-3.996667	2.110000	-2.160000	-0.503333	-2.31333
591	131272	Grp28	0.630000	1.013333	1.326667	1.710000	0.420000	1.183333	0.10000
1792	154633	Grp359	0.326667	1.166667	-0.830000	1.280000	0.876667	-0.246667	0.15000
1801	15506	Grp787	-6.630000	4.976667	-7.366667	2.733333	-4.823333	-0.820000	-4.84000
2031	204064	Grp1317	0.743333	0.980000	-0.883333	1.693333	0.033333	0.320000	0.32666
...	...	...	...	...	...	...	...	...	
225728	42609	Grp994	-1.500000	2.250000	-2.323333	3.033333	-1.653333	-0.273333	-2.77666
225924	88307	Grp375	0.133333	1.536667	-0.330000	1.686667	-0.166667	0.486667	-0.09333
226338	141258	Grp1210	0.353333	1.820000	-1.483333	2.310000	-0.076667	-0.096667	-1.23333
227267	249607	Grp1370	-1.793333	-1.816667	-0.900000	1.910000	2.733333	-1.470000	-1.27666
227383	154693	Grp1071	-0.163333	2.120000	-1.926667	2.453333	0.060000	-0.150000	-0.98666

394 rows × 33 columns

In [ ]:

In [132…]:
```
fraudlent_transaction = (394/(227451+394))*100
fraudlent_transaction
```

Out[132]:
```
0.17292457591783889
```

In [ ]:

In [89]:
```
test.head()
```

Out[89]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	
227845	146574	Grp229	-0.300000	1.540000	0.220000	-0.280000	0.570000	0.260000	0.700000	1
227846	268759	Grp141	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0
227847	59727	Grp188	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0
227848	151544	Grp426	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0
227849	155008	Grp443	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0

5 rows × 33 columns

In [88]:
```
test.columns
```

Out[88]:
```
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',
       'lambda_wt', 'qsets_normalized_tat'],
      dtype='object')
```

In [90]:
```
x_test = test.drop(['id','Group','Target','data'], axis=1)
y_test = test['Target']
```

In [91]:
```
x_test.head()
```

Out[91]:

	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
227846	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0.550000	0.543333
227847	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0.686667	0.593333
227848	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0.663333	1.156667
227849	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0.686667	1.476667

5 rows × 29 columns

In [92]:
```
y_test.head()
```

Out[92]:
```
227845   NaN
227846   NaN
227847   NaN
227848   NaN
227849   NaN
Name: Target, dtype: float64
```

In [94]:
```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 227845 entries, 0 to 227844
Data columns (total 29 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Per1                  227845 non-null  float64
 1   Per2                  227845 non-null  float64
 2   Per3                  227845 non-null  float64
 3   Per4                  227845 non-null  float64
 4   Per5                  227845 non-null  float64
 5   Per6                  227845 non-null  float64
 6   Per7                  227845 non-null  float64
 7   Per8                  227845 non-null  float64
 8   Per9                  227845 non-null  float64
 9   Dem1                  227845 non-null  float64
 10  Dem2                  227845 non-null  float64
 11  Dem3                  227845 non-null  float64
 12  Dem4                  227845 non-null  float64
 13  Dem5                  227845 non-null  float64
 14  Dem6                  227845 non-null  float64
 15  Dem7                  227845 non-null  float64
 16  Dem8                  227845 non-null  float64
 17  Dem9                  227845 non-null  float64
 18  Cred1                 227845 non-null  float64
 19  Cred2                 227845 non-null  float64
 20  Cred3                 227845 non-null  float64
 21  Cred4                 227845 non-null  float64
 22  Cred5                 227845 non-null  float64
 23  Cred6                 227845 non-null  float64
 24  Normalised_FNT        227845 non-null  float64
 25  geo_score             227845 non-null  float64
 26  instance_scores       227845 non-null  float64
 27  lambda_wt             227845 non-null  float64
 28  qsets_normalized_tat  227845 non-null  float64
dtypes: float64(29)
memory usage: 52.1 MB
```

In [95]:
```
x_train.describe()
```

Out[95]:

	Per1	Per2	Per3	Per4	Per5	Per6
count	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000
mean	0.666006	0.667701	0.666315	0.666687	0.666723	0.667378
std	0.654133	0.548305	0.506357	0.471956	0.461393	0.444573
min	-18.136667	-23.573333	-15.443333	-1.226667	-37.246667	-8.053333
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	8.020000	3.793333	6.163333	12.266667	25.100000

8 rows × 29 columns

In [ ]:
```
# Please do EDA part - heatmap, pairplot, distplot, Pandas profiling, Dtale, datapr
```

In [97]:
```
# Feature scaling
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
sc_x_train = sc.fit_transform(x_train)
sc_x_test = sc.fit_transform(x_test)
```

In [ ]:

In [99]:
```
# split the data into training and test
from sklearn.model_selection import train_test_split
x_train1, x_test1, y_train1, y_test1 = train_test_split(sc_x_train, y_train, test_s
```

Logistic Regression

In [100…
```
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression()
logit.fit(x_train1, y_train1)
```

Out[100]:
```
LogisticRegression()
```

In [101…
```
y_pred_logit_train = logit.predict(x_train1)
y_pred_logit_test = logit.predict(x_test1)
```

In [102…
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [103…
```
confusion_matrix(y_test1, y_pred_logit_test)
```

Out[103]:
```
array([[45471,     5],
       [   50,    43]], dtype=int64)
```

In [107…
```
print(classification_report(y_train1,y_pred_logit_train))
print("************************"*10)
print(classification_report(y_test1,y_pred_logit_test))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181975
         1.0       0.87      0.64      0.74       301

    accuracy                           1.00    182276
   macro avg       0.94      0.82      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     45476
         1.0       0.90      0.46      0.61        93

    accuracy                           1.00     45569
   macro avg       0.95      0.73      0.80     45569
weighted avg       1.00      1.00      1.00     45569
```

In [108…
```
print(accuracy_score(y_train1,y_pred_logit_train))
print("************************"*10)
print(accuracy_score(y_test1,y_pred_logit_test))
```

```
0.9992483925475653
**********************************************************************************
**********************************************************************************
**********************************************************************
0.998793039127477
```

RandomForest Classification Model

In [110…
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(criterion='entropy', random_state=1)
rf.fit(x_train1, y_train1)
```

Out[110]:
```
RandomForestClassifier(criterion='entropy', random_state=1)
```

In [111…
```
y_pred_rf_train = rf.predict(x_train1)
y_pred_rf_test = rf.predict(x_test1)
```

In [112…
```
print(classification_report(y_train1,y_pred_rf_train))
print("************************"*10)
print(classification_report(y_test1,y_pred_rf_test))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181975
         1.0       1.00      1.00      1.00       301

    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     45476
         1.0       0.94      0.69      0.80        93

    accuracy                           1.00     45569
   macro avg       0.97      0.84      0.90     45569
weighted avg       1.00      1.00      1.00     45569
```

In [113…
```
print(accuracy_score(y_train1,y_pred_rf_train))
print("************************"*10)
print(accuracy_score(y_test1,y_pred_rf_test))
```

```
1.0
**********************************************************************************
**********************************************************************************
**********************************************************************
0.9992758234764862
```

In [ ]:

XGBoost Classifier

In [ ]:
```
# !pip install xgboost
```

In [115…
```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb = xgb.fit(x_train1, y_train1)
```

In [117…
```
y_pred_xgb_train = xgb.predict(x_train1)
y_pred_xgb_test = xgb.predict(x_test1)
```

In [118…
```
print(classification_report(y_train1,y_pred_xgb_train))
print("**********"*10)
print(classification_report(y_test1,y_pred_xgb_test))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    181975
         1.0       1.00      1.00      1.00       301

    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     45476
         1.0       0.94      0.73      0.82        93

    accuracy                           1.00     45569
   macro avg       0.97      0.87      0.91     45569
weighted avg       1.00      1.00      1.00     45569
```

In [119…
```
print(accuracy_score(y_train1,y_pred_xgb_train))
print("**********"*10)
print(accuracy_score(y_test1,y_pred_xgb_test))
```

```
1.0
**********
**********
**********
0.9993636024490333
```

In [122…
```
x_test.head()
```

Out[122]:

	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
227846	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0.550000	0.543333
227847	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0.686667	0.593333
227848	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0.663333	1.156667
227849	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0.686667	1.476667

5 rows × 29 columns

In [123…
```
final_target_value = xgb.predict(sc_x_test)
```

In [124…
```
final_target_value
```

Out[124]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [ ]:
```

```

In [ ]:
```

```

ISOLATION FOREST, Local Outlier Factor , OneClassSVM - These models dedicated for anomaly detection

In [121…
```
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
```

In [ ]:
```
OneClassSVM()
```

In [125…
```
classification = {'IsolationForest' : IsolationForest(n_estimators=100,max_samples=
                                                      contamination= fraudlent_transa

                  "LocalOutlierFactor" : LocalOutlierFactor(n_neighbors=20, contamir
                  "OneClassSVM" : OneClassSVM()}
```

In [136…
```
n_outlier = len(fraud)
n_outlier
```

Out[136]:
```
394
```

In [129…
```
isolaction = IsolationForest(n_estimators=100,max_samples=len(x_train), contaminati
```

In [130…
```
isolaction.fit(x_train1, y_train1)
```

Out[130]:
```
IsolationForest(contamination=0.17322412299792042, max_samples=227845)
```

In [133…
```
y_pred_isolaction_train = isolaction.predict(x_train1)
y_pred_isolaction_test = isolaction.predict(x_test1)
```

In [134…
```
print(classification_report(y_train1,y_pred_isolaction_train))
print("**********************"*10)
print(classification_report(y_test1,y_pred_isolaction_test))
```

```
              precision    recall  f1-score   support

        -1.0       0.00      0.00      0.00         0
         0.0       0.00      0.00      0.00    181975
         1.0       0.00      0.07      0.00       301

    accuracy                           0.00    182276
   macro avg       0.00      0.02      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     45476
         1.0       0.00      0.14      0.00        93

    accuracy                           0.00     45569
   macro avg       0.00      0.05      0.00     45569
weighted avg       0.00      0.00      0.00     45569
```

In [135…

```
print(accuracy_score(y_train1,y_pred_isolaction_train))
print("************************"*10)
print(accuracy_score(y_test1,y_pred_isolaction_test))
```

```
0.00012069608725229871
**************************************************************************************
**************************************************************************************
**************************************************************************
0.0002852816607781606
```

In [ ]: