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.

- **[geo_scores]** - location of the transactions
- **[Lambda_wts]** - their own proprietary index
- **[Qset_tats]** - on network turn around times (Transaction time)
- **[instance_scores]** - vulnerability qualification score (cibil scores)

Training data contains masked variables pertaining to each transaction id .

prediction target here is 'Target' .

1: Fraudulent transactions 0: Clean transactions

Importing Libraries

In [5]:

```
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')

# Display all the columns of the dataframe
pd.pandas.set_option('display.max_columns',None)
```

Importing Datasets

In [6]:

```
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 = pd.read_csv('test_share.csv')
train = pd.read_csv('train.csv')
```

Dataset Information

In [7]:

```
datasets = [geo, instance, lambdawts, qset, test, train]
names = ["geo_scores", "instance_scores", "lambdawts", "qset", "test", "tra

# ANSI escape code for bold text
bold_text = "\033[1m"
# ANSI escape code to reset text formatting
reset_text = "\033[0m"

for name, dataset in zip(names, datasets):
    # Print shape
    print(f"{bold_text} {name} shape:{reset_text} {dataset.shape}")
    print("---------" * 10)

    # Print head
    print(f"{bold_text}{name} head:{reset_text}")
    print(dataset.head())
    print("---------" * 10)
```

```
 geo_scores shape: (1424035, 2)
-----------------------------------------------------------------------------
-------------
geo_scores head:
       id  geo_score
0   26674       4.48
1  204314       4.48
2  176521       5.17
3   48812      -2.41
4  126870       6.55
-----------------------------------------------------------------------------
-------------
 instance_scores shape: (1424035, 2)
-----------------------------------------------------------------------------
-------------
instance_scores head:
       id  instance_scores
0  173444            -0.88
1  259378             1.50
2  161170             0.44
3  191161             0.76
4   34521            -0.84
-----------------------------------------------------------------------------
-------------
 lambdawts shape: (1400, 2)
-----------------------------------------------------------------------------
-------------
lambdawts head:
     Group  lambda_wt
0   Grp936       3.41
1   Grp347      -2.88
2   Grp188       0.39
3  Grp1053      -2.75
4    Grp56      -0.83
-----------------------------------------------------------------------------
-------------
 qset shape: (1424035, 2)
-----------------------------------------------------------------------------
-------------
qset head:
       id  qsets_normalized_tat
0    9983                  2.41
1  266000                  3.10
2   77525                  1.03
3  160765                -11.63
4  138220                 -4.48
-----------------------------------------------------------------------------
-------------
 test shape: (56962, 27)
-----------------------------------------------------------------------------
-------------
test head:
       id   Group      Per1      Per2      Per3      Per4      Per5      Per6
\
0  146574  Grp229 -0.300000  1.540000  0.220000 -0.280000  0.570000  0.260000
1  268759  Grp141  0.633333  0.953333  0.810000  0.466667  0.910000  0.253333
2   59727  Grp188  1.043333  0.740000  0.860000  1.006667  0.583333  0.616667
3  151544  Grp426  1.283333  0.300000  0.576667  0.636667  0.256667  0.543333
4  155008  Grp443  1.186667  0.326667  0.476667  0.866667  0.436667  0.680000

       Per7      Per8      Per9      Dem1      Dem2      Dem3      Dem4  \
0  0.700000  1.076667  0.930000  0.156667  0.546667  0.530000  0.876667
1  1.040000  0.550000  0.543333  0.433333  0.966667  0.760000  0.576667
2  0.630000  0.686667  0.593333  1.250000  0.826667  0.826667  0.653333
3  0.356667  0.663333  1.156667  1.186667  0.900000  0.433333  0.230000
```

```
4  0.476667  0.686667  1.476667  1.213333  0.853333  0.583333  0.850000

       Dem5      Dem6      Dem7      Dem8      Dem9     Cred1     Cred2  \
0  0.450000  0.370000  0.786667  0.546667  0.313333  0.703333  0.813333
1  0.653333  0.553333  0.636667  0.770000  0.993333  0.536667  0.703333
2  0.663333  0.453333  0.626667  0.756667  0.953333  0.623333  0.753333
3  1.323333  0.403333  0.480000  0.460000  0.260000  0.800000  0.606667
4  1.090000  0.550000  0.706667  0.740000  0.823333  0.670000  0.896667

      Cred3     Cred4     Cred5     Cred6  Normalised_FNT
0  0.776667  0.796667  0.823333  0.783333       -249.7500
1  0.806667  0.630000  0.673333  0.673333       -249.8125
2  0.870000  0.596667  0.680000  0.670000       -248.1200
3  0.456667  0.320000  0.676667  0.660000       -222.9875
4  0.566667  0.546667  0.650000  0.663333       -196.2200
-----------------------------------------------------------------------------
-------------
```

train shape: (227845, 28)

```
-----------------------------------------------------------------------------
-------------
```

train head:

```
       id   Group      Per1      Per2      Per3      Per4      Per5      Per6
\
0  112751  Grp169  1.070000  0.580000  0.480000  0.766667  1.233333  1.993333
1   18495  Grp161  0.473333  1.206667  0.883333  1.430000  0.726667  0.626667
2   23915  Grp261  1.130000  0.143333  0.946667  0.123333  0.080000  0.836667
3   50806  Grp198  0.636667  1.090000  0.750000  0.940000  0.743333  0.346667
4  184244  Grp228  0.560000  1.013333  0.593333  0.416667  0.773333  0.460000

       Per7      Per8      Per9      Dem1      Dem2      Dem3      Dem4  \
0  0.340000  1.010000  0.863333  0.460000  0.643333  0.736667  0.756667
1  0.810000  0.783333  0.190000  0.470000  0.613333  0.883333  0.653333
2  0.056667  0.756667  0.226667  0.660000  0.730000  0.873333  0.923333
3  0.956667  0.633333  0.486667  1.096667  0.466667  0.670000  0.526667
4  0.853333  0.796667  0.516667  0.756667  0.683333  0.296667  0.780000

       Dem5      Dem6      Dem7      Dem8      Dem9     Cred1     Cred2  \
0  0.813333  0.693333  0.666667  0.680000  0.726667  0.606667  1.010000
1  0.463333  0.483333  0.583333  0.716667  0.743333  0.680000  0.690000
2  1.223333  0.686667  0.606667  0.690000  0.820000  0.600000  0.383333
3  0.783333  0.856667  0.716667  0.720000  0.900000  0.680000  0.846667
4  0.636667  0.783333  0.630000  0.603333  0.486667  0.693333  0.526667

      Cred3     Cred4     Cred5     Cred6  Normalised_FNT  Target
0  0.933333  0.603333  0.686667  0.673333       -245.7500       0
1  0.560000  0.670000  0.553333  0.653333       -248.0000       0
2  0.763333  0.670000  0.686667  0.673333       -233.1250       0
3  0.423333  0.520000  0.846667  0.760000       -249.7775       0
4  0.520000  0.716667  0.706667  0.673333       -247.5775       0
-----------------------------------------------------------------------------
-------------
```

In [8]:

```
for name, dataset in zip(names, datasets):
    # Print dataset info
    print(f"* Information of {bold_text}{name}:{reset_text}")
    print("-------" * 10)
    dataset.info()
    print("-------" * 10)

```

```
* Information of geo_scores:
----------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1424035 entries, 0 to 1424034
Data columns (total 2 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   id         1424035 non-null  int64
 1   geo_score  1352492 non-null  float64
dtypes: float64(1), int64(1)
memory usage: 21.7 MB
----------------------------------------------------------------------
* Information of instance_scores:
----------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1424035 entries, 0 to 1424034
Data columns (total 2 columns):
 #   Column           Non-Null Count    Dtype
---  ------           --------------    -----
 0   id               1424035 non-null  int64
 1   instance_scores  1424035 non-null  float64
dtypes: float64(1), int64(1)
memory usage: 21.7 MB
----------------------------------------------------------------------
* Information of lambdawts:
----------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1400 entries, 0 to 1399
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Group      1400 non-null   object
 1   lambda_wt  1400 non-null   float64
dtypes: float64(1), object(1)
memory usage: 22.0+ KB
----------------------------------------------------------------------
* Information of qset:
----------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1424035 entries, 0 to 1424034
Data columns (total 2 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   id                     1424035 non-null  int64
 1   qsets_normalized_tat   1320834 non-null  float64
dtypes: float64(1), int64(1)
memory usage: 21.7 MB
----------------------------------------------------------------------
* Information of test:
----------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56962 entries, 0 to 56961
Data columns (total 27 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              56962 non-null  int64
 1   Group           56962 non-null  object
 2   Per1            56962 non-null  float64
 3   Per2            56962 non-null  float64
 4   Per3            56962 non-null  float64
 5   Per4            56962 non-null  float64
 6   Per5            56962 non-null  float64
 7   Per6            56962 non-null  float64
 8   Per7            56962 non-null  float64
 9   Per8            56962 non-null  float64
```

```
 10  Per9             56962 non-null  float64
 11  Dem1             56962 non-null  float64
 12  Dem2             56962 non-null  float64
 13  Dem3             56962 non-null  float64
 14  Dem4             56962 non-null  float64
 15  Dem5             56962 non-null  float64
 16  Dem6             56962 non-null  float64
 17  Dem7             56962 non-null  float64
 18  Dem8             56962 non-null  float64
 19  Dem9             56962 non-null  float64
 20  Cred1            56962 non-null  float64
 21  Cred2            56962 non-null  float64
 22  Cred3            56962 non-null  float64
 23  Cred4            56962 non-null  float64
 24  Cred5            56962 non-null  float64
 25  Cred6            56962 non-null  float64
 26  Normalised_FNT   56962 non-null  float64
dtypes: float64(25), int64(1), object(1)
memory usage: 11.7+ MB
----------------------------------------------------------------------
* Information of train:
----------------------------------------------------------------------
<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
----------------------------------------------------------------------
```

Data Preprocessing

- Checking for Duplicates

In [9]:

```python
for name, dataset in zip(names, datasets):
    print(f"* Duplicated values in {bold_text}{name}:{reset_text}",dataset
    print("---------" * 10)
```

```
* Duplicated values in geo_scores: 55349
-----------------------------------------------------------------------------
-------------
* Duplicated values in instance_scores: 33600
-----------------------------------------------------------------------------
-------------
* Duplicated values in lambdawts: 0
-----------------------------------------------------------------------------
-------------
* Duplicated values in qset: 59311
-----------------------------------------------------------------------------
-------------
* Duplicated values in test: 0
-----------------------------------------------------------------------------
-------------
* Duplicated values in train: 0
-----------------------------------------------------------------------------
-------------
```

- Checking for Missing Values

In [10]:
```
for name, dataset in zip(names, datasets):
    missing_values = dataset.isnull().sum()

    print(f"* Details of Missing values in {bold_text}{name}:{reset_text}")
    print("---------" * 10)
    print(missing_values if not missing_values.empty else "No missing valu
    print("---------" * 10)

```

∗ Details of Missing values in **geo_scores:**

```
-------------------------------------------------------------------------------------------
id                0
geo_score     71543
dtype: int64
-------------------------------------------------------------------------------------------
```

∗ Details of Missing values in **instance_scores:**

```
-------------------------------------------------------------------------------------------
id                  0
instance_scores     0
dtype: int64
-------------------------------------------------------------------------------------------
```

∗ Details of Missing values in **lambdawts:**

```
-------------------------------------------------------------------------------------------
Group        0
lambda_wt    0
dtype: int64
-------------------------------------------------------------------------------------------
```

∗ Details of Missing values in **qset:**

```
-------------------------------------------------------------------------------------------
id                            0
qsets_normalized_tat     103201
dtype: int64
-------------------------------------------------------------------------------------------
```

∗ Details of Missing values in **test:**

```
-------------------------------------------------------------------------------------------
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
dtype: int64
-------------------------------------------------------------------------------------------
```

```
* Details of Missing values in train:
-------------------------------------------------------------------------------------
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 [11]:

```
# checking the missing values percentages

geo.isnull().sum()/len(geo)*100
```

Out[11]:

```
id           0.000000
geo_score    5.023964
dtype: float64
```

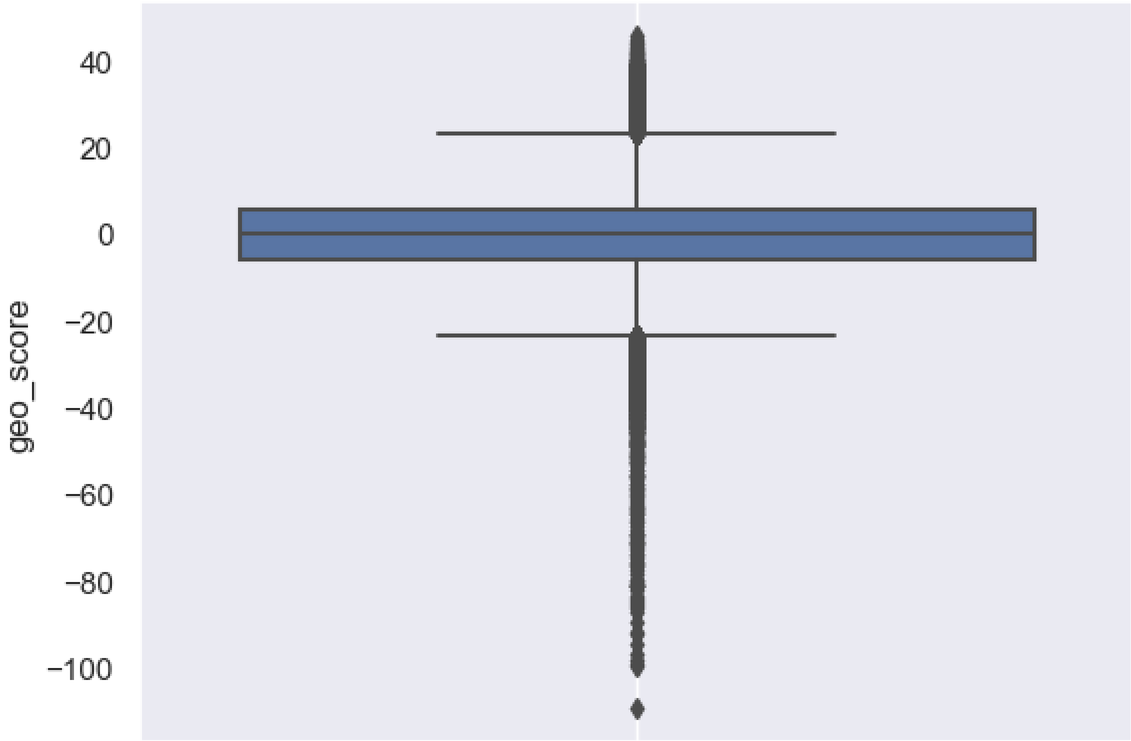
In [12]:

```
geo.describe()
```

Out[12]:

	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 [13]:
```
sns.boxplot(y='geo_score', data=geo)
plt.grid();
```



In [14]:
```
geo['geo_score'] = geo['geo_score'].fillna(geo['geo_score'].median())
```

In [15]:
```
geo['geo_score'].isnull().sum()
```

Out[15]:
```
0
```

In [16]:
```
qset.isnull().sum()/len(qset)*100
```

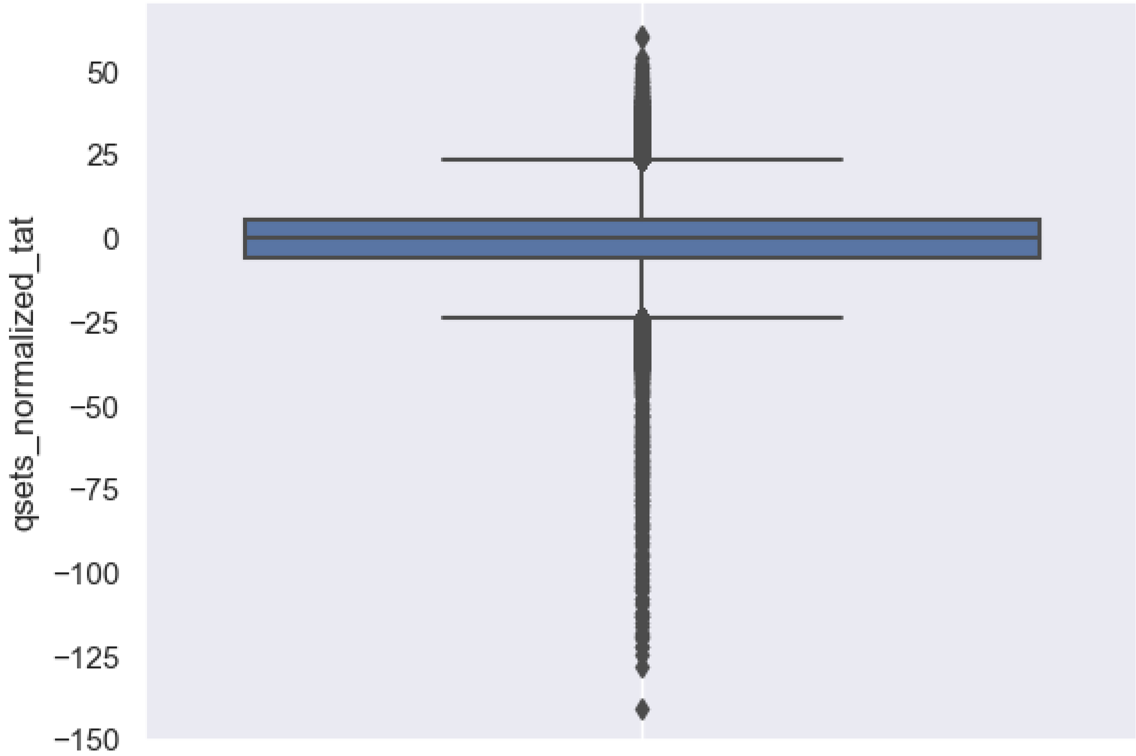
Out[16]:
```
id                        0.000000
qsets_normalized_tat      7.247083
dtype: float64
```

In [17]:
```
qset.describe()
```

Out[17]:

	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 [18]:
```python
sns.boxplot(y='qsets_normalized_tat', data=qset)
plt.grid();
```



In [19]:
```python
qset['qsets_normalized_tat'] = qset['qsets_normalized_tat'].fillna(qset['qs
```

In [20]:
```python
qset['qsets_normalized_tat'].isnull().sum()
```

Out[20]: 0

- Checking for duplicated values

In [21]:
```python
for name, dataset in zip(names, datasets):
    duplicate_values = dataset.duplicated().sum()

    print(f"* Details of Duplicated values in {bold_text}{name}:{reset_text
    print("-------" * 10)
```

```
* Details of Duplicated values in geo_scores:55349
----------------------------------------------------------------------
* Details of Duplicated values in instance_scores:33600
----------------------------------------------------------------------
* Details of Duplicated values in lambdawts:0
----------------------------------------------------------------------
* Details of Duplicated values in qset:59314
----------------------------------------------------------------------
* Details of Duplicated values in test:0
----------------------------------------------------------------------
* Details of Duplicated values in train:0
----------------------------------------------------------------------
```

In [22]:

```
1 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 [23]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56962 entries, 0 to 56961
Data columns (total 27 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              56962 non-null  int64  
 1   Group           56962 non-null  object 
 2   Per1            56962 non-null  float64
 3   Per2            56962 non-null  float64
 4   Per3            56962 non-null  float64
 5   Per4            56962 non-null  float64
 6   Per5            56962 non-null  float64
 7   Per6            56962 non-null  float64
 8   Per7            56962 non-null  float64
 9   Per8            56962 non-null  float64
 10  Per9            56962 non-null  float64
 11  Dem1            56962 non-null  float64
 12  Dem2            56962 non-null  float64
 13  Dem3            56962 non-null  float64
 14  Dem4            56962 non-null  float64
 15  Dem5            56962 non-null  float64
 16  Dem6            56962 non-null  float64
 17  Dem7            56962 non-null  float64
 18  Dem8            56962 non-null  float64
 19  Dem9            56962 non-null  float64
 20  Cred1           56962 non-null  float64
 21  Cred2           56962 non-null  float64
 22  Cred3           56962 non-null  float64
 23  Cred4           56962 non-null  float64
 24  Cred5           56962 non-null  float64
 25  Cred6           56962 non-null  float64
 26  Normalised_FNT  56962 non-null  float64
dtypes: float64(25), int64(1), object(1)
memory usage: 11.7+ MB
```

In [24]:

```
print("geo id :", geo['id'].nunique())
print("--------"*10)
print("instance id:", instance['id'].nunique())
print("--------"*10)
print("lambdawts Group :", lambdawts['Group'].nunique())
print("--------"*10)
print("qset id :", qset['id'].nunique())
print("--------"*10)
print("test - id :", test['id'].nunique())
print("--------"*10)
print("test - Group :", test['Group'].nunique())
print("--------"*10)
print("train - id :", train['id'].nunique())
print("--------"*10)
print("train - Group :", train['Group'].nunique())
```

```
geo id : 284807
--------------------------------------------------------------------------------
instance id: 284807
--------------------------------------------------------------------------------
lambdawts Group : 1400
--------------------------------------------------------------------------------
qset id : 284807
--------------------------------------------------------------------------------
test - id : 56962
--------------------------------------------------------------------------------
test - Group : 915
--------------------------------------------------------------------------------
train - id : 227845
--------------------------------------------------------------------------------
train - Group : 1301
```

- geo_scores,instance_scores and Qset_stats have similar number of unique ids = **284807**
- Train has **227845** number of unique ids.
- Test has **56962** number of unique ids.
- combining id column of Train and Test : **227845 + 56962 = 284807** unique transactions ids.
- combining Group column of Train and Test : **915 + 1301 = 1406**

In [25]:

```
print(train.shape)
print(test.shape)
```

```
(227845, 28)
(56962, 27)
```

In [26]:

```
total_rows = train.shape[0] + test.shape[0]

print(f"Combining the Train and Test : {bold_text}{total_rows}{reset_text}'
```

Combining the Train and Test : **284807**

In [27]:

```
train['data'] ='train'
test['data'] = 'test'      #to recognize the train and test data
```

In [28]:

```
all_data = pd.concat([train, test], axis=0)  #combine the train and test da
```

In [29]:

```
all_data.head()
```

Out[29]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Pe
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.8633
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.1900
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.2266
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.4866
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.5166

In [30]:

```
all_data.tail()
```

Out[30]:

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

In [31]:

```
all_data.isnull().sum()
```

Out[31]:

```
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 [32]:

```
all_data.shape
```

Out[32]:

```
(284807, 29)
```

- Joining all the Datasets one by one who is having **same id** and **same Group**

In [33]:
```
geo.shape
```
Out[33]: (1424035, 2)

In [34]:
```
geo.head()
```
Out[34]:

	id	geo_score
0	26674	4.48
1	204314	4.48
2	176521	5.17
3	48812	-2.41
4	126870	6.55

In [35]:
```
geo = geo.groupby('id').mean()
```

In [36]:
```
all_data = pd.merge(all_data, geo, on='id', how='left')
```

In [37]:
```
all_data.head()
```
Out[37]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Pe
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.8633
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.1900
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.2266
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.4866
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.5166

In [38]:
```
all_data.shape
```
Out[38]: (284807, 30)

In [39]:
```
instance['id'].nunique()
```
Out[39]: 284807

In [40]:
```
instance.shape
```
Out[40]: (1424035, 2)

In [41]:
```
instance = instance.groupby('id').mean()
```

In [42]:
```
instance.shape
```
Out[42]: (284807, 1)

In [43]:
```
all_data = pd.merge(all_data, instance, on='id', how='left')
```

In [44]:
```
all_data.shape
```
Out[44]: (284807, 31)

In [45]:
```
all_data.head()
```
Out[45]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Pe
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.8633
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.1900
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.2266
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.4866
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.5166

In [46]:
```
lambdawts['Group'].nunique()
```
Out[46]: 1400

In [47]:
```
lambdawts.shape
```
Out[47]: (1400, 2)

In [48]:
```
all_data.shape
```
Out[48]: (284807, 31)

In [49]:
```
all_data['Group'].nunique()
```
Out[49]: 1400

In [50]:
```
all_data = pd.merge(all_data, lambdawts, on='Group', how='left')
```

In [51]:
```
all_data.shape
```
Out[51]: (284807, 32)

In [52]:
```
all_data.head()
```
Out[52]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Pe
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.8633
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.1900
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.2266
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.4866
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.5166

In [53]:
```
qset['id'].nunique()
```
Out[53]: 284807

In [54]:
```
qset.shape
```
Out[54]: (1424035, 2)

In [55]:
```
qset = qset.groupby('id').mean()
```

In [56]:
```
qset.shape
```
Out[56]: (284807, 1)

In [57]:

```
all_data = pd.merge(all_data, qset, on='id', how='left')
```

In [58]:

```
all_data.head()
```

Out[58]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Pe
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.8633
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.1900
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.2266
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.4866
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.5166

In [59]:

```
all_data.tail()
```

Out[59]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8
284802	18333	Grp102	0.553333	1.043333	1.096667	0.686667	0.673333	0.340000	0.900000	0.643333
284803	244207	Grp504	1.353333	0.616667	0.276667	0.783333	0.690000	0.650000	0.473333	0.670000
284804	103277	Grp78	1.083333	0.433333	0.806667	0.490000	0.243333	0.316667	0.533333	0.606667
284805	273294	Grp134	0.566667	1.153333	0.370000	0.616667	0.793333	0.226667	0.910000	0.696667
284806	223337	Grp18	1.426667	0.110000	-0.006667	-0.200000	0.983333	1.870000	0.033333	0.963333

In [60]:

```
# split the train and test data seperately
train = all_data[all_data['data']=='train']
test = all_data[all_data['data']=='test']
```

In [61]:

```
print(train.shape)
print(test.shape)
```

```
(227845, 33)
(56962, 33)
```

In [62]:

```
# Target - train dataset
train.head()
```

Out[62]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Pe
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.8633
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.1900
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.2266
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.4866
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.5166

In [63]:

```
Fraud = train[train['Target']==1]
Valid = train[train['Target']==0]
outlier_fraction = (len(Fraud)/(len(train)))*100
print(outlier_fraction)
```

```
0.17292457591783889
```

In [64]:

```
print(len(Fraud))
print(len(Valid))
```

```
394
227451
```

In [65]:

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

In [66]:

```
x.head()
```

Out[66]:

	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	0.6
1	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000	0.6
2	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000	0.7
3	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667	0.4
4	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667	0.6

In [67]:

```
y.head()
```

Out[67]:

	Target
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

In [68]:

```
# Test dataset
test.head()
```

Out[68]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8
227845	146574	Grp229	-0.300000	1.540000	0.220000	-0.280000	0.570000	0.260000	0.700000	1.076667
227846	268759	Grp141	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0.550000
227847	59727	Grp188	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0.686667
227848	151544	Grp426	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0.663333
227849	155008	Grp443	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0.686667

In [69]:

```
test = test.drop(['id', 'Group', 'Target', 'data'], axis=1)
```

In [70]:

```
test.head()
```

Out[70]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Der
227845	-0.300000	1.540000	0.220000	-0.280000	0.570000	0.260000	0.700000	1.076667	0.930000	0.1566
227846	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0.550000	0.543333	0.4333
227847	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0.686667	0.593333	1.2500
227848	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0.663333	1.156667	1.1866
227849	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0.686667	1.476667	1.2133

In [71]:

```
x.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 [72]:

```
x.describe()
```

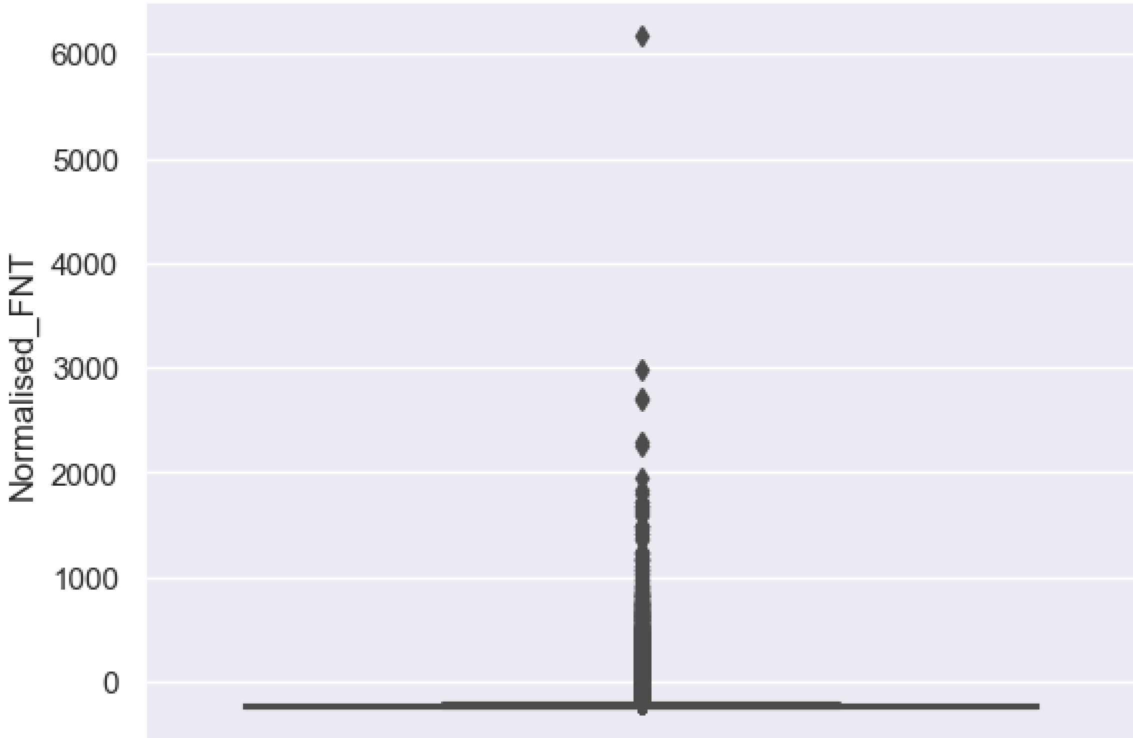
Out[72]:

	Per1	Per2	Per3	Per4	Per5	Per6	
count	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845
mean	0.666006	0.667701	0.666315	0.666687	0.666723	0.667378	0
std	0.654133	0.548305	0.506357	0.471956	0.461393	0.444573	0
min	-18.136667	-23.573333	-15.443333	-1.226667	-37.246667	-8.053333	-13
25%	0.360000	0.470000	0.370000	0.383333	0.436667	0.410000	0
50%	0.670000	0.690000	0.726667	0.660000	0.650000	0.576667	0
75%	1.103333	0.933333	1.010000	0.913333	0.870000	0.800000	0
max	1.483333	8.020000	3.793333	6.163333	12.266667	25.100000	40

In [73]:
```
x['Normalised_FNT'].describe()
```

Out[73]:
```
count    227845.000000
mean       -227.954170
std          61.951661
min        -250.000000
25%        -248.617500
50%        -244.510000
75%        -230.750000
max        6172.790000
Name: Normalised_FNT, dtype: float64
```

In [74]:
```
sns.boxplot(y='Normalised_FNT', data=x)
plt.show()
```



In [75]:
```
IQR = -230.750000 + 248.617500
IQR
```

Out[75]: 17.867500000000007

In [76]:
```
# pos_outlier_range = Q3 + 1.5*IQR
pos_outlier_range = -230.750000 + (1.5*IQR)
pos_outlier_range
```

Out[76]: -203.94875

- Holding capping method right now as positive outlier range is -203.94 which is very less and only few data are above this range

Feature scaling

In [77]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
```

In [78]:

```
pd.DataFrame(sc_x).describe()
```

Out[78]:

	0	1	2	3	4	5	
count	2.278450e+05	2.278450e+05	2.278450e+05	2.278450e+05	2.278450e+05	2.278450e+05	2.27845
mean	2.469880e-16	-1.035354e-16	2.169253e-16	3.414173e-16	-1.947214e-16	-2.275907e-16	7.03541
std	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.00000
min	-2.874447e+01	-4.421098e+01	-3.181486e+01	-4.011726e+00	-8.217170e+01	-1.961595e+01	-3.49334
25%	-4.678047e-01	-3.605683e-01	-5.851915e-01	-6.003821e-01	-4.986139e-01	-5.789326e-01	-4.41712
50%	6.105872e-03	4.066939e-02	1.191877e-01	-1.416771e-02	-3.624526e-02	-2.040406e-01	3.14356
75%	6.685615e-01	4.844626e-01	6.787413e-01	5.226069e-01	4.405724e-01	2.983147e-01	4.56467
max	1.249484e+00	1.340918e+01	6.175532e+00	1.164655e+01	2.514117e+01	5.495757e+01	9.67060

Check imbalance dataset

In [79]:

```
y.value_counts()
```

Out[79]:

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

In [80]:

```
fraud_per = 394/(394+227451)*100
fraud_per
```

Out[80]:

```
0.17292457591783889
```

In [81]:

```
x.shape
```

Out[81]:

```
(227845, 29)
```

In [82]:

```
# Since data is imbalance, so we can build model with both aproach
# 1) balance the data and perform model building
# 2) model building with balance the data
```

In [83]:
```
import imblearn
from imblearn.over_sampling import SMOTE
ros = SMOTE()
x_ros, y_ros = ros.fit_resample(sc_x, y)
print("before data is imbalance")
print(y.value_counts())
print()
print("after balancing the data: ")
print(y_ros.value_counts())
```

```
before data is imbalance
Target
0.0      227451
1.0         394
dtype: int64

after balancing the data:
Target
0.0      227451
1.0      227451
dtype: int64
```

Split the data into training and testing for model building

In [84]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_ros, y_ros, test_size
```

Logistic Regression

In [85]:
```
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression()
logit.fit(x_train, y_train)
```

Out[85]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [86]:
```
y_pred_train = logit.predict(x_train)
y_pred_test = logit.predict(x_test)
```

In [87]:
```
from sklearn.metrics import confusion_matrix, classification_report, accura
```

In [88]:
```
print(confusion_matrix(y_train, y_pred_train))
print()
print(confusion_matrix(y_test, y_pred_test))
```

```
[[165950   4677]
 [ 15655 154894]]

[[55239  1585]
 [ 5258 51644]]
```

In [89]:
```
(15477+4647)/(165980+4647+15477+155072)
```

Out[89]:
```
0.05898421928857833
```

In [90]:
```
print(classification_report(y_train, y_pred_train))
print()
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

         0.0       0.91      0.97      0.94    170627
         1.0       0.97      0.91      0.94    170549

    accuracy                           0.94    341176
   macro avg       0.94      0.94      0.94    341176
weighted avg       0.94      0.94      0.94    341176


              precision    recall  f1-score   support

         0.0       0.91      0.97      0.94     56824
         1.0       0.97      0.91      0.94     56902

    accuracy                           0.94    113726
   macro avg       0.94      0.94      0.94    113726
weighted avg       0.94      0.94      0.94    113726
```

In [91]:
```
print(accuracy_score(y_train, y_pred_train))
print()
print(accuracy_score(y_test, y_pred_test))
```

```
0.940406124698103

0.9398290628352356
```

Decision Tree

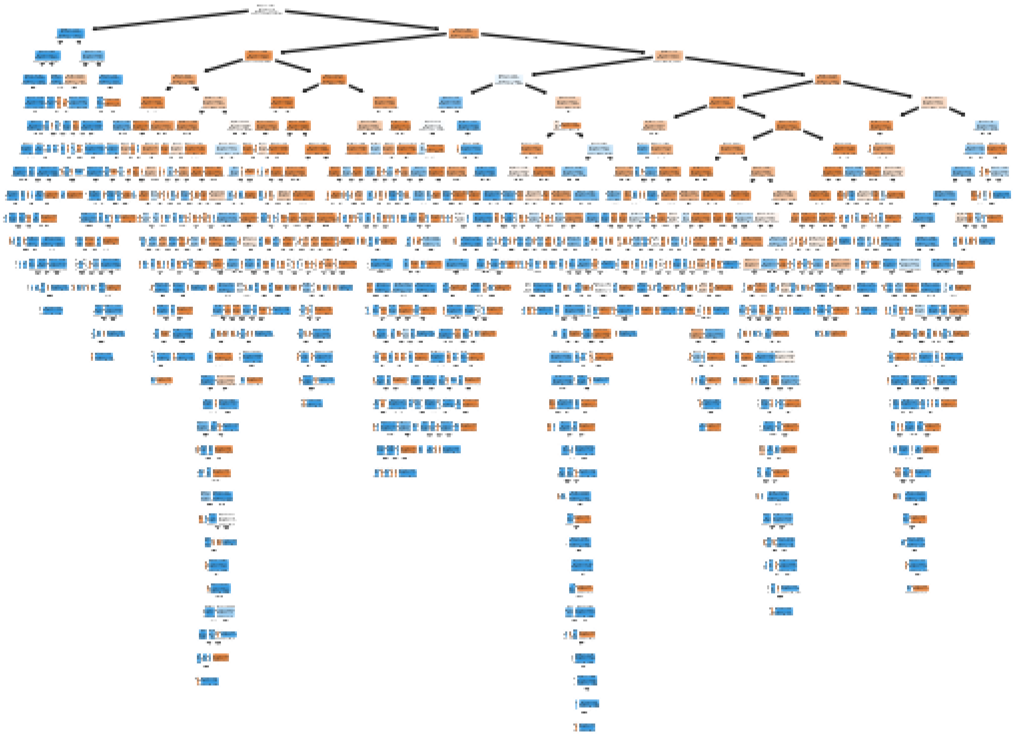
In [92]:
```
from sklearn.tree import DecisionTreeClassifier
dtree= DecisionTreeClassifier(criterion='entropy')
dtree.fit(x_train, y_train)
y_pred_train_dt = dtree.predict(x_train)
y_pred_test_dt = dtree.predict(x_test)
print(accuracy_score(y_train, y_pred_train_dt))
print()
print(accuracy_score(y_test, y_pred_test_dt))
```

```
1.0

0.9984524207305278
```

In [93]:

```
from sklearn import tree
tree.plot_tree(dtree, filled=True)
plt.show()
```



RandomForest Classification

In [94]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100,criterion='entropy')
rf.fit(x_train, y_train)
y_pred_train_rf = rf.predict(x_train)
y_pred_test_rf = rf.predict(x_test)
print(accuracy_score(y_train, y_pred_train_rf))
print()
print(accuracy_score(y_test, y_pred_test_rf))
```

```
1.0

0.9998856901675958
```

In [95]:

```
from xgboost import XGBClassifier
```

XGBoost Classifier

In [96]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(x_train, y_train)
```

Out[96]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [97]:
```
y_pred_train_xg = xgb.predict(x_train)
y_pred_test_xg = xgb.predict(x_test)
print(accuracy_score(y_train, y_pred_train_xg))
print()
print(accuracy_score(y_test, y_pred_test_xg))
```

```
0.9999970689614744

0.9997362080790673
```

Stacking Classifier

In [98]:
```
from mlxtend.classifier import StackingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
```

In [99]:
```
clf1 = LogisticRegression()
clf2 = DecisionTreeClassifier(criterion='entropy')
clf3 = RandomForestClassifier(n_estimators=100,criterion='entropy')
clf4 = XGBClassifier()
clf5 = GaussianNB()
sclf = StackingClassifier(classifiers=[clf2, clf3, clf4, clf5], meta_class
print('3-fold cross validation : \n')
for clf, label in zip([clf2, clf3, clf4, clf5, sclf],['Dtree','RForest','X
    scores = cross_val_score(clf, x_train, y_train, cv=3, scoring='accuracy
    print("Accuracy : %0.2f (+/-%0.2f)[%s]" % (scores.mean(), scores.std()
```

```
3-fold cross validation :

Accuracy : 1.00 (+/-0.00)[Dtree]
Accuracy : 1.00 (+/-0.00)[RForest]
Accuracy : 1.00 (+/-0.00)[XGBoost]
Accuracy : 0.91 (+/-0.00)[Naive_Bayes]
Accuracy : nan (+/-nan)[StackingClassifier]
```

In [100]:
```
# Anomaly Detection Model
# 1) IsolationForest - RF
# 2) LocalOutlierFector -knn
# 3) OneClassSVM - SVM
```

In [101]:
```
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
```

In [102]:
```
classification = {'IsolationForest' : IsolationForest(contamination=outlie
                  "LocalOutlierFactor": LocalOutlierFactor(contamination=ou
                  "OneClassSVM" : OneClassSVM()}
```

In [103]:
```
n_outlier = len(Fraud)
n_outlier
```

Out[103]: 394

In [104]:
```
fraud_percent=len(Fraud)*100/len(x)
print('Percentage of Fraud :',fraud_percent)
```

```
Percentage of Fraud : 0.17292457591783889
```

In [105]:
```
from sklearn.metrics import confusion_matrix, classification_report, accura
```

In [106]:
```
x.shape
```

Out[106]: (227845, 29)

In [107]:
```
y.shape
```

Out[107]: (227845, 1)

In [108]:
```
y.head()
```

Out[108]:

	Target
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

Anamoly Detection:

In [ ]:

```python
for i, (clf_name, clf) in enumerate(classification.items()):
    if clf_name =='LocalOutlierFactor':
        y_pred = clf.fit_predict(x)
        score_prediction = clf.negative_outlier_factor_

    elif clf_name=='OneClassSVM':
        clf.fit(x)
        y_pred = clf.predict(x)

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

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

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

    print("Accuracy Score :")
    print(accuracy_score(y, y_pred))
    print()

    print("Classification Report :")
    print(classification_report(y, y_pred))
```

```
IsolationForest : 188445

Accuracy Score :
0.8284974434374246

Classification Report :
              precision    recall  f1-score   support

         0.0       1.00      0.83      0.91    227451
         1.0       0.01      0.91      0.02       394

    accuracy                           0.83    227845
   macro avg       0.50      0.87      0.46    227845
weighted avg       1.00      0.83      0.90    227845

LocalOutlierFactor : 188445

Accuracy Score :
0.8265926397331519

Classification Report :
              precision    recall  f1-score   support

         0.0       1.00      0.83      0.91    227451
         1.0       0.00      0.36      0.01       394

    accuracy                           0.83    227845
   macro avg       0.50      0.59      0.46    227845
weighted avg       1.00      0.83      0.90    227845
```

In [ ]:

```
1
```