Fraud Detection

ATM fraud detection is of paramount importance in the banking sector for the following reasons:

1. Financial Loss Prevention: ATM fraud can result in significant financial losses for both banks and customers. Detecting and preventing fraudulent activities helps minimize these losses, ensuring the financial stability of the banking institution and the trust of its customers.
2. Customer Trust and Confidence: Effective ATM fraud detection mechanisms reassure customers that their funds and personal information are secure. Maintaining trust is essential for retaining existing customers and attracting new ones.
3. Reputation Management: A bank's reputation is a critical asset. Incidents of ATM fraud can damage a bank's reputation and lead to a loss of customers. Robust fraud detection and prevention measures demonstrate a commitment to security and can help safeguard a bank's reputation.
4. Operational Efficiency: ATM fraud can lead to operational disruptions, including customer service inquiries, investigations, and card replacements. Efficient fraud detection reduces the workload associated with managing fraudulent transactions.
5. Technological Advancements: As technology evolves, so do the tactics used by fraudsters. Continuous investment in fraud detection systems is essential to stay ahead of emerging threats.

Hence, Fraud detection is a critical component of the banking sector's security framework. It safeguards financial institutions, their customers, and their reputations, while also ensuring regulatory compliance and operational efficiency.

About Dataset

Building a Fraudelent prediction model, dataset used is provided by an Australian bank, whose profitability and reputation are being hit by fraudulent ATM transactions. They want help in reducing and if possible completely eliminating such fraudulent transactions. This can be done by building a predictive model to catch such fraudulent transactions in real time and decline them. 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. Training data contains masked variables pertaining to each transaction id . The prediction target here is 'Target'.

1: Fraudulent transactions

0: Clean transactions

Features

The dataset contains masked variable for both train and test data, to protect the confidentiality.

In [1]:
```
# Importing required Libraries
import os, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
sns.set()
pd.set_option('display.max_rows',None)
pd.set_option('display.max_columns',None)
sns.set_style('whitegrid')
```

In [2]:
```
# Checking the version of python used as here, EDA is done using Dataprep package and is compatible with 3.10

import sys
print(sys.version)
```

```
3.10.13 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:24:38) [MSC v.1916 64 bit (AMD64)]
```

In [3]:
```
# Loading the datasets

geo_df = pd.read_csv('Geo_scores.csv')
lambda_wts_df = pd.read_csv('Lambda_wts.csv')
qset_df = pd.read_csv('Qset_tats.csv')
instances_df = pd.read_csv('instance_scores.csv')
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test_share.csv')
```

In [4]:
```
# Columns present in the above datasets

print(geo_df.columns)
print('--'*50)
print('\n')
print(lambda_wts_df.columns)
print('--'*50)
print('\n')
print(qset_df.columns)
print('--'*50)
print('\n')
print(instances_df.columns)
print('--'*50)
print('\n')
print(train_df.columns)
print('--'*50)
print('\n')
print(test_df.columns)
print('--'*50)
```

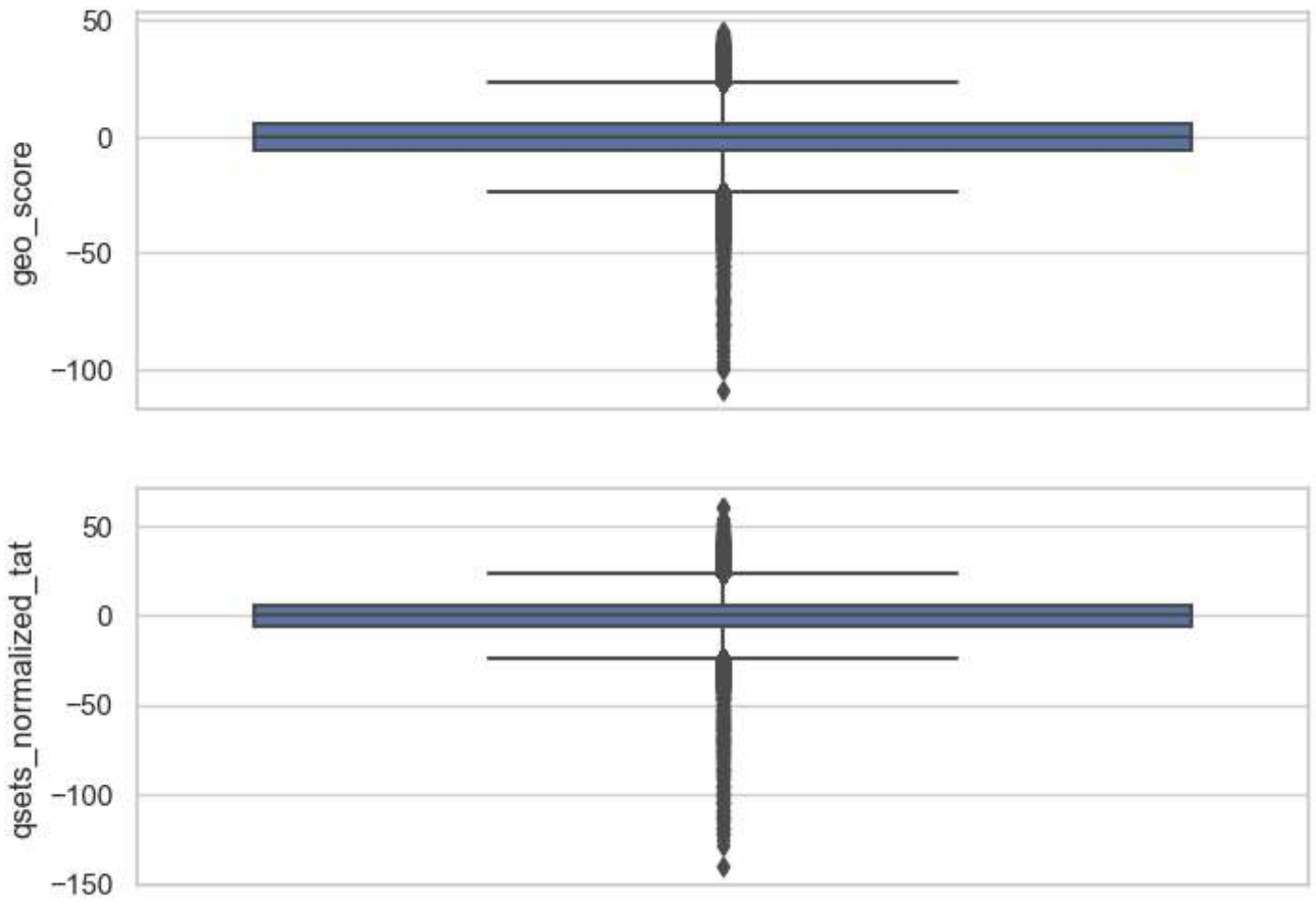
```
Index(['id', 'geo_score'], dtype='object')
----------------------------------------------------------------------------------------------------


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


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


Index(['id', 'instance_scores'], 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')
----------------------------------------------------------------------------------------------------


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')
----------------------------------------------------------------------------------------------------
```

Now since the data is distributed among different tables, the task is to group all the data into one dataframe which can aid in data preprocesing, analysis and model building.

In [5]:
```
# To do this first the missing values are to be handled, missing values are handled individually to avoid the
# leakage.

print(geo_df.isnull().sum())
print('--'*50)
print('\n')
print(lambda_wts_df.isnull().sum())
print('--'*50)
print('\n')
print(qset_df.isnull().sum())
print('--'*50)
print('\n')
print(instances_df.isnull().sum())
print('--'*50)
print('\n')
print(train_df.isnull().sum())
print('--'*50)
print('\n')
print(test_df.isnull().sum())
print('--'*50)
```

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


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


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


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

In [6]:
```
# There are missing values present in only geo_df and qset_df.
# Checking the box plot to check the presence of outliers, if there are outliers present going with median im

plt.figure(figsize=(8,6))
plt.subplot(2,1,1)
sns.boxplot(data=geo_df, y = 'geo_score')
plt.subplot(2,1,2)
sns.boxplot(data=qset_df, y = 'qsets_normalized_tat')
plt.show()
```



In [7]:
```
# SInce there is the presence of outliers going with median imputation

geo_df['geo_score'] = geo_df['geo_score'].fillna(geo_df['geo_score'].median())
qset_df['qsets_normalized_tat'] = qset_df['qsets_normalized_tat'].fillna(qset_df['qsets_normalized_tat'].medi
```

In [8]:
```
# Confirming that the missing values are handled:

print(geo_df.isnull().sum())
print('--'*50)
print('\n')
print(qset_df.isnull().sum())
print('--'*50)
```

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


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

In [9]:
```
# Now checking if there are any duplicate values in all the tables:

print(geo_df.duplicated().sum())
print('--'*50)
print('\n')
print(lambda_wts_df.duplicated().sum())
print('--'*50)
print('\n')
print(qset_df.duplicated().sum())
print('--'*50)
print('\n')
print(instances_df.duplicated().sum())
print('--'*50)
print('\n')
print(train_df.duplicated().sum())
print('--'*50)
print('\n')
print(test_df.duplicated().sum())
print('--'*50)
```

```
55349
----------------------------------------------------------------------------------------------------


0
----------------------------------------------------------------------------------------------------


59314
----------------------------------------------------------------------------------------------------


33600
----------------------------------------------------------------------------------------------------


0
----------------------------------------------------------------------------------------------------


0
----------------------------------------------------------------------------------------------------
```

There are duplicates present in geo_df, qset_df and instances_df, however the same will not be ommited, this is because it is possible and mostly obvious that the same customer with a unique customer ID has made a transaction at the same geographic location, same goes with the time and the instance score.

In [10]:
```
# Now merging the tables:

print(geo_df['id'].nunique())
print('--'*50)
print('\n')
print(lambda_wts_df['Group'].nunique())
print('--'*50)
print('\n')
print(qset_df['id'].nunique())
print('--'*50)
print('\n')
print(instances_df['id'].nunique())
print('--'*50)
print('\n')
print(train_df['id'].nunique())
print('--'*50)
print('\n')
print(train_df['Group'].nunique())
print('--'*50)
print('\n')
print(test_df['id'].nunique())
print('--'*50)
print('\n')
print(test_df['Group'].nunique())
print('--'*50)
```

```
284807
----------------------------------------------------------------------------------------------------


1400
----------------------------------------------------------------------------------------------------


284807
----------------------------------------------------------------------------------------------------


284807
----------------------------------------------------------------------------------------------------


227845
----------------------------------------------------------------------------------------------------


1301
----------------------------------------------------------------------------------------------------


56962
----------------------------------------------------------------------------------------------------


915
----------------------------------------------------------------------------------------------------
```

Now we can observe that the sum of train data unique values of id and test data unique values of id add up to 284807, which is same as that of geo_df, qset_df and instances_df

In [11]:
```
227845+56962
```

Out[11]:
```
284807
```

In [12]:
```
# To mark the train and test datasets in the concatenated df lets take a new column data in both train and te
# train and test

train_df['Data'] = 'train'
test_df['Data'] = 'test'
```

In [13]:
```
# Now merging train and test tables

full_df = pd.concat([train_df, test_df])
```

In [14]:
```
# Checking the head of the full_df

full_df.head()
```

Out[14]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	D
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.460000	0.643333	0.736
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000	0.613333	0.883
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000	0.730000	0.873
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667	0.466667	0.670
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667	0.683333	0.296

In [15]:
```
# Checking the tail of the full_df

full_df.tail()
```

Out[15]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2
56957	18333	Grp102	0.553333	1.043333	1.096667	0.686667	0.673333	0.340000	0.900000	0.643333	0.543333	0.586667	0.660000
56958	244207	Grp504	1.353333	0.616667	0.276667	0.783333	0.690000	0.650000	0.473333	0.670000	1.023333	0.293333	0.940000
56959	103277	Grp78	1.083333	0.433333	0.806667	0.490000	0.243333	0.316667	0.533333	0.606667	0.420000	0.480000	0.490000
56960	273294	Grp134	0.566667	1.153333	0.370000	0.616667	0.793333	0.226667	0.910000	0.696667	0.600000	0.500000	0.896667
56961	223337	Grp18	1.426667	0.110000	-0.006667	-0.200000	0.983333	1.870000	0.033333	0.963333	0.266667	0.586667	0.793333

In [16]:
```
# Now to merge all other tables with the full_df:

print(full_df['id'].nunique())
print('--'*50)
print(geo_df['id'].nunique())
print('--'*50)
print(full_df.shape[0])
print('--'*50)
print(geo_df.shape[0])
```

```
284807
----------------------------------------------------------------------------------------------------
284807
----------------------------------------------------------------------------------------------------
284807
----------------------------------------------------------------------------------------------------
1424035
```

In [17]:
```
# Although the unique id values are same, the number of observations are different, hence first need to
# group the geo_df by id, grouping the observations by the average value to retain the center value of the gr

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

In [18]:
```
geo_df.shape[0]
```

Out[18]:
```
284807
```

In [19]:
```
# Now merging both geo_df and full_df

full_df = pd.merge(full_df, geo_df, how='left', on = 'id')
```

In [20]:
```
# Checking if the geo_df has been added to full_df

full_df.head()
```

Out[20]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	D
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.460000	0.643333	0.736
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000	0.613333	0.883
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000	0.730000	0.873
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667	0.466667	0.670
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667	0.683333	0.296

In [21]:
```
# Similarly for instances_df

print(full_df.shape[0])
print('--'*50)
print(instances_df.shape[0])
```

```
284807
----------------------------------------------------------------------------------------------------
1424035
```

In [22]:
```
instances_df = instances_df.groupby('id').mean()
print(instances_df.shape[0])
full_df = pd.merge(full_df, instances_df, how='left', on='id')
```

```
284807
```

In [23]:
```
# Checking if the instances_df has been added to full_df

full_df.head()
```

Out[23]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	D
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.460000	0.643333	0.736
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000	0.613333	0.883
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000	0.730000	0.873
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667	0.466667	0.670
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667	0.683333	0.296

In [24]:
```
# Similarly for qset_df

print(full_df.shape[0])
print('--'*50)
print(qset_df.shape[0])
```

```
284807
----------------------------------------------------------------------------------------------------
1424035
```

In [25]:
```
qset_df = qset_df.groupby('id').mean()
full_df = pd.merge(full_df, qset_df, how='left', on='id')
```

In [26]:
```
# Checking if the instances_df has been added to full_df

full_df.head()
```

Out[26]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	D
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.460000	0.643333	0.736
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000	0.613333	0.883
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000	0.730000	0.873
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667	0.466667	0.670
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667	0.683333	0.296

In [27]:
```
# Similarly for lambda_wts_df

print(full_df['Group'].nunique())
print('--'*50)
print(lambda_wts_df.shape[0])
```

```
1400
----------------------------------------------------------------------------------------------------
1400
```

In [28]:
```
# Checking if the instances_df has been added to full_df

full_df = pd.merge(full_df, lambda_wts_df, how='left', on='Group')
```

In [29]:
```
full_df.head()
```

Out[29]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	D
0	112751	Grp169	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.460000	0.643333	0.736
1	18495	Grp161	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000	0.613333	0.883
2	23915	Grp261	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000	0.730000	0.873
3	50806	Grp198	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667	0.466667	0.670
4	184244	Grp228	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667	0.683333	0.296

In [30]:
```
# Now separating the train and test datasets again:

train_df = full_df[full_df['Data'] == 'train']
test_df = full_df[full_df['Data'] == 'test']
```

In [31]:
```
train_df.shape
```

Out[31]: (227845, 33)

In [32]:
```
test_df.shape
```

Out[32]: (56962, 33)

In [33]:
```
from dataprep.eda import create_report
train_eda_report = create_report(train_df, title='Train Data Report')
```

```
  0%|                                                                  | 0/7282 [00:0
0<…
```

In [34]:
```
train_eda_report
```

Out[34]:

Train Data Report | Overview | Variables ≡ | Interactions | Correlations | Missing Values

Overview

Dataset Statistics

Number of Variables	33
Number of Rows	227845
Missing Cells	0
Missing Cells (%)	0.0%
Duplicate Rows	0
Duplicate Rows (%)	0.0%
Total Size in Memory	82.7 MB
Average Row Size in Memory	380.5 B
Variable Types	Numerical: 30 Categorical: 3

Dataset Insights

`id` is uniformly distributed	Uniform
`Per4` and `Dem1` have similar distributions	Similar Distribution
`Per5` and `Per7` have similar distributions	Similar Distribution
`Per5` and `Per9` have similar distributions	Similar Distribution
`Per5` and `Dem1` have similar distributions	Similar Distribution
`Per5` and `Dem2` have similar distributions	Similar Distribution
`Per7` and `Dem2` have similar distributions	Similar Distribution
`Per7` and `Dem3` have similar distributions	Similar Distribution
`Per7` and `Dem4` have similar distributions	Similar Distribution
`Per7` and `Dem5` have similar distributions	Similar Distribution

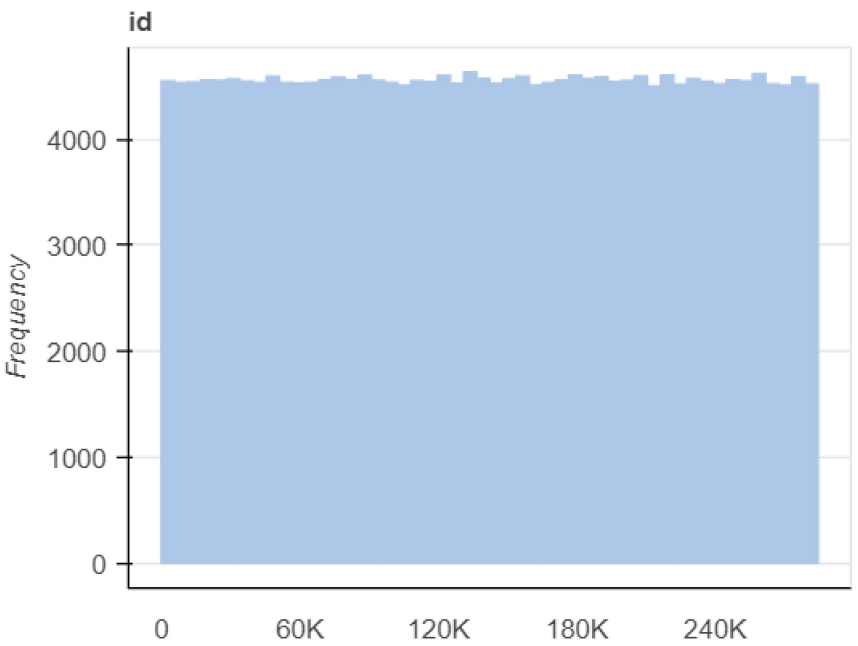
1 2 3 4 5 6 7

Variables

Sort by Feature order ☐ Reverse order

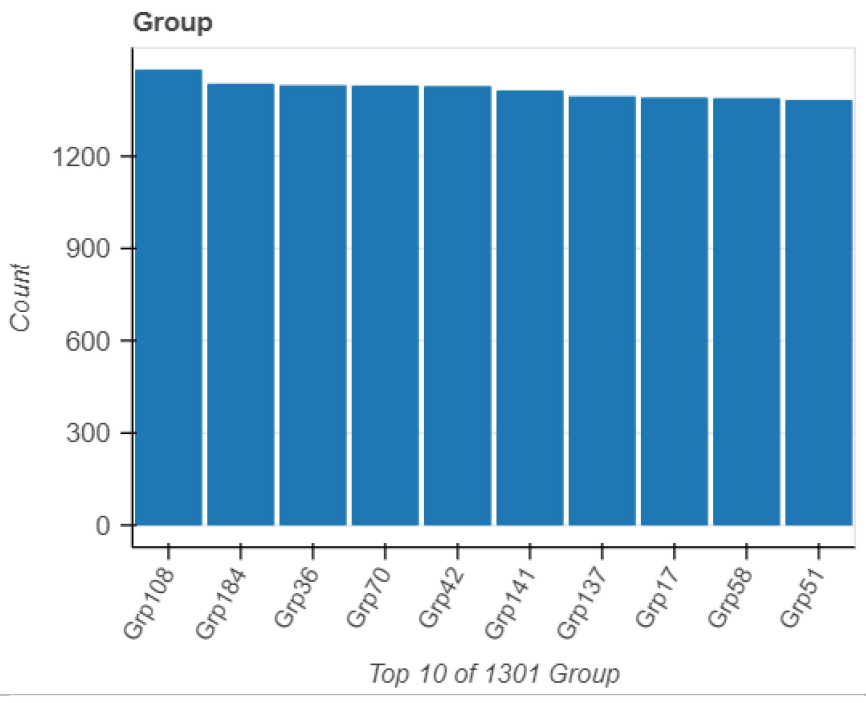
id
numerical
Show Details

Approximate Distinct Count	227845	Memory Size	3645520
Approximate Unique (%)	100.0%	Mean	142404.0
Missing	0	Minimum	0
Missing (%)	0.0%	Maximum	284805
Infinite	0	Zeros	1
Infinite (%)	0.0%	Zeros (%)	0.0%
		Negatives	0
		Negatives (%)	0.0%



Group
categorical
Show Details

Approximate Distinct Count	1301
Approximate Unique (%)	0.6%
Missing	0
Missing (%)	0.0%
Memory Size	16074208



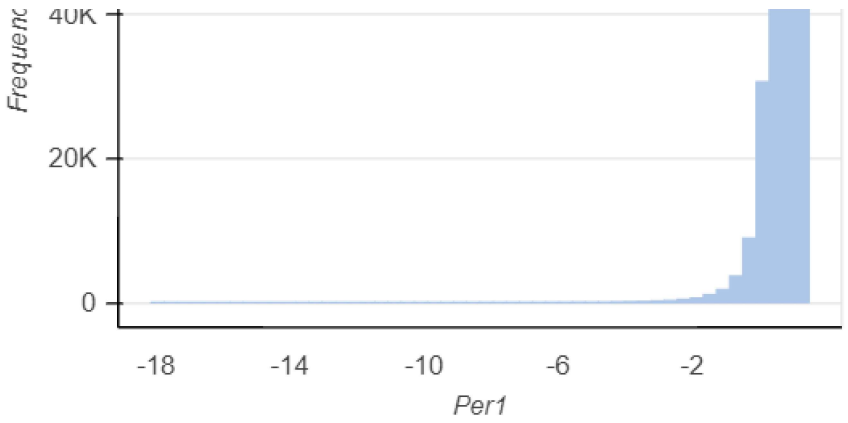
Approximate Distinct Count	1778	Approximate Unique (%)	0.8%
		Missing	0

Per1
numerical

Show Details

Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.666

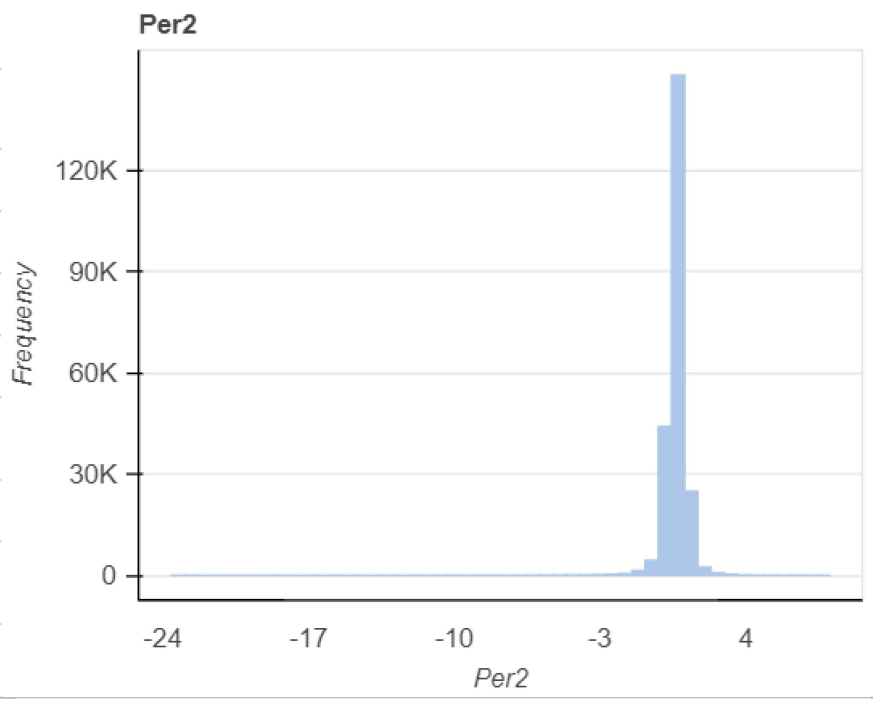
Minimum	-18.1367
Maximum	1.4833
Zeros	190
Zeros (%)	0.1%
Negatives	20867
Negatives (%)	9.2%



Per2
numerical

Show Details

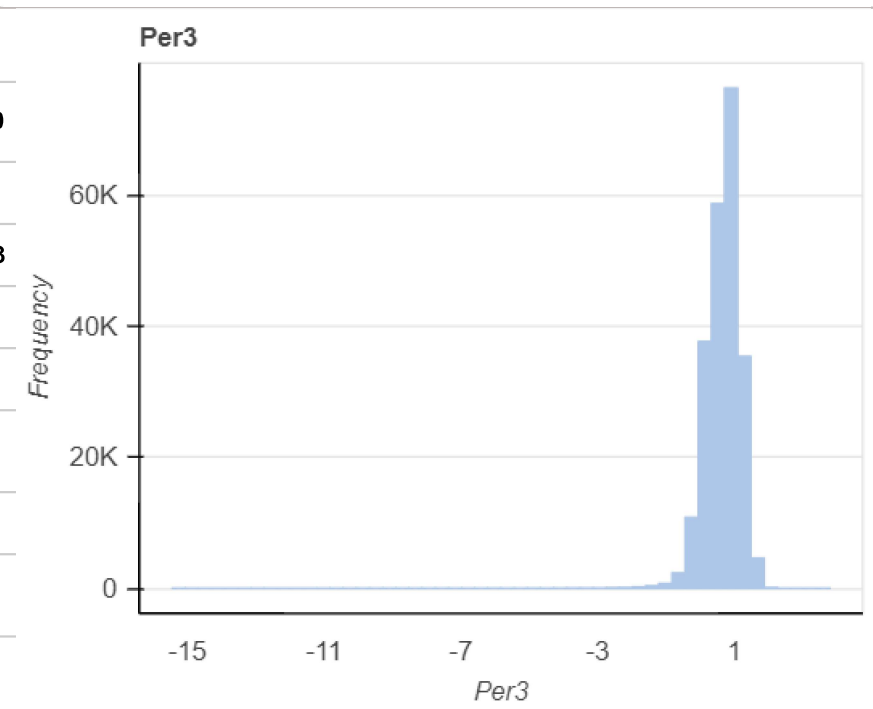
Approximate Distinct Count	2319
Approximate Unique (%)	1.0%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.6677
Minimum	-23.5733
Maximum	8.02
Zeros	98
Zeros (%)	0.0%
Negatives	11024
Negatives (%)	4.8%



Per3
numerical

Show Details

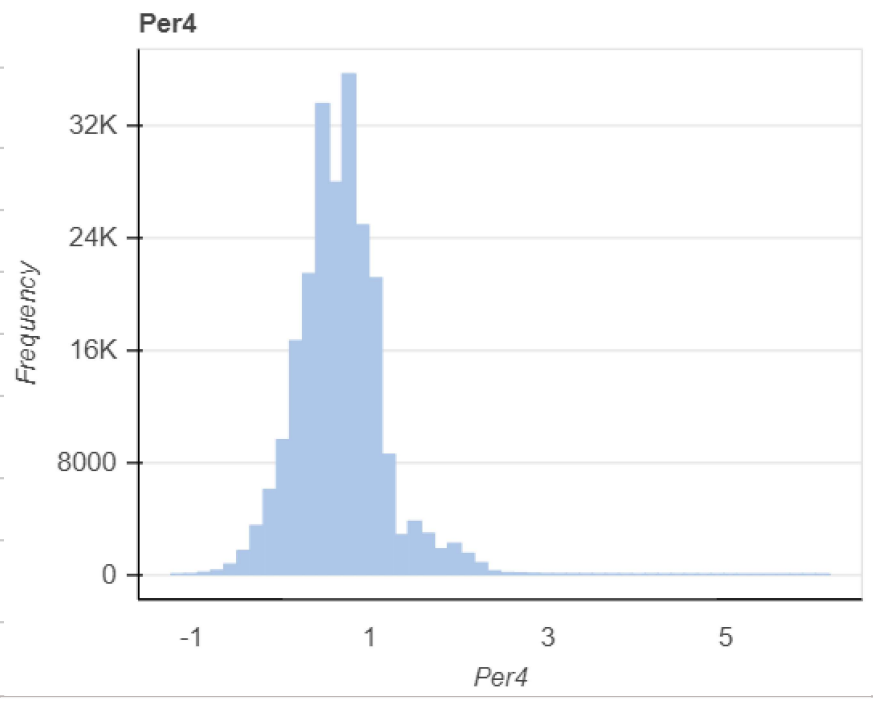
Approximate Distinct Count	1469
Approximate Unique (%)	0.6%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.6663
Minimum	-15.4433
Maximum	3.7933
Zeros	178
Zeros (%)	0.1%
Negatives	17921
Negatives (%)	7.9%



Per4
numerical

Show Details

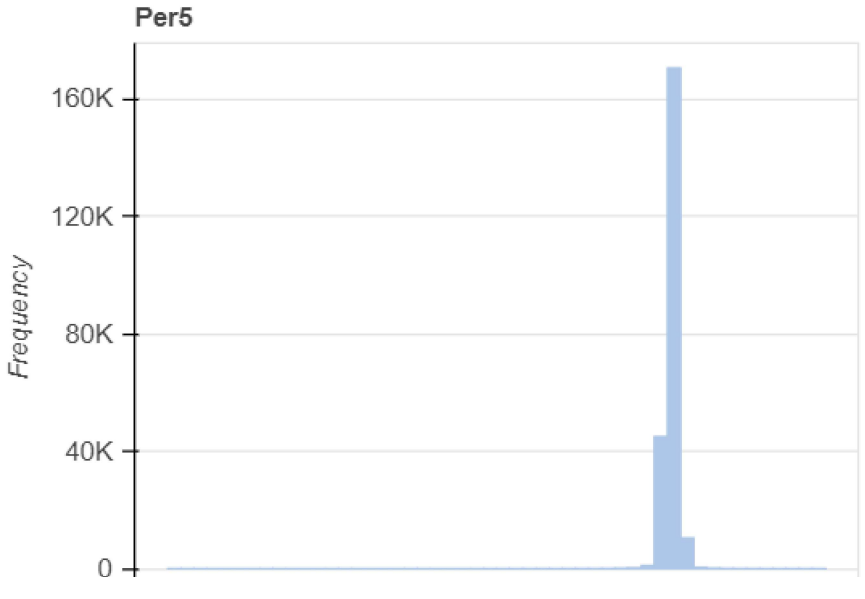
Approximate Distinct Count	1292
Approximate Unique (%)	0.6%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.6667
Minimum	-1.2267
Maximum	6.1633
Zeros	173
Zeros (%)	0.1%
Negatives	14208
Negatives (%)	6.2%



Per5
numerical

Show Details

Approximate Distinct Count	1881
Approximate Unique (%)	0.8%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.6667
Minimum	-37.2467
Maximum	12.2667
Zeros	108
Zeros (%)	0.0%
Negatives	6813

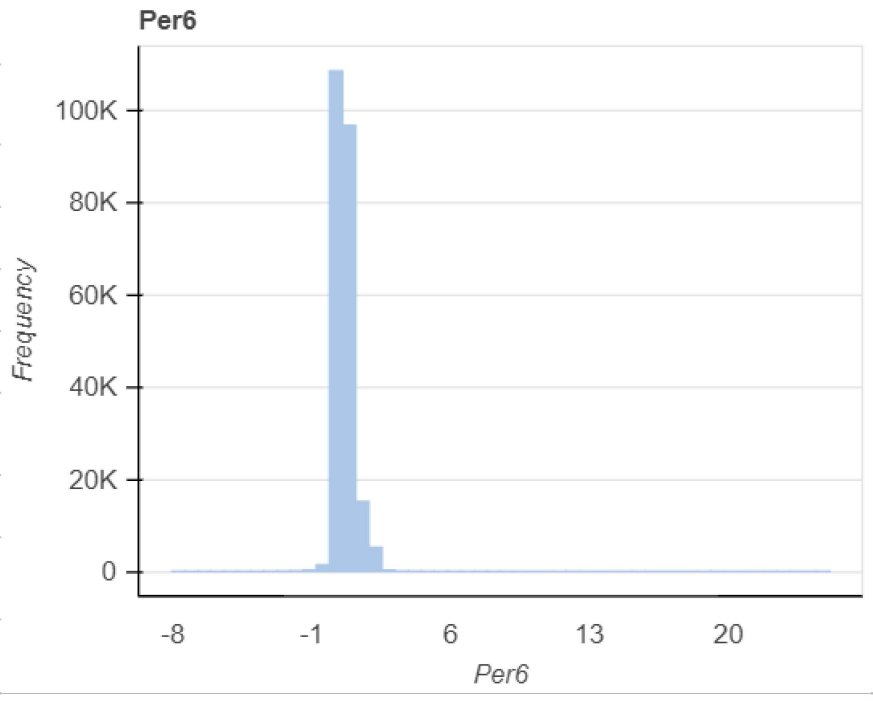


Negatives (%)	3.0%

Per6
numerical
Show Details

Approximate Distinct Count	1477
Approximate Unique (%)	0.6%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

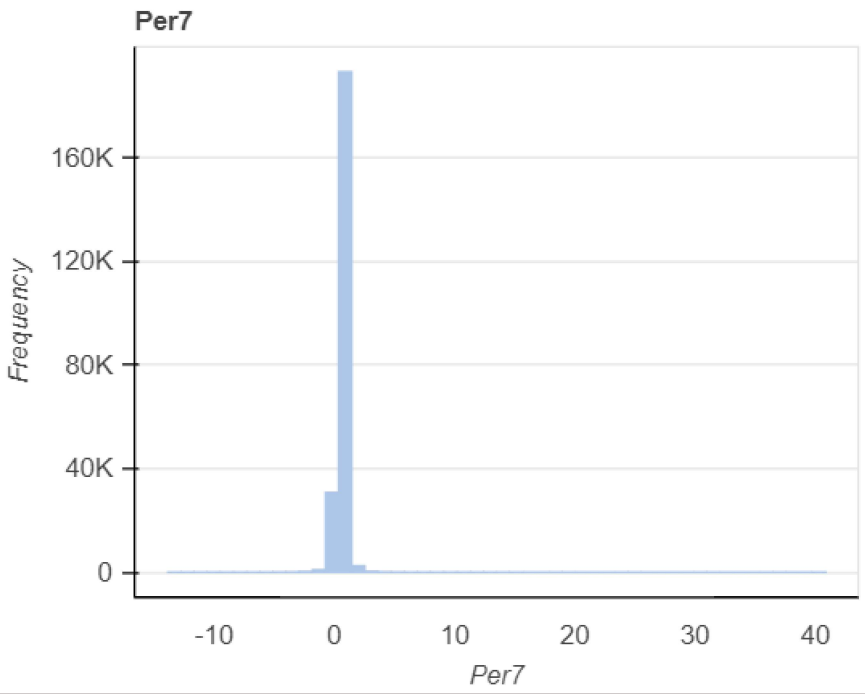
Memory Size	3645520
Mean	0.6674
Minimum	-8.0533
Maximum	25.1
Zeros	65
Zeros (%)	0.0%
Negatives	2740
Negatives (%)	1.2%



Per7
numerical
Show Details

Approximate Distinct Count	1901
Approximate Unique (%)	0.8%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

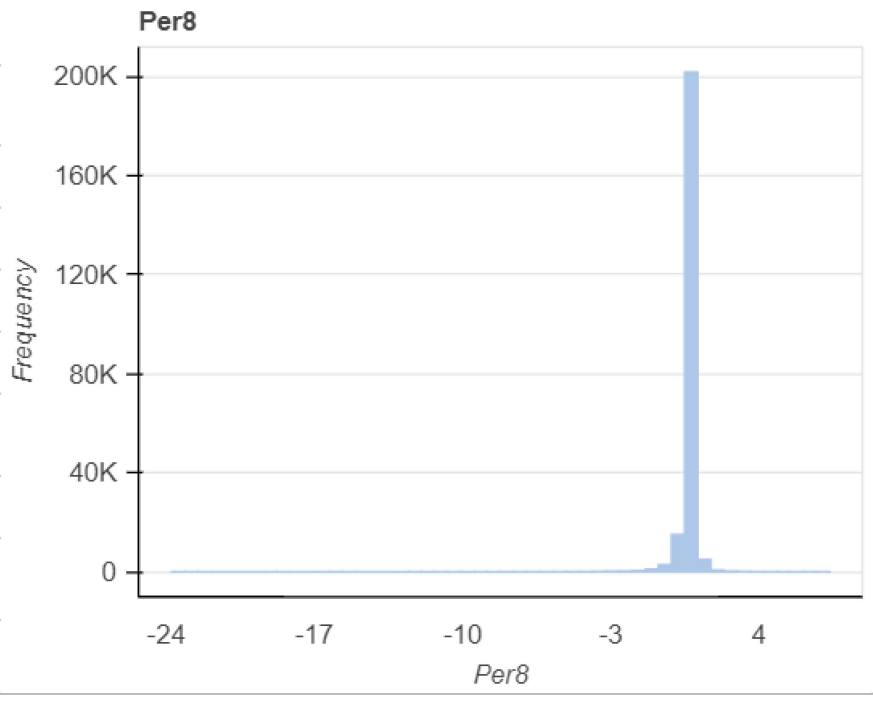
Memory Size	3645520
Mean	0.6669
Minimum	-13.8533
Maximum	40.8633
Zeros	66
Zeros (%)	0.0%
Negatives	4835
Negatives (%)	2.1%



Per8
numerical
Show Details

Approximate Distinct Count	2047
Approximate Unique (%)	0.9%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

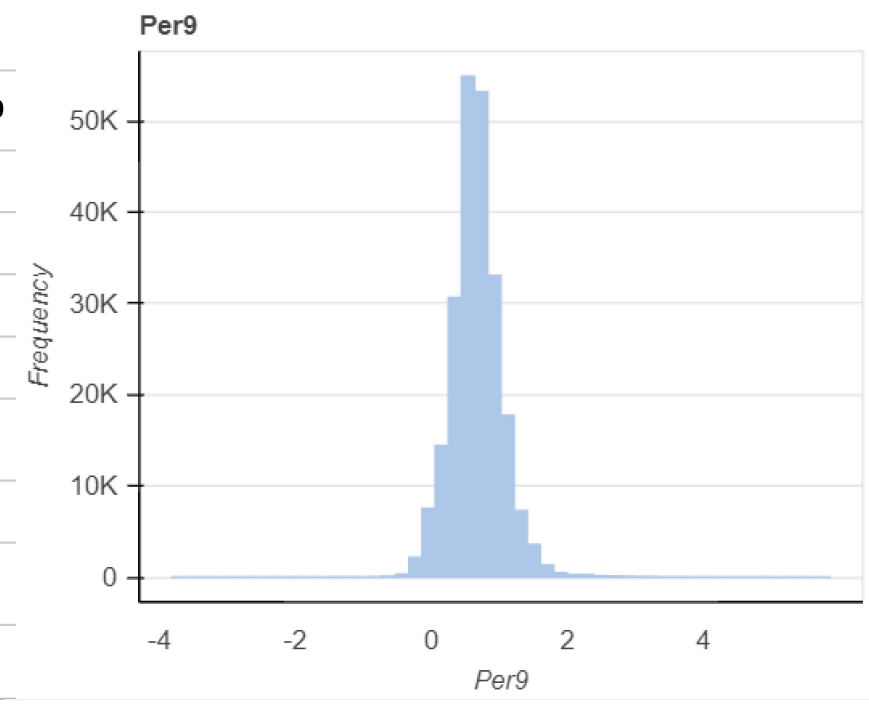
Memory Size	3645520
Mean	0.6663
Minimum	-23.74
Maximum	7.3367
Zeros	19
Zeros (%)	0.0%
Negatives	6025
Negatives (%)	2.6%



Per9
numerical
Show Details

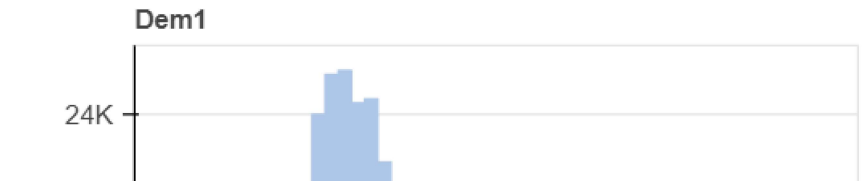
Approximate Distinct Count	1203
Approximate Unique (%)	0.5%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Memory Size	3645520
Mean	0.6667
Minimum	-3.81
Maximum	5.8633
Zeros	143
Zeros (%)	0.1%
Negatives	7145
Negatives (%)	3.1%



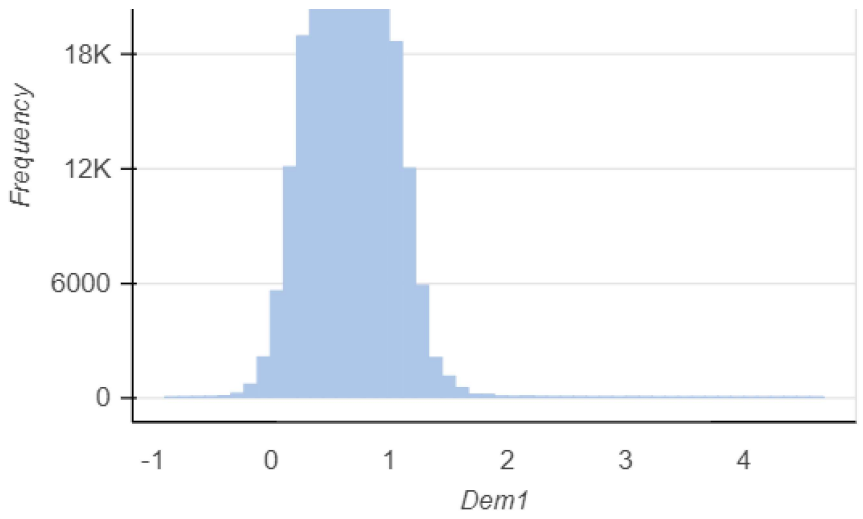
Approximate Distinct Count	910

Approximate Unique (%)	0.4%



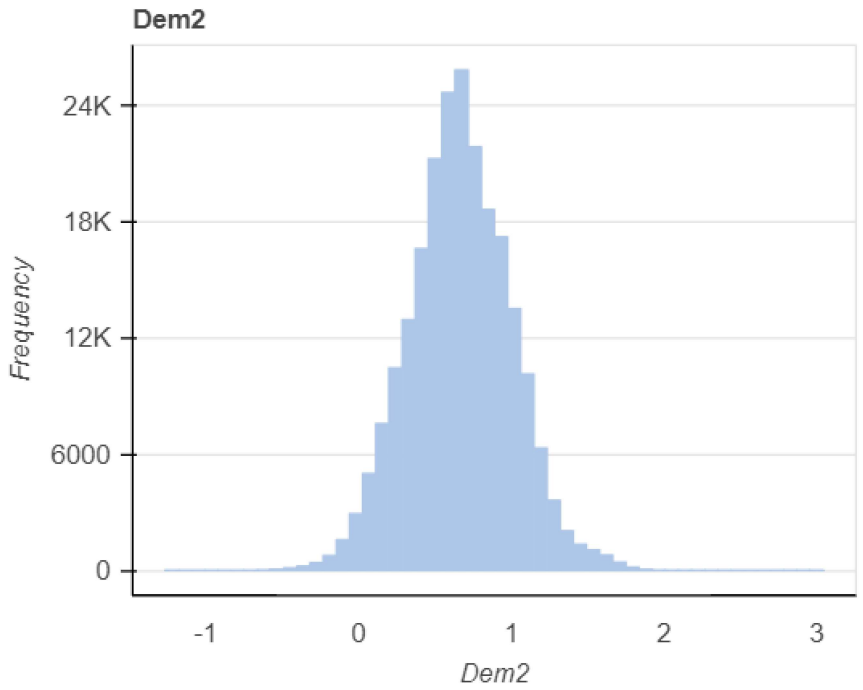
Dem1
numerical
Show Details

Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.6666
Minimum	-0.8933
Maximum	4.6733
Zeros	88
Zeros (%)	0.0%
Negatives	3049
Negatives (%)	1.3%



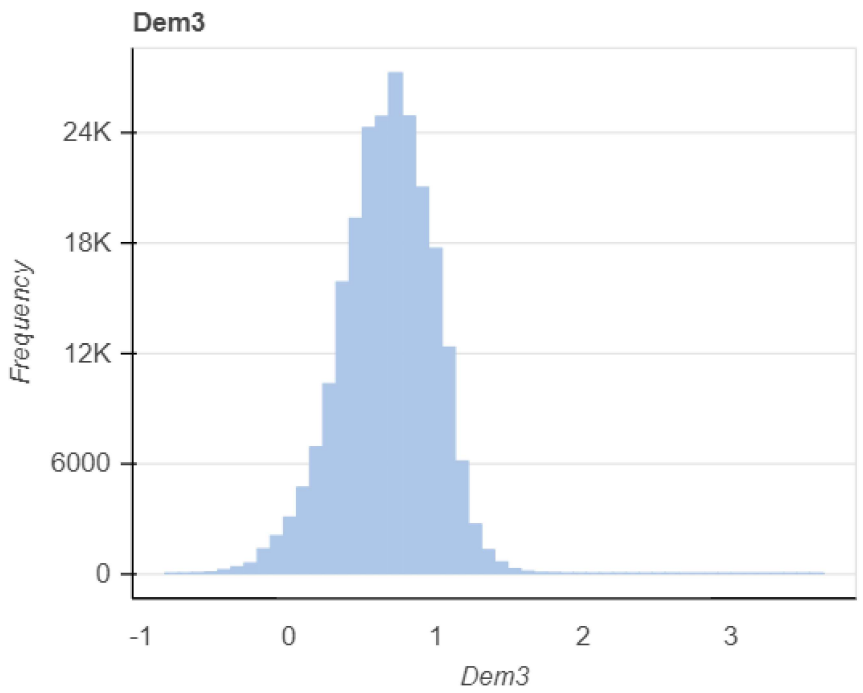
Dem2
numerical
Show Details

Approximate Distinct Count	762
Approximate Unique (%)	0.3%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.6663
Minimum	-1.2633
Maximum	3.0433
Zeros	130
Zeros (%)	0.1%
Negatives	4778
Negatives (%)	2.1%



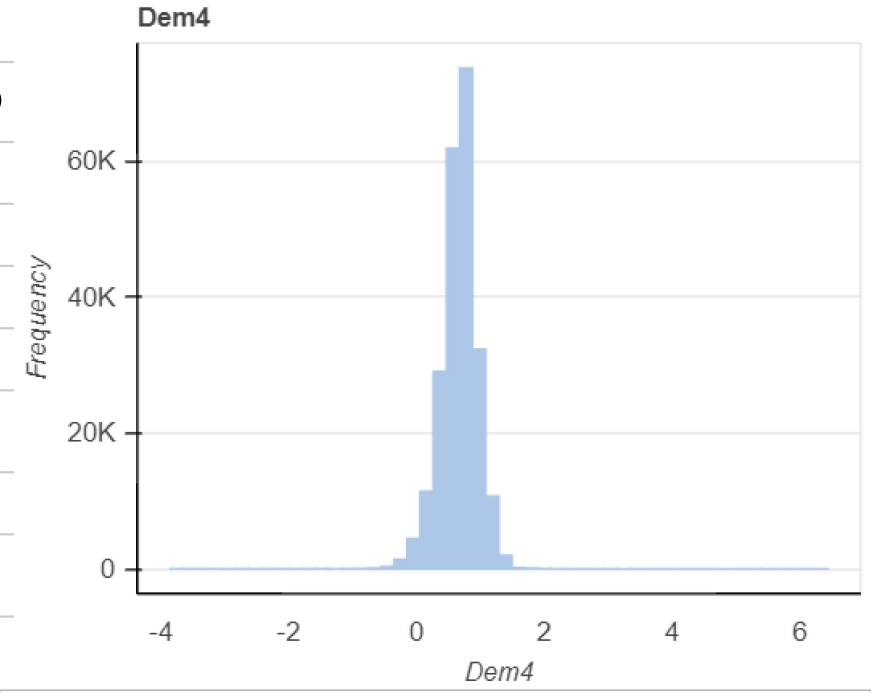
Dem3
numerical
Show Details

Approximate Distinct Count	747
Approximate Unique (%)	0.3%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.6664
Minimum	-0.8333
Maximum	3.6267
Zeros	104
Zeros (%)	0.0%
Negatives	5397
Negatives (%)	2.4%



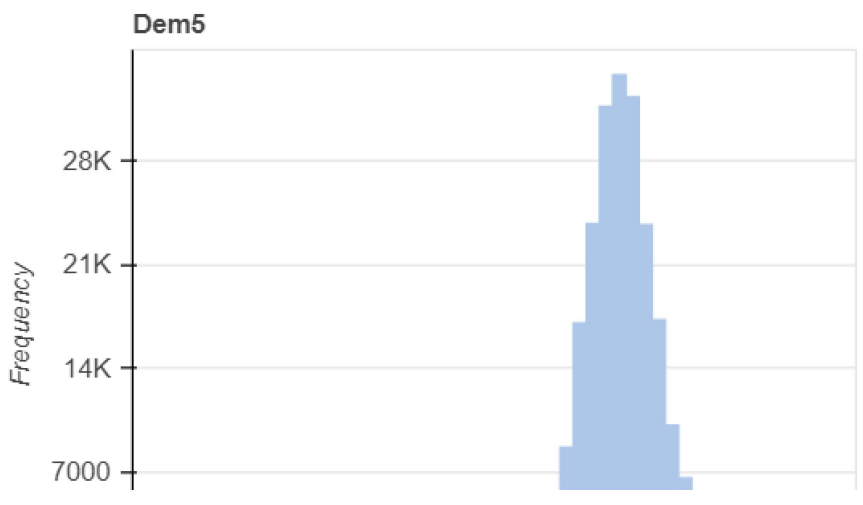
Dem4
numerical
Show Details

Approximate Distinct Count	983
Approximate Unique (%)	0.4%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.6669
Minimum	-3.8533
Maximum	6.44
Zeros	88
Zeros (%)	0.0%
Negatives	4755
Negatives (%)	2.1%

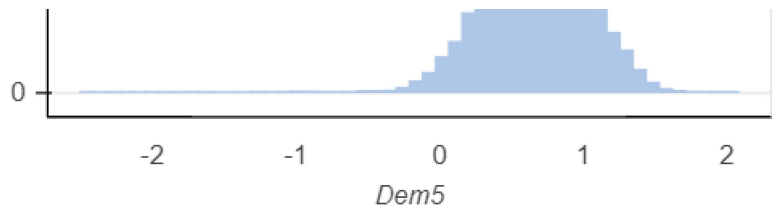


Dem5
numerical
Show Details

Approximate Distinct Count	838
Approximate Unique (%)	0.4%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	3645520
Mean	0.6665
Minimum	-2.5
Maximum	2.08

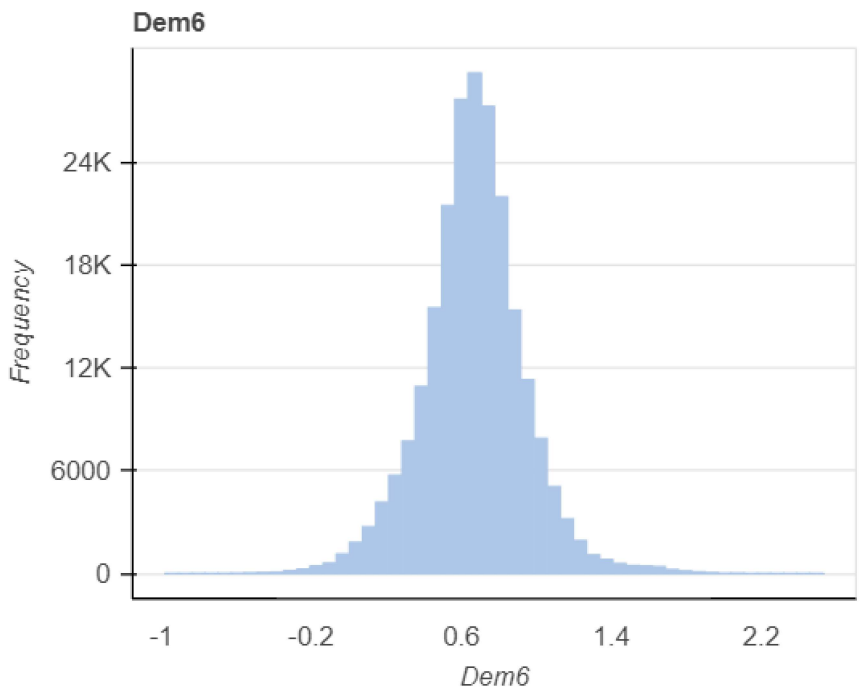


Zeros	73	Negatives	3180
Zeros (%)	0.0%	Negatives (%)	1.4%



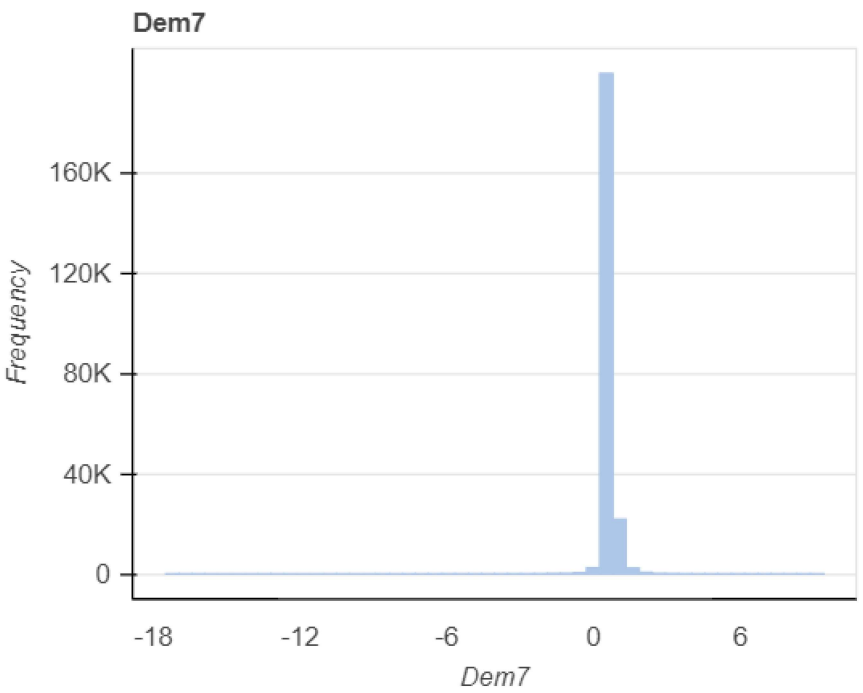
Dem6
numerical
Show Details

Approximate Distinct Count	817	Memory Size	3645520
Approximate Unique (%)	0.4%	Mean	0.6667
Missing	0	Minimum	-0.9767
Missing (%)	0.0%	Maximum	2.53
Infinite	0	Zeros	68
Infinite (%)	0.0%	Zeros (%)	0.0%
		Negatives	2645
		Negatives (%)	1.2%



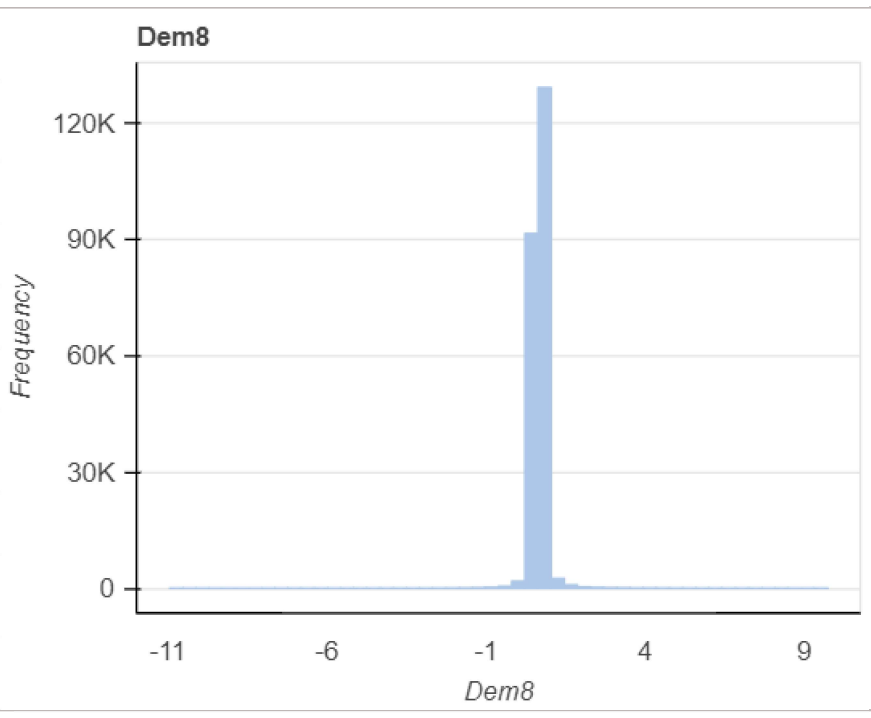
Dem7
numerical
Show Details

Approximate Distinct Count	1505	Memory Size	3645520
Approximate Unique (%)	0.7%	Mean	0.6664
Missing	0	Minimum	-17.5
Missing (%)	0.0%	Maximum	9.4133
Infinite	0	Zeros	12
Infinite (%)	0.0%	Zeros (%)	0.0%
		Negatives	1560
		Negatives (%)	0.7%



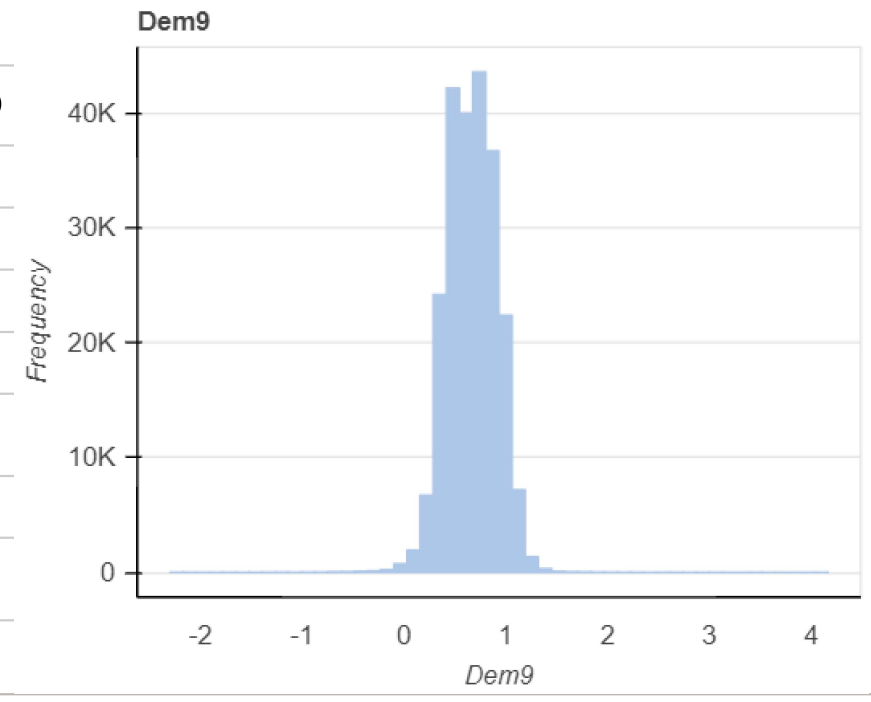
Dem8
numerical
Show Details

Approximate Distinct Count	1537	Memory Size	3645520
Approximate Unique (%)	0.7%	Mean	0.6665
Missing	0	Minimum	-10.9433
Missing (%)	0.0%	Maximum	9.7333
Infinite	0	Zeros	8
Infinite (%)	0.0%	Zeros (%)	0.0%
		Negatives	1427
		Negatives (%)	0.6%



Dem9
numerical
Show Details

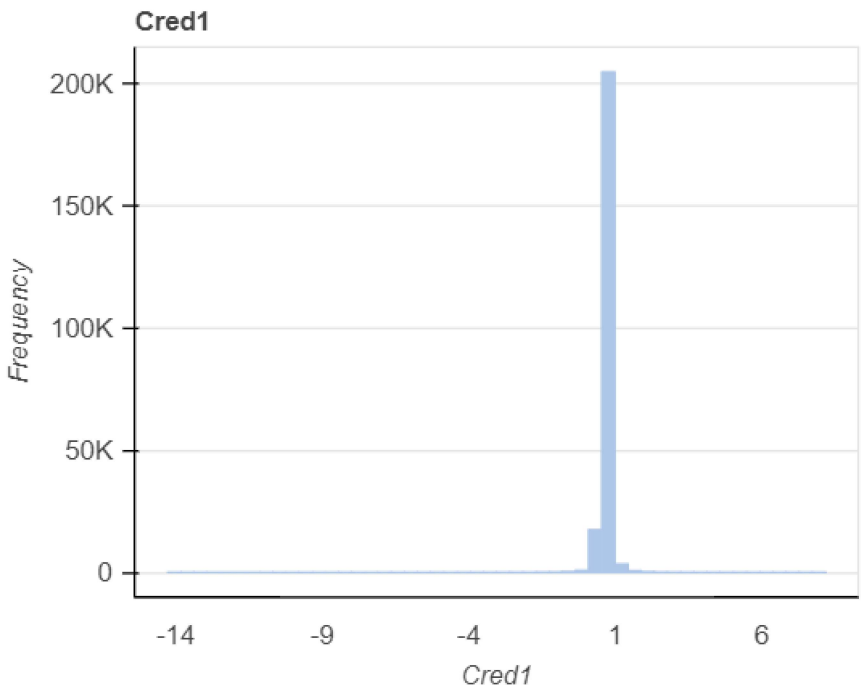
Approximate Distinct Count	812	Memory Size	3645520
Approximate Unique (%)	0.4%	Mean	0.6665
Missing	0	Minimum	-2.2967
Missing (%)	0.0%	Maximum	4.1667
Infinite	0	Zeros	24
Infinite (%)	0.0%	Zeros (%)	0.0%
		Negatives	1036
		Negatives (%)	0.5%



Cred1
numerical
Show Details

Approximate Distinct Count	1307
Approximate Unique (%)	0.6%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

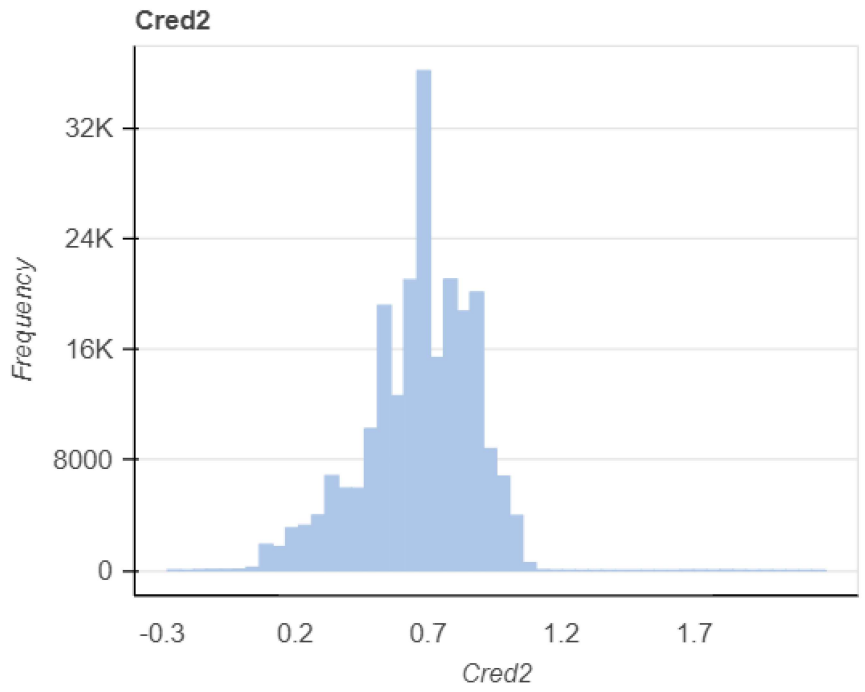
Memory Size	3645520
Mean	0.6667
Minimum	-14.27
Maximum	8.1767
Zeros	8
Zeros (%)	0.0%
Negatives	1032
Negatives (%)	0.5%



Cred2
numerical
Show Details

Approximate Distinct Count	494
Approximate Unique (%)	0.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

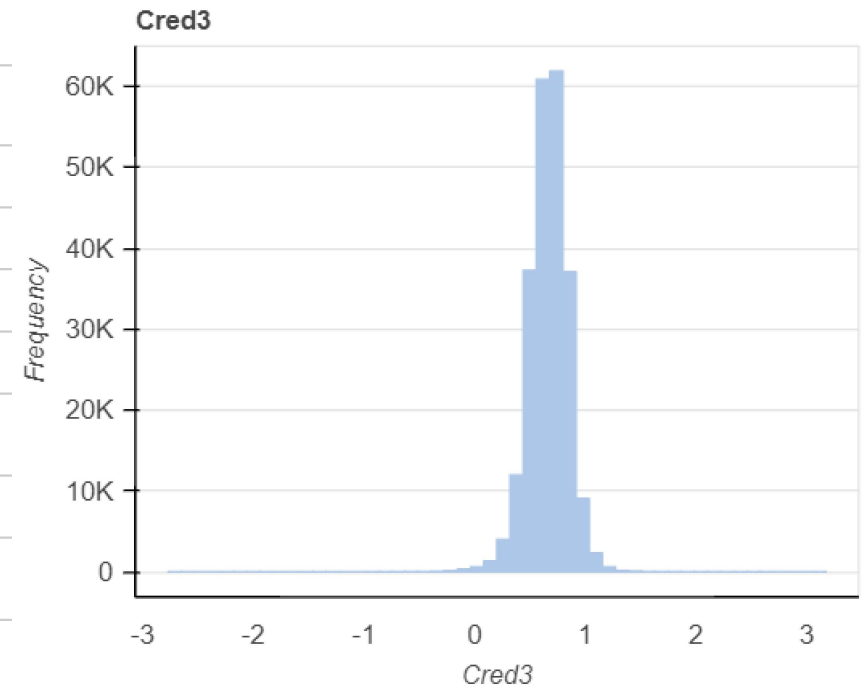
Memory Size	3645520
Mean	0.6663
Minimum	-0.28
Maximum	2.1933
Zeros	7
Zeros (%)	0.0%
Negatives	230
Negatives (%)	0.1%



Cred3
numerical
Show Details

Approximate Distinct Count	636
Approximate Unique (%)	0.3%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

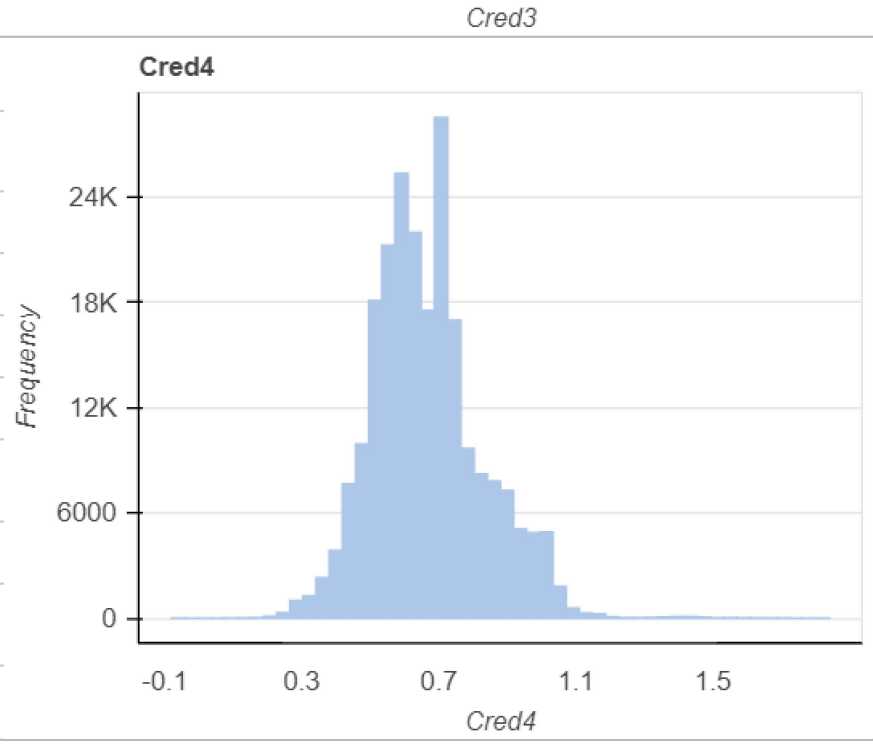
Memory Size	3645520
Mean	0.6668
Minimum	-2.7667
Maximum	3.1733
Zeros	9
Zeros (%)	0.0%
Negatives	662
Negatives (%)	0.3%



Cred4
numerical
Show Details

Approximate Distinct Count	460
Approximate Unique (%)	0.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Memory Size	3645520
Mean	0.6669
Minimum	-0.08
Maximum	1.84
Zeros	0
Zeros (%)	0.0%
Negatives	2
Negatives (%)	0.0%



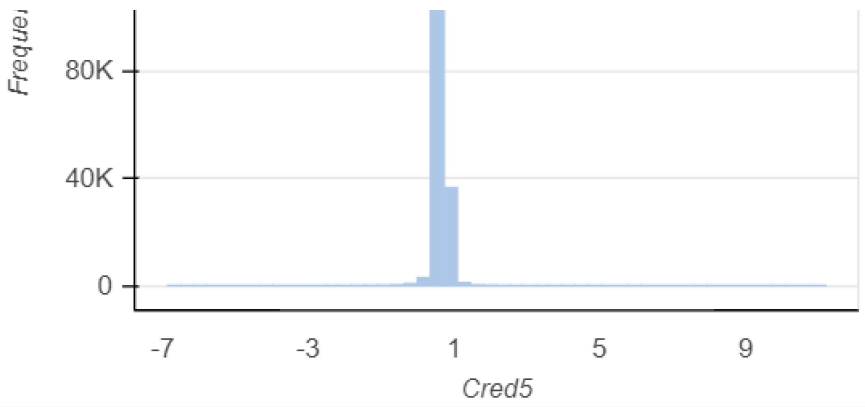
Cred5

Approximate Distinct Count	920
Approximate Unique (%)	0.4%

Missing	0
Missing (%)	0.0%
Infinite	0

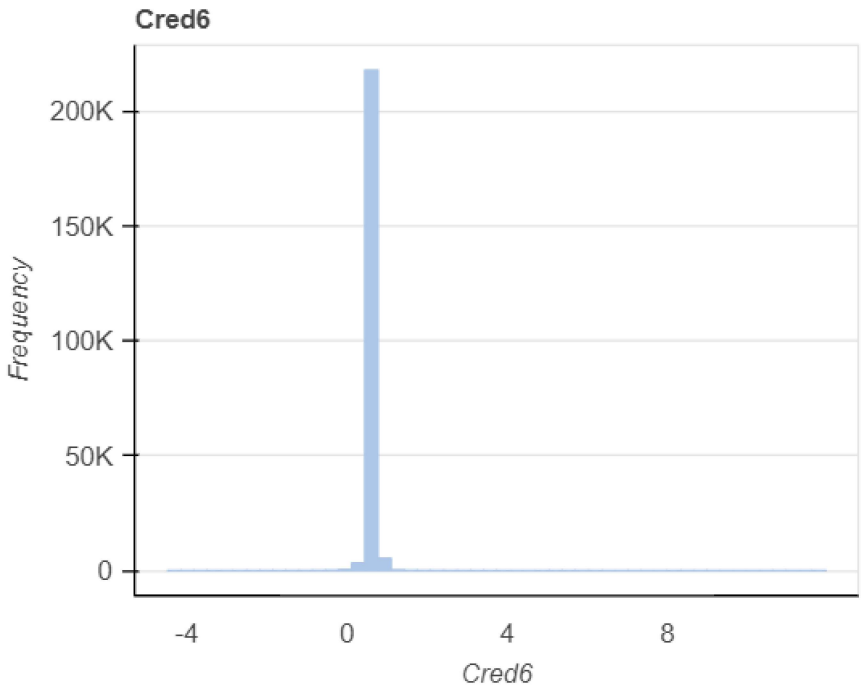
Cred5
numerical
Show Details

Infinite (%)	0.0%	Zeros	11
Memory Size	3645520	Zeros (%)	0.0%
Mean	0.6666	Negatives	1010
Minimum	-6.8567	Negatives (%)	0.4%
Maximum	11.2033		



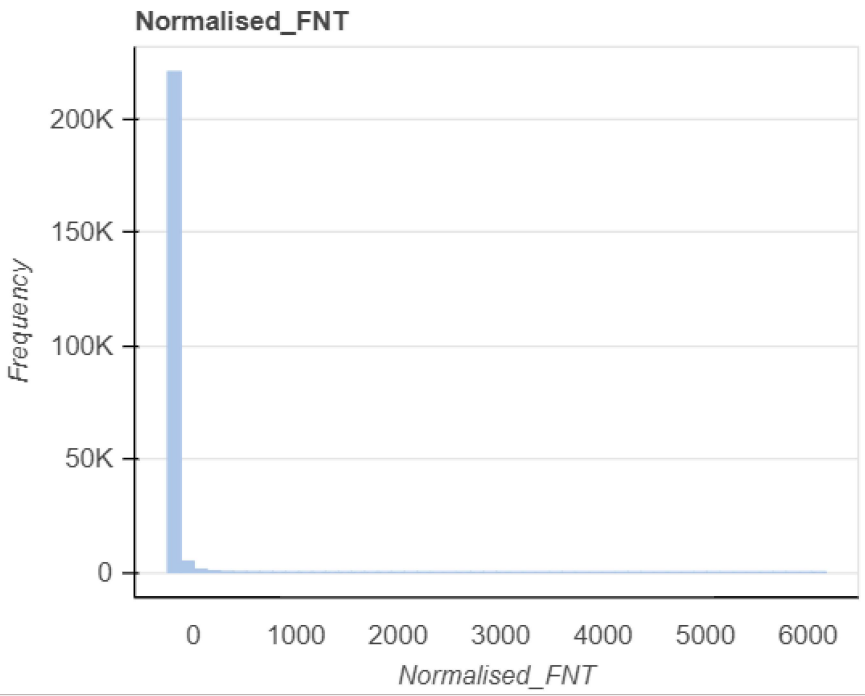
Cred6
numerical
Show Details

Approximate Distinct Count	778	Memory Size	3645520
Approximate Unique (%)	0.3%	Mean	0.6668
Missing	0	Minimum	-4.4767
Missing (%)	0.0%	Maximum	11.95
Infinite	0	Zeros	3
Infinite (%)	0.0%	Zeros (%)	0.0%
		Negatives	384
		Negatives (%)	0.2%



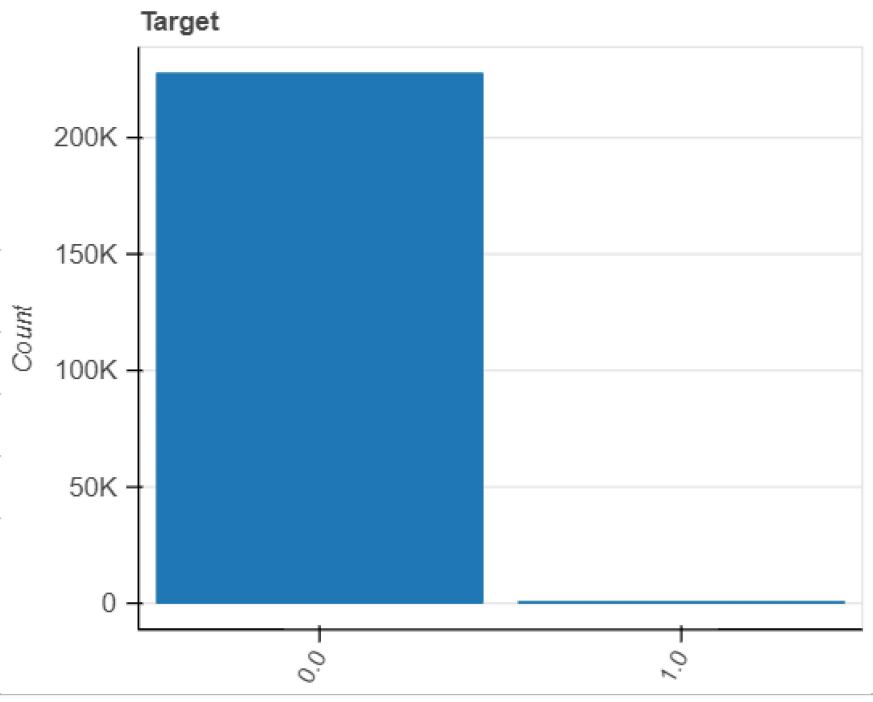
Normalised_FNT
numerical
Show Details

Approximate Distinct Count	29299	Memory Size	3645520
Approximate Unique (%)	12.9%	Mean	-227.954
Missing	0	Minimum	-250
Missing (%)	0.0%	Maximum	6172.79
Infinite	0	Zeros	102
Infinite (%)	0.0%	Zeros (%)	0.0%
		Negatives	225396
		Negatives (%)	98.9%



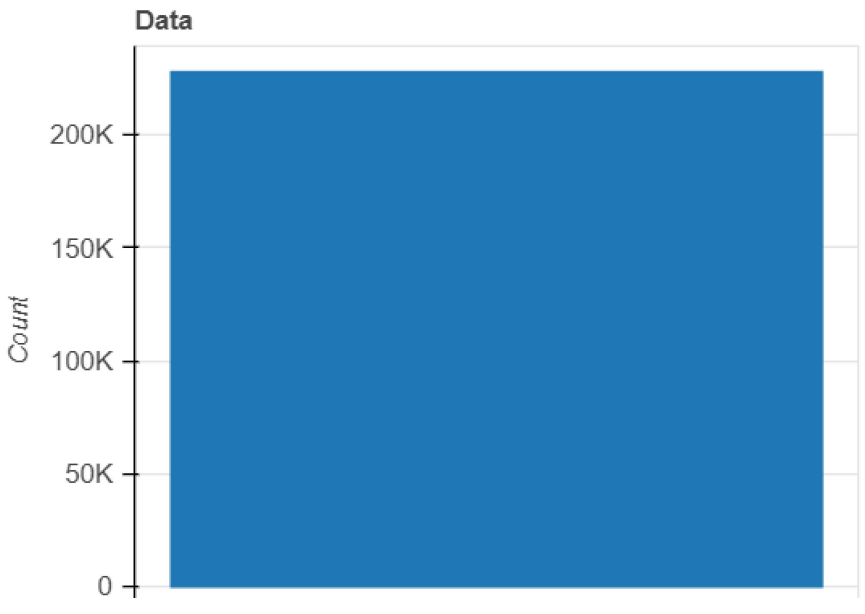
Target
categorical
Show Details

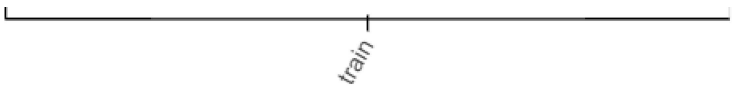
Approximate Distinct Count	2
Approximate Unique (%)	0.0%
Missing	0
Missing (%)	0.0%
Memory Size	15493460



Data
categorical
Show Details

Approximate Distinct Count	1
Approximate Unique (%)	0.0%
Missing	0
Missing (%)	0.0%
Memory Size	15949150



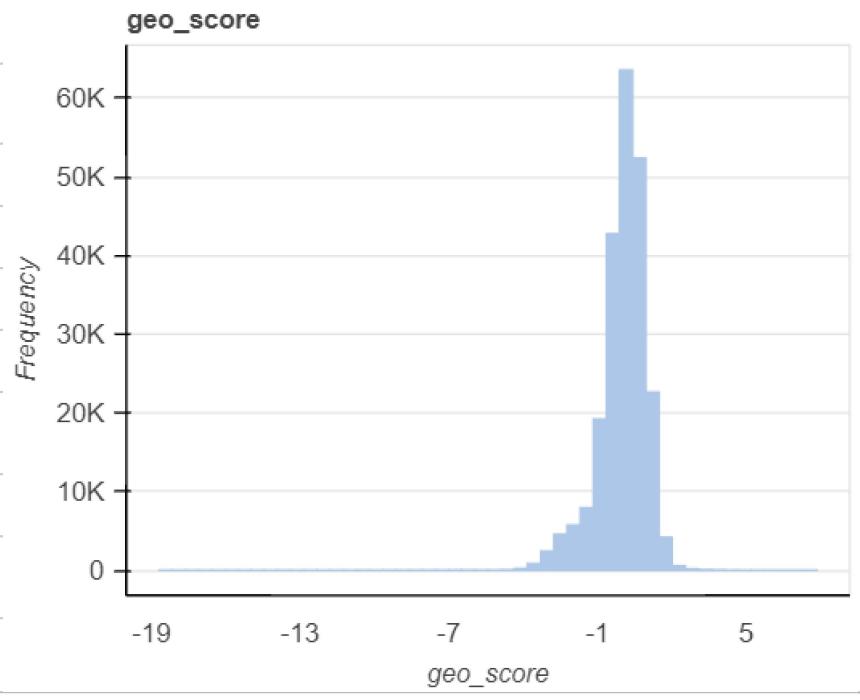


geo_score
numerical

Show Details

Approximate Distinct Count	14082
Approximate Unique (%)	6.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Memory Size	3645520
Mean	0.008908
Minimum	-18.68
Maximum	7.85
Zeros	486
Zeros (%)	0.2%
Negatives	95677
Negatives (%)	42.0%

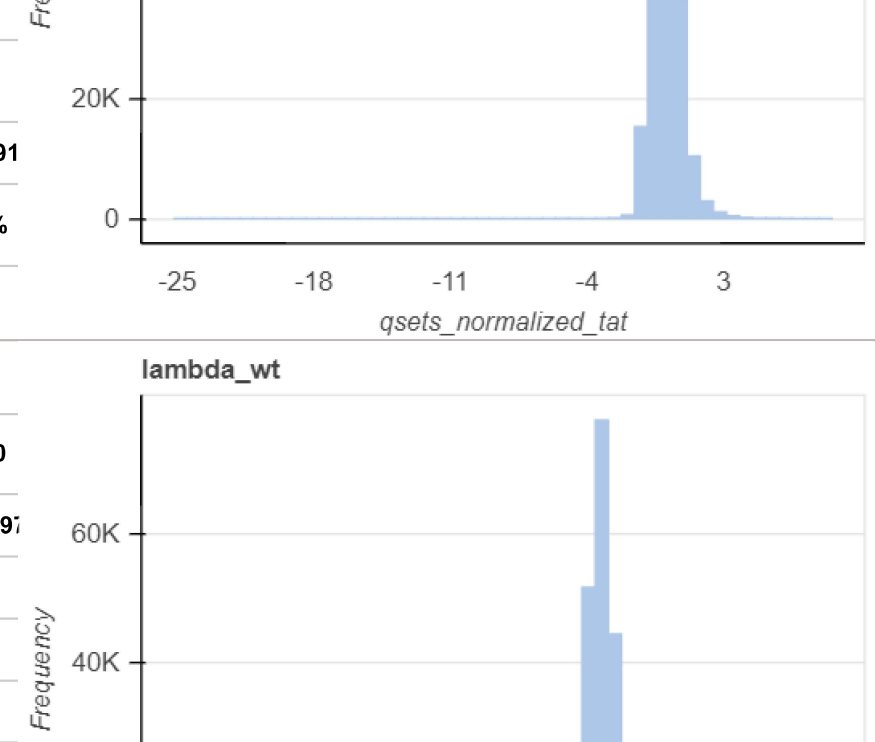


instance_scores
numerical

Show Details

Approximate Distinct Count	4175
Approximate Unique (%)	1.8%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Memory Size	3645520
Mean	-0.00012
Minimum	-24.59
Maximum	23.75
Zeros	642
Zeros (%)	0.3%
Negatives	127483
Negatives (%)	56.0%

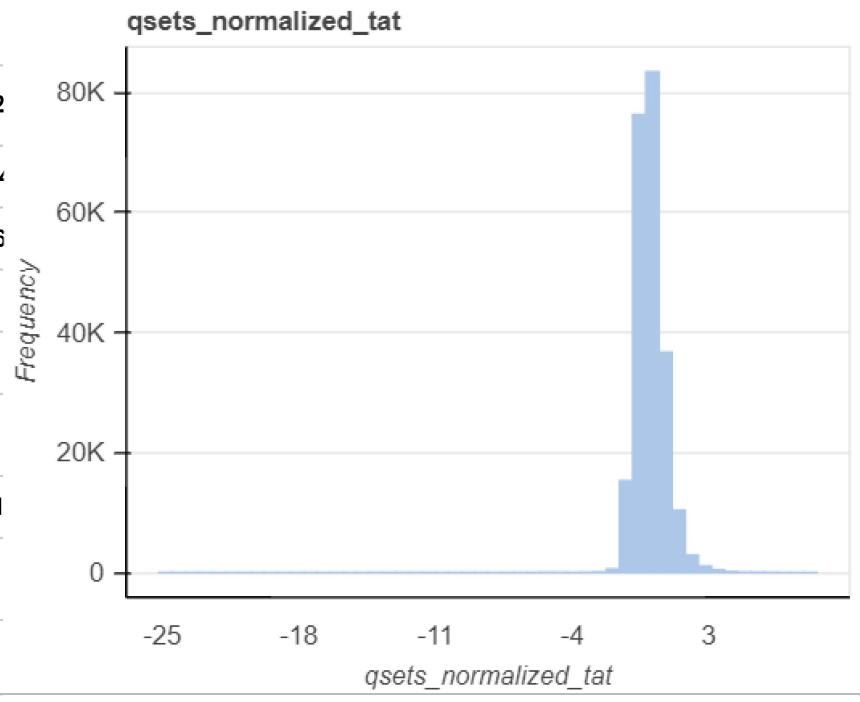


qsets_normalized_tat
numerical

Show Details

Approximate Distinct Count	14223
Approximate Unique (%)	6.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Memory Size	364552
Mean	0.0013
Minimum	-25.156
Maximum	3.54
Zeros	488
Zeros (%)	0.2%
Negatives	122491
Negatives (%)	53.8%



lambda_wt
numerical

Show Details

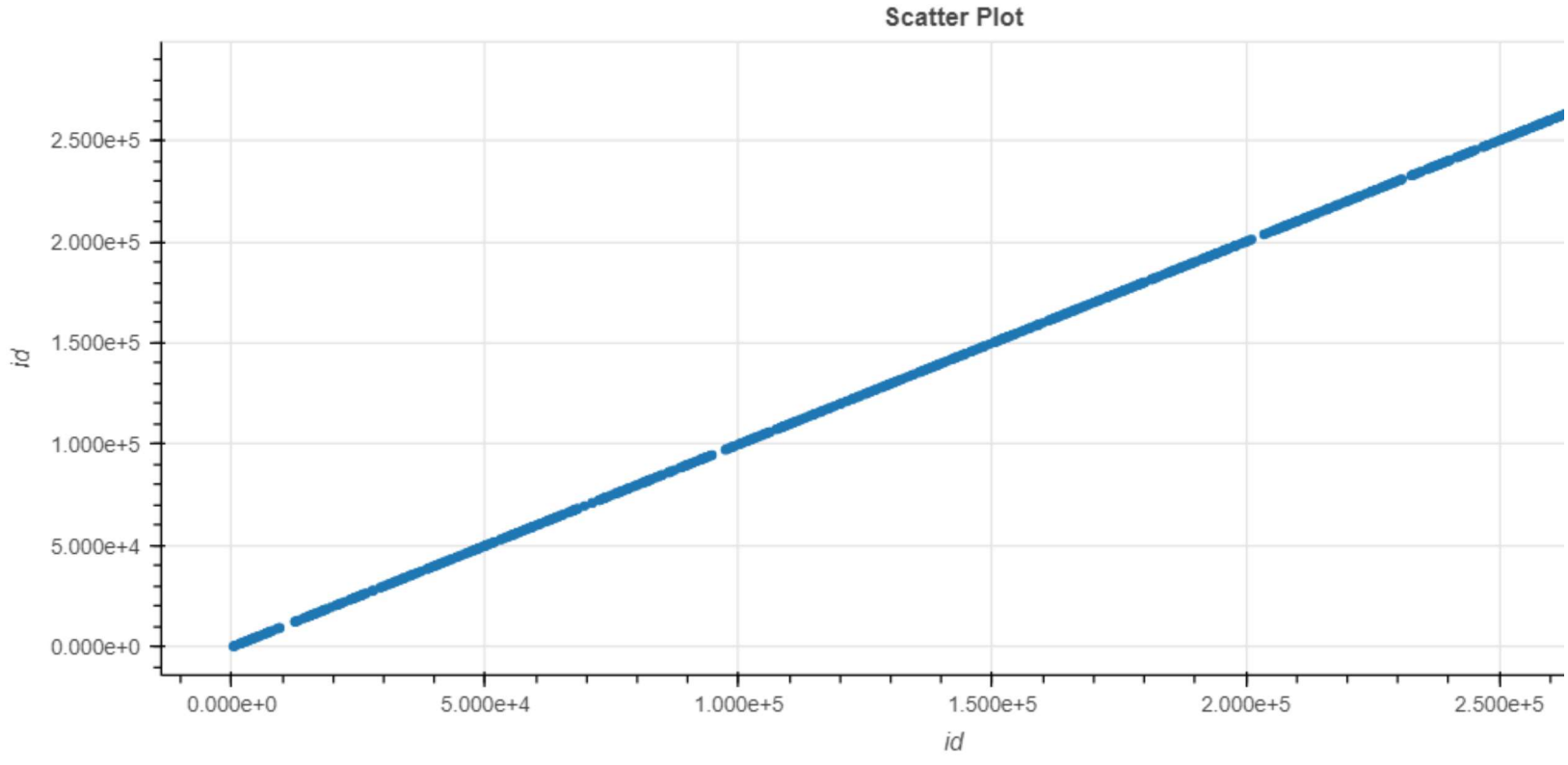
Approximate Distinct Count	1301
Approximate Unique (%)	0.6%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Memory Size	3645520
Mean	0.000349
Minimum	-19.21
Maximum	10.53
Zeros	1351
Zeros (%)	0.6%
Negatives	106356
Negatives (%)	46.7%

Interactions

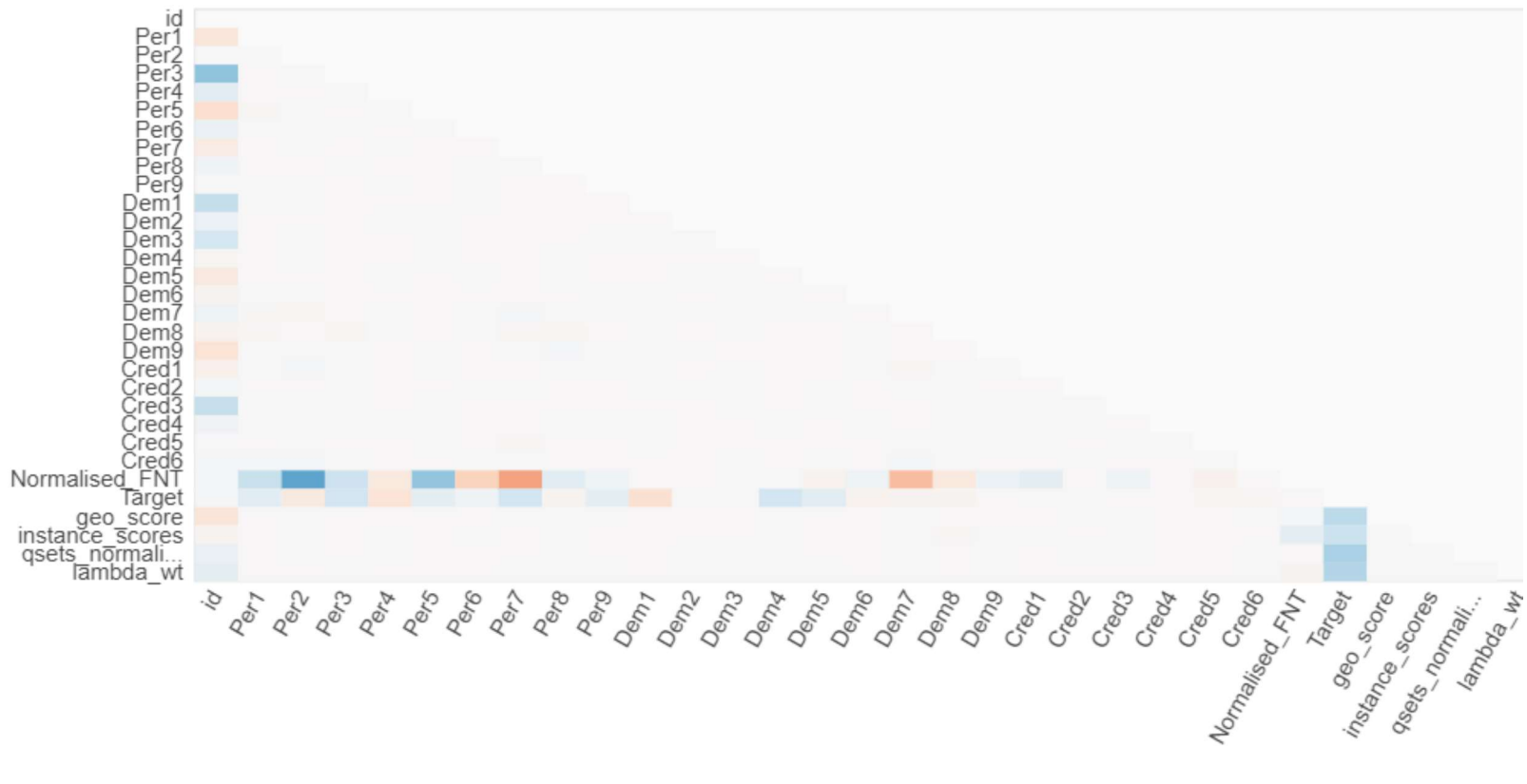
X-Axis: id

Y-Axis: id



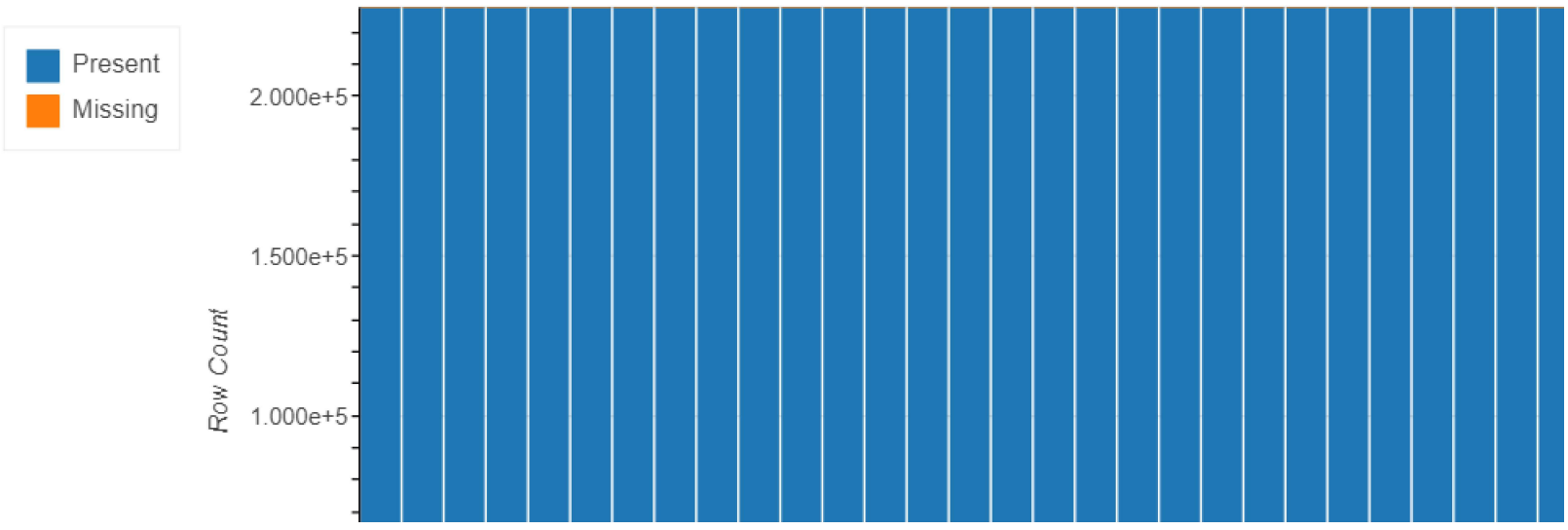
Correlations

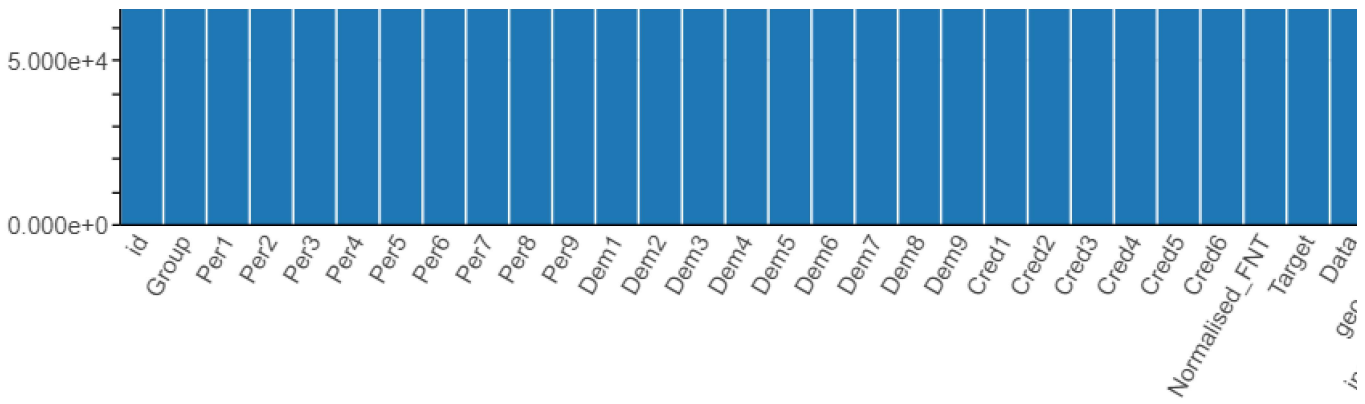
Pearson | Spearman | KendallTau



Missing Values

Bar Chart | Spectrum | Heat Map | Dendrogram





Report generated with [DataPrep (https://dataprep.ai/)](https://dataprep.ai/)

In [35]:
```
# It is obvious that ID will not contribute in determining the fraud transactions. Hence dropping id and the
# both train and test datasets.

train_df.drop(['id', 'Data'], axis=1, inplace = True)
test_df.drop(['id', 'Data'], axis=1, inplace = True)
```

In [36]:
```
# Group also does not impact the target variable as the lambda wts which specifies the proprietary index valu

train_df.drop(['Group'], axis=1, inplace = True)
test_df.drop(['Group'], axis=1, inplace = True)
```

In [37]:
```
# Checking the first 10 rows
train_df.head(10)
```

Out[37]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	Dem3	Dem4
0	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.340000	1.010000	0.863333	0.460000	0.643333	0.736667	0.756667
1	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.810000	0.783333	0.190000	0.470000	0.613333	0.883333	0.653333
2	1.130000	0.143333	0.946667	0.123333	0.080000	0.836667	0.056667	0.756667	0.226667	0.660000	0.730000	0.873333	0.923333
3	0.636667	1.090000	0.750000	0.940000	0.743333	0.346667	0.956667	0.633333	0.486667	1.096667	0.466667	0.670000	0.526667
4	0.560000	1.013333	0.593333	0.416667	0.773333	0.460000	0.853333	0.796667	0.516667	0.756667	0.683333	0.296667	0.780000
5	0.873333	0.140000	0.836667	0.273333	0.190000	0.653333	0.503333	0.723333	1.233333	1.150000	0.533333	0.986667	0.513333
6	0.980000	0.546667	0.820000	0.863333	0.680000	1.163333	0.486667	0.893333	0.806667	1.300000	0.050000	0.780000	0.223333
7	-0.136667	0.360000	0.366667	0.996667	0.473333	0.706667	0.676667	0.933333	0.166667	0.066667	0.783333	0.823333	-0.010000
8	1.010000	0.606667	0.823333	1.066667	0.516667	0.653333	0.633333	0.703333	0.790000	0.846667	0.343333	0.420000	0.683333
9	-0.133333	0.890000	0.210000	0.333333	0.743333	1.023333	0.416667	0.096667	0.403333	0.453333	0.890000	0.306667	0.906667

In [38]:
```
# Checking the last 10 rows
train_df.tail(10)
```

Out[38]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	Dem3	D
227835	1.306667	0.906667	0.153333	1.836667	1.046667	0.616667	0.903333	0.613333	0.080000	0.870000	0.513333	0.236667	0.92
227836	0.736667	0.123333	0.996667	1.156667	0.090000	0.676667	0.563333	0.726667	0.860000	1.060000	0.290000	1.023333	1.006
227837	0.470000	-0.420000	0.720000	0.966667	0.116667	1.070000	0.733333	0.763333	0.950000	1.086667	0.486667	0.543333	0.553
227838	0.523333	0.603333	1.410000	0.783333	0.403333	0.876667	0.756667	0.696667	1.086667	0.010000	0.316667	0.233333	0.520
227839	1.080000	0.760000	0.726667	0.833333	0.610000	0.480000	0.653333	0.656667	0.606667	1.100000	0.673333	0.813333	0.926
227840	0.476667	1.013333	0.536667	0.576667	1.406667	1.846667	0.600000	1.103333	0.356667	0.530000	0.683333	1.096667	0.750
227841	1.363333	0.730000	0.060000	0.776667	0.883333	0.466667	0.733333	0.590000	0.806667	0.436667	0.906667	0.736667	0.773
227842	1.060000	0.756667	0.906667	0.896667	0.503333	0.396667	0.683333	0.620000	0.630000	0.870000	0.986667	1.053333	0.606
227843	0.433333	1.013333	1.163333	0.940000	0.930000	0.900000	0.813333	0.720000	1.020000	0.413333	1.043333	0.720000	0.610
227844	1.006667	0.553333	0.946667	1.206667	0.406667	0.750000	0.520000	0.756667	1.053333	0.273333	-0.046667	0.503333	0.426

In [39]:
```
# Checking the number of rows and columns.

print('Total number of observations/rows are:', train_df.shape[0])
print('Total number of features/columns are: ', train_df.shape[1])
```

```
Total number of observations/rows are: 227845
Total number of features/columns are:  30
```

In [40]:
```
# Checking the basic info of the data.

train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 227845 entries, 0 to 227844
Data columns (total 30 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  Target                227845 non-null  float64
 26  geo_score             227845 non-null  float64
 27  instance_scores       227845 non-null  float64
 28  qsets_normalized_tat  227845 non-null  float64
 29  lambda_wt             227845 non-null  float64
dtypes: float64(30)
memory usage: 53.9 MB
```

In [41]:
```
# Checking the data description of all the variables, which gives us the count, mean, std, minimum, 25th quar
# median, 75th quartile and maximum values in the respective numerical columns.

train_df.describe().T
```

Out[41]:

	count	mean	std	min	25%	50%	75%	max
Per1	227845.0	0.666006	0.654133	-18.136667	0.360000	0.670000	1.103333	1.483333
Per2	227845.0	0.667701	0.548305	-23.573333	0.470000	0.690000	0.933333	8.020000
Per3	227845.0	0.666315	0.506357	-15.443333	0.370000	0.726667	1.010000	3.793333
Per4	227845.0	0.666687	0.471956	-1.226667	0.383333	0.660000	0.913333	6.163333
Per5	227845.0	0.666723	0.461393	-37.246667	0.436667	0.650000	0.870000	12.266667
Per6	227845.0	0.667378	0.444573	-8.053333	0.410000	0.576667	0.800000	25.100000
Per7	227845.0	0.666934	0.415657	-13.853333	0.483333	0.680000	0.856667	40.863333
Per8	227845.0	0.666279	0.401546	-23.740000	0.596667	0.673333	0.776667	7.336667
Per9	227845.0	0.666688	0.366537	-3.810000	0.453333	0.650000	0.866667	5.863333
Dem1	227845.0	0.666576	0.340436	-0.893333	0.413333	0.656667	0.913333	4.673333
Dem2	227845.0	0.666329	0.332016	-1.263333	0.450000	0.663333	0.886667	3.043333
Dem3	227845.0	0.666380	0.305384	-0.833333	0.473333	0.683333	0.883333	3.626667
Dem4	227845.0	0.666871	0.292013	-3.853333	0.510000	0.690000	0.840000	6.440000
Dem5	227845.0	0.666518	0.279469	-2.500000	0.500000	0.666667	0.833333	2.080000
Dem6	227845.0	0.666748	0.271590	-0.976667	0.513333	0.666667	0.820000	2.530000
Dem7	227845.0	0.666394	0.257631	-17.500000	0.596667	0.646667	0.710000	9.413333
Dem8	227845.0	0.666540	0.244423	-10.943333	0.590000	0.656667	0.730000	9.733333
Dem9	227845.0	0.666535	0.241902	-2.296667	0.486667	0.670000	0.843333	4.166667
Cred1	227845.0	0.666684	0.207575	-14.270000	0.613333	0.663333	0.716667	8.176667
Cred2	227845.0	0.666264	0.202204	-0.280000	0.546667	0.680000	0.813333	2.193333
Cred3	227845.0	0.666755	0.174204	-2.766667	0.560000	0.673333	0.783333	3.173333
Cred4	227845.0	0.666878	0.160803	-0.080000	0.556667	0.650000	0.746667	1.840000
Cred5	227845.0	0.666566	0.135762	-6.856667	0.643333	0.666667	0.696667	11.203333
Cred6	227845.0	0.666776	0.111612	-4.476667	0.650000	0.670000	0.693333	11.950000
Normalised_FNT	227845.0	-227.954170	61.951661	-250.000000	-248.617500	-244.510000	-230.750000	6172.790000
Target	227845.0	0.001729	0.041548	0.000000	0.000000	0.000000	0.000000	1.000000
geo_score	227845.0	0.008908	0.997629	-18.680000	-0.400000	0.150000	0.630000	7.850000
instance_scores	227845.0	-0.000123	1.091488	-24.590000	-0.540000	-0.090000	0.450000	23.750000
qsets_normalized_tat	227845.0	0.001346	0.850169	-25.156000	-0.480000	-0.066000	0.400000	8.540000
lambda_wt	227845.0	0.000350	0.957957	-19.210000	-0.430000	0.050000	0.490000	10.530000

Data Preprocessing

Handling missing values

The misssing values are aready handled in the individual tables before the merge.

In [42]:
```
# Checking what percent of values are missing in each feature.

pd.DataFrame(train_df.isnull().sum()/len(train_df)*100, columns=['Missing Value %'], index = train_df.columns
```

Out[42]:

	Missing Value %
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	0.0
geo_score	0.0
instance_scores	0.0
qsets_normalized_tat	0.0
lambda_wt	0.0

In [43]:
```
# Data Types of each variable:

pd.DataFrame(train_df.dtypes, columns=['Datatype'], index = train_df.columns)
```

Out[43]:

	Datatype
Per1	float64
Per2	float64
Per3	float64
Per4	float64
Per5	float64
Per6	float64
Per7	float64
Per8	float64
Per9	float64
Dem1	float64
Dem2	float64
Dem3	float64
Dem4	float64
Dem5	float64
Dem6	float64
Dem7	float64
Dem8	float64
Dem9	float64
Cred1	float64
Cred2	float64
Cred3	float64
Cred4	float64
Cred5	float64
Cred6	float64
Normalised_FNT	float64
Target	float64
geo_score	float64
instance_scores	float64
qsets_normalized_tat	float64
lambda_wt	float64

Handling Outliers

In [44]:
```
# Here it can be observed from the above data description that only Normalised_FNT might potentially have out

sns.boxplot(data=train_df, y='Normalised_FNT')
plt.show()
```

In [45]:
```
train_df['Normalised_FNT'].describe()
```

Out[45]:
```
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 [46]:
```
# Now to find the maximum limit using inter quartile method:

Q1 = -248.617500
Q3 = -230.750000
IQR = Q3 - Q1
lower_limit = Q1 - 1.5*IQR
upper_limit = Q3 + 1.5*IQR
print(lower_limit)
print(upper_limit)
```

```
-275.41875000000005
-203.94875
```

In [47]:
```
# Checking how many observations are outliers (above the upper limit):

train_df[train_df['Normalised_FNT'] > upper_limit].count()[0]
```

Out[47]: 25486

Since there are 25486 values which are outliers, we might not handle them for now as these values might be critical in determining if the transaction is a fraudulent one.

Splitting the data into independent variables and dependent variables

In [48]:
```
x = train_df.drop('Target', axis=1)
y = train_df['Target']
```

Feature Scaling

In [49]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_sc = scaler.fit_transform(x)
```

In [50]:
```
x_sc
```

Out[50]:
```
array([[ 6.17603382e-01, -1.59949449e-01, -3.67953077e-01, ...,
        -5.48582452e-02, -8.24950841e-01, -1.36070832e-01],
       [-2.94547070e-01,  9.82969984e-01,  4.28587917e-01, ...,
         4.76527663e-01,  1.63090340e-01,  6.88602704e-01],
       [ 7.09328008e-01, -9.56345437e-01,  5.53664602e-01, ...,
         1.42935757e+00, -5.07366176e-01, -5.32749242e-01],
       ...,
       [ 6.02315944e-01,  1.62256561e-01,  4.74668801e-01, ...,
        -1.18991027e-01,  1.01792044e-02,  1.56218522e-01],
       [-3.55696821e-01,  6.30367180e-01,  9.81558524e-01, ...,
        -6.04567805e-01,  4.50092778e-01,  1.59678748e+00],
       [ 5.20782943e-01, -2.08584319e-01,  5.53664602e-01, ...,
        -2.01447461e-01,  6.45348535e-01, -3.65060454e-04]])
```

In [51]:
```
y.value_counts()
```

Out[51]:
```
0.0    227451
1.0       394
Name: Target, dtype: int64
```

In [52]:
```
# Checking the percentage of fraudulent transactions out of entire data:

fraud_percent = round((394/(227451+394))*100,3)
valid_transaction_percent = round((227451/(227451+394))*100,3)
print("Percentage of valid transcation is: ",valid_transaction_percent,"%", sep='')
print("Percentage of fraud transcation is: ",fraud_percent,"%", sep='')
```

```
Percentage of valid transcation is: 99.827%
Percentage of fraud transcation is: 0.173%
```

Only 0.17% of the recorded transactions are fraudulent.

Imbalance Treatment of the Target variable

In [53]:
```
y.value_counts()
```

Out[53]:
```
0.0    227451
1.0       394
Name: Target, dtype: int64
```

In [55]:
```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler()
x_bal, y_bal = ros.fit_resample(x_sc, y)
print(y.value_counts()) # data before balancing
print(y_bal.value_counts()) # target variable after balancing
```

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

Splitting the balanced data into train and test

In [57]:
```
from sklearn.model_selection import train_test_split, cross_val_score

x_train, x_test, y_train, y_test = train_test_split(x_bal, y_bal, test_size = 0.25, random_state = 101)
```

In [58]:
```
# importing necessary models from sklearn for model building:
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# Importing performance metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, recall_score
```

Logistic Regression

In [59]:
```
# Building Binary classification Logistic regression model.

logit = LogisticRegression()
logit.fit(x_train, y_train)
pred_train_logit = logit.predict(x_train)
pred_test_logit = logit.predict(x_test)
```

In [60]:
```
print("Classification Report, Confusion matrix, Accuracy score and Recall_Score for Train data using Logistic
print(classification_report(y_train, pred_train_logit))
print('\n')
print(confusion_matrix(y_train, pred_train_logit))
print('\n')
print("Accuracy Score:",accuracy_score(y_train, pred_train_logit))
print('\n')
print("Recall Score:",recall_score(y_train, pred_train_logit))
```

```
Classification Report, Confusion matrix, Accuracy score and Recall_Score for Train data using Logistic Regre
ssion model:
              precision    recall  f1-score   support

         0.0       0.92      0.98      0.95    170627
         1.0       0.97      0.91      0.94    170549

    accuracy                           0.94    341176
   macro avg       0.95      0.94      0.94    341176
weighted avg       0.95      0.94      0.94    341176



[[166390   4237]
 [ 15044 155505]]


Accuracy Score: 0.9434866461884776


Recall Score: 0.9117907463544201
```

In [61]:
```
print("Classification Report, Confusion matrix, Accuracy score and Recall_Score for Test data using Logistic
print(classification_report(y_test, pred_test_logit))
print('\n')
print(confusion_matrix(y_test, pred_test_logit))
print('\n')
print("Accuracy Score:",accuracy_score(y_test, pred_test_logit))
print('\n')
print("Recall Score:",recall_score(y_test, pred_test_logit))
```

```
Classification Report, Confusion matrix, Accuracy score and Recall_Score for Test data using Logistic Regres
sion model:
              precision    recall  f1-score   support

         0.0       0.92      0.98      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



[[55407  1417]
 [ 5048 51854]]


Accuracy Score: 0.9431528410389884


Recall Score: 0.911286070788373
```

In [62]:
```
# Cross Validation test for the model:

accuracy_logit_train = cross_val_score(estimator=logit, X=x_train, y=y_train, cv=10, scoring='accuracy')
accuracy_logit_test = cross_val_score(estimator=logit, X=x_test, y=y_test, cv=10, scoring='accuracy')
print("Cross Validation Score for Train data", np.mean(accuracy_logit_train))
print("Cross Validation Score for Test data", np.mean(accuracy_logit_test))
```

```
Cross Validation Score for Train data 0.9434895765280686
Cross Validation Score for Test data 0.9429505913367272
```

Random Forest Model

In [63]:
```
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(x_train, y_train)
pred_train_rfc = rfc.predict(x_train)
pred_test_rfc = rfc.predict(x_test)
```

In [64]:
```
print("Classification Report, Confusion matrix, Accuracy score and Recall_Score for Train data using Random F
print(classification_report(y_train, pred_train_rfc))
print('\n')
print(confusion_matrix(y_train, pred_train_rfc))
print('\n')
print("Accuracy Score:",accuracy_score(y_train, pred_train_rfc))
print('\n')
print("Recall Score:",recall_score(y_train, pred_train_rfc))
```

```
Classification Report, Confusion matrix, Accuracy score and Recall_Score for Train data using Random Forest
model:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    170627
         1.0       1.00      1.00      1.00    170549

    accuracy                           1.00    341176
   macro avg       1.00      1.00      1.00    341176
weighted avg       1.00      1.00      1.00    341176



[[170627      0]
 [     0 170549]]


Accuracy Score: 1.0


Recall Score: 1.0
```

In [65]:
```
print("Classification Report, Confusion matrix, Accuracy score and Recall_Score for Test data using Logistic
print(classification_report(y_test, pred_test_rfc))
print('\n')
print(confusion_matrix(y_test, pred_test_rfc))
print('\n')
print("Accuracy Score:",accuracy_score(y_test, pred_test_rfc))
print('\n')
print("Recall Score:",recall_score(y_test, pred_test_rfc))
```

```
Classification Report, Confusion matrix, Accuracy score and Recall_Score for Test data using Logistic Random
Forest model:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     56824
         1.0       1.00      1.00      1.00     56902

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726



[[56817     7]
 [    0 56902]]


Accuracy Score: 0.9999384485517824


Recall Score: 1.0
```

In [66]:
```
# Cross Validation test for the model:

accuracy_rfc_train = cross_val_score(estimator=rfc, X=x_train, y=y_train, cv=10, scoring='accuracy')
accuracy_rfc_test = cross_val_score(estimator=rfc, X=x_test, y=y_test, cv=10, scoring='accuracy')
print("Cross Validation Score for Train data", np.mean(accuracy_rfc_train))
print("Cross Validation Score for Test data", np.mean(accuracy_rfc_test))
```

```
Cross Validation Score for Train data 0.999953103332035
Cross Validation Score for Test data 0.9998593120577122
```

XG Boost Classifier Model

In [67]:
```
xgbc = XGBClassifier()
xgbc.fit(x_train, y_train)
pred_train_xgbc = xgbc.predict(x_train)
pred_test_xgbc = xgbc.predict(x_test)
```

In [68]:
```
print("Classification Report, Confusion matrix, Accuracy score and Recall_Score for Train data using XG Boost
print(classification_report(y_train, pred_train_xgbc))
print('\n')
print(confusion_matrix(y_train, pred_train_xgbc))
print('\n')
print("Accuracy Score:",accuracy_score(y_train, pred_train_xgbc))
print('\n')
print("Recall Score:",recall_score(y_train, pred_train_xgbc))
```

```
Classification Report, Confusion matrix, Accuracy score and Recall_Score for Train data using XG Boost Class
ifier model:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    170627
         1.0       1.00      1.00      1.00    170549

    accuracy                           1.00    341176
   macro avg       1.00      1.00      1.00    341176
weighted avg       1.00      1.00      1.00    341176



[[170627      0]
 [     0 170549]]


Accuracy Score: 1.0


Recall Score: 1.0
```

In [69]:
```
print("Classification Report, Confusion matrix, Accuracy score and Recall_Score for Test data using Logistic
print(classification_report(y_test, pred_test_xgbc))
print('\n')
print(confusion_matrix(y_test, pred_test_xgbc))
print('\n')
print("Accuracy Score:",accuracy_score(y_test, pred_test_xgbc))
print('\n')
print("Recall Score:",recall_score(y_test, pred_test_xgbc))
```

```
Classification Report, Confusion matrix, Accuracy score and Recall_Score for Test data using Logistic XG Boo
st Classifier model:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     56824
         1.0       1.00      1.00      1.00     56902

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726



[[56810    14]
 [    0 56902]]


Accuracy Score: 0.9998768971035648


Recall Score: 1.0
```

In [70]:
```
# Cross Validation test for the model:

accuracy_xgbc_train = cross_val_score(estimator=xgbc, X=x_train, y=y_train, cv=10, scoring='accuracy')
accuracy_xgbc_test = cross_val_score(estimator=xgbc, X=x_test, y=y_test, cv=10, scoring='accuracy')
print("Cross Validation Score for Train data", np.mean(accuracy_xgbc_train))
print("Cross Validation Score for Test data", np.mean(accuracy_xgbc_test))
```

```
Cross Validation Score for Train data 0.9999237930326836
Cross Validation Score for Test data 0.9997801733988148
```

Checking performance of the model using ROC-AUC Curve

In [71]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
```

In [72]:
```
rfc_roc_under_auc = roc_auc_score(y_test,pred_test_rfc)
```

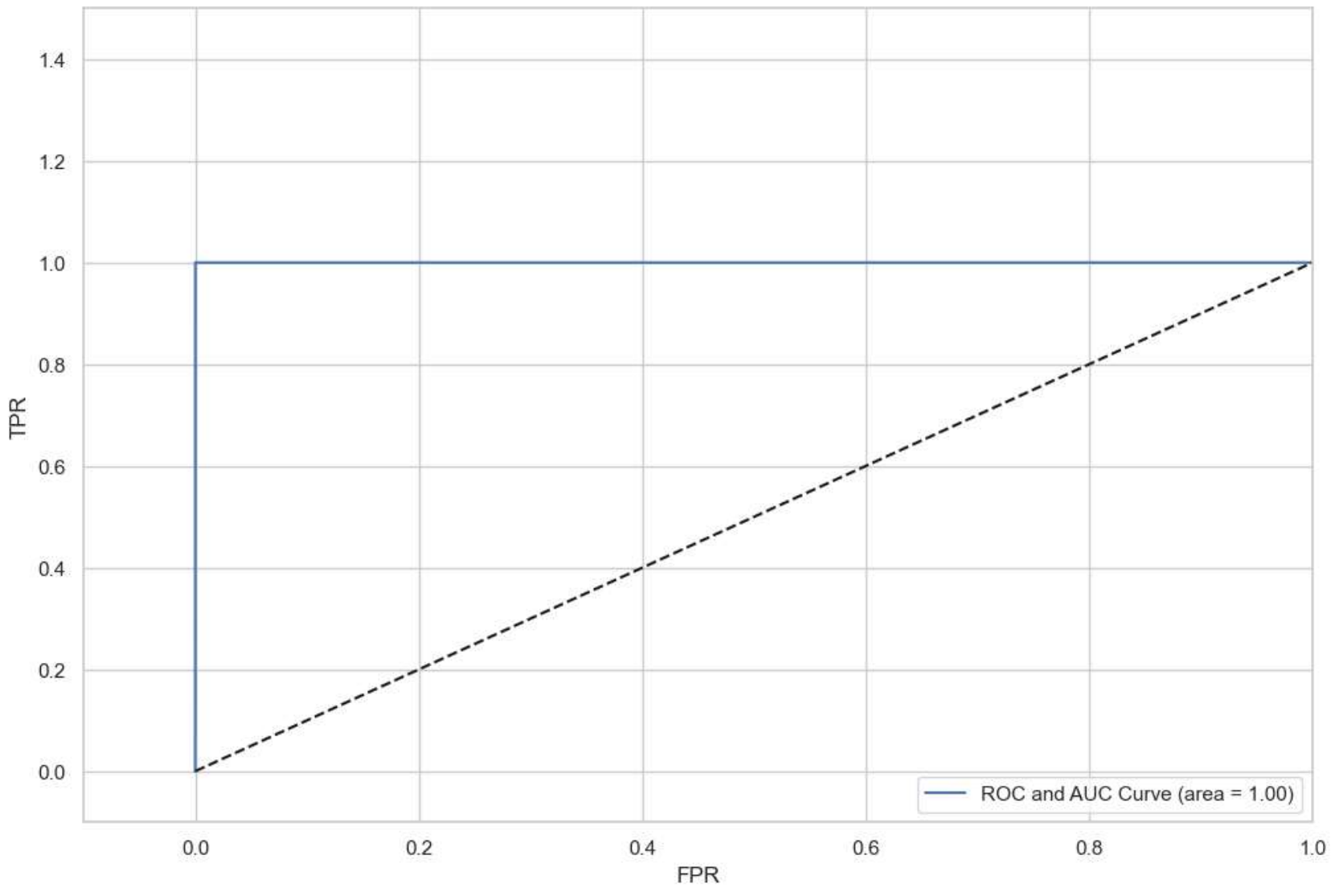
In [73]:
```
rfc_roc_under_auc
```

Out[73]:
```
0.9999384063071942
```

In [74]:
```
fpr, tpr, thres = roc_curve(y_test, pred_test_rfc)
display(fpr[:10])
display(tpr[:10])
display(thres[:10])
```

```
array([0.00000000e+00, 1.23187386e-04, 1.00000000e+00])

array([0., 1., 1.])

array([inf,  1.,  0.])
```

In [75]:
```
plt.figure(figsize=(12,8))
plt.plot(fpr, tpr, label = 'ROC and AUC Curve (area = %0.2f)' % rfc_roc_under_auc)
plt.plot([0,1],[0,1], 'k--')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.xlim([-0.1,1.0])
plt.ylim([-0.1,1.5])
plt.legend(loc='lower right')
plt.show()
```



Concluding: Since the model is showing remarkable performance in the ROC-AUC Curve, the model using Random Forest with Train accuracy of 100% and Test accuracy of 99.99% and Cross Validation mean Accuracy scores for training and test data is 99.995% and 99.986% respectively, Hence the model can be said to be giving striking performance and can be used for deployment in production.

Note: Since this is a Fraud detection model, the False Negative value should be as low as possible as we would not want to classify a transaction as not fraudulent when they are actually fraudulent.

Now predicting the model using the test data:

In [77]:
```
# Removing the Target column from test_df

test_df.drop('Target', axis = 1, inplace = True)
```

In [78]:
```
test_df.head()
```

Out[78]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	Dem3	D
227845	-0.300000	1.540000	0.220000	-0.280000	0.570000	0.260000	0.700000	1.076667	0.930000	0.156667	0.546667	0.530000	0.876
227846	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	0.550000	0.543333	0.433333	0.966667	0.760000	0.576
227847	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	0.686667	0.593333	1.250000	0.826667	0.826667	0.653
227848	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	0.663333	1.156667	1.186667	0.900000	0.433333	0.230
227849	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	0.686667	1.476667	1.213333	0.853333	0.583333	0.850

In [79]:
```
pred_test_rfc = rfc.predict(scaler.transform(test_df))
```

In [80]:
```
test_df['Predicted'] = pred_test_rfc
```

In [81]:
```
pd.DataFrame(test_df['Predicted'].value_counts())
```

Out[81]:

	Predicted
0.0	56872
1.0	90

In [82]:
```
test_df[test_df['Predicted'] == 1.0].head(20)
```

Out[82]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	Dem3
228121	-1.300000	2.146667	-2.210000	3.263333	-0.996667	0.266667	-1.890000	2.043333	-1.106667	2.073333	1.110000	0.356667
228229	0.526667	2.233333	-1.840000	3.223333	0.753333	-0.216667	-0.283333	0.986667	-0.863333	2.183333	0.586667	1.030000
230343	-3.730000	4.930000	-2.636667	1.773333	-0.933333	2.586667	-5.583333	-11.783333	0.536667	2.136667	0.030000	0.590000
230497	-0.733333	0.730000	-0.640000	1.973333	0.040000	-0.040000	-0.536667	1.230000	-0.063333	2.126667	0.233333	1.000000
231658	-0.050000	1.396667	-0.476667	0.706667	0.610000	0.763333	-0.270000	-0.226667	0.480000	1.300000	0.340000	0.436667
233094	-0.650000	1.626667	-1.473333	1.476667	0.223333	-0.040000	-0.410000	0.436667	0.926667	2.306667	1.173333	0.783333
233851	-0.116667	1.170000	-0.216667	1.250000	0.803333	-0.063333	0.436667	0.483333	0.633333	1.390000	0.740000	0.720000
233890	0.343333	1.366667	-1.073333	2.836667	-0.730000	1.370000	0.983333	-0.150000	-0.400000	1.016667	0.856667	0.556667
235342	-4.423333	3.443333	-6.780000	4.630000	-2.240000	-0.106667	-4.733333	0.836667	-1.443333	3.980000	0.906667	0.556667
235731	-5.316667	4.356667	-6.246667	2.693333	-3.796667	-0.843333	-4.396667	4.700000	-0.676667	2.690000	0.666667	0.673333
236785	-3.730000	4.930000	-2.636667	1.773333	-0.933333	2.586667	-5.583333	-11.783333	0.536667	2.136667	0.030000	0.590000
238225	-4.790000	3.740000	-7.090000	4.613333	-2.486667	-0.150000	-4.976667	1.126667	-1.430000	3.896667	0.946667	0.540000
238307	0.156667	1.133333	0.470000	1.393333	-0.046667	0.500000	0.260000	0.986667	0.043333	1.590000	0.490000	0.873333
238578	-0.370000	0.863333	-1.133333	1.273333	0.386667	-0.313333	-0.070000	1.020000	0.123333	1.310000	0.290000	0.713333
238697	-4.866667	4.180000	-5.940000	2.673333	-3.676667	-0.710000	-4.040000	4.386667	-0.700000	2.776667	0.620000	0.686667
238918	-2.876667	3.083333	-5.226667	3.400000	-2.723333	-0.836667	-3.660000	2.926667	-0.886667	3.463333	0.780000	0.853333
239007	0.633333	1.600000	-1.473333	1.750000	0.123333	-0.256667	-0.103333	0.986667	0.033333	1.683333	0.176667	1.333333
239785	-4.240000	3.293333	-6.623333	4.636667	-2.116667	-0.086667	-4.610000	0.693333	-1.453333	4.020000	0.883333	0.563333
240568	-1.080000	0.873333	-1.260000	1.226667	0.546667	0.506667	-0.760000	0.433333	-0.323333	1.260000	0.263333	0.760000
242665	0.206667	1.760000	-1.370000	2.006667	-0.116667	-0.236667	-0.706667	1.123333	-0.253333	2.086667	0.166667	1.016667

As per the model there are only 90 out of 56962 transactions that are classified as potentially fraudulent transactions

Solving the problem using Anomaly Detection

The use of implementing an anomaly detection model in fraud detection is to automatically identify and flag unusual or potentially fraudulent activities within a dataset. It helps detect fraudulent transactions, behaviors, or events that deviate significantly from the expected or normal patterns, allowing organizations to take action and prevent financial losses or security breaches.

An anomaly detection algorithm in fraud detection is a specific method or technique used to find these anomalies or outliers within the data. It leverages statistical, machine learning, or deep learning approaches to distinguish between normal and abnormal data points. Examples of such algorithms include Isolation Forest, One-Class SVM, and clustering-based methods, which aim to identify suspicious activities that may indicate fraudulent behavior.

In [83]:
```
fraud_percent1 = (394/(227451+394))
fraud_percent1
```

Out[83]:
```
0.001729245759178389
```

In [84]:
```
# Considering the train data before balancing target variable using Random Over Sampling technique
```

In [85]:
```
x_train1, x_test1, y_train1, y_test1 = train_test_split(x_sc, y, test_size = 0.25, random_state = 101)
```

In [86]:
```
from sklearn.ensemble import IsolationForest

# Create an Isolation Forest instance
isolation_forest = IsolationForest(contamination=fraud_percent1)

# Fit the model to your train data
isolation_forest.fit(x_train1)
```

Out[86]:
```
IsolationForest(contamination=0.001729245759178389)
```

In [87]:
```
y_pred = isolation_forest.predict(x_test1)
```

In [88]:
```
# Count the number of outliers

n_outliers = np.sum(y_pred == -1)
print(n_outliers)
```

```
89
```

In [89]:
```
# Get the indices of the outliers

outlier_indices = np.where(y_pred == -1)[0]
print(outlier_indices)
```

```
[  247   409   969  1330  2467  2568  2817  3258  4301  4852  5881  6066
  6241  6789  7234  8325  8722  8750  9902 11852 12516 12538 12926 14063
 14395 16983 17331 19337 19857 20579 21046 21989 22265 22343 22531 22615
 23248 23567 24088 24482 25924 27428 27444 27728 29001 29088 29861 30012
 30035 31094 31982 32445 32481 32617 32861 33771 34950 37218 37418 37579
 38446 39831 41072 41206 41419 41634 42191 42738 42745 43521 44437 45360
 45705 47639 48144 48282 48774 50728 50812 51616 51618 52954 53907 54057
 54553 55183 55640 56344 56742]
```

In [90]:
```
test_df.drop('Predicted', axis=1, inplace = True)
```

In [91]:
```
# Predicting the Fraudulent transactions for test data

y_pred_test = isolation_forest.predict(scaler.transform(test_df))
```

In [92]:
```
# Number of Fraud Transactions as per the prediction model

n_outliers_test = np.sum(y_pred_test == -1)
print("Number of Fraud Transactions as per the prediction model out of 56962 transactions:",n_outliers_test)
```

```
Number of Fraud Transactions as per the prediction model out of 56962 transactions: 80
```

In [93]:
```
# Fetching the indices of the flagged transactions

outlier_indices_test = np.where(y_pred_test == -1)[0]
```

In [94]:
```
# Below are the 20 transactions out of 9765 flagged as fraudulent (potentially)

test_df.reset_index(drop=True, inplace = True)
selected_data = test_df.loc[outlier_indices_test]
selected_data.head(20)
```

Out[94]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	Per9	Dem1	Dem2	Dem3
276	-1.300000	2.146667	-2.210000	3.263333	-0.996667	0.266667	-1.890000	2.043333	-1.106667	2.073333	1.110000	0.356667
442	-2.370000	-4.483333	-1.570000	0.706667	-2.713333	2.963333	5.793333	0.280000	0.673333	0.890000	1.000000	0.526667
751	-4.473333	-3.760000	0.256667	2.106667	2.600000	0.356667	0.770000	-0.136667	1.833333	1.570000	1.533333	1.036667
1829	-4.213333	-8.850000	-3.570000	2.626667	-1.953333	1.490000	4.640000	-0.216667	-0.073333	0.780000	0.990000	0.690000
1986	-2.486667	-0.513333	-2.200000	0.813333	-4.326667	3.240000	4.780000	-0.150000	0.973333	0.476667	1.233333	0.946667
2498	-3.730000	4.930000	-2.636667	1.773333	-0.933333	2.586667	-5.583333	-11.783333	0.536667	2.136667	0.030000	0.590000
3495	-10.556667	-15.143333	-1.733333	4.020000	10.340000	-6.016667	-5.460000	1.266667	1.220000	0.866667	1.306667	1.086667
5331	-10.850000	-19.486667	-6.446667	6.293333	-5.743333	2.780000	8.806667	-0.986667	0.730000	0.640000	1.913333	1.526667
5729	-3.023333	3.893333	-1.473333	0.600000	-0.973333	2.263333	-4.473333	-7.506667	-0.306667	0.126667	-0.223333	0.400000
5829	-5.120000	-3.500000	-0.400000	1.666667	0.410000	1.083333	2.206667	-0.880000	2.600000	1.700000	0.653333	1.316667
5918	-6.356667	-7.983333	-0.540000	2.293333	5.310000	-2.473333	-2.626667	0.963333	0.866667	0.346667	0.873333	1.096667
6263	-8.753333	-8.183333	-4.643333	2.983333	-5.733333	5.153333	8.573333	-2.473333	2.420000	2.006667	1.596667	2.273333
7389	-5.393333	-8.700000	-1.496667	1.706667	3.503333	-2.130000	1.096667	0.650000	1.080000	0.653333	0.510000	0.550000
7497	-4.423333	3.443333	-6.780000	4.630000	-2.240000	-0.106667	-4.733333	0.836667	-1.443333	3.980000	0.906667	0.556667
7886	-5.316667	4.356667	-6.246667	2.693333	-3.796667	-0.843333	-4.396667	4.700000	-0.676667	2.690000	0.666667	0.673333
8343	-1.803333	-2.016667	0.463333	0.700000	1.456667	0.386667	1.580000	-0.623333	1.750000	1.253333	0.556667	0.686667
8940	-3.730000	4.930000	-2.636667	1.773333	-0.933333	2.586667	-5.583333	-11.783333	0.536667	2.136667	0.030000	0.590000
9277	-7.140000	-1.536667	-3.120000	2.703333	-0.766667	0.696667	-2.683333	-3.266667	0.350000	0.013333	0.656667	1.166667
9448	-3.556667	-2.210000	-2.106667	1.293333	-6.686667	5.556667	8.620000	-1.386667	1.506667	1.713333	0.980000	1.183333
9703	-8.130000	-5.253333	-2.760000	4.136667	-2.483333	2.740000	3.236667	-0.103333	2.500000	0.910000	1.273333	1.356667