Untitled13

June 2, 2024

[1]:
```
import os                   #reading or writing to the file system - File and␣
 ↪Directory Operations/ Platform Independence/ Path Manipulation/ Process␣
 ↪Management
import numpy as np          #Array and Matrix Operations/ Mathematical Functions/␣
 ↪Data Manipulation/ Integration with Other Libraries
import pandas as pd         #easily read, write, filter, and transform data
import matplotlib.pyplot as plt
%matplotlib inline          #Matplotlib provides a flexible foundation for␣
 ↪creating a wide range of static, animated, and interactive plots, while␣
 ↪Seaborn builds on Matplotlib to offer more aesthetically pleasing and␣
 ↪statistically sophisticated plots with simpler commands.
import seaborn as sns       #styling to plots, making them visually appealing and␣
 ↪consistent.
sns.set()
import warnings             # simply it show warn as message is "ignore"
warnings.filterwarnings("ignore")
```

[3]:
```
geo = pd.read_csv('Geo_scores.csv')          # simple code to read file
instance = pd.read_csv("instance_scores.csv")
lambdawts = pd.read_csv("Lambda_wts.csv")
qset = pd.read_csv("Qset_tats.csv")
test_data = pd.read_csv("test_share.csv")
train_data = pd.read_csv('train.csv')
```

[27]:
```
test_data = pd.read_csv("test_share.csv")
```

[5]:
```
print(geo.shape)     # row * column
print()              # for space only
print(instance.shape)
print()
print(lambdawts.shape)
print()
print(qset.shape)
print()
print(test_data.shape)
print()
print(train_data.shape)
```

```
(1424035, 2)

(1424035, 2)

(1400, 2)

(1424035, 2)

(56962, 27)

(227845, 28)
```

[6]:
```
print(geo.columns)              # display all columns

print(instance.columns)

print(lambdawts.columns)

print(qset.columns)

print(test_data.columns)

print(train_data.columns)
```

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

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

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

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

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

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

[9]:
```
print("geo id", geo['id'].nunique())        # print number of null value in "id"
print()
print("instance id", instance['id'].nunique())
print()
```

```
print("Lambda Group", lambdawts['Group'].nunique())
print()
print("qset id", qset['id'].nunique())
print()
print("Test id", test_data['id'].nunique())
print()
print("Train Id", train_data['id'].nunique())
print()
print("Test Group", test_data['Group'].nunique())
print()
print("Train Group",train_data['Group'].nunique())
```

```
geo id 284807

instance id 284807

Lambda Group 1400

qset id 284807

Test id 56962

Train Id 227845

Test Group 915

Train Group 1301
```

[30]:
```
test_data.head(1)  # display first row
```

[30]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	…	\
0	146574	Grp229	-0.3	1.54	0.22	-0.28	0.57	0.26	0.7	1.076667	…	

	Dem8	Dem9	Cred1	Cred2	Cred3	Cred4	Cred5	\
0	0.546667	0.313333	0.703333	0.813333	0.776667	0.796667	0.823333	

	Cred6	Normalised_FNT	data
0	0.783333	-249.75	test

[1 rows x 28 columns]

[10]:
```
train_data['data'] = 'train'   # add column data with each row fill by train
test_data['data'] = 'test'     # add column data with each row fill by test
```

[29]:
```
test_data['data'] = 'test'
```

[11]:
```
train_data.columns
```

[11]:
```
Index(['id', 'Group', 'Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7',
       'Per8', 'Per9', 'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7',
       'Dem8', 'Dem9', 'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6',
       'Normalised_FNT', 'Target', 'data'],
      dtype='object')
```

[12]:
```
test_data.columns
```

[12]:
```
Index(['id', 'Group', 'Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7',
       'Per8', 'Per9', 'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7',
       'Dem8', 'Dem9', 'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6',
       'Normalised_FNT', 'data'],
      dtype='object')
```

[13]:
```
train_data.head()
```

[13]:

	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	…	Dem9	Cred1	Cred2	Cred3	Cred4	\
0	0.340000	1.010000	…	0.726667	0.606667	1.010000	0.933333	0.603333	
1	0.810000	0.783333	…	0.743333	0.680000	0.690000	0.560000	0.670000	
2	0.056667	0.756667	…	0.820000	0.600000	0.383333	0.763333	0.670000	
3	0.956667	0.633333	…	0.900000	0.680000	0.846667	0.423333	0.520000	
4	0.853333	0.796667	…	0.486667	0.693333	0.526667	0.520000	0.716667	

	Cred5	Cred6	Normalised_FNT	Target	data
0	0.686667	0.673333	-245.7500	0	train
1	0.553333	0.653333	-248.0000	0	train
2	0.686667	0.673333	-233.1250	0	train
3	0.846667	0.760000	-249.7775	0	train
4	0.706667	0.673333	-247.5775	0	train

[5 rows x 29 columns]

[14]:
```
train_data.tail()
```

[14]:

	id	Group	Per1	Per2	Per3	Per4	Per5	\
227840	97346	Grp232	0.476667	1.013333	0.536667	0.576667	1.406667	
227841	147361	Grp199	1.363333	0.730000	0.060000	0.776667	0.883333	
227842	50989	Grp36	1.060000	0.756667	0.906667	0.896667	0.503333	
227843	149780	Grp445	0.433333	1.013333	1.163333	0.940000	0.930000	
227844	22175	Grp143	1.006667	0.553333	0.946667	1.206667	0.406667	

```
            Per6      Per7      Per8  …      Dem9     Cred1     Cred2  \
227840  1.846667  0.600000  1.103333  …  0.630000  0.633333  0.996667
227841  0.466667  0.733333  0.590000  …  0.356667  0.766667  0.730000
227842  0.396667  0.683333  0.620000  …  0.510000  0.740000  0.873333
227843  0.900000  0.813333  0.720000  …  0.606667  0.540000  0.643333
227844  0.750000  0.520000  0.756667  …  0.646667  0.636667  0.683333

            Cred3     Cred4     Cred5     Cred6  Normalised_FNT  Target   data
227840  0.646667  0.533333  0.680000  0.693333       -246.5025       0  train
227841  0.596667  0.730000  0.646667  0.656667       -249.7775       0  train
227842  0.700000  0.696667  0.663333  0.673333       -249.7775       0  train
227843  0.906667  0.540000  0.766667  0.710000       -242.7500       0  train
227844  0.843333  0.580000  0.683333  0.676667       -235.0000       0  train

[5 rows x 29 columns]
```

[ ]:

[ ]:

[23]:
```
test_data.head(1)
```

[23]:
```
       id   Group  Per1  Per2  Per3  Per4  Per5  Per6  Per7      Per8  …  \
0  56962  Grp229  -0.3  1.54  0.22 -0.28  0.57  0.26   0.7  1.076667  …

       Dem8      Dem9     Cred1     Cred2     Cred3     Cred4     Cred5  \
0  0.546667  0.313333  0.703333  0.813333  0.776667  0.796667  0.823333

      Cred6  Normalised_FNT  data
0  0.783333         -249.75  test

[1 rows x 28 columns]
```

[25]:
```
del test_data
```

[26]:
```
test_data
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 test_data

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

[31]:
```
all_data = pd.concat([train_data, test_data], axis=0)
# add rows-wise (vertically)
```

```
/*
    'id': [1, 2, 3],
    'feature1': [10, 20, 30],
    'feature2': ['A', 'B', 'C'],
    'label': [0, 1, 0]

        'id': [4, 5, 6],
    'feature1': [40, 50, 60],
    'feature2': ['D', 'E', 'F']

   id  feature1 feature2  label
0   1        10        A    0.0
1   2        20        B    1.0
2   3        30        C    0.0
3   4        40        D    NaN
4   5        50        E    NaN
5   6        60        F    NaN


*/
```

[32]:
```
all_data.head()
```

[32]:
```
       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  …      Dem9     Cred1     Cred2     Cred3     Cred4  \
0  0.340000  1.010000  …  0.726667  0.606667  1.010000  0.933333  0.603333
1  0.810000  0.783333  …  0.743333  0.680000  0.690000  0.560000  0.670000
2  0.056667  0.756667  …  0.820000  0.600000  0.383333  0.763333  0.670000
3  0.956667  0.633333  …  0.900000  0.680000  0.846667  0.423333  0.520000
4  0.853333  0.796667  …  0.486667  0.693333  0.526667  0.520000  0.716667

      Cred5     Cred6  Normalised_FNT  Target   data
0  0.686667  0.673333        -245.7500     0.0  train
1  0.553333  0.653333        -248.0000     0.0  train
2  0.686667  0.673333        -233.1250     0.0  train
3  0.846667  0.760000        -249.7775     0.0  train
4  0.706667  0.673333        -247.5775     0.0  train

[5 rows x 29 columns]
```

[71]:
```
# (56962, 27)

# (227845, 28)

all_data.shape
```

[71]:
```
(284807, 32)
```

[34]:
```
all_data.columns
```

[34]:
```
Index(['id', 'Group', 'Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7',
       'Per8', 'Per9', 'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7',
       'Dem8', 'Dem9', 'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6',
       'Normalised_FNT', 'Target', 'data'],
      dtype='object')
```

[35]:
```
train_data.head(1)
```

[35]:

	id	Group	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	…	Dem9	Cred1	Cred2	Cred3	Cred4	Cred5	Cred6	Normalised_FNT	Target	data
0	112751	Grp169	1.07	0.58	0.48	0.766667	1.233333	1.993333	0.34	1.01	…	0.726667	0.606667	1.01	0.933333	0.603333	0.686667	0.673333	-245.75	0	train

[1 rows x 29 columns]

[36]:
```
print("all_data id", all_data['id'].nunique())  # 284807 null rows at id column
print()
print("all_data group", all_data['Group'].nunique())  # 1400 null rows at id␣
 ↪column
```

```
all_data id 284807

all_data group 1400
```

[37]:
```
print(geo.isnull().sum())  # in each column - total null value
print()
print(instance.isnull().sum())
print()
print(lambdawts.isnull().sum())
print()
print(qset.isnull().sum())
print()
print(all_data.isnull().sum())
```

```
id               0
geo_score    71543
dtype: int64

id                 0
instance_scores    0
dtype: int64

Group        0
lambda_wt    0
dtype: int64

id                           0
qsets_normalized_tat    103201
dtype: int64

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

[38]:
```
geo.head(1)
```

[38]:
```
       id  geo_score
0  26674       4.48
```

[39]:
```
print(geo.describe())
print()
print(qset.describe())
```

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

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

[40]:
```
geo['geo_score'] = geo['geo_score'].fillna(geo['geo_score'].median())

# fillna() method is used to replace all NaN values in the geo_score column␣
 ↪with the median value calculated in the previous step.
qset['qsets_normalized_tat'] = qset['qsets_normalized_tat'].
 ↪fillna(qset['qsets_normalized_tat'].median())
```

[41]:
```
geo.shape
```

[41]:
```
(1424035, 2)
```

[42]:
```
geo['id'].nunique()   # count the number of unique values in each column of a␣
 ↪DataFrame
```

[42]:
```
284807
```

[43]:
```
geo = geo.groupby('id').mean()
/*performs a group-by operation on the geo DataFrame
using the id column and then calculates the mean of each group.
```

```
*/
```

[44]:
```
geo.shape
```

[44]:
```
(284807, 1)
```

[45]:
```
geo
```

[45]:
```
        geo_score
id
0          -0.620
1           1.106
2           0.070
3           0.180
4           0.540
…           …
284802      2.710
284803      0.956
284804      0.060
284805     -0.960
284806     -0.030

[284807 rows x 1 columns]
```

[46]:
```
qset = qset.groupby('id').mean()    # grouping and than taking mean
```

[50]:
```
qset.head(1)
```

[50]:
```
    qsets_normalized_tat
id
0                  0.214
```

[47]:
```
qset.shape
```

[47]:
```
(284807, 1)
```

[48]:
```
instance.shape
```

[48]:
```
(1424035, 2)
```

[49]:
```
instance = instance.groupby('id').mean()
```

[51]:
```
instance.head(1)
```

[51]:
```
    instance_scores
id
0              0.09
```

[52]:
```
instance.shape
```

[52]:
```
(284807, 1)
```

[53]:
```
lambdawts.shape
```

[53]:
```
(1400, 2)
```

[54]:
```
print(geo.shape)
print()
print(instance.shape)
print()
print(lambdawts.shape)
print()
print(qset.shape)
print()
print(all_data.shape)
```

```
(284807, 1)

(284807, 1)

(1400, 2)

(284807, 1)

(284807, 29)
```

[55]:
```
all_data.head()
```

[55]:

	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	…	Dem9	Cred1	Cred2	Cred3	Cred4	\
0	0.340000	1.010000	…	0.726667	0.606667	1.010000	0.933333	0.603333	
1	0.810000	0.783333	…	0.743333	0.680000	0.690000	0.560000	0.670000	
2	0.056667	0.756667	…	0.820000	0.600000	0.383333	0.763333	0.670000	
3	0.956667	0.633333	…	0.900000	0.680000	0.846667	0.423333	0.520000	
4	0.853333	0.796667	…	0.486667	0.693333	0.526667	0.520000	0.716667	

	Cred5	Cred6	Normalised_FNT	Target	data
0	0.686667	0.673333	-245.7500	0.0	train
1	0.553333	0.653333	-248.0000	0.0	train
2	0.686667	0.673333	-233.1250	0.0	train

```
3  0.846667  0.760000       -249.7775     0.0  train
4  0.706667  0.673333       -247.5775     0.0  train

[5 rows x 29 columns]
```

[56]:
```
instance.head()
```

[56]:
```
    instance_scores
id
0              0.09
1             -0.17
2              0.21
3             -0.05
4              0.75
```

[57]:
```
all_data = pd.merge(all_data,instance , on='id', how='left')
```

[58]:
```
all_data.head(1)
```

[58]:
```
       id   Group  Per1  Per2  Per3      Per4      Per5      Per6  Per7  Per8  \
0  112751  Grp169  1.07  0.58  0.48  0.766667  1.233333  1.993333  0.34  1.01

   …     Cred1  Cred2     Cred3     Cred4     Cred5     Cred6  \
0  …  0.606667   1.01  0.933333  0.603333  0.686667  0.673333

   Normalised_FNT  Target   data  instance_scores
0         -245.75     0.0  train            -0.06

[1 rows x 30 columns]
```

[59]:
```
qset.head(2)
```

[59]:
```
    qsets_normalized_tat
id
0                  0.214
1                 -0.110
```

[60]:
```
all_data.shape
```

[60]:
```
(284807, 30)
```

[72]:
```
all_data.columns
```

[72]:
```
Index(['id', 'Group', 'Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7',
       'Per8', 'Per9', 'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7',
       'Dem8', 'Dem9', 'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6',
       'Normalised_FNT', 'Target', 'data', 'instance_scores',
       'qsets_normalized_tat', 'lambda_wt'],
```

```
        dtype='object')
```

[61]:
```
all_data['Group'].nunique()
```

[61]:
```
1400
```

[62]:
```
all_data = pd.merge(all_data,qset , on='id', how='left')
```

[73]:
```
all_data.shape
```

[73]:
```
(284807, 32)
```

[74]:
```
all_data.head(1)
```

[74]:
```
        id   Group  Per1  Per2  Per3      Per4      Per5      Per6  Per7  Per8  \
0  112751  Grp169  1.07  0.58  0.48  0.766667  1.233333  1.993333  0.34  1.01

   …     Cred3     Cred4     Cred5     Cred6  Normalised_FNT  Target   data  \
0  …  0.933333  0.603333  0.686667  0.673333         -245.75     0.0  train

   instance_scores  qsets_normalized_tat  lambda_wt
0            -0.06                  -0.7      -0.13

[1 rows x 32 columns]
```

[63]:
```
lambdawts.head(2)
```

[63]:
```
    Group  lambda_wt
0  Grp936       3.41
1  Grp347      -2.88
```

[64]:
```
lambdawts.shape
```

[64]:
```
(1400, 2)
```

[65]:
```
lambdawts['Group'].nunique()
```

[65]:
```
1400
```

[66]:
```
all_data = pd.merge(all_data,lambdawts , on='Group', how='left')
```

[67]:
```
all_data.head()
```

[67]:
```
        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  …     Cred3     Cred4     Cred5     Cred6  \
0  0.340000  1.010000  …  0.933333  0.603333  0.686667  0.673333
1  0.810000  0.783333  …  0.560000  0.670000  0.553333  0.653333
2  0.056667  0.756667  …  0.763333  0.670000  0.686667  0.673333
3  0.956667  0.633333  …  0.423333  0.520000  0.846667  0.760000
4  0.853333  0.796667  …  0.520000  0.716667  0.706667  0.673333

   Normalised_FNT  Target   data  instance_scores  qsets_normalized_tat  \
0       -245.7500     0.0  train            -0.06                -0.700
1       -248.0000     0.0  train             0.52                 0.140
2       -233.1250     0.0  train             1.56                -0.430
3       -249.7775     0.0  train             0.70                -0.302
4       -247.5775     0.0  train            -0.47                -0.630

   lambda_wt
0      -0.13
1       0.66
2      -0.51
3       0.72
4       0.60

[5 rows x 32 columns]
```

[68]:
```
all_data['lambda_wt'].count()
/*expression all_data['lambda_wt'].count() is used to count the number of␣
 ↪non-null
(non-missing) entries in the column 'lambda_wt' of the DataFrame all_data.*/
```

[68]: 284807

[75]:
```
all_data['lambda_wt'].nunique()
```

[75]: 1400

[76]:
```
train_data = all_data[all_data['data']=='train']
test_data = all_data[all_data['data']=='test']
```

[77]:
```
train_data.shape
```

[77]: (227845, 32)

[78]:
```
test_data.shape
```

[78]: (56962, 32)

[79]:
```
train_data.columns
```

[79]:
```
Index(['id', 'Group', 'Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7',
       'Per8', 'Per9', 'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7',
       'Dem8', 'Dem9', 'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6',
       'Normalised_FNT', 'Target', 'data', 'instance_scores',
       'qsets_normalized_tat', 'lambda_wt'],
      dtype='object')
```

[80]:
```
plt.figure(figsize=(20,12)) #This creates a new figure with a specified size of␣
 ↪20 inches in width and 12 inches in height
sns.heatmap(train_data.corr(), annot=True, cmap='coolwarm')
/*The given code snippet is used to create a heatmap of the correlation matrix␣
 ↪of a DataFrame `train_data` using Seaborn and Matplotlib. This visualization␣
 ↪helps in understanding the relationships between different features in the␣
 ↪dataset. Here is a detailed explanation of each step involved:

### Code Breakdown

1. **Setting the Figure Size**:
   ```python
   plt.figure(figsize=(20, 12))
   ```
   - `plt.figure(figsize=(20, 12))`: This creates a new figure with a specified␣
 ↪size of 20 inches in width and 12 inches in height. The `figsize` parameter␣
 ↪controls the dimensions of the figure.

2. **Creating the Heatmap**:
   ```python
   sns.heatmap(train_data.corr(), annot=True, cmap='coolwarm')
   ```
   - `sns.heatmap(...)`: This function from the Seaborn library is used to␣
 ↪create a heatmap.
   - `train_data.corr()`: This calculates the correlation matrix of the␣
 ↪DataFrame `train_data`. The correlation matrix is a table showing␣
 ↪correlation coefficients between variables. Each cell in the table shows the␣
 ↪correlation between two variables.
   - `annot=True`: This parameter adds the correlation coefficient values␣
 ↪inside the cells of the heatmap, making it easier to read the exact values.
   - `cmap='coolwarm'`: This specifies the color map to use for the heatmap.␣
 ↪The 'coolwarm' color map displays a gradient from cool (blue) to warm (red)␣
 ↪colors, which helps in visually distinguishing the correlation values.

3. **Displaying the Plot**:
   ```python
   plt.show()
   ```
   - `plt.show()`: This function from the Matplotlib library displays the␣
 ↪figure.
```

```
### Detailed Example

To make this more concrete, let's go through an example with sample data.

#### Example

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Create a sample DataFrame
np.random.seed(0)
data = {
    'A': np.random.rand(100),
    'B': np.random.rand(100),
    'C': np.random.rand(100),
    'D': np.random.rand(100),
    'E': np.random.rand(100)
}
train_data = pd.DataFrame(data)

# Add some correlation between columns for illustration
train_data['B'] = train_data['A'] * 0.5 + np.random.rand(100) * 0.5
train_data['C'] = train_data['A'] * (-0.5) + np.random.rand(100) * 0.5

# Plotting the heatmap
plt.figure(figsize=(20, 12))
sns.heatmap(train_data.corr(), annot=True, cmap='coolwarm') #train_data.corr(): This computes the Pearson correlation coefficient between all pairs of columns in train_data.
plt.show()
```

Explanation of the Example

1. **Creating the DataFrame**:
   - `data` dictionary contains random values generated using `np.random.rand(100)` for columns `A`, `B`, `C`, `D`, and `E`.
   - `train_data` DataFrame is created from this dictionary.
   - To illustrate correlations, columns `B` and `C` are adjusted to have some correlation with `A`.

2. **Calculating the Correlation Matrix**:
```

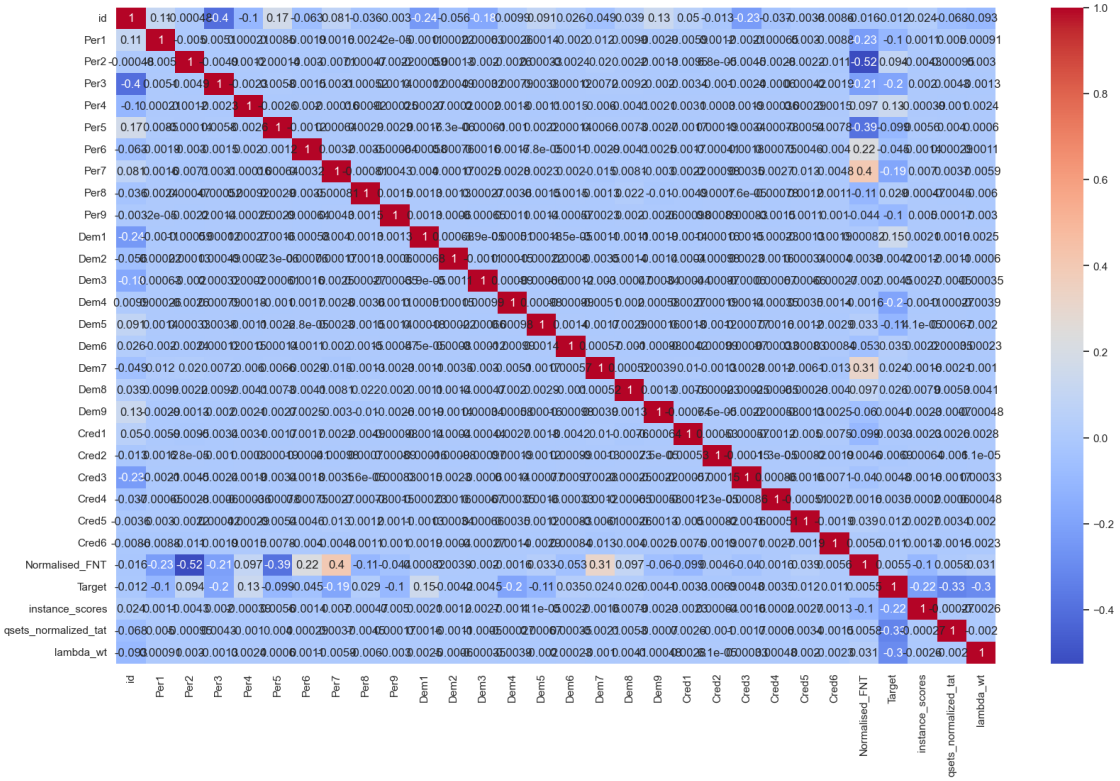
```
   - `train_data.corr()`: This computes the Pearson correlation coefficient␣
 ↪between all pairs of columns in `train_data`.

3. **Plotting the Heatmap**:
   - `plt.figure(figsize=(20, 12))`: This sets the size of the figure.
   - `sns.heatmap(train_data.corr(), annot=True, cmap='coolwarm')`: This␣
 ↪creates the heatmap with annotations (correlation values) and a 'coolwarm'␣
 ↪color scheme.
   - `plt.show()`: This displays the plot.

### Interpretation of the Heatmap

- **Diagonal**: The diagonal elements of the heatmap are all 1, as each␣
 ↪variable is perfectly correlated with itself.
- **Off-diagonal**: The off-diagonal elements show the correlation between␣
 ↪different variables.
  - Positive values indicate a positive correlation.
  - Negative values indicate a negative correlation.
  - Values close to 1 or -1 indicate strong correlation, while values close to␣
 ↪0 indicate weak or no correlation.
- **Color Map**: The 'coolwarm' color map helps to visually distinguish␣
 ↪positive correlations (red shades) from negative correlations (blue shades).

By using this heatmap, you can quickly identify relationships between different␣
 ↪features in your dataset, which is useful for feature selection,␣
 ↪understanding data structure, and identifying potential multicollinearity␣
 ↪issues.*/
plt.show()
```



[81]:
```
# splitting the data into independent and dependent variable
x = train_data.drop(['id', 'Group','Target', 'data'], axis=1) # ind variable
y = train_data['Target'] # dependent
```

[82]:
```
x.head(2)
```

[82]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	Per8	\
0	1.070000	0.580000	0.480000	0.766667	1.233333	1.993333	0.34	1.010000	
1	0.473333	1.206667	0.883333	1.430000	0.726667	0.626667	0.81	0.783333	

	Per9	Dem1	…	Cred1	Cred2	Cred3	Cred4	Cred5	\
0	0.863333	0.46	…	0.606667	1.01	0.933333	0.603333	0.686667	
1	0.190000	0.47	…	0.680000	0.69	0.560000	0.670000	0.553333	

	Cred6	Normalised_FNT	instance_scores	qsets_normalized_tat	lambda_wt
0	0.673333	-245.75	-0.06	-0.70	-0.13
1	0.653333	-248.00	0.52	0.14	0.66

[2 rows x 28 columns]

[83]:
```
x.columns
```

[83]:
```
Index(['Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7', 'Per8', 'Per9',
       'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7', 'Dem8', 'Dem9',
       'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6', 'Normalised_FNT',
       'instance_scores', 'qsets_normalized_tat', 'lambda_wt'],
      dtype='object')
```

[84]:
```
x.shape
```

[84]:
```
(227845, 28)
```

[85]:
```
y.head()
```

[85]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: Target, dtype: float64
```

[86]:
```
test_data.columns
```

[86]:
```
Index(['id', 'Group', 'Per1', 'Per2', 'Per3', 'Per4', 'Per5', 'Per6', 'Per7',
       'Per8', 'Per9', 'Dem1', 'Dem2', 'Dem3', 'Dem4', 'Dem5', 'Dem6', 'Dem7',
       'Dem8', 'Dem9', 'Cred1', 'Cred2', 'Cred3', 'Cred4', 'Cred5', 'Cred6',
       'Normalised_FNT', 'Target', 'data', 'instance_scores',
       'qsets_normalized_tat', 'lambda_wt'],
      dtype='object')
```

[87]:
```
test_data.isnull().sum()/len(test_data)*100
```

[87]:
```
id                      0.0
Group                   0.0
Per1                    0.0
Per2                    0.0
Per3                    0.0
Per4                    0.0
Per5                    0.0
Per6                    0.0
Per7                    0.0
Per8                    0.0
Per9                    0.0
Dem1                    0.0
Dem2                    0.0
Dem3                    0.0
Dem4                    0.0
Dem5                    0.0
Dem6                    0.0
Dem7                    0.0
```

```
Dem8                      0.0
Dem9                      0.0
Cred1                     0.0
Cred2                     0.0
Cred3                     0.0
Cred4                     0.0
Cred5                     0.0
Cred6                     0.0
Normalised_FNT            0.0
Target                  100.0
data                      0.0
instance_scores           0.0
qsets_normalized_tat      0.0
lambda_wt                 0.0
dtype: float64
```

[88]:
```
test_data = test_data.drop(['id','Group','Target','data'], axis=1)
/*axis=1: Specifies that the operation is to be performed along the columns. If␣
 ↪axis=0, it would drop rows.*/
```

[89]:
```
# Task :
# This data is for prediction whether listed customer will do fraudulent or not
test_data.head()
```

[89]:

	Per1	Per2	Per3	Per4	Per5	Per6	Per7	\
227845	-0.300000	1.540000	0.220000	-0.280000	0.570000	0.260000	0.700000	
227846	0.633333	0.953333	0.810000	0.466667	0.910000	0.253333	1.040000	
227847	1.043333	0.740000	0.860000	1.006667	0.583333	0.616667	0.630000	
227848	1.283333	0.300000	0.576667	0.636667	0.256667	0.543333	0.356667	
227849	1.186667	0.326667	0.476667	0.866667	0.436667	0.680000	0.476667	

	Per8	Per9	Dem1	…	Cred1	Cred2	Cred3	\
227845	1.076667	0.930000	0.156667	…	0.703333	0.813333	0.776667	
227846	0.550000	0.543333	0.433333	…	0.536667	0.703333	0.806667	
227847	0.686667	0.593333	1.250000	…	0.623333	0.753333	0.870000	
227848	0.663333	1.156667	1.186667	…	0.800000	0.606667	0.456667	
227849	0.686667	1.476667	1.213333	…	0.670000	0.896667	0.566667	

	Cred4	Cred5	Cred6	Normalised_FNT	instance_scores	\
227845	0.796667	0.823333	0.783333	-249.7500	-0.04	
227846	0.630000	0.673333	0.673333	-249.8125	-0.77	
227847	0.596667	0.680000	0.670000	-248.1200	0.11	
227848	0.320000	0.676667	0.660000	-222.9875	0.33	
227849	0.546667	0.650000	0.663333	-196.2200	-0.37	

	qsets_normalized_tat	lambda_wt
227845	-0.426	0.76

```
227846                -0.620       0.18
227847                -0.406       0.39
227848                 0.374       1.80
227849                -0.130       1.89

[5 rows x 28 columns]
```

[92]:
```
# Actual Data */
```

[93]:
```
x.head()
```

[93]:
```
       Per1      Per2      Per3      Per4      Per5      Per6      Per7  \
0  1.070000  0.580000  0.480000  0.766667  1.233333  1.993333  0.340000
1  0.473333  1.206667  0.883333  1.430000  0.726667  0.626667  0.810000
2  1.130000  0.143333  0.946667  0.123333  0.080000  0.836667  0.056667
3  0.636667  1.090000  0.750000  0.940000  0.743333  0.346667  0.956667
4  0.560000  1.013333  0.593333  0.416667  0.773333  0.460000  0.853333

       Per8      Per9      Dem1  …     Cred1     Cred2     Cred3     Cred4  \
0  1.010000  0.863333  0.460000  …  0.606667  1.010000  0.933333  0.603333
1  0.783333  0.190000  0.470000  …  0.680000  0.690000  0.560000  0.670000
2  0.756667  0.226667  0.660000  …  0.600000  0.383333  0.763333  0.670000
3  0.633333  0.486667  1.096667  …  0.680000  0.846667  0.423333  0.520000
4  0.796667  0.516667  0.756667  …  0.693333  0.526667  0.520000  0.716667

      Cred5     Cred6  Normalised_FNT  instance_scores  qsets_normalized_tat  \
0  0.686667  0.673333        -245.7500            -0.06                -0.700
1  0.553333  0.653333        -248.0000             0.52                 0.140
2  0.686667  0.673333        -233.1250             1.56                -0.430
3  0.846667  0.760000        -249.7775             0.70                -0.302
4  0.706667  0.673333        -247.5775            -0.47                -0.630

   lambda_wt
0      -0.13
1       0.66
2      -0.51
3       0.72
4       0.60

[5 rows x 28 columns]
```

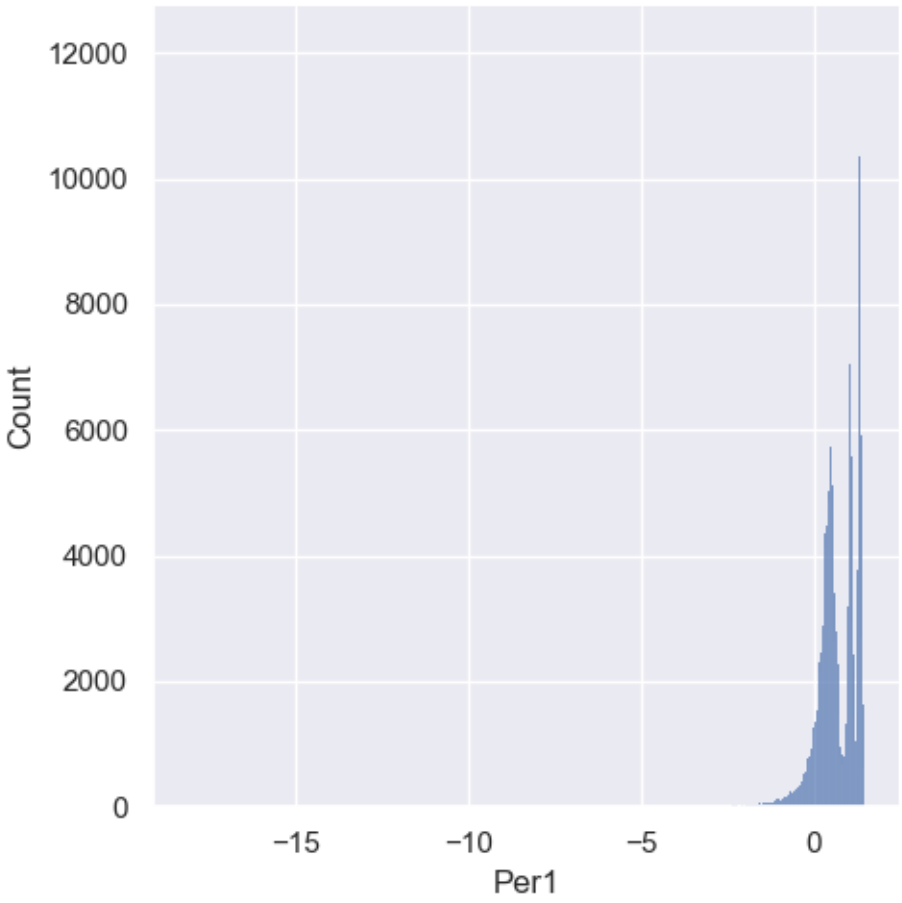
[94]:
```
x.isnull().any()
```

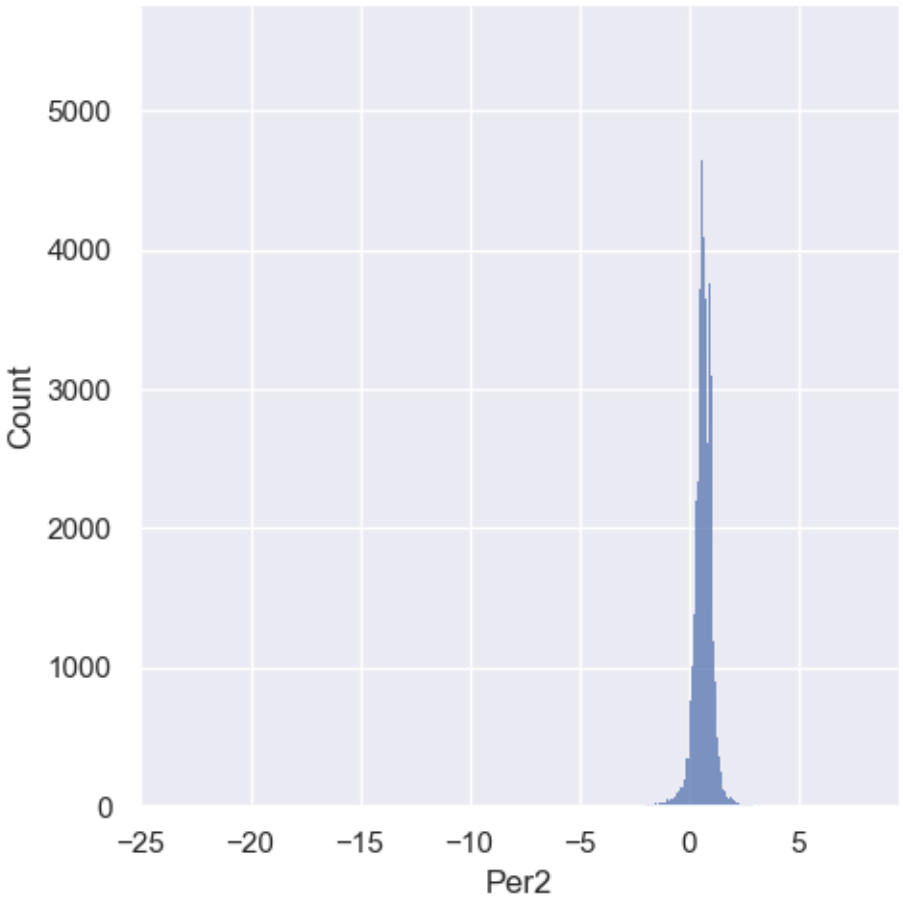
[94]:
```
Per1                  False
Per2                  False
Per3                  False
Per4                  False
```

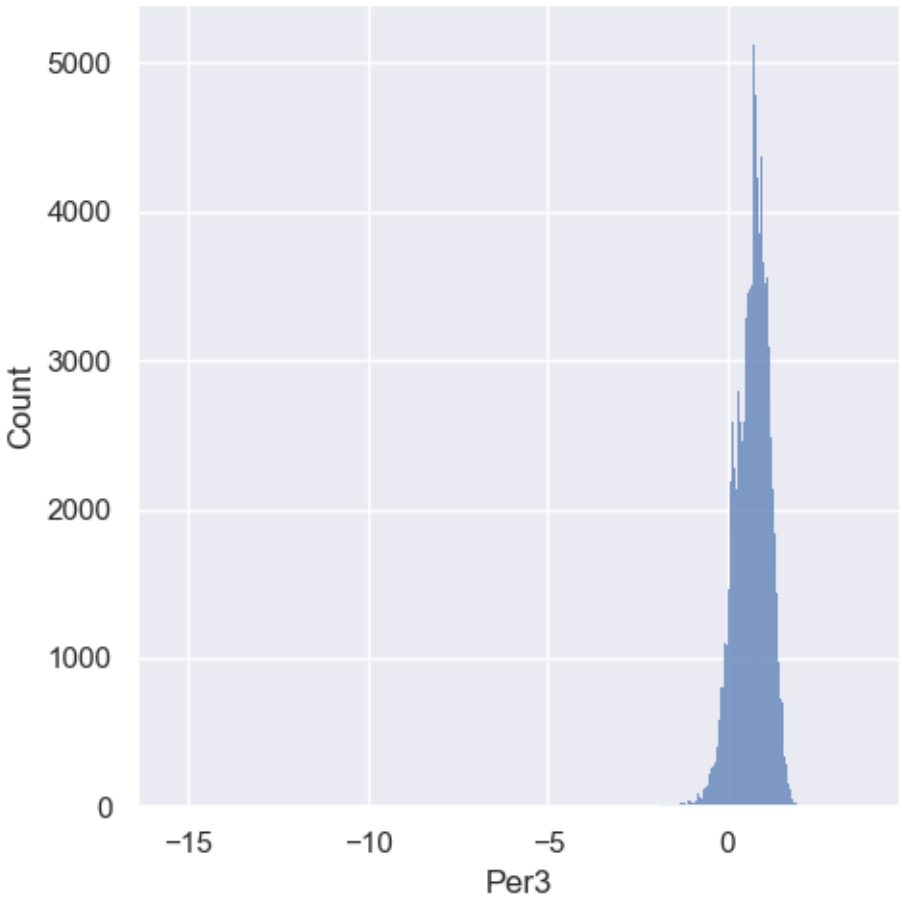
```
Per5                    False
Per6                    False
Per7                    False
Per8                    False
Per9                    False
Dem1                    False
Dem2                    False
Dem3                    False
Dem4                    False
Dem5                    False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

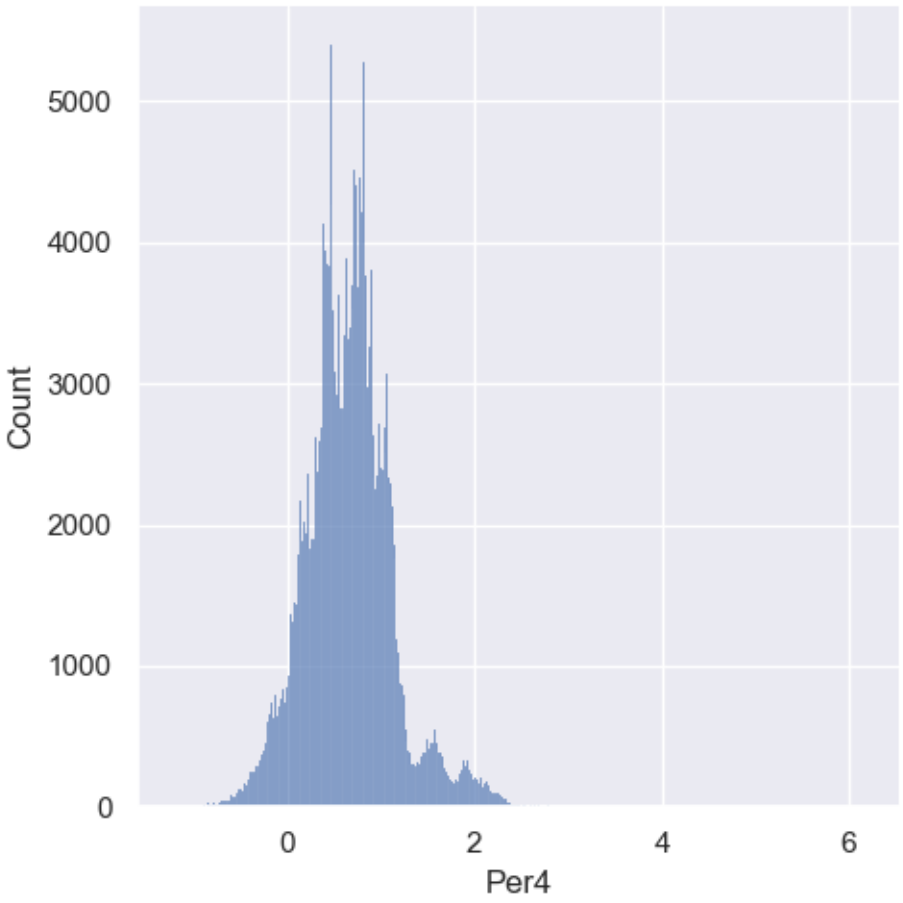
[ ]:
```
def distplots(col):
    sns.displot(x[col])
    plt.show()

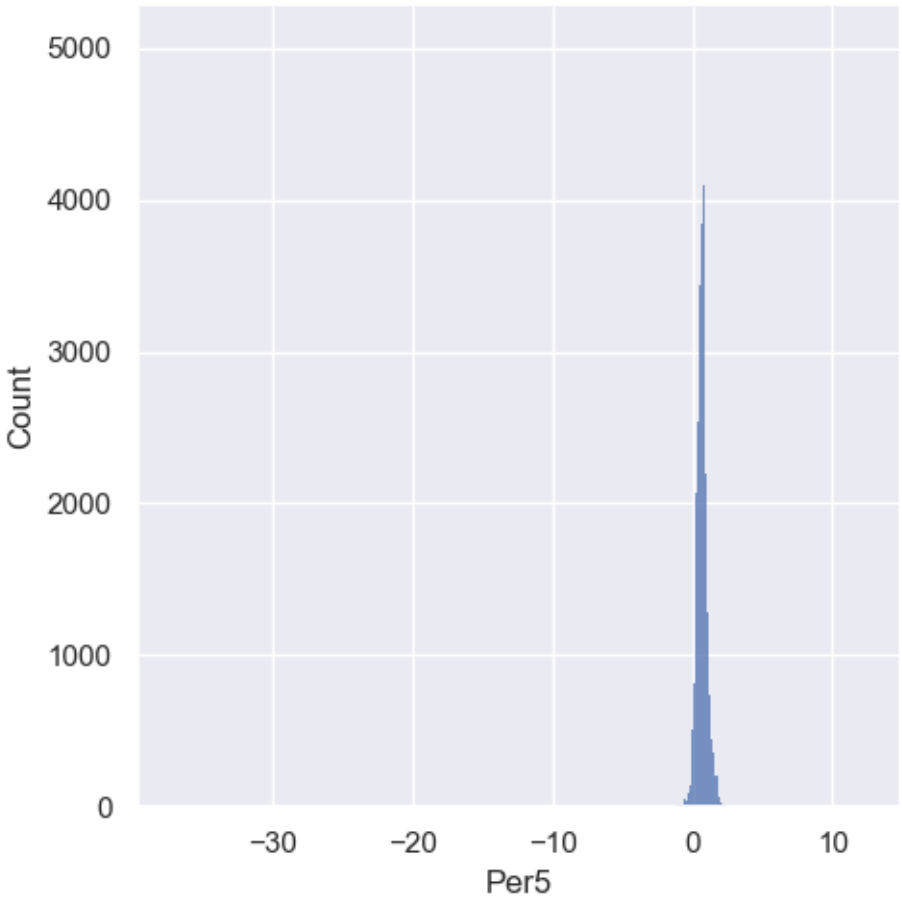
for i in list(x.columns)[0:]:
    distplots(i)
```

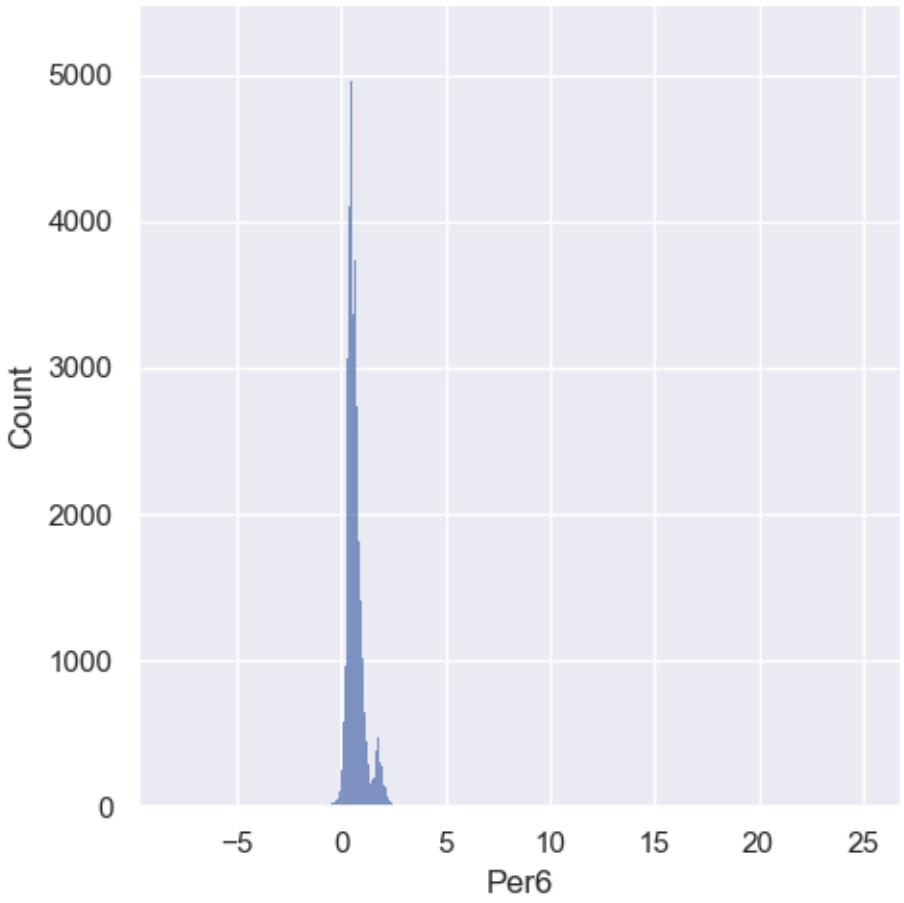

12000
10000
8000
6000
4000
2000
0
Count
−15
−10
−5
0
Per1

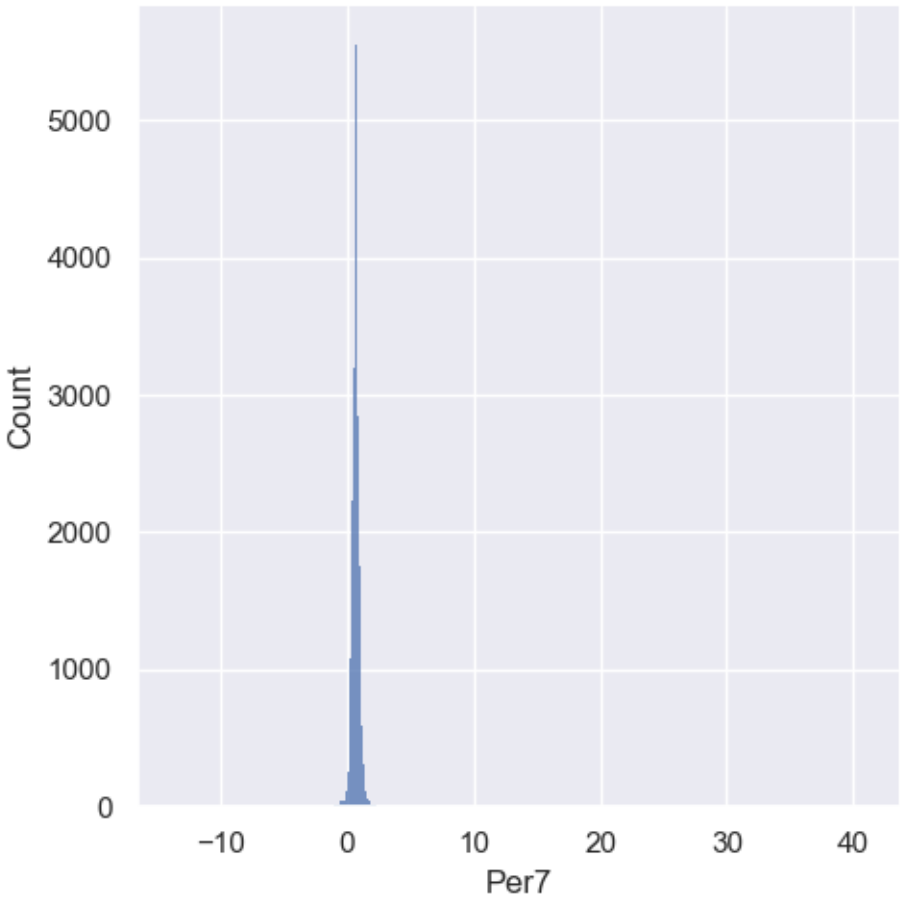

5000
4000
3000
2000
1000
0
Count
−25
−20
−15
−10
−5
0
5
Per2

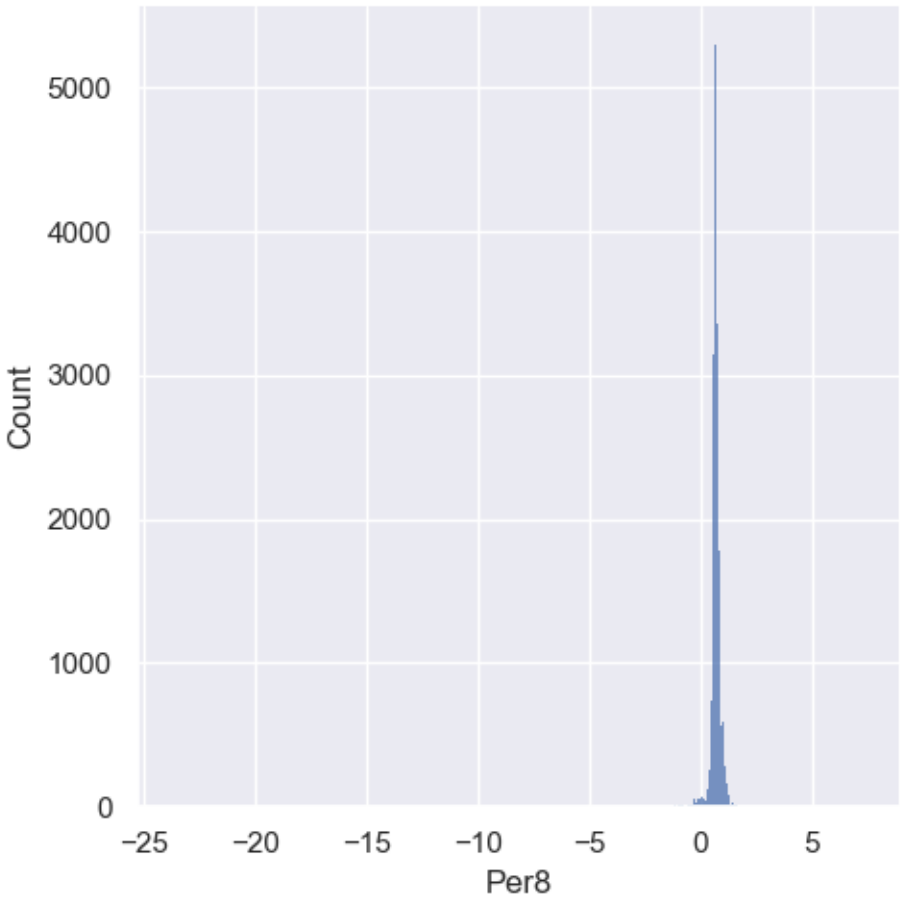

5000
4000
3000
2000
1000
0
Count
−15
−10
−5
0
Per3

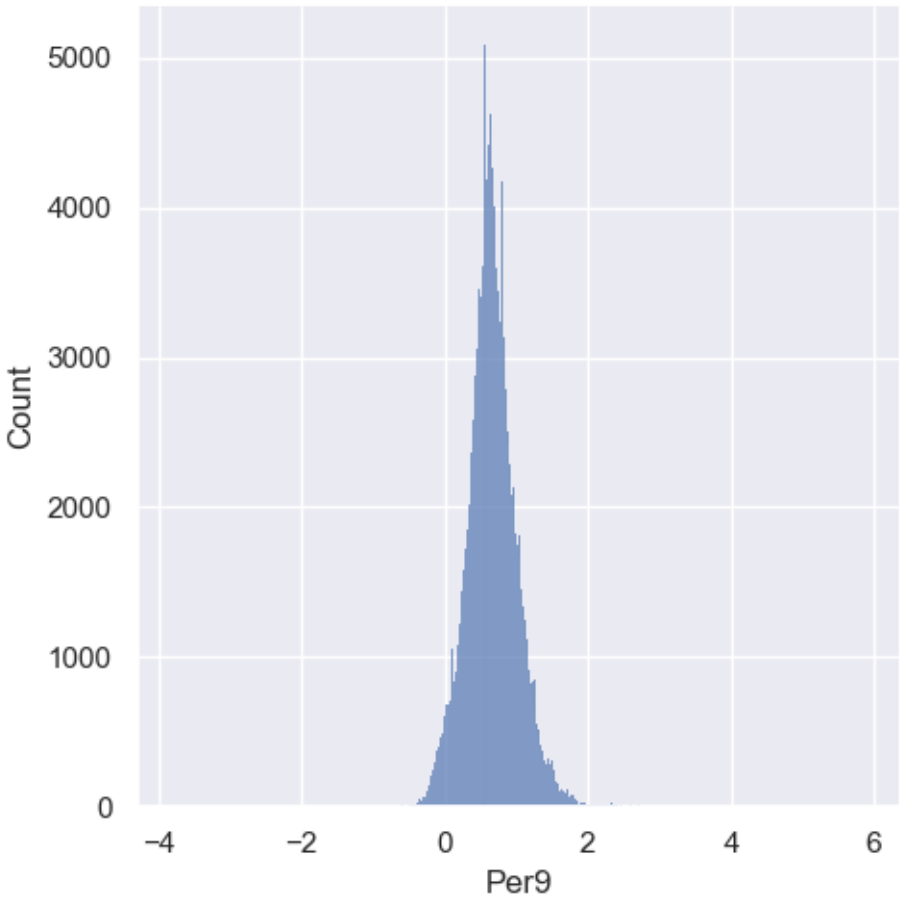

5000
4000
3000
2000
1000
0
Count
0
2
4
6
Per4

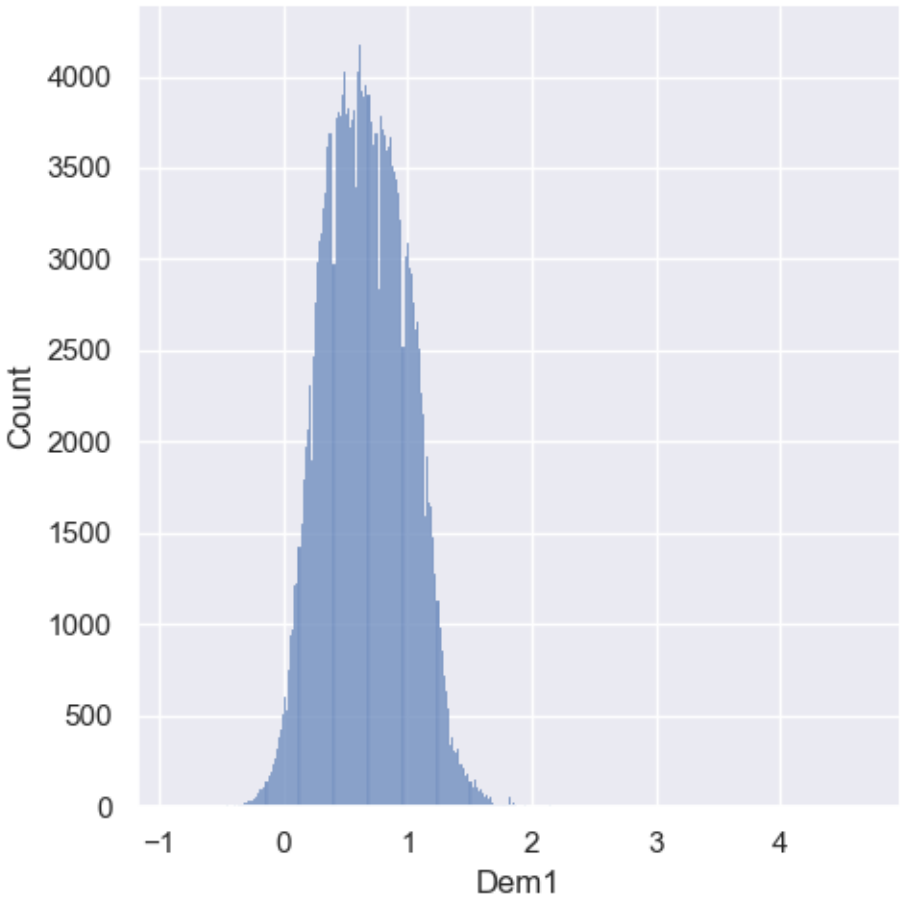

5000
4000
3000
2000
1000
0
Count
−30
−20
−10
0
10
Per5

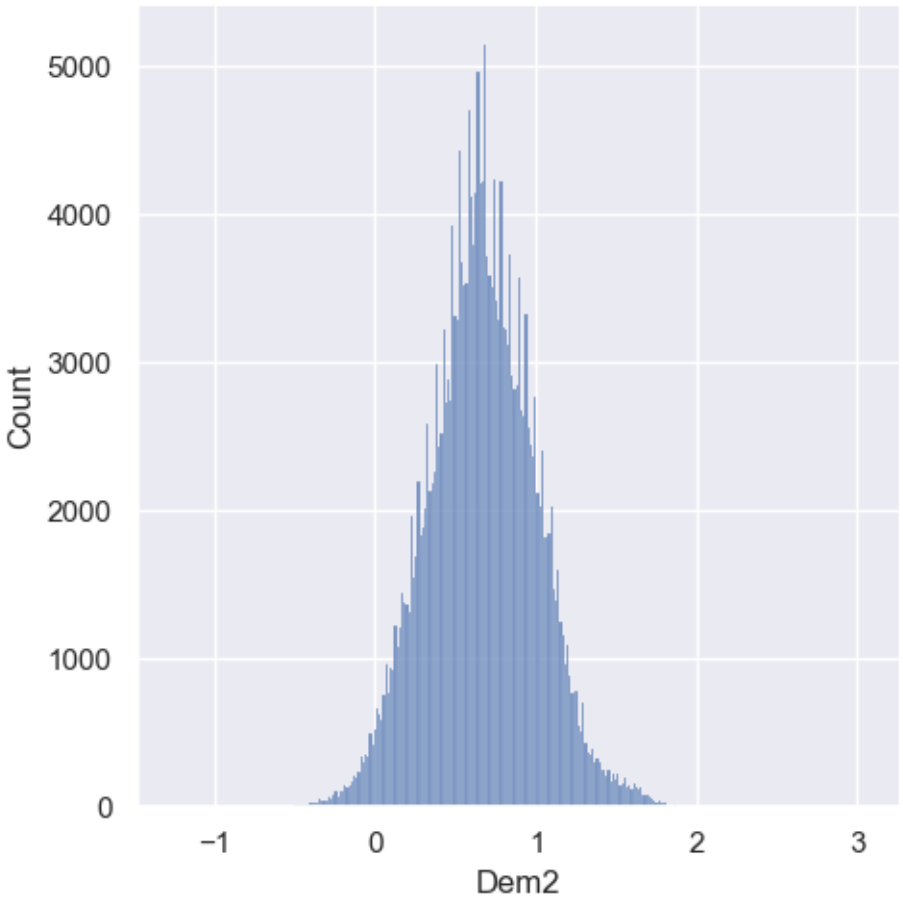

5000
4000
3000
2000
1000
0
Count
−5
0
5
10
15
20
25
Per6

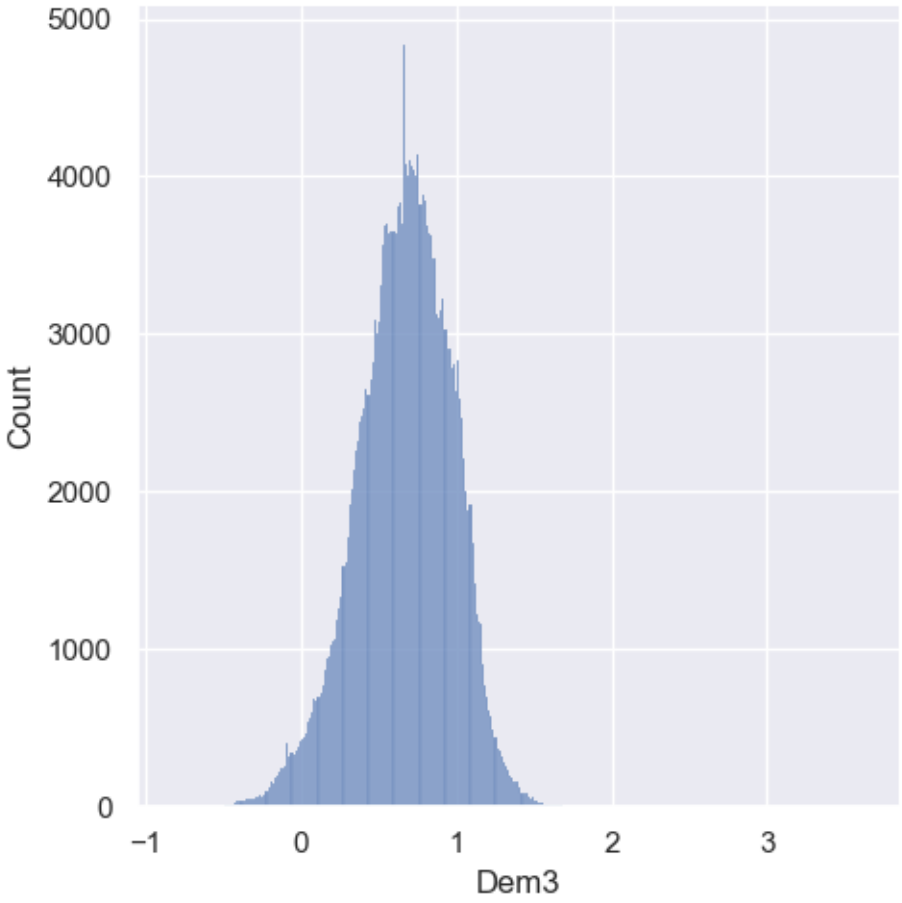

5000
4000
3000
2000
1000
0
Count
−10
0
10
20
30
40
Per7

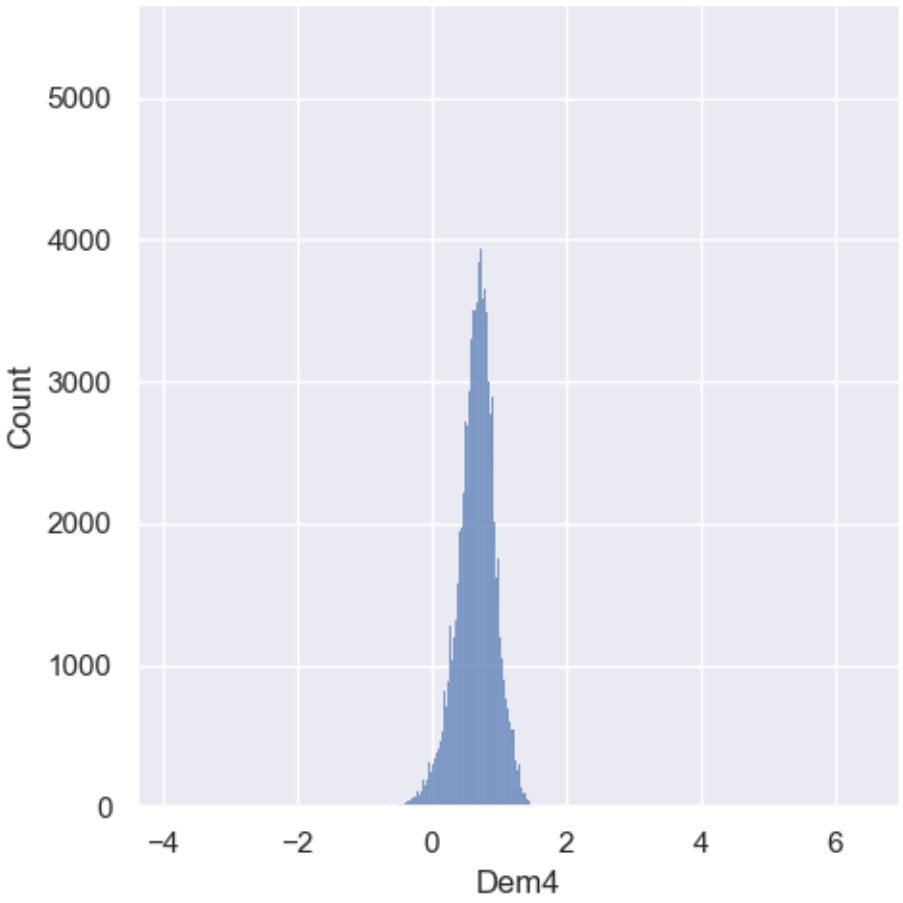

5000
4000
3000
2000
1000
0
Count
−25
−20
−15
−10
−5
0
5
Per8

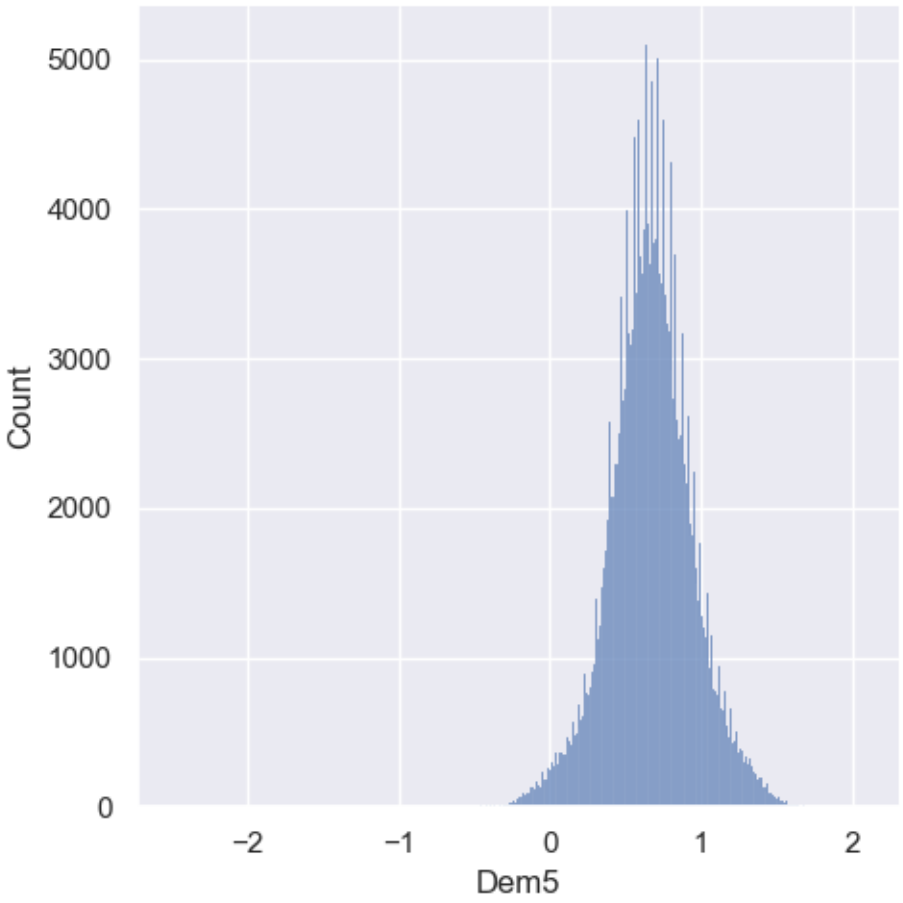

5000
4000
3000
2000
1000
0
Count
−4
−2
0
2
4
6
Per9

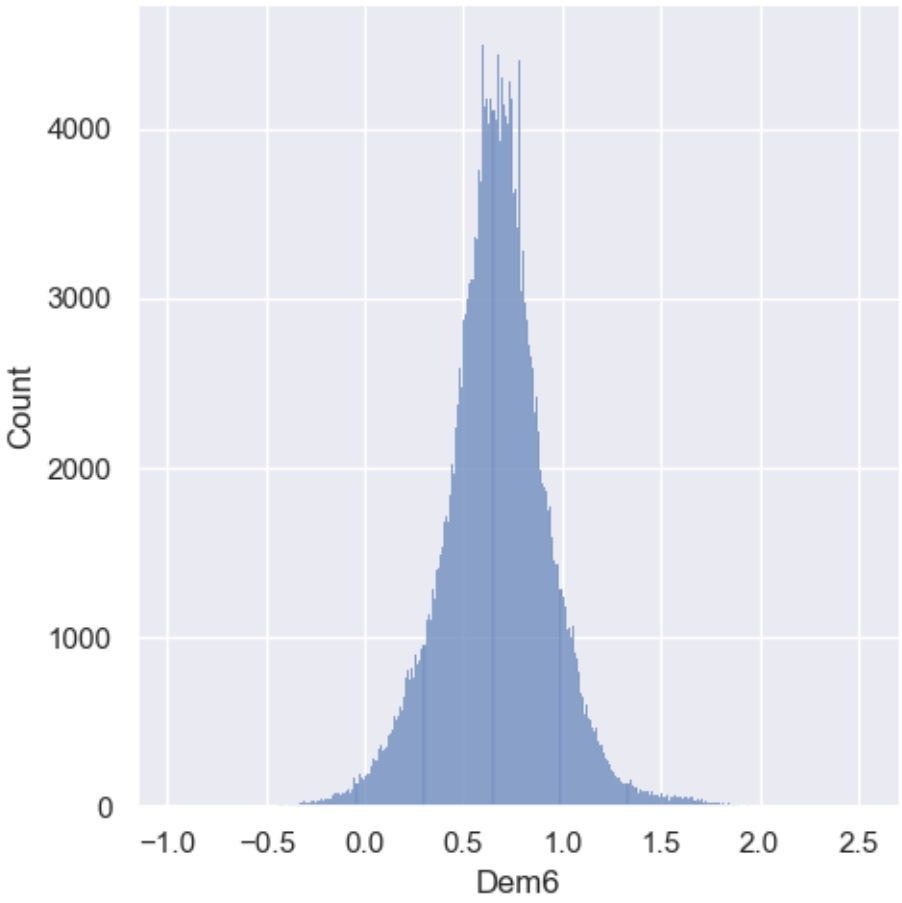

4000
3500
3000
2500
2000
1500
1000
500
0
Count
−1
0
1
2
3
4
Dem1

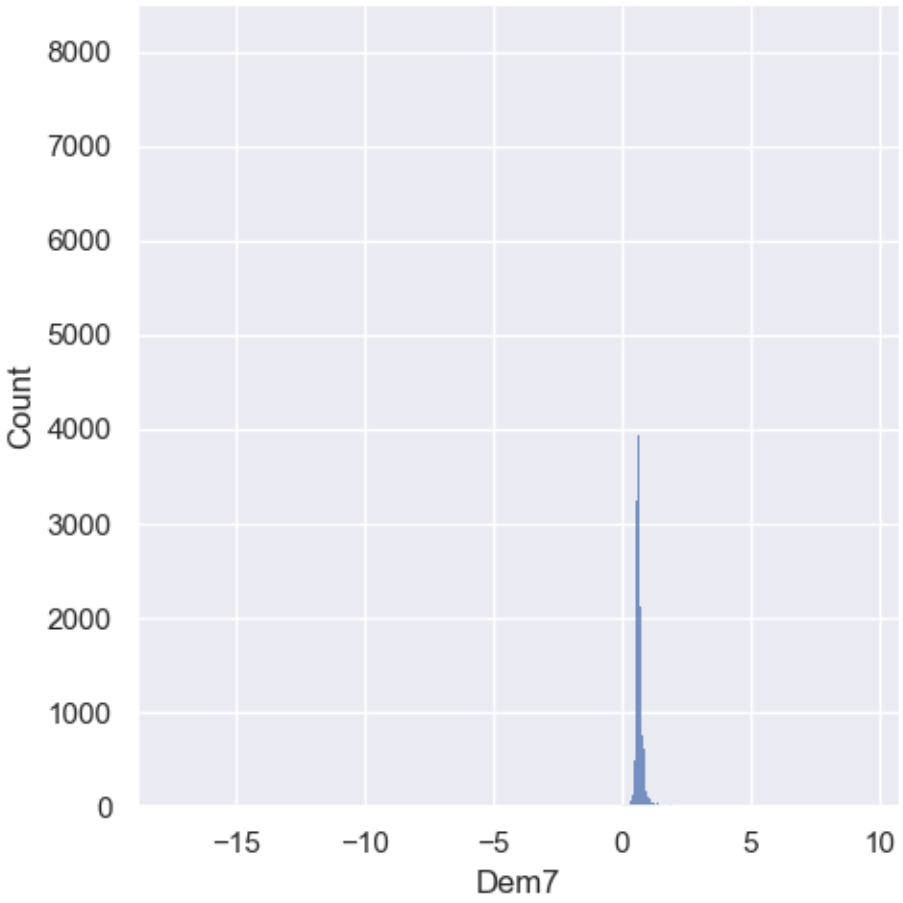

5000
4000
3000
2000
1000
0
Count
−1
0
1
2
3
Dem2

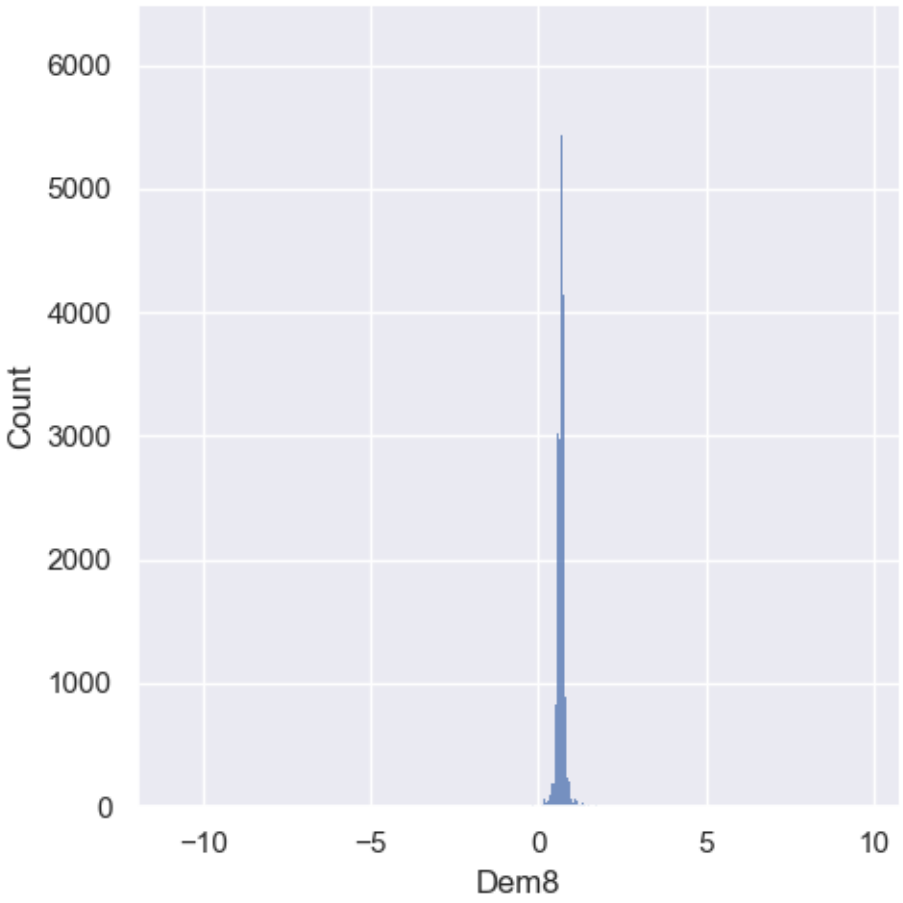

5000
4000
3000
2000
1000
0
Count
−1
0
1
2
3
Dem3

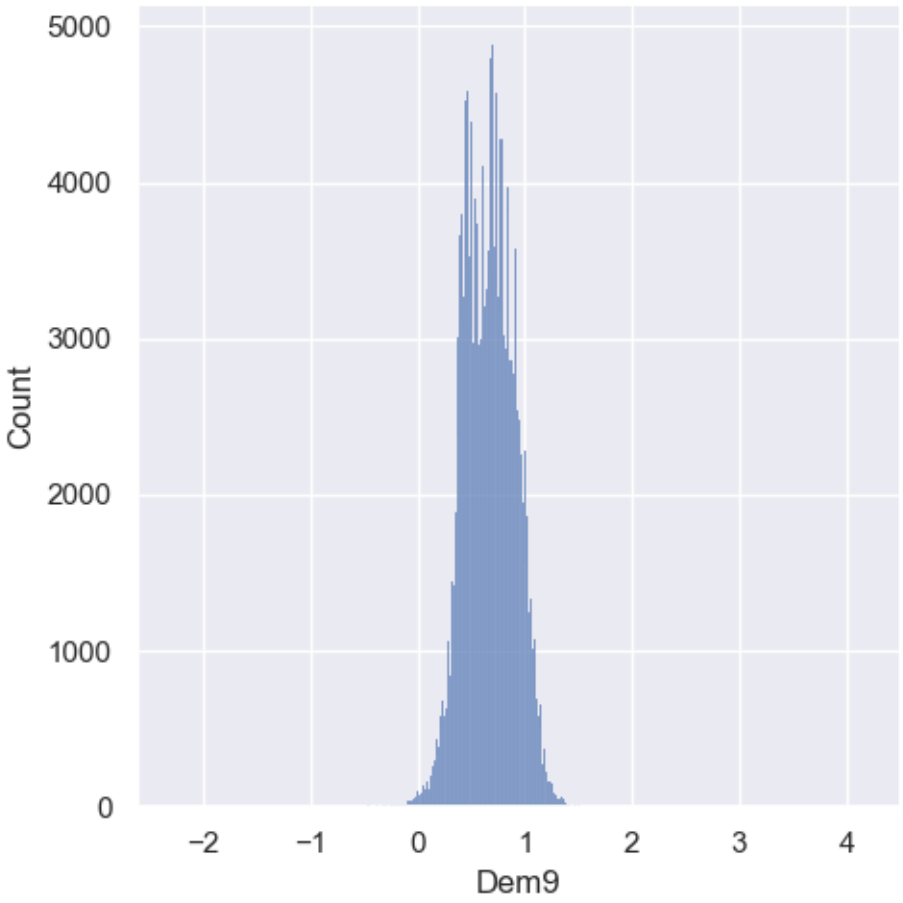

5000
4000
3000
2000
1000
0
Count
−4
−2
0
2
4
6
Dem4

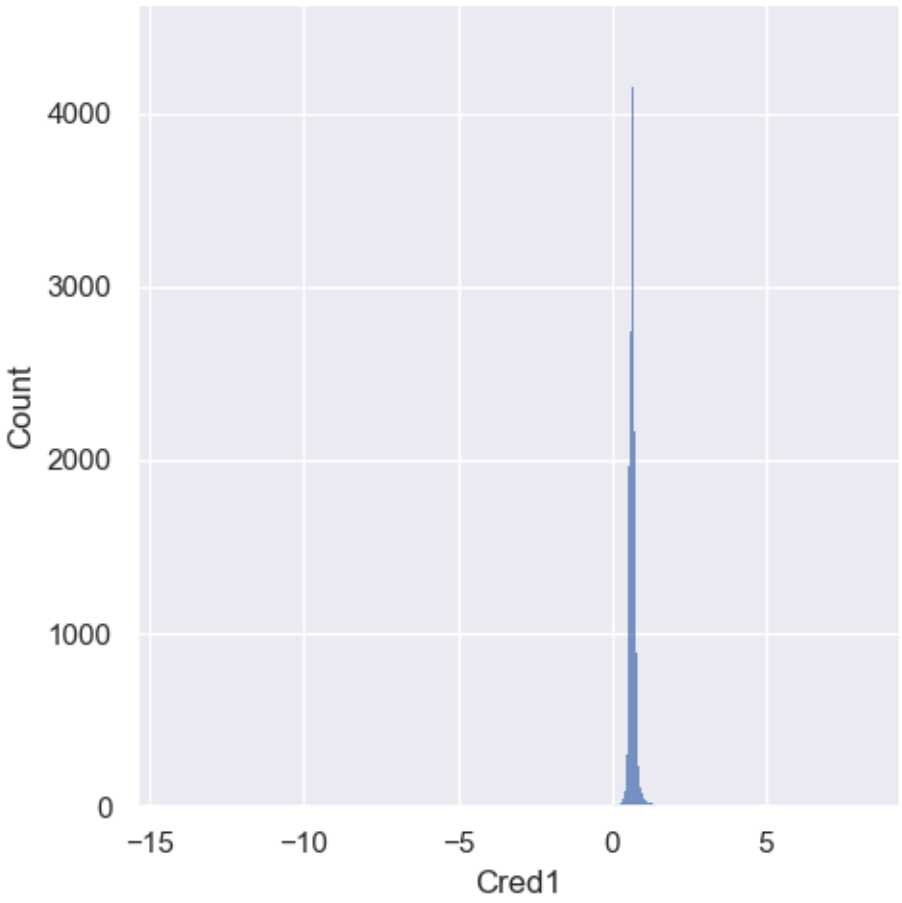

5000
4000
3000
2000
1000
0
Count
−2
−1
0
1
2
Dem5

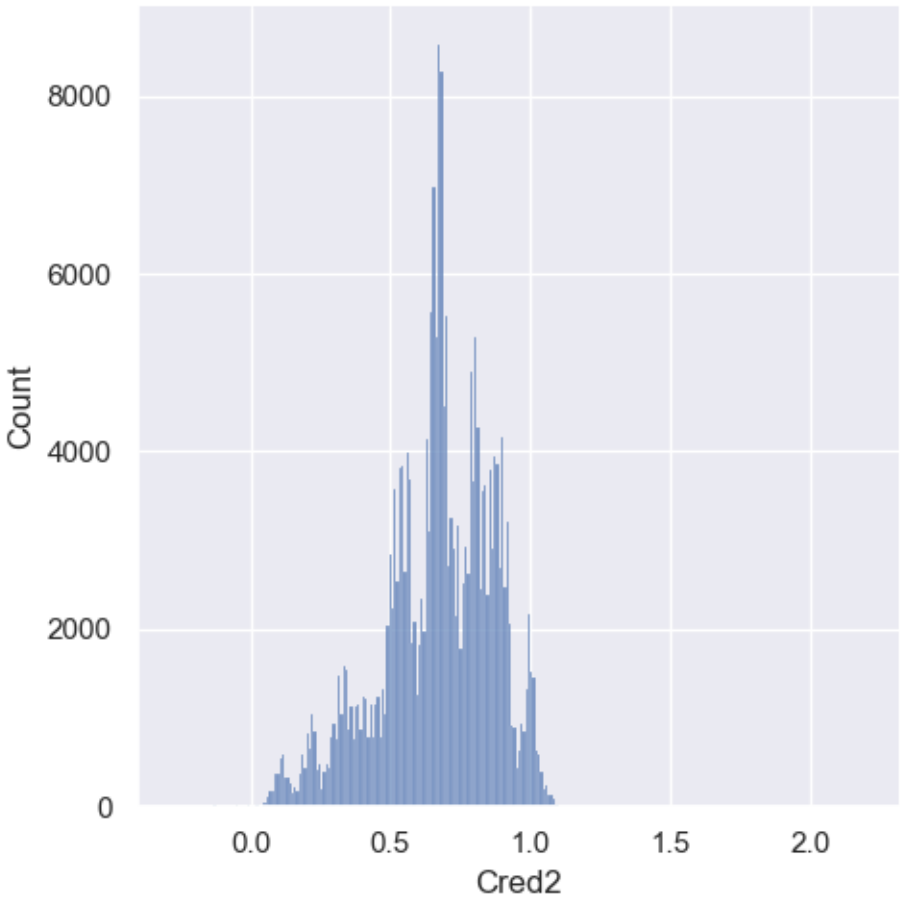

4000
3000
2000
1000
0
Count
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Dem6

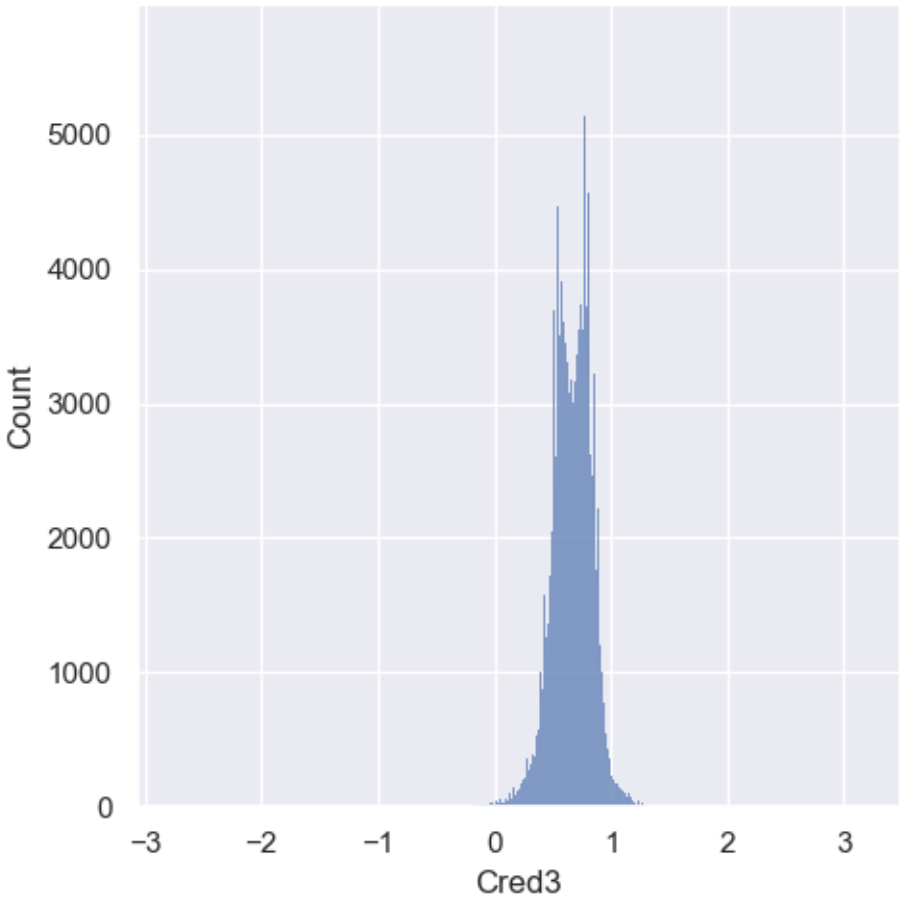

8000
7000
6000
5000
4000
3000
2000
1000
0
Count
−15
−10
−5
0
5
10
Dem7

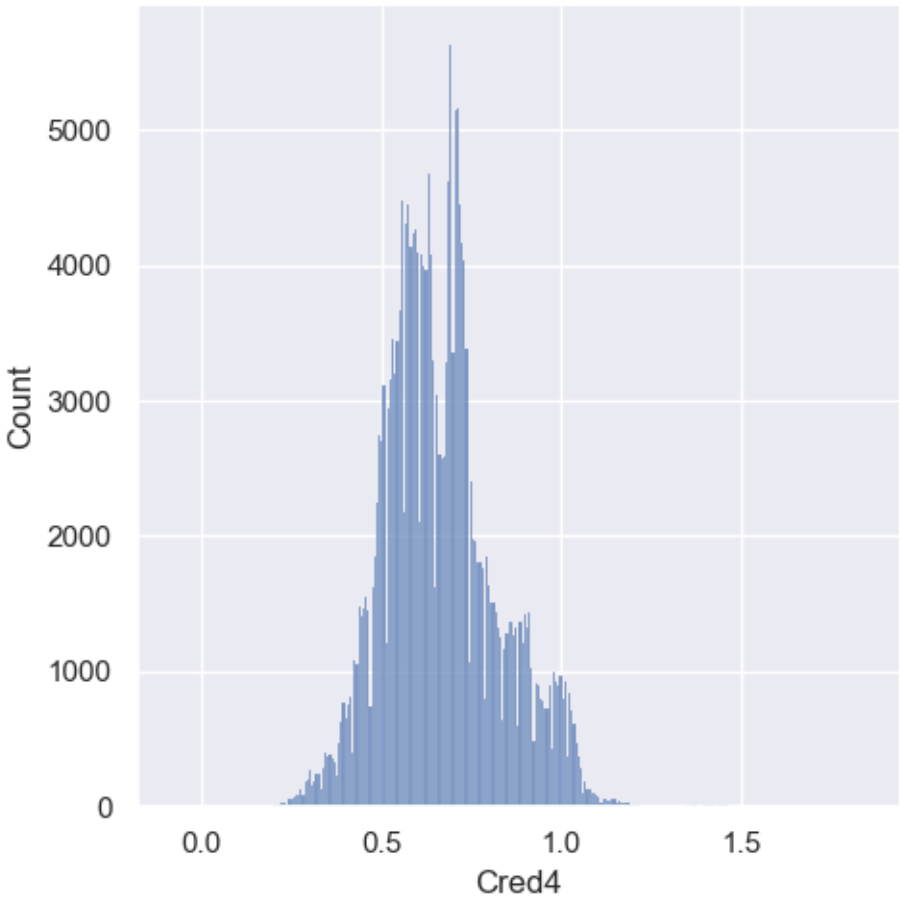

6000
5000
4000
3000
2000
1000
0
Count
−10
−5
0
5
10
Dem8

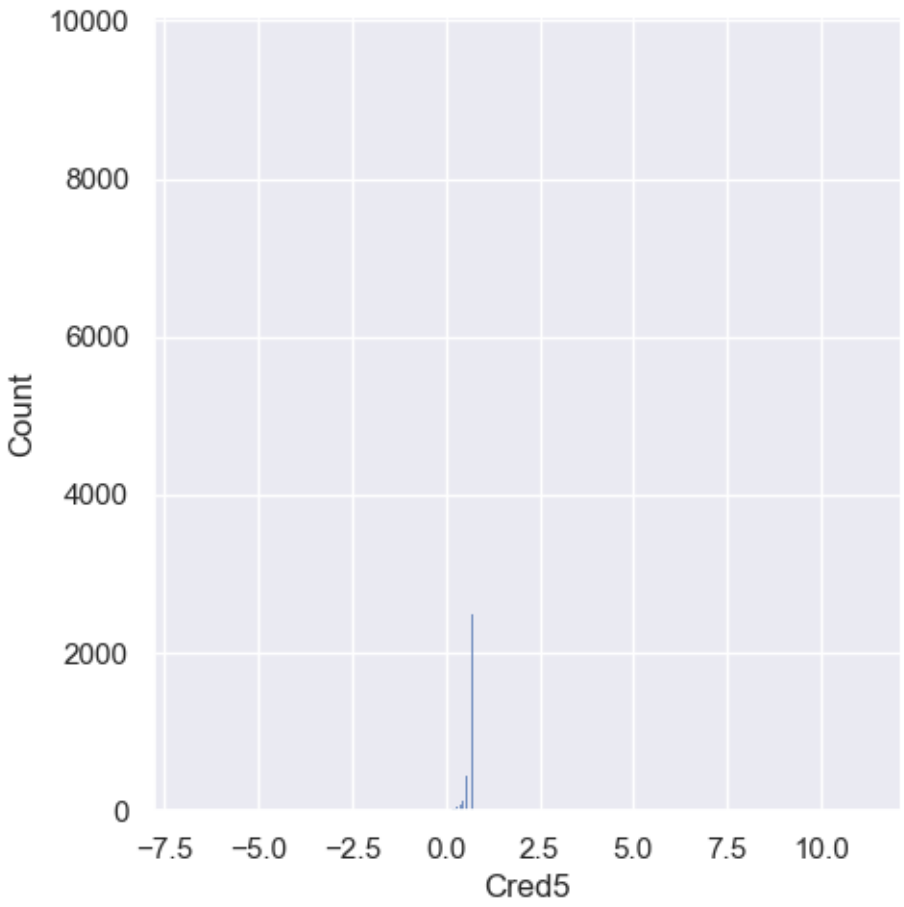

5000
4000
3000
2000
1000
0
Count
−2
−1
0
1
2
3
4
Dem9


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


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


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


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



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

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

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: