Devices Life Prediction Version 0.2

Developed by A.Okada, T.Shirakami, K.Kuramitsu, K.Iino and N.Yamazaki

Start from June 8, 2021

https://github.com/AISTARWORKS/CONVENTION.git

git push --set-upstream origin master

Sample Generator

In [33]:

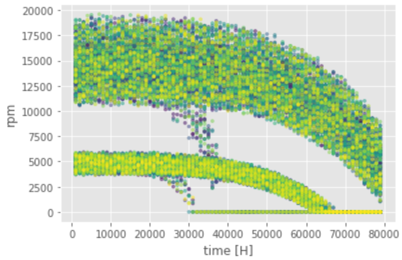
```
# Library import
import pandas as pd # 表=DataFrameに関する
import numpy as np # 行列に関する
import random # ramdom値生成
import math # 数学演算、logとかsin, cosとか
import matplotlib.pyplot as plt # グラフに関する
import matplotlib as mpl # 同じくグラフ関係、要らないかも。
import seaborn as sns

# DataFrameの値の小数点以下桁数はここで調整。
pd.options.display.precision = 2

Sampling = 1000
NumSample = 100
ArrheniusA = np.e
ArrheniusB = 1000
ArrheniusC = 0.035

# 複数FANタイプに対応するため関数化。
def sample_generator(RpmSpec,
                     PowerSpec,
                     TempSpec,
                     DiameterSpec,
                     ThicknessSpec,
                     QSpec,
                     PSpec,
                     LifeSpec,
                     Sampling,
                     NumSample):

    data_list = [[]] # sampling毎のlistなので2次元list
#     cum_rpm = 0
#     cum_temp = 0
#     cum_power = 0

    for sample_id in range(NumSample):

        rpm = RpmSpec # 初期値。最初から0だと困る。
        cum_rpm = 0
        cum_temp = 0
        cum_power = 0
        cumurated_life_impact_factor = 0

        if np.random.random() < 0.1: # ランダムに10%は寿命の短い不良品が混入  ==>rpmが下がりpowerが増える
            defect = 1
        else:
            defect = 0

        for time in range(Sampling, LifeSpec*2, Sampling):

            temp = 25 + random.uniform(-5,5)

            if rpm <= 0: # 前のforループで死んでたら後は永久に死。
                power = 0
                death = 1
            else:

                if defect == 1: # 不良品の場合
                    # rpm: 40degCでフル回転, 時間とともに低下、 +/-5% 誤差考慮でランダム
                    rpm = (-1 * ((time + Sampling)/8000) ** 6 + RpmSpec * temp / TempSpec) * (1-random.uniform(-0.05,0.05))
                    if rpm < 0.1*RpmSpec:
                        rpm = 0
                        power = 0
                        death = 1
                        remaining_life = 0
                    else:
                        #power: rpmの低下により増加する成分と、温度に追従する成分をもつ。+/-5%誤差考慮でランダム
                        power = (0.5 * (4000/rpm) ** 1.2 + PowerSpec * (temp / TempSpec)) * (1 - random.uniform(-0.05,0.05))
                        death = 0
                        # remaining_life
                        k = ArrheniusA ** (ArrheniusB/(273 + temp)) * ArrheniusC
                        remaining_life = k * ((rpm*temp/TempSpec - 0.1*cum_rpm/time)**(1/6)) * 8000 * (1 - random.uniform(-0.05,0.05)) - time
                        if remaining_life < 0:
                            remaining_life = 0
                else: # 良品の場合
                    rpm = (-1 * ((time + Sampling)/8000) ** 4 + RpmSpec * temp / TempSpec) * (1-random.uniform(-0.05,0.05))
                    if rpm < 100:
                        rpm = 0
                        power = 0
                        death = 1
                        remaining_life = 0
                    else:
                        power = (0.5 * (4000/rpm) ** 1.1 + PowerSpec * (temp / TempSpec)) * (1 - random.uniform(-0.05,0.05))
                        death = 0
                        k = ArrheniusA ** (ArrheniusB/(273 + temp)) * ArrheniusC
                        remaining_life = k * ((rpm*temp/TempSpec - 0.1*cum_rpm/time)**(1/4)) * 8000 * (1 - random.uniform(-0.05,0.05)) -time
                        if remaining_life < 0:
                            remaining_life = 0

            # 累積。cum = cumurated = 累積
            cum_rpm += rpm
            cum_temp += temp
            cum_power += power

            # FANの寿命に与えるファクタとして、累積rpmの逆数, 累積temp, 累積powerの積の対数
            # cumurated_life_impact_factorは、0を中心に+/-1以内で推移。大きいと寿命に与える影響大。

            cumurated_life_impact_factor = math.log(10, ((1/cum_rpm) ** 0.5) * cum_temp * cum_power)

            data_list.append([sample_id,
                              defect,
                              time,
                              rpm,
                              temp,
                              power,
                              cum_rpm,
                              cum_temp,
                              cum_power,
                              RpmSpec,
                              PowerSpec,
                              DiameterSpec,
                              ThicknessSpec,
                              QSpec,
```

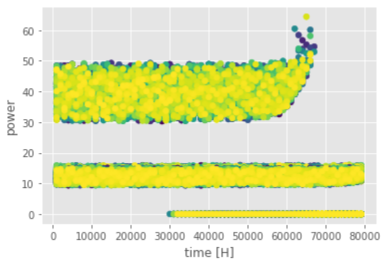
```python
                            PSpec,
                            LifeSpec,
                            cumurated_life_impact_factor,
                            death,
                            k,
                            remaining_life])
    return data_list

fan40 = sample_generator(RpmSpec = 25000,
                         PowerSpec = 20.16,
                         TempSpec = 40,
                         DiameterSpec = 40,
                         ThicknessSpec = 28,
                         QSpec = 0.83, # m^3/min
                         PSpec = 1100, # Pa
                         LifeSpec = 40000,
                         Sampling = Sampling,
                         NumSample = NumSample)

fan40cr = sample_generator(RpmSpec = 22000,
                         PowerSpec = 19.2,
                         TempSpec = 40,
                         DiameterSpec = 40,
                         ThicknessSpec = 56,
                         QSpec = 0.9, # m^3/min
                         PSpec = 1045, # Pa
                         LifeSpec = 40000,
                         Sampling = Sampling,
                         NumSample = NumSample)

fan120 = sample_generator(RpmSpec = 7650,
                         PowerSpec = 1.3 * 48,
                         TempSpec = 40,
                         DiameterSpec = 120,
                         ThicknessSpec = 38,
                         QSpec = 7.49, # m^3/min
                         PSpec = 532.5, # Pa,
                         LifeSpec = 40000,
                         Sampling = Sampling,
                         NumSample = NumSample)
# 各fan統合
list = fan40
for data in fan40cr:
    list.append(data)

for data in fan120:
    list.append(data)

df = pd.DataFrame(list,    # listは3種ファン統合。別々にやる場合は、fan40, fan40cr or fan120
                 columns=['sample_id',
                          'defect',
                          'time',
                          'rpm',
                          'temp',
                          'power',
                          'cum_rpm',
                          'cum_temp',
                          'cum_power',
                          'RpmSpec',
                          'PowerSpec',
                          'DiameterSpec',
                          'ThicknessSpec',
                          'QSpec',
                          'PSpec',
                          'LifeSpec',
                          'cumurated_life_impact_factor',
                          'death',
                          'k',
                          'remaining_life'])

# df.to_csv("./sample_data_check3.csv")

print('df= ', df.info())

fig, ax = plt.subplots()
ax.scatter(df['time'], df['rpm'], c=df['sample_id'], s=10, alpha=0.5)
plt.xlabel('time [H]',size=12)
plt.ylabel('rpm',size=12)

fig, ax = plt.subplots()
ax.scatter(df['time'], df['power'], c=df['sample_id'])
plt.xlabel('time [H]',size=12)
plt.ylabel('power',size=12)

fig, ax = plt.subplots()
ax.scatter(df['time'], df['remaining_life'], c=df['sample_id'])
plt.xlabel('time [H]',size=12)
plt.ylabel('remaining_life [H]',size=12)

fig, ax = plt.subplots()
ax.scatter(df['time'], df['k'], c=df['sample_id'])
plt.xlabel('time [H]',size=12)
plt.ylabel('k (Arrhenius coefficient)',size=12)

fig, ax = plt.subplots()
ax.scatter(df['time'], df['cum_rpm'], c=df['sample_id'])
plt.xlabel('time [H]',size=12)
plt.ylabel('cum_rpm',size=12)

fig, ax = plt.subplots()
ax.scatter(df['time'], df['cum_power'], c=df['sample_id'])
plt.xlabel('time [H]',size=12)
plt.ylabel('cum_power',size=12)

fig, ax = plt.subplots()
ax.scatter(df['time'], df['cum_temp'], c=df['sample_id'])
plt.xlabel('time [H]',size=12)
plt.ylabel('cum_temp',size=12)

fig, ax = plt.subplots()
ax.scatter(df['time'], df['cumurated_life_impact_factor'], c=df['sample_id'])
plt.xlabel('time [H]',size=12)
plt.ylabel('cumurated_life_impact_factor',size=12)

# Corelation Analysis

sns.pairplot(df.loc[: ,'defect':'cumurated_life_impact_factor'])
```

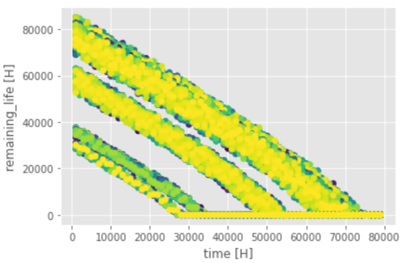
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23703 entries, 0 to 23702
Data columns (total 20 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   sample_id                     23700 non-null  float64
 1   defect                        23700 non-null  float64
 2   time                          23700 non-null  float64
 3   rpm                           23700 non-null  float64
 4   temp                          23700 non-null  float64
 5   power                         23700 non-null  float64
 6   cum_rpm                       23700 non-null  float64
 7   cum_temp                      23700 non-null  float64
 8   cum_power                     23700 non-null  float64
 9   RpmSpec                       23700 non-null  float64
 10  PowerSpec                     23700 non-null  float64
 11  DiameterSpec                  23700 non-null  float64
```

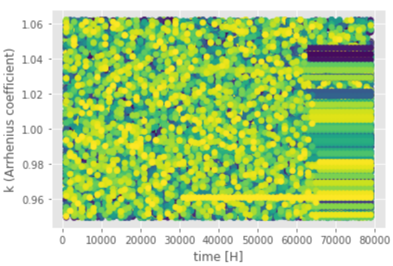
```
 12  ThicknessSpec                  23700 non-null  float64
 13  QSpec                          23700 non-null  float64
 14  PSpec                          23700 non-null  float64
 15  LifeSpec                       23700 non-null  float64
 16  cumurated_life_impact_factor   23700 non-null  float64
 17  death                          23700 non-null  float64
 18  k                              23700 non-null  float64
 19  remaining_life                 23700 non-null  float64
dtypes: float64(20)
memory usage: 3.6 MB
df=  None
```

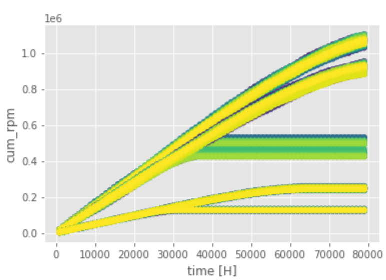
Out[33]: <seaborn.axisgrid.PairGrid at 0x25312571af0>

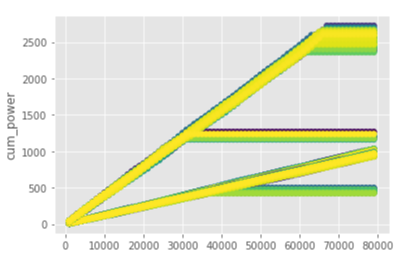


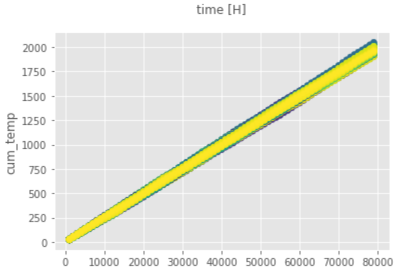


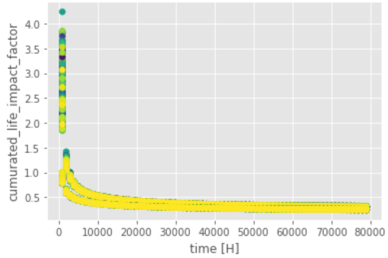


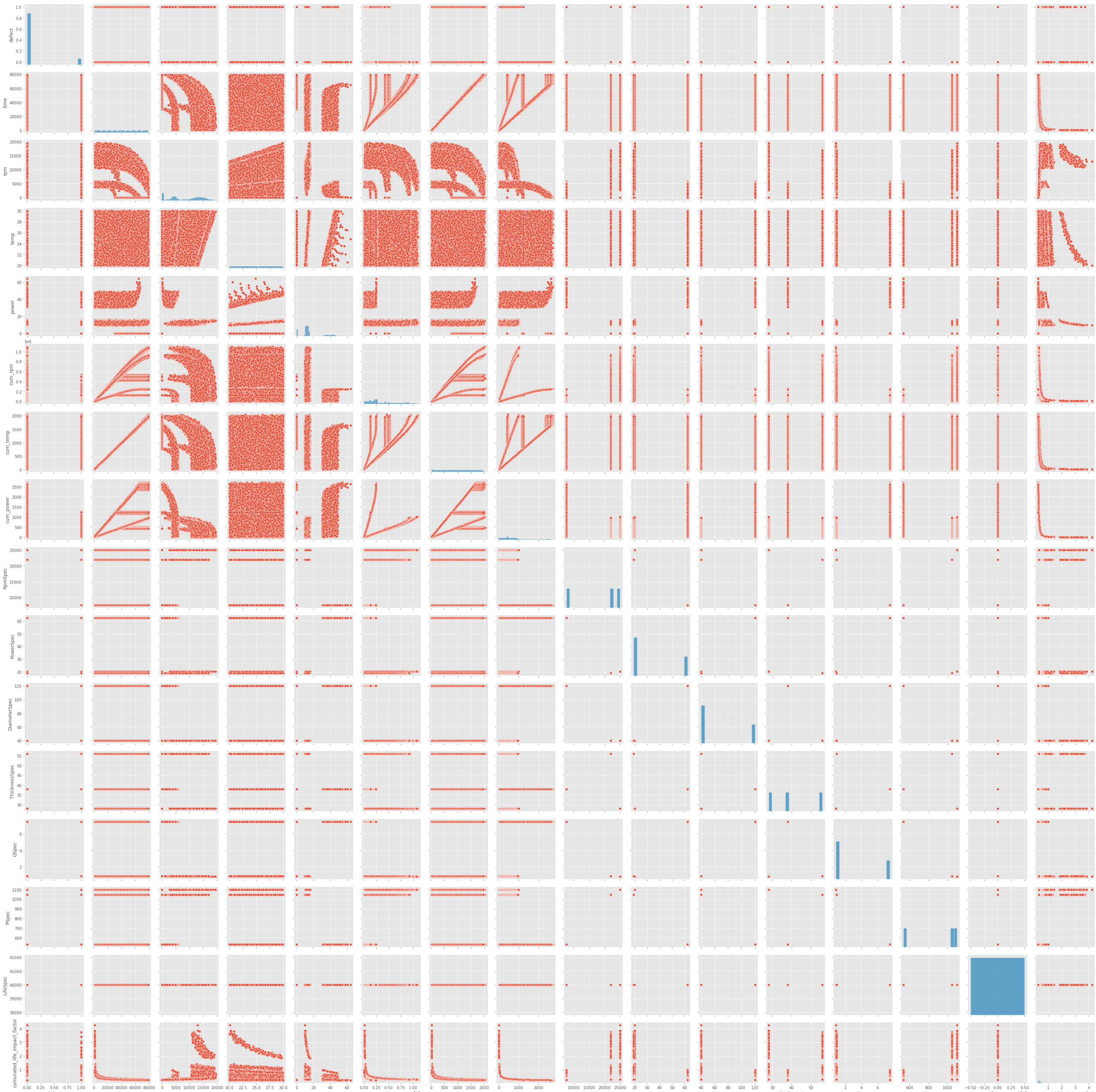












In [34]:

```
df = df.dropna(how="any")

indexNames = df[ df['death'] == 1 ].index
df.drop(indexNames , inplace=True)

indexNames = df[ df['remaining_life'] == 0 ].index
df.drop(indexNames , inplace=True)

df.to_csv('./sample_data_v02.csv')
df
```

Out[34]:

	sample_id	defect	time	rpm	temp	power	cum_rpm	cum_temp	cum_power	RpmSpec	PowerSpec	DiameterSpec	ThicknessSpec	QSpec	PSpec	LifeSpec	cumurated_life_impact_factor	death	k	remaining_life
1	0.0	0.0	1000.0	13152.68	20.37	10.29	13152.68	20.37	10.29	25000.0	20.16	40.0	28.0	0.83	1100.0	40000.0	3.82	0.0	1.06	71790.06
2	0.0	0.0	2000.0	16087.30	25.77	12.48	29239.98	46.14	22.77	25000.0	20.16	40.0	28.0	0.83	1100.0	40000.0	1.27	0.0	0.99	76168.90
3	0.0	0.0	3000.0	14229.76	22.91	11.28	43469.74	69.05	34.05	25000.0	20.16	40.0	28.0	0.83	1100.0	40000.0	0.95	0.0	1.03	71599.91

	sample_id	defect	time	rpm	temp	power	cum_rpm	cum_temp	cum_power	RpmSpec	PowerSpec	DiameterSpec	ThicknessSpec	QSpec	PSpec	LifeSpec	cumurated_life_impact_factor	death	k	remaining_life
4	0.0	0.0	4000.0	15643.55	26.26	13.03	59113.29	95.31	47.08	25000.0	20.16	40.0	28.0	0.83	1100.0	40000.0	0.79	0.0	0.99	71797.83
5	0.0	0.0	5000.0	14929.23	24.91	13.17	74042.52	120.22	60.25	25000.0	20.16	40.0	28.0	0.83	1100.0	40000.0	0.70	0.0	1.00	76479.44
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23646	99.0	1.0	23000.0	4867.03	29.25	43.84	108524.75	583.75	921.12	7650.0	62.40	120.0	38.0	7.49	532.5	40000.0	0.31	0.0	0.96	5863.14
23647	99.0	1.0	24000.0	3971.75	25.87	40.03	112496.49	609.62	961.15	7650.0	62.40	120.0	38.0	7.49	532.5	40000.0	0.31	0.0	0.99	6597.50
23648	99.0	1.0	25000.0	2745.68	21.04	34.63	115242.17	630.65	995.78	7650.0	62.40	120.0	38.0	7.49	532.5	40000.0	0.31	0.0	1.05	2160.39
23649	99.0	1.0	26000.0	3756.97	28.09	44.60	118999.14	658.74	1040.37	7650.0	62.40	120.0	38.0	7.49	532.5	40000.0	0.30	0.0	0.97	1999.14
23650	99.0	1.0	27000.0	2779.55	24.01	40.12	121778.69	682.75	1080.50	7650.0	62.40	120.0	38.0	7.49	532.5	40000.0	0.30	0.0	1.01	2062.68

17819 rows × 20 columns

Prediction

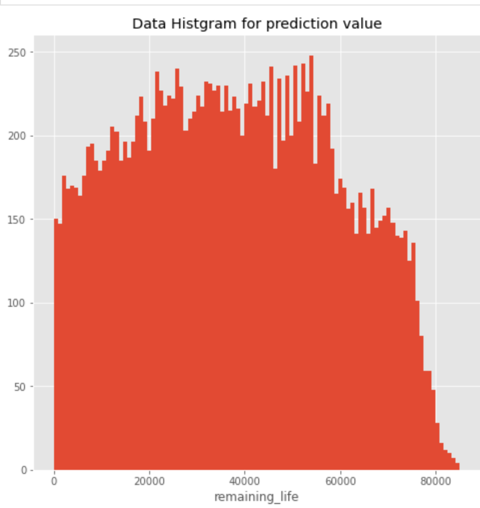
In [35]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
import seaborn as sns

df = pd.read_csv('sample_data_v02.csv', index_col=[0])
df = df.dropna(how="any")

# print(df.head(), df.tail())
df.info()
# print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17819 entries, 1 to 23650
Data columns (total 20 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   sample_id                     17819 non-null  float64
 1   defect                        17819 non-null  float64
 2   time                          17819 non-null  float64
 3   rpm                           17819 non-null  float64
 4   temp                          17819 non-null  float64
 5   power                         17819 non-null  float64
 6   cum_rpm                       17819 non-null  float64
 7   cum_temp                      17819 non-null  float64
 8   cum_power                     17819 non-null  float64
 9   RpmSpec                       17819 non-null  float64
 10  PowerSpec                     17819 non-null  float64
 11  DiameterSpec                  17819 non-null  float64
 12  ThicknessSpec                 17819 non-null  float64
 13  QSpec                         17819 non-null  float64
 14  PSpec                         17819 non-null  float64
 15  LifeSpec                      17819 non-null  float64
 16  cumurated_life_impact_factor  17819 non-null  float64
 17  death                         17819 non-null  float64
 18  k                             17819 non-null  float64
 19  remaining_life                17819 non-null  float64
dtypes: float64(20)
memory usage: 2.9 MB
```

In [36]:

```
plt.figure(figsize=(8, 8))
plt.hist(df['remaining_life'], bins=100)
plt.title('Data Histgram for prediction value')
plt.xlabel('remaining_life',size=12)
plt.show()
```



In [37]:

```
from sklearn.model_selection import train_test_split
import xgboost as xgb

# 機械学習では、学習させる特徴量をX, 求める答えをyで表す。
X = df.drop(columns=['remaining_life', 'sample_id', 'defect', 'k'])
y = df['remaining_life']

print(X.shape)
print(y.shape)

# trainとは学習に用いるデータ、testとは検証用に取っておくまだ見ぬデータ。
# 自動的に検証用のまだ見ぬデータを20%とっておく。
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)
print(X_train.shape)
print(X_test.shape)
```

```
(17819, 16)
(17819,)
(14255, 16)
(3564, 16)
```

In [39]:

```
params = {
    'silent': 1,
    'max_depth': 6,
    'min_child_weight': 1,
    'eta': 0.1,
    'tree_method': 'exact',
    'objective': 'reg:linear',
    'eval_metric': 'rmse',
    'predictor': 'cpu_predictor'
}
```

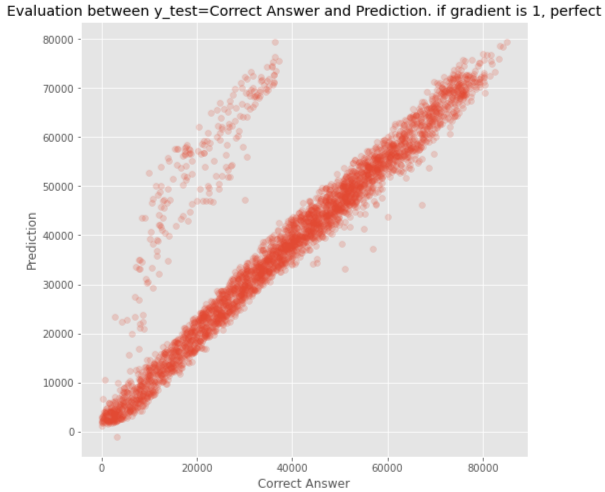
```
# Xgboost params tutorial
# https://qiita.com/FJyusk56/items/0649f4362587261bd57a

## objective
# reg:linear(線形回帰)
# reg:logistic(ロジスティック回帰)
# binary:logistic(2項分類で確率を返す)
# multi:softmax(多項分類でクラスの値を返す)

## eval_metric
# rmse(2乗平均平方根誤差)
# logloss(負の対数尺度)
# error(2-クラス分類のエラー率)
# merror(多クラス分類のエラー率)
# mlogloss(多クラスの対数損失)
# auc(ROC曲線下の面積で性能の良さを表す)
# mae(平均絶対誤差)



# GPUの場合
# params = {
#     'silent': 1,
#     'max_depth': 6,
#     'min_child_weight': 1,
#     'eta': 0.1,
#     'tree_method': 'gpu_exact',
#     'objective': 'gpu:reg:linear',
#     'eval_metric': 'rmse',
#     'predictor': 'gpu_predictor'
# }

dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
model = xgb.train(params=params,
                  dtrain=dtrain,
                  num_boost_round=1000,
                  early_stopping_rounds=5,
                  evals=[(dtest, 'test')])
```

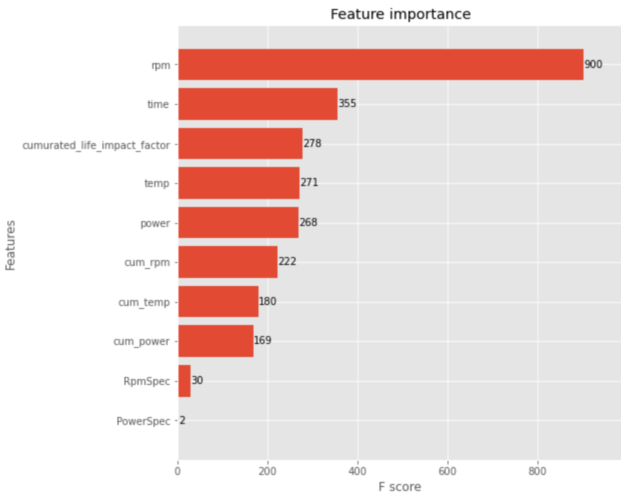
```
[13:35:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarederror.
[13:35:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "silent" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[0]	test-rmse:39310.63672
[1]	test-rmse:35561.77734
[2]	test-rmse:32217.49805
[3]	test-rmse:29230.38477
[4]	test-rmse:26562.93555
[5]	test-rmse:24197.17188
[6]	test-rmse:22088.08398
[7]	test-rmse:20229.36914
[8]	test-rmse:18586.19922
[9]	test-rmse:17149.95117
[10]	test-rmse:15891.49219
[11]	test-rmse:14784.81738
[12]	test-rmse:13826.41504
[13]	test-rmse:13002.63867
[14]	test-rmse:12290.92090
[15]	test-rmse:11695.05762
[16]	test-rmse:11178.50000
[17]	test-rmse:10750.60449
[18]	test-rmse:10394.81641
[19]	test-rmse:10108.81445
[20]	test-rmse:9869.28613
[21]	test-rmse:9666.28027
[22]	test-rmse:9495.73047
[23]	test-rmse:9369.89453
[24]	test-rmse:9263.78613
[25]	test-rmse:9184.93652
[26]	test-rmse:9120.52148
[27]	test-rmse:9057.99414
[28]	test-rmse:9019.41504
[29]	test-rmse:8956.44141
[30]	test-rmse:8919.71777
[31]	test-rmse:8878.77930
[32]	test-rmse:8865.41113
[33]	test-rmse:8854.47949
[34]	test-rmse:8845.56836
[35]	test-rmse:8835.10449
[36]	test-rmse:8837.10644
[37]	test-rmse:8822.32910
[38]	test-rmse:8822.80859
[39]	test-rmse:8823.57422
[40]	test-rmse:8813.61914
[41]	test-rmse:8815.13379
[42]	test-rmse:8810.94727
[43]	test-rmse:8806.53906
[44]	test-rmse:8806.91113
[45]	test-rmse:8799.20019
[46]	test-rmse:8794.03516
[47]	test-rmse:8797.21484
[48]	test-rmse:8796.31152
[49]	test-rmse:8800.11719
[50]	test-rmse:8801.24121
[51]	test-rmse:8801.96680
```

In [40]:

```
print(model)
model.save_model('./xgb1.model')

model.load_model('./xgb1.model')

prediction = model.predict(xgb.DMatrix(X_test),
                           ntree_limit=model.best_ntree_limit)

plt.figure(figsize=(8, 8))
# plt.scatter(y_test[:1000], prediction[:1000], alpha=0.2)
plt.scatter(y_test, prediction, alpha=0.2)
plt.title('Evaluation between y_test=Correct Answer and Prediction. if gradient is 1, perfect')
plt.xlabel('Correct Answer',size=12)
plt.ylabel('Prediction',size=12)
plt.show()

fig, ax = plt.subplots(figsize=(8, 8))
xgb.plot_importance(model, max_num_features=12, height=0.8, ax=ax)
plt.show()
```

```
<xgboost.core.Booster object at 0x000002531A32E040>
[13:35:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarederror.
C:¥Users¥nnroc¥anaconda3¥lib¥site-packages¥xgboost¥core.py:101: UserWarning: ntree_limit is deprecated, use `iteration_range` or model slicing instead.
  warnings.warn(
```





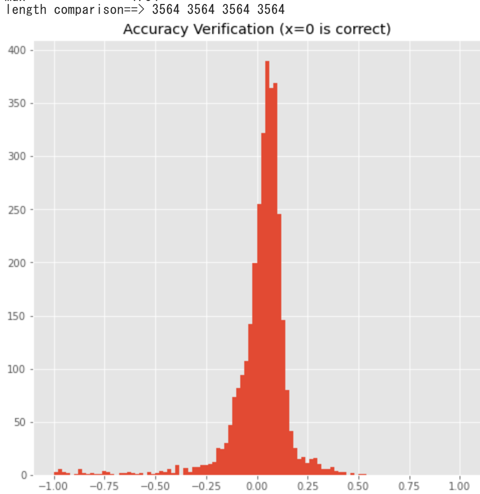
In [ ]:

In [41]:

```
X_test_df = pd.DataFrame(X_test)
y_test_df = pd.DataFrame(y_test)
prediction_df = pd.DataFrame(prediction, columns=['remaining_life_pred'])

X_test_df_reset = X_test_df.reset_index()
y_test_df_reset = y_test_df.reset_index()
prediction_df_reset = prediction_df.reset_index()

print(y_test_df_reset)

y_test_list = y_test_df_reset['remaining_life'].to_list()
prediction_list = prediction_df_reset['remaining_life_pred'].to_list()

print(y_test_list[:5], len(y_test_list))
print(prediction_list[:5], len(prediction_list))
difference_list = []

for i in range(len(y_test_list)):
    diff = (y_test_list[i] - prediction_list[i]) / y_test_list[i]
    difference_list.append(diff)

print(difference_list[:5])

difference_df = pd.DataFrame(difference_list, columns=['defference_rate'])
print(difference_df)
print('')
print('difference_df.info()==>', difference_df.info())
print('difference_df.describe()==>', difference_df.describe())
print('length comparison==>', len(X_test_df_reset), len(y_test_df_reset), len(prediction_df_reset), len(difference_df))

difference_df_reset = difference_df.reset_index()

plt.figure(figsize=(8, 8))
plt.hist(difference_df['defference_rate'], bins=100, range=(-1,1))
plt.title('Accuracy Verification (x=0 is correct)')
plt.show()

report_df = pd.concat([X_test_df_reset, y_test_df_reset, prediction_df_reset, difference_df_reset], axis=1)

report_df

report_df.to_csv("./report_remaining_life.csv")
```

```
      index  remaining_life
0      3046        36717.70
1      6370        24083.21
2     20084        44009.49
3     21500        45145.44
4     20102        20291.89
...     ...             ...
3559  20636        41471.89
3560  19298        38627.77
3561  12974        60833.30
3562  17817        17294.74
3563   6645        28772.73

[3564 rows x 2 columns]
[36717.69783062507, 24083.21150757832, 44009.48647900557, 45145.44324472488, 20291.89056514258] 3564
[35666.8046875, 24106.26953125, 44355.98828125, 45107.58984375, 22094.65625] 3564
[0.02862088870529761, -0.0009574314316188985, -0.007873343453114903, 0.0008384766712706823, -0.08884168180732309]
      defference_rate
0            2.86e-02
1           -9.57e-04
2           -7.87e-03
3            8.38e-04
4           -8.88e-02
...               ...
3559         3.75e-02
```

```
3560         1.22e-01
3561         4.85e-02
3562         1.31e-01
3563        -1.13e+00

[3564 rows x 1 columns]

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3564 entries, 0 to 3563
Data columns (total 1 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   defference_rate  3564 non-null   float64
dtypes: float64(1)
memory usage: 28.0 KB
difference_df.info()==> None
difference_df.describe()==>        defference_rate
count          3564.00
mean             -0.18
std               2.10
min            -108.14
25%              -0.04
50%               0.04
75%               0.09
max               1.34
length comparison==> 3564 3564 3564 3564
```



In [ ]:

In [ ]:

Can Defects be detected?

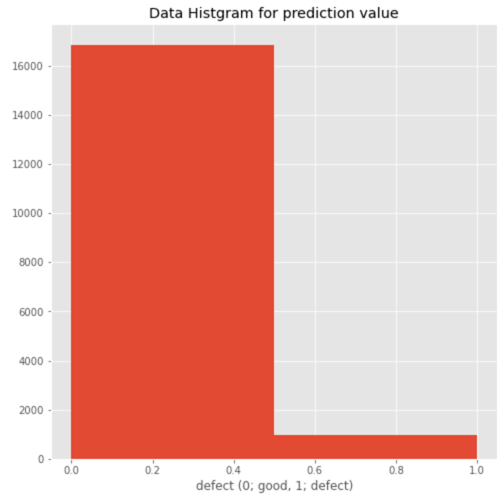
In [45]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
import seaborn as sns

df = pd.read_csv('sample_data_v02.csv', index_col=[0])
df = df.dropna(how="any")

# print(df.head(), df.tail())
df.info()
# print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17819 entries, 1 to 23650
Data columns (total 20 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   sample_id                      17819 non-null  float64
 1   defect                         17819 non-null  float64
 2   time                           17819 non-null  float64
 3   rpm                            17819 non-null  float64
 4   temp                           17819 non-null  float64
 5   power                          17819 non-null  float64
 6   cum_rpm                        17819 non-null  float64
 7   cum_temp                       17819 non-null  float64
 8   cum_power                      17819 non-null  float64
 9   RpmSpec                        17819 non-null  float64
 10  PowerSpec                      17819 non-null  float64
 11  DiameterSpec                   17819 non-null  float64
 12  ThicknessSpec                  17819 non-null  float64
 13  QSpec                          17819 non-null  float64
 14  PSpec                          17819 non-null  float64
 15  LifeSpec                       17819 non-null  float64
 16  cumurated_life_impact_factor   17819 non-null  float64
 17  death                          17819 non-null  float64
 18  k                              17819 non-null  float64
 19  remaining_life                 17819 non-null  float64
dtypes: float64(20)
memory usage: 2.9 MB
```

In [46]:

```python
plt.figure(figsize=(8, 8))
plt.hist(df['defect'], bins=2)
plt.title('Data Histgram for prediction value')
plt.xlabel('defect (0: good, 1: defect)',size=12)
plt.show()
```



In [47]:

```
from sklearn.model_selection import train_test_split
import xgboost as xgb

X = df.drop(columns=['remaining_life', 'sample_id', 'defect', 'k'])
y = df['defect']

print(X.shape)
print(y.shape)

# trainとは学習に用いるデータ、testとは検証用に取っておくまだ見ぬデータ。
# 自動的に検証用のまだ見ぬデータを20%とっておく。
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)
print(X_train.shape)
print(X_test.shape)
```

```
(17819, 16)
(17819,)
(14255, 16)
(3564, 16)
```

In [48]:

```
params = {
    'silent': 1,
    'max_depth': 6,
    'min_child_weight': 1,
    'eta': 0.1,
    'tree_method': 'exact',
    'objective': 'reg:linear',
    'eval_metric': 'rmse',
    'predictor': 'cpu_predictor'
}

# GPUの場合
# params = {
#     'silent': 1,
#     'max_depth': 6,
#     'min_child_weight': 1,
#     'eta': 0.1,
#     'tree_method': 'gpu_exact',
#     'objective': 'gpu:reg:linear',
#     'eval_metric': 'rmse',
#     'predictor': 'gpu_predictor'
# }

dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
model = xgb.train(params=params,
                  dtrain=dtrain,
                  num_boost_round=1000,
                  early_stopping_rounds=5,
                  evals=[(dtest, 'test')])
```

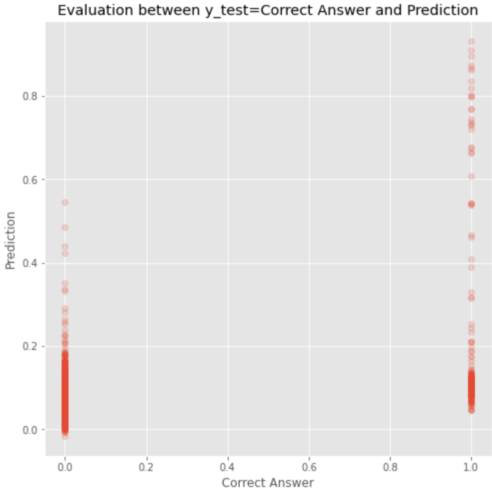
```
[13:37:32] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarederror.
[13:37:32] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "silent" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[0]	test-rmse:0.46060
[1]	test-rmse:0.42523
[2]	test-rmse:0.39492
[3]	test-rmse:0.36789
[4]	test-rmse:0.34453
[5]	test-rmse:0.32442
[6]	test-rmse:0.30723
[7]	test-rmse:0.29272
[8]	test-rmse:0.28045
[9]	test-rmse:0.27014
[10]	test-rmse:0.26160
[11]	test-rmse:0.25425
[12]	test-rmse:0.24810
[13]	test-rmse:0.24291
[14]	test-rmse:0.23838
[15]	test-rmse:0.23482
[16]	test-rmse:0.23214
[17]	test-rmse:0.22987
[18]	test-rmse:0.22779
[19]	test-rmse:0.22617
[20]	test-rmse:0.22515
[21]	test-rmse:0.22433
[22]	test-rmse:0.22368
[23]	test-rmse:0.22290
[24]	test-rmse:0.22260
[25]	test-rmse:0.22180
[26]	test-rmse:0.22150
[27]	test-rmse:0.22090
[28]	test-rmse:0.22076
[29]	test-rmse:0.22074
[30]	test-rmse:0.22070
[31]	test-rmse:0.22057
[32]	test-rmse:0.22057
[33]	test-rmse:0.22051
[34]	test-rmse:0.22050
[35]	test-rmse:0.22043
[36]	test-rmse:0.22025
[37]	test-rmse:0.22028
[38]	test-rmse:0.22028
[39]	test-rmse:0.22031
[40]	test-rmse:0.22041
[41]	test-rmse:0.22046
```

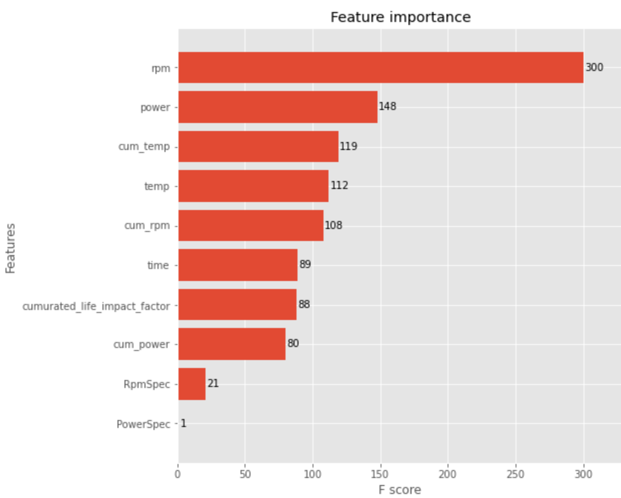
In [49]:

```
print(model)
model.save_model('./xgb1.model')
```

```
model.load_model('./xgb1.model')

prediction = model.predict(xgb.DMatrix(X_test),
                           ntree_limit=model.best_ntree_limit)

plt.figure(figsize=(8, 8))
# plt.scatter(y_test[:1000], prediction[:1000], alpha=0.2)
plt.scatter(y_test, prediction, alpha=0.2)
plt.title('Evaluation between y_test=Correct Answer and Prediction')
plt.xlabel('Correct Answer',size=12)
plt.ylabel('Prediction',size=12)
plt.show()

fig, ax = plt.subplots(figsize=(8, 8))
xgb.plot_importance(model, max_num_features=12, height=0.8, ax=ax)
plt.show()
```

```
<xgboost.core.Booster object at 0x000002533AE9F580>
[13:37:36] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now deprecated in favor of reg:squarederror.
C:¥Users¥nnroc¥anaconda3¥lib¥site-packages¥xgboost¥core.py:101: UserWarning: ntree_limit is deprecated, use `iteration_range` or model slicing instead.
  warnings.warn(
```



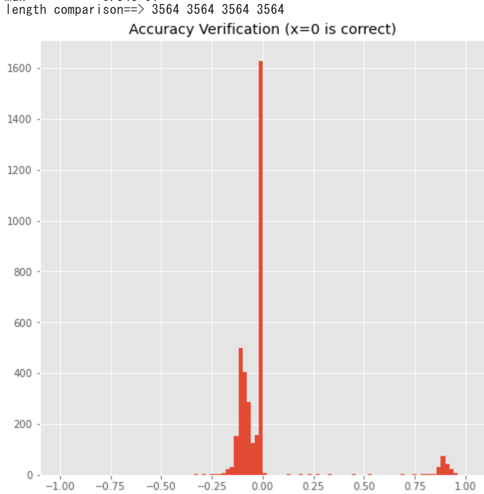


In [50]:

```
X_test_df = pd.DataFrame(X_test)
y_test_df = pd.DataFrame(y_test)
prediction_df = pd.DataFrame(prediction, columns=['defect_pred'])

X_test_df_reset = X_test_df.reset_index()
y_test_df_reset = y_test_df.reset_index()
prediction_df_reset = prediction_df.reset_index()

print(y_test_df_reset)

y_test_list = y_test_df_reset['defect'].to_list()
prediction_list = prediction_df_reset['defect_pred'].to_list()

print(y_test_list[:5], len(y_test_list))
print(prediction_list[:5], len(prediction_list))
difference_list = []

for i in range(len(y_test_list)):
    diff = (y_test_list[i] - prediction_list[i])
    difference_list.append(diff)

print(difference_list[:5])

difference_df = pd.DataFrame(difference_list, columns=['defference_rate'])
print(difference_df)
print('')
print('difference_df.info()==>', difference_df.info())
print('difference_df.describe()==>', difference_df.describe())
print('length comparison==>', len(X_test_df_reset), len(y_test_df_reset), len(prediction_df_reset), len(difference_df))

difference_df_reset = difference_df.reset_index()

plt.figure(figsize=(8, 8))
plt.hist(difference_df['defference_rate'], bins=100, range=(-1,1))
plt.title('Accuracy Verification (x=0 is correct)')
plt.show()

report_df = pd.concat([X_test_df_reset, y_test_df_reset, prediction_df_reset, difference_df_reset], axis=1)

report_df

report_df.to_csv("./report_detect_defect.csv")
```

```
       index  defect
0       3046     0.0
1       6370     0.0
2      20084     0.0
```

```
3     21500     0.0
4     20102     0.0
...     ...     ...
3559  20636     0.0
3560  19298     0.0
3561  12974     0.0
3562  17817     0.0
3563   6645     1.0

[3564 rows x 2 columns]
[0.0, 0.0, 0.0, 0.0, 0.0] 3564
[0.010021302849054337, 0.009678870439529419, 0.07789280265569687, 0.0894283801317215, 0.008862419053912163] 3564
[-0.010021302849054337, -0.009678870439529419, -0.07789280265569687, -0.0894283801317215, -0.008862419053912163]
      defference_rate
0           -1.00e-02
1           -9.68e-03
2           -7.79e-02
3           -8.94e-02
4           -8.86e-03
...               ...
3559        -8.67e-02
3560        -7.83e-02
3561        -1.02e-01
3562        -1.10e-02
3563         8.81e-01

[3564 rows x 1 columns]

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3564 entries, 0 to 3563
Data columns (total 1 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   defference_rate  3564 non-null   float64
dtypes: float64(1)
memory usage: 28.0 KB
difference_df.info()==> None
difference_df.describe()==>        defference_rate
count          3.56e+03
mean           1.04e-03
std            2.20e-01
min           -5.46e-01
25%           -9.35e-02
50%           -1.30e-02
75%           -1.10e-02
max            9.54e-01
length comparison==> 3564 3564 3564 3564
```



In [51]:
```
print('Completed!!Completed!!!Completed!!!!Completed!!!!Completed!!!!Completed!!!!Completed!!!')
```

```
Completed!!Completed!!!Completed!!!!Completed!!!!Completed!!!!Completed!!!!Completed!!!
```

In [ ]:

In [ ]: