life_prediction02

June 12, 2021

0.1 Devices Life Prediction Version 0.2

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

Start from June 8, 2021

0.1.1 Sample Generator

[1]:

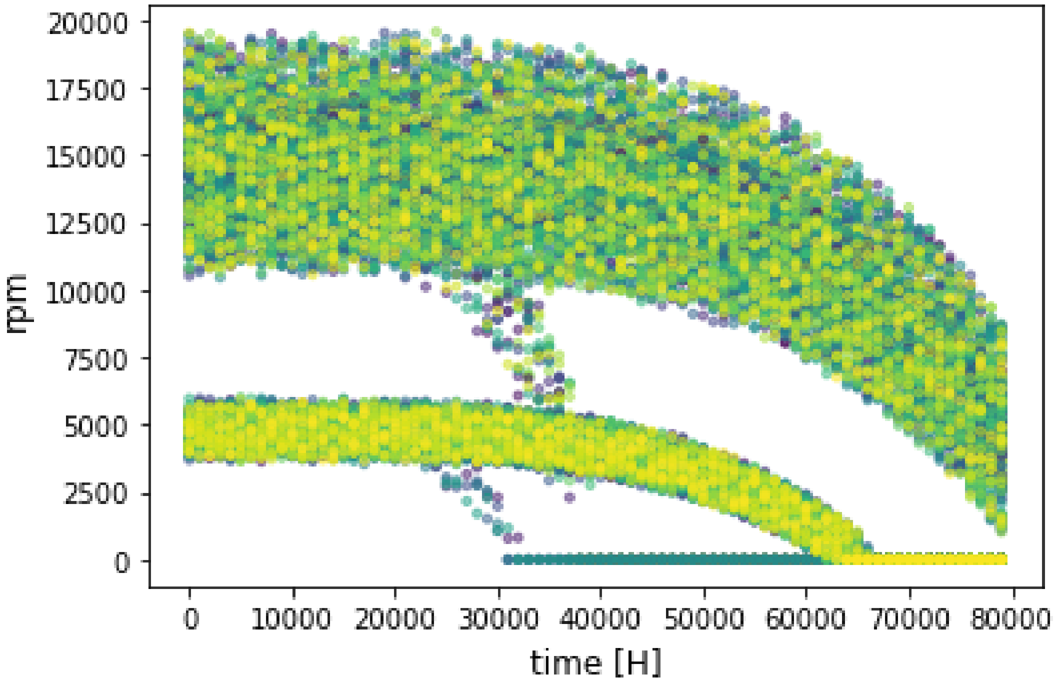
```
# Library import
import pandas as pd #  =DataFrame
import numpy as np #
import random # ramdom
import math #    log sin, cos
import matplotlib.pyplot as plt #
import matplotlib as mpl #

# DataFrame
pd.options.display.precision = 2

Sampling = 1000
NumSample = 100

#  FAN
def sample_generator(RpmSpec,
                     PowerSpec,
                     TempSpec,
                     DiameterSpec,
                     ThicknessSpec,
                     QSpec,
                     PSpec,
                     LifeSpec,
                     Sampling,
                     NumSample):

    data_list = [[]] # sampling list  2 list
#     cum_rpm = 0
#     cum_temp = 0
#     cum_power = 0

    for sample_id in range(NumSample):

        rpm = RpmSpec #      0
        cum_rpm = 0
        cum_temp = 0
        cum_power = 0
        cumurated_life_impact_factor = 0

        if np.random.random() < 0.1: #    10%        ==>rpm  power
            defect = 1
        else:
            defect = 0

        for time in range(0, LifeSpec*2, Sampling):

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

            if rpm <= 0: #  for
                power = 0
                death = 1
            else:

                if defect == 1: #
                    # rpm; 40degC   ,       +/-5%
                    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              +/-5%
                        power = (0.5 * (4000/rpm) ** 1.2 + PowerSpec * (temp /␣
 ↪TempSpec)) * (1 - random.uniform(-0.05,0.05))
                        death = 0
                        # remaining_life
                        remaining_life = ((RpmSpec*temp/TempSpec - 0.
 ↪1*RpmSpec)**(1/6)) * 8000 * (1 - random.uniform(-0.05,0.05)) - time
                        if remaining_life < 0:
                            remaining_life = 0
                else: #
                    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
                        remaining_life = ((RpmSpec*temp/TempSpec - 0.1*RpmSpec)**(1/4)) * 8000 * (1 - random.uniform(-0.05,0.05)) -time
                        if remaining_life < 0:
                            remaining_life = 0

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

            # FAN         rpm  ,  temp,  power
            # cumurated_life_impact_factor 0  +/-1

            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,
                            PSpec,
                            LifeSpec,
                            cumurated_life_impact_factor,
                            death,
                            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)
#  fan
list = fan40
for data in fan40cr:
    list.append(data)

for data in fan120:
    list.append(data)

df = pd.DataFrame(list,    # list 3        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',
                          '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['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)
```

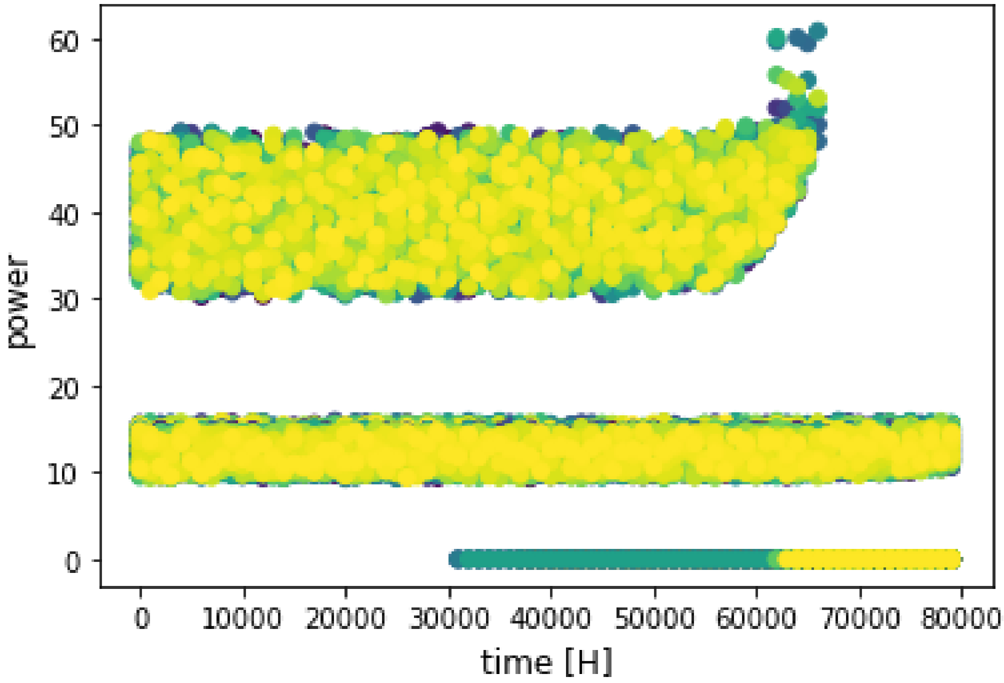
```
fig, ax = plt.subplots()
ax.scatter(df['time'], df['cumurated_life_impact_factor'], c=df['sample_id'])
plt.xlabel('time [H]',size=12)
plt.ylabel('cumurated_life_impact_factor',size=12)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24003 entries, 0 to 24002
Data columns (total 19 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   sample_id                     24000 non-null  float64
 1   defect                        24000 non-null  float64
 2   time                          24000 non-null  float64
 3   rpm                           24000 non-null  float64
 4   temp                          24000 non-null  float64
 5   power                         24000 non-null  float64
 6   cum_rpm                       24000 non-null  float64
 7   cum_temp                      24000 non-null  float64
 8   cum_power                     24000 non-null  float64
 9   RpmSpec                       24000 non-null  float64
 10  PowerSpec                     24000 non-null  float64
 11  DiameterSpec                  24000 non-null  float64
 12  ThicknessSpec                 24000 non-null  float64
 13  QSpec                         24000 non-null  float64
 14  PSpec                         24000 non-null  float64
 15  LifeSpec                      24000 non-null  float64
 16  cumurated_life_impact_factor  24000 non-null  float64
 17  death                         24000 non-null  float64
 18  remaining_life                24000 non-null  float64
dtypes: float64(19)
memory usage: 3.5 MB
df=  None
```

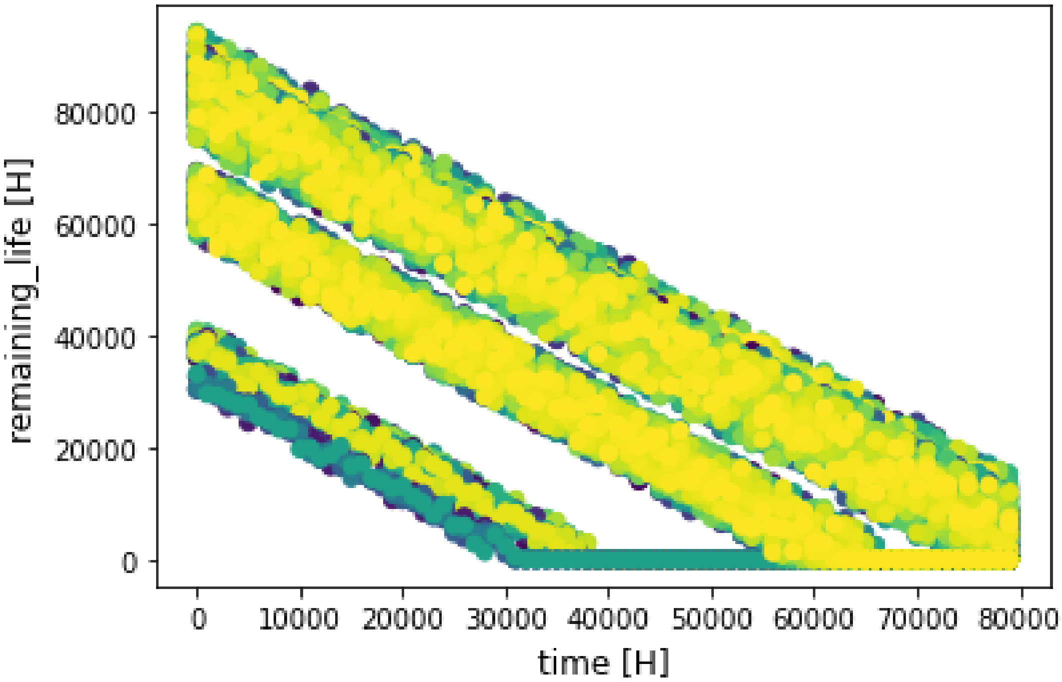
```
[1]: Text(0, 0.5, 'cumurated_life_impact_factor')
```

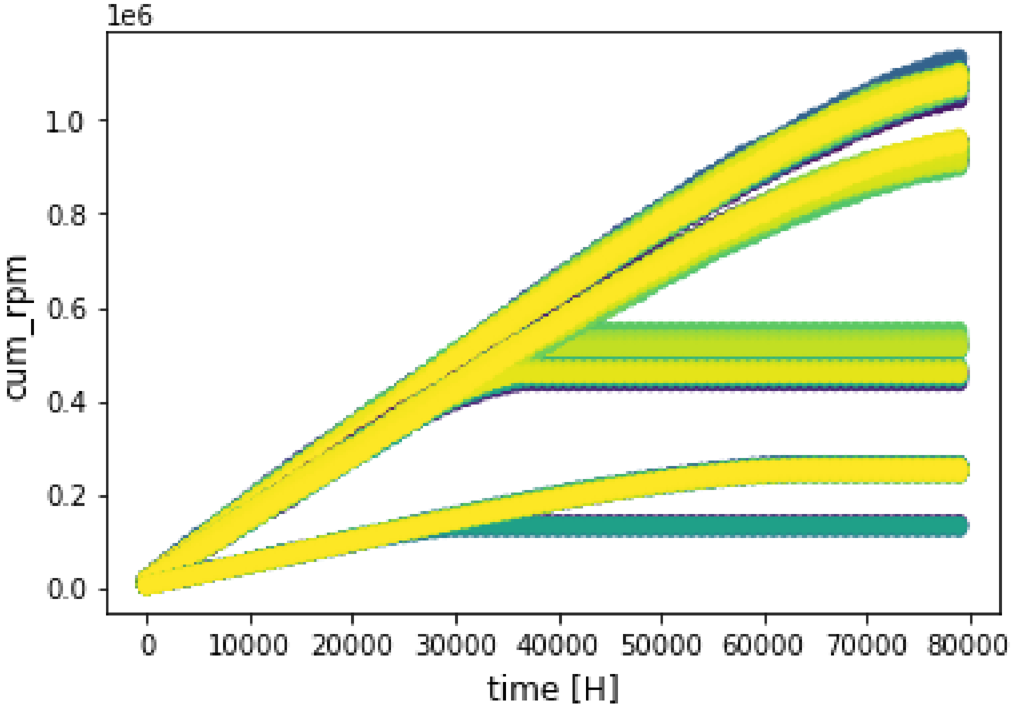


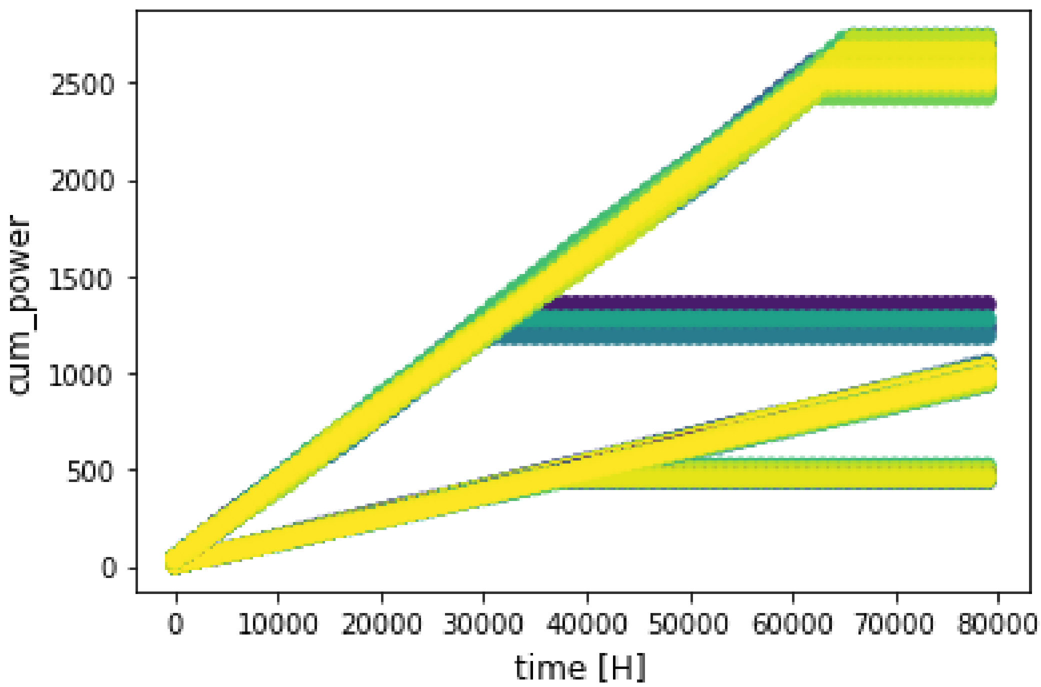
20000
17500
15000
12500
10000
7500
5000
2500
0
rpm
0
10000
20000
30000
40000
50000
60000
70000
80000
time [H]

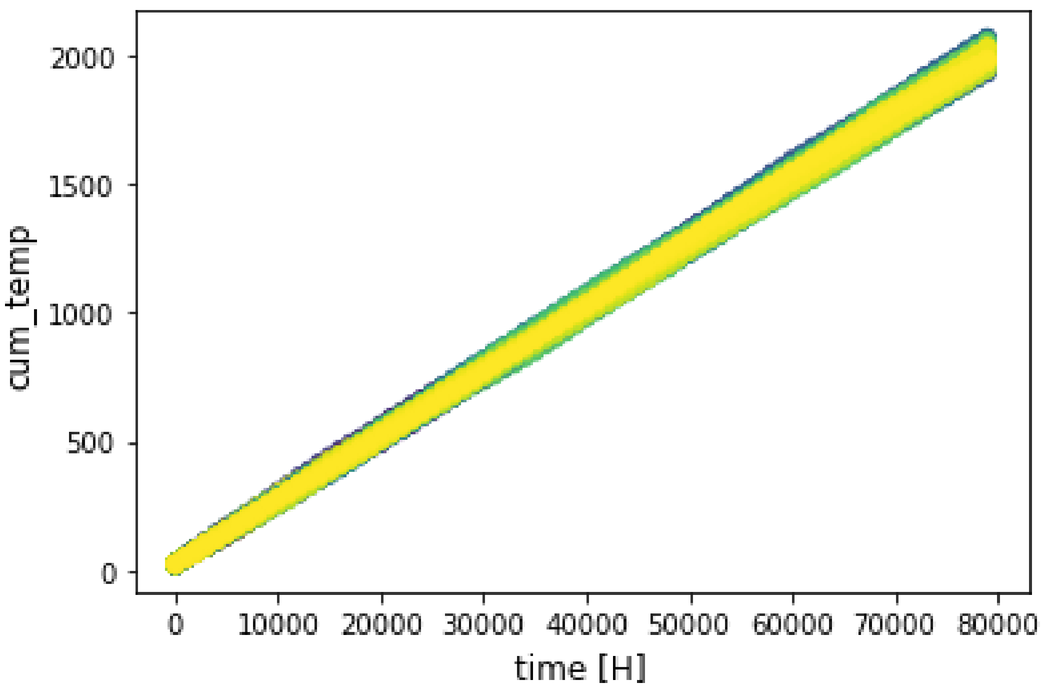


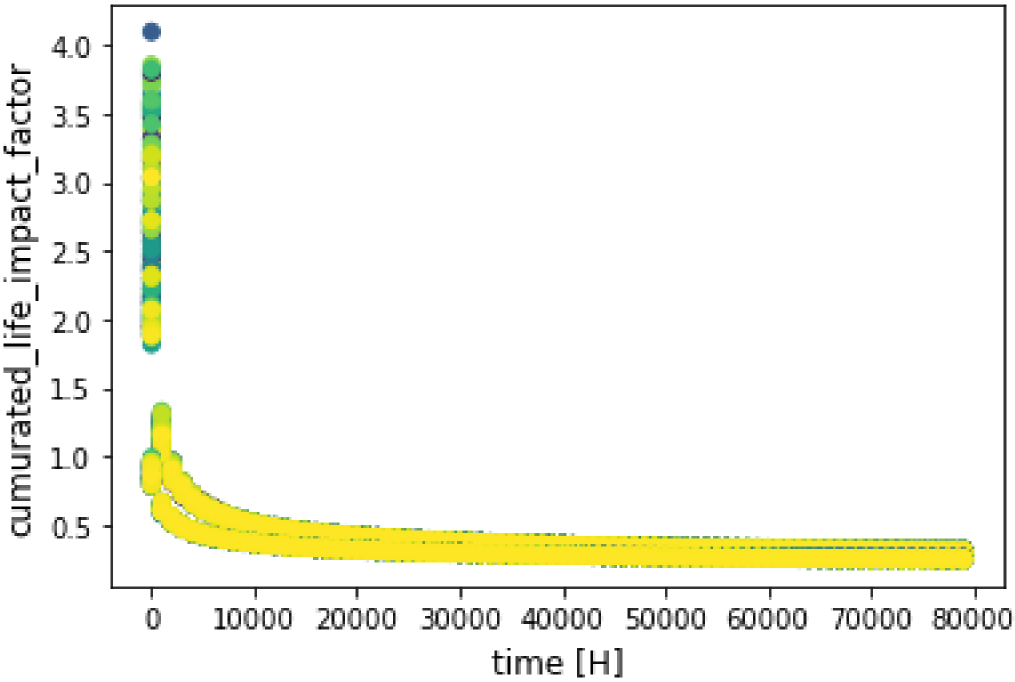
60
50
40
30
20
10
0
power
0
10000
20000
30000
40000
50000
60000
70000
80000
time [H]


remaining_life [H]
80000
60000
40000
20000
0
0
10000
20000
30000
40000
50000
60000
70000
80000
time [H]


1e6
cum_rpm
1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000
50000
60000
70000
80000
time [H]


cum_power
0
500
1000
1500
2000
2500
0
10000
20000
30000
40000
50000
60000
70000
80000
time [H]


cum_temp
0
500
1000
1500
2000
0
10000
20000
30000
40000
50000
60000
70000
80000
time [H]



```
[2]: 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.csv')
     df
```

```
[2]:        sample_id  defect     time       rpm   temp  power    cum_rpm  \
     1            0.0     0.0      0.0  14399.74  22.95  11.21   14399.74
     2            0.0     0.0   1000.0  14942.86  23.66  11.88   29342.61
     3            0.0     0.0   2000.0  14045.38  22.89  11.78   43387.99
     4            0.0     0.0   3000.0  12900.79  20.52  10.32   56288.78
     5            0.0     0.0   4000.0  14429.85  23.39  12.27   70718.63
     …              …     …        …         …     …      …          …
     23981       99.0     0.0  58000.0   1247.23  22.25  36.17  247698.27
     23982       99.0     0.0  59000.0   1344.03  23.52  36.72  249042.30
     23983       99.0     0.0  60000.0   1961.37  28.01  44.69  251003.67
     23984       99.0     0.0  61000.0    644.90  22.33  36.86  251648.56
     23985       99.0     0.0  62000.0   1452.19  27.35  42.94  253100.76
```

```
       cum_temp  cum_power  RpmSpec  PowerSpec  DiameterSpec  ThicknessSpec  \
1         22.95      11.21  25000.0      20.16          40.0           28.0
2         46.61      23.09  25000.0      20.16          40.0           28.0
3         69.50      34.88  25000.0      20.16          40.0           28.0
4         90.01      45.20  25000.0      20.16          40.0           28.0
5        113.40      57.47  25000.0      20.16          40.0           28.0
…         …        …      …        …            …            …
23981   1493.18    2370.01   7650.0      62.40         120.0           38.0
23982   1516.69    2406.73   7650.0      62.40         120.0           38.0
23983   1544.70    2451.42   7650.0      62.40         120.0           38.0
23984   1567.03    2488.27   7650.0      62.40         120.0           38.0
23985   1594.39    2531.21   7650.0      62.40         120.0           38.0

       QSpec  PSpec  LifeSpec  cumurated_life_impact_factor  death  \
1       0.83  1100.0   40000.0                          3.02    0.0
2       0.83  1100.0   40000.0                          1.25    0.0
3       0.83  1100.0   40000.0                          0.94    0.0
4       0.83  1100.0   40000.0                          0.81    0.0
5       0.83  1100.0   40000.0                          0.72    0.0
…      …     …       …                            …    …
23981   7.49   532.5   40000.0                          0.26    0.0
23982   7.49   532.5   40000.0                          0.26    0.0
23983   7.49   532.5   40000.0                          0.26    0.0
23984   7.49   532.5   40000.0                          0.26    0.0
23985   7.49   532.5   40000.0                          0.26    0.0

       remaining_life
1            86425.74
2            85610.30
3            83857.05
4            80516.46
5            82687.82
…               …
23981         1504.73
23982         6273.60
23983         3065.98
23984         2969.42
23985         6288.92

[21041 rows x 19 columns]
```

0.1.2 Prediction

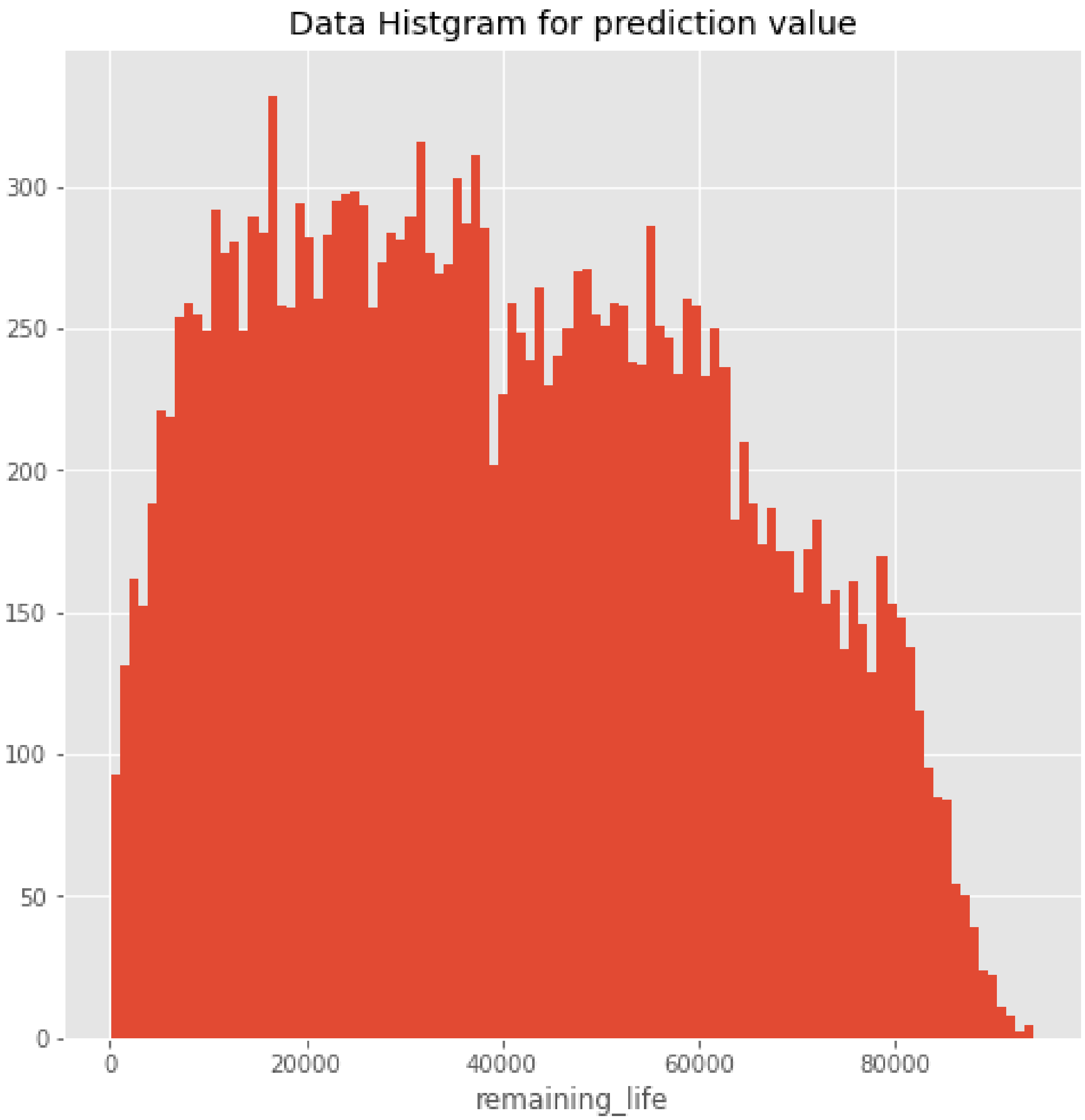
[3]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
plt.style.use('ggplot')
import seaborn as sns

df = pd.read_csv('sample_data.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: 21041 entries, 1 to 23985
Data columns (total 19 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   sample_id                      21041 non-null  float64
 1   defect                         21041 non-null  float64
 2   time                           21041 non-null  float64
 3   rpm                            21041 non-null  float64
 4   temp                           21041 non-null  float64
 5   power                          21041 non-null  float64
 6   cum_rpm                        21041 non-null  float64
 7   cum_temp                       21041 non-null  float64
 8   cum_power                      21041 non-null  float64
 9   RpmSpec                        21041 non-null  float64
 10  PowerSpec                      21041 non-null  float64
 11  DiameterSpec                   21041 non-null  float64
 12  ThicknessSpec                  21041 non-null  float64
 13  QSpec                          21041 non-null  float64
 14  PSpec                          21041 non-null  float64
 15  LifeSpec                       21041 non-null  float64
 16  cumurated_life_impact_factor   21041 non-null  float64
 17  death                          21041 non-null  float64
 18  remaining_life                 21041 non-null  float64
dtypes: float64(19)
memory usage: 3.2 MB
```

[4]:

```
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()
```



[5]:

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

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

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

# train       test
#          20%
```

```python
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)
```

```
(21041, 16)
(21041,)
(16832, 16)
(4209, 16)
```

[6]:

```python
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
# 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')])
```

```
[09:07:11] 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.
[09:07:11] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.4.0/src/learner.cc:573:
Parameters: { "silent" } might not be used.
```

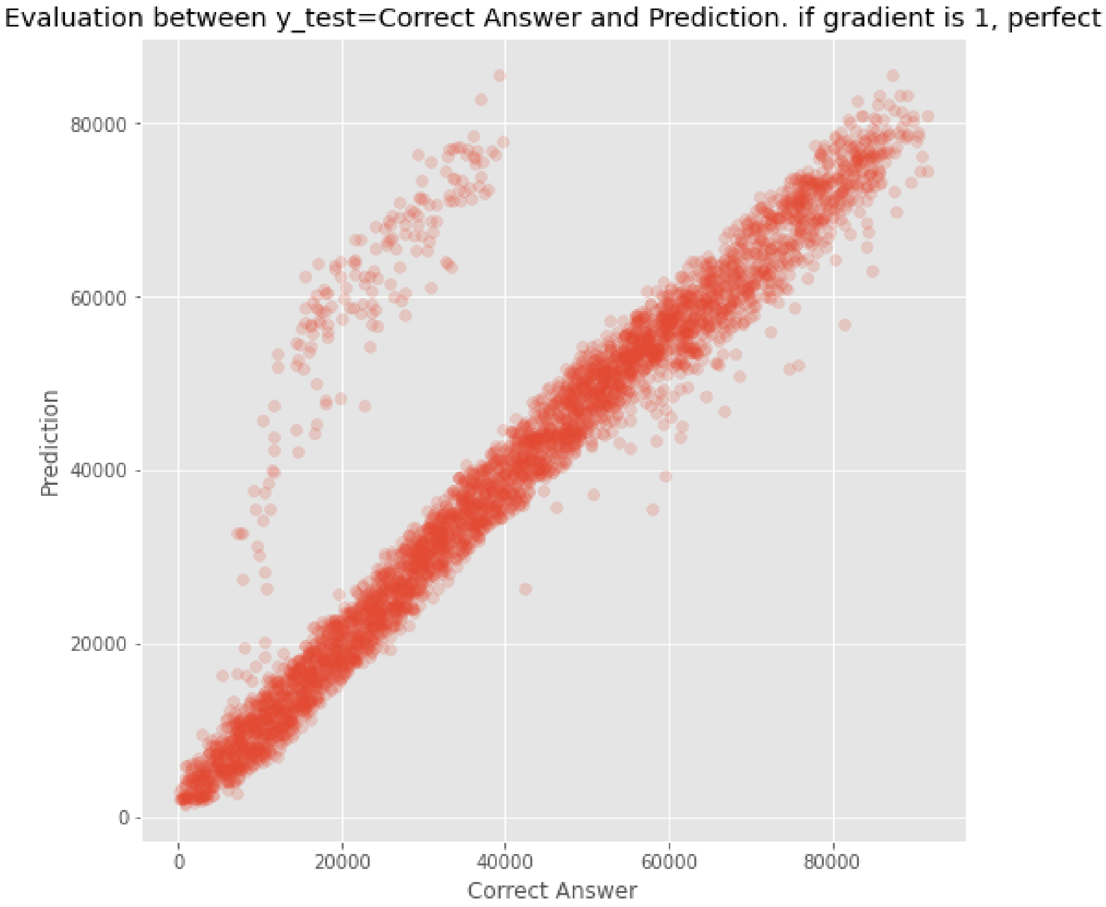
```
  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


[0]     test-rmse:41641.05859
[1]     test-rmse:37699.17969
[2]     test-rmse:34178.75391
[3]     test-rmse:31037.64648
[4]     test-rmse:28216.15430
[5]     test-rmse:25723.76758
[6]     test-rmse:23487.06055
[7]     test-rmse:21513.18359
[8]     test-rmse:19757.56250
[9]     test-rmse:18200.48047
[10]    test-rmse:16831.53711
[11]    test-rmse:15633.85352
[12]    test-rmse:14587.16113
[13]    test-rmse:13696.62305
[14]    test-rmse:12913.14648
[15]    test-rmse:12243.71387
[16]    test-rmse:11663.33301
[17]    test-rmse:11173.53906
[18]    test-rmse:10760.42383
[19]    test-rmse:10407.97461
[20]    test-rmse:10125.70508
[21]    test-rmse:9876.97754
[22]    test-rmse:9673.81543
[23]    test-rmse:9503.46191
[24]    test-rmse:9354.51172
[25]    test-rmse:9235.05664
[26]    test-rmse:9142.36914
[27]    test-rmse:9063.54785
[28]    test-rmse:8998.35938
[29]    test-rmse:8949.75684
[30]    test-rmse:8905.86914
[31]    test-rmse:8875.48047
[32]    test-rmse:8846.91406
[33]    test-rmse:8817.47266
[34]    test-rmse:8808.79102
[35]    test-rmse:8796.17481
[36]    test-rmse:8787.79199
[37]    test-rmse:8781.47266
[38]    test-rmse:8772.75488
[39]    test-rmse:8768.42481
[40]    test-rmse:8771.42676
```

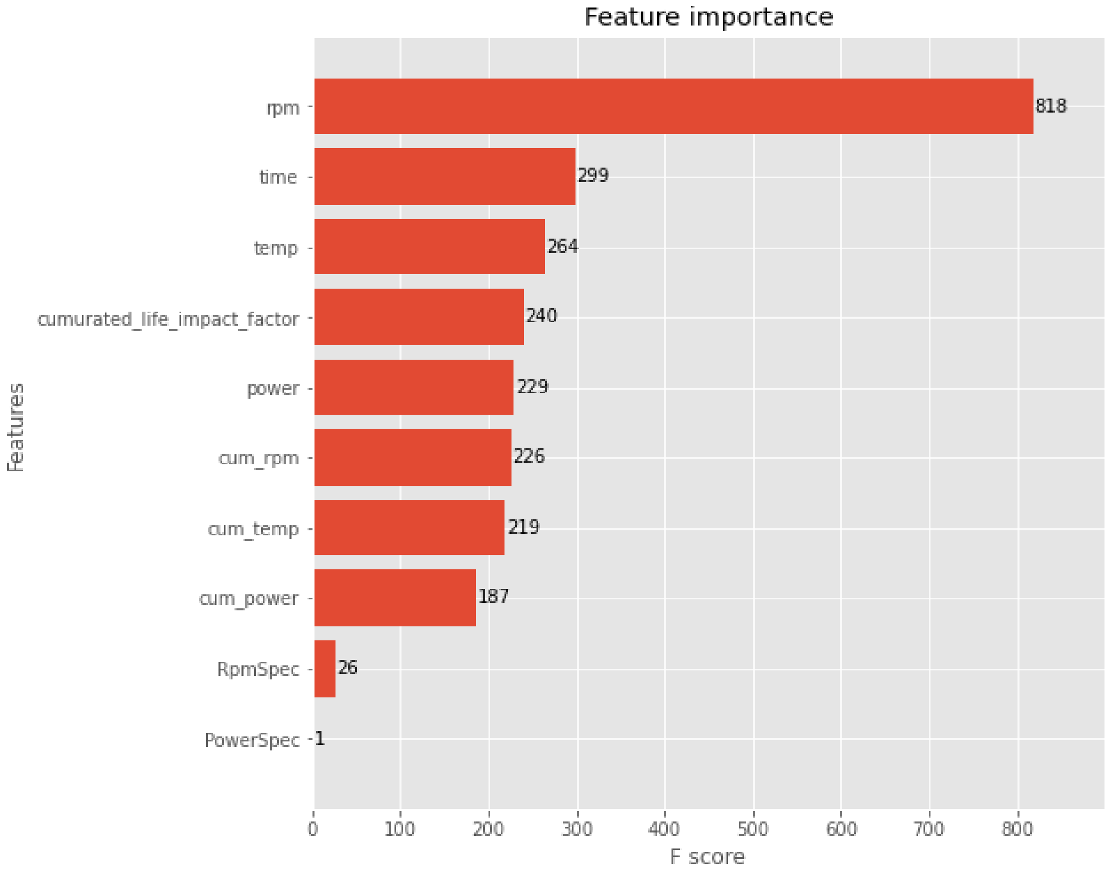
```
[41]    test-rmse:8767.30469
[42]    test-rmse:8767.16504
[43]    test-rmse:8757.79981
[44]    test-rmse:8759.59180
[45]    test-rmse:8764.60059
[46]    test-rmse:8767.15234
[47]    test-rmse:8771.22070
[48]    test-rmse:8771.06641
```

[7]:

```
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 0x000001A25D6E7EB0>
[09:07:12] 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(
```


Evaluation between y_test=Correct Answer and Prediction. if gradient is 1, perfect
Prediction
80000
60000
40000
20000
0
0
20000
40000
60000
80000
Correct Answer



[ ]:

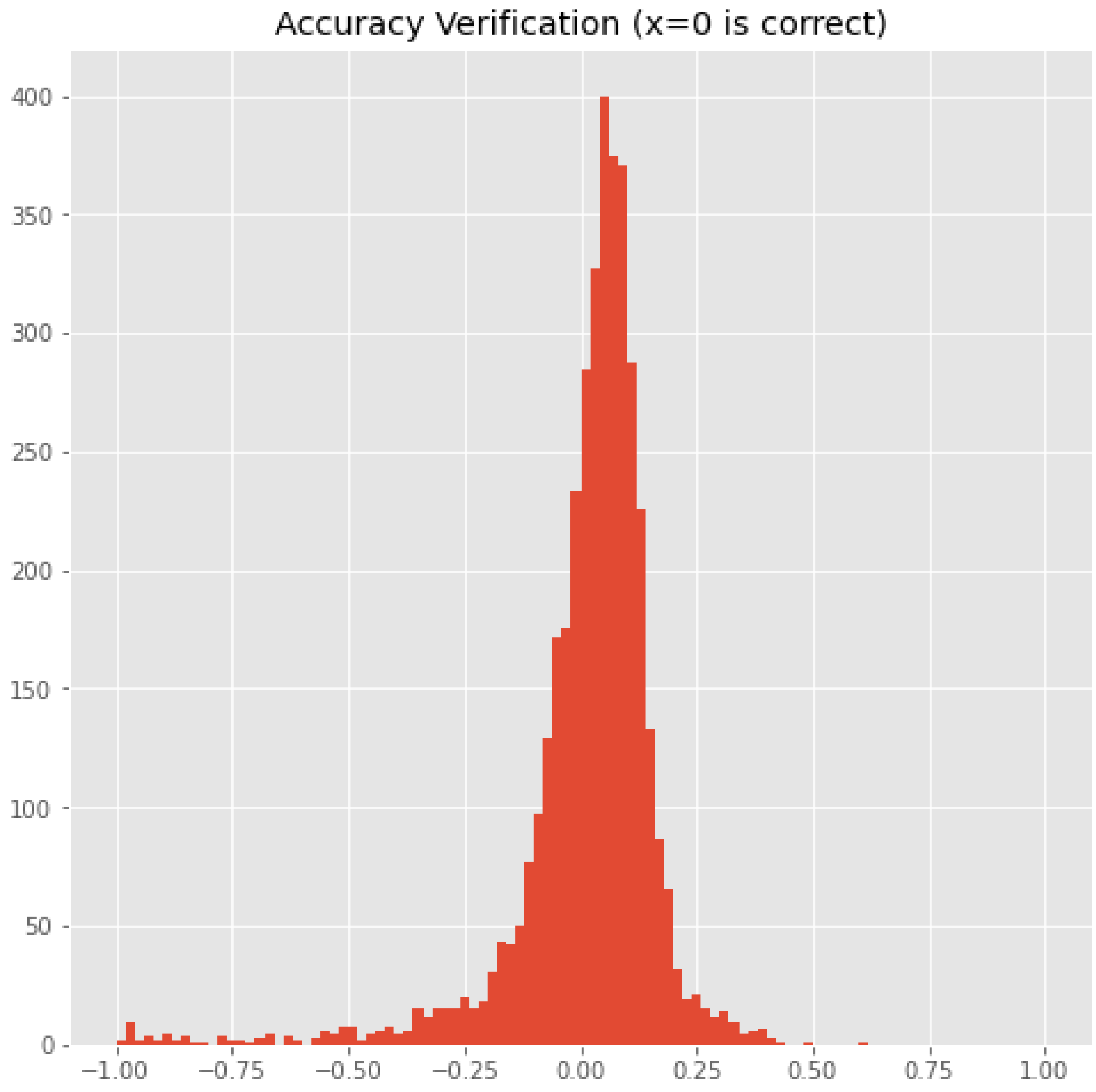
[8]:

```
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     23242        21799.93
1     12692        27539.23
2      4334        66073.56
3     11191        17004.39
4     20434        27982.34
…     …                 …
4204  11936        69472.39
4205   3946        62116.93
4206   5034        11273.14
4207   1578        32261.88
4208  17770        59830.38

[4209 rows x 2 columns]
[21799.9262274026, 27539.23311048046, 66073.56435337233, 17004.385899645626,
27982.33607424148] 4209
[21000.203125, 31099.66015625, 59397.828125, 13450.2841796875, 29800.15625] 4209
[0.036684670125045886, -0.1292856279434508, 0.101034903954468,
0.20901088348225463, -0.06496313141746136]
```

```
      defference_rate
0                0.04
1               -0.13
2                0.10
3                0.21
4               -0.06
…               …
4204             0.03
4205             0.11
4206             0.31
4207             0.11
4208             0.10

[4209 rows x 1 columns]

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Data columns (total 1 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   defference_rate  4209 non-null   float64
dtypes: float64(1)
memory usage: 33.0 KB
difference_df.info()==> None
difference_df.describe()==>        defference_rate
count          4209.00
mean             -0.09
std               0.67
min             -25.61
25%              -0.04
50%               0.04
75%               0.09
max               0.62
length comparison==> 4209 4209 4209 4209
```



[ ]:

[ ]:

Can Defects be detected?

[9]:

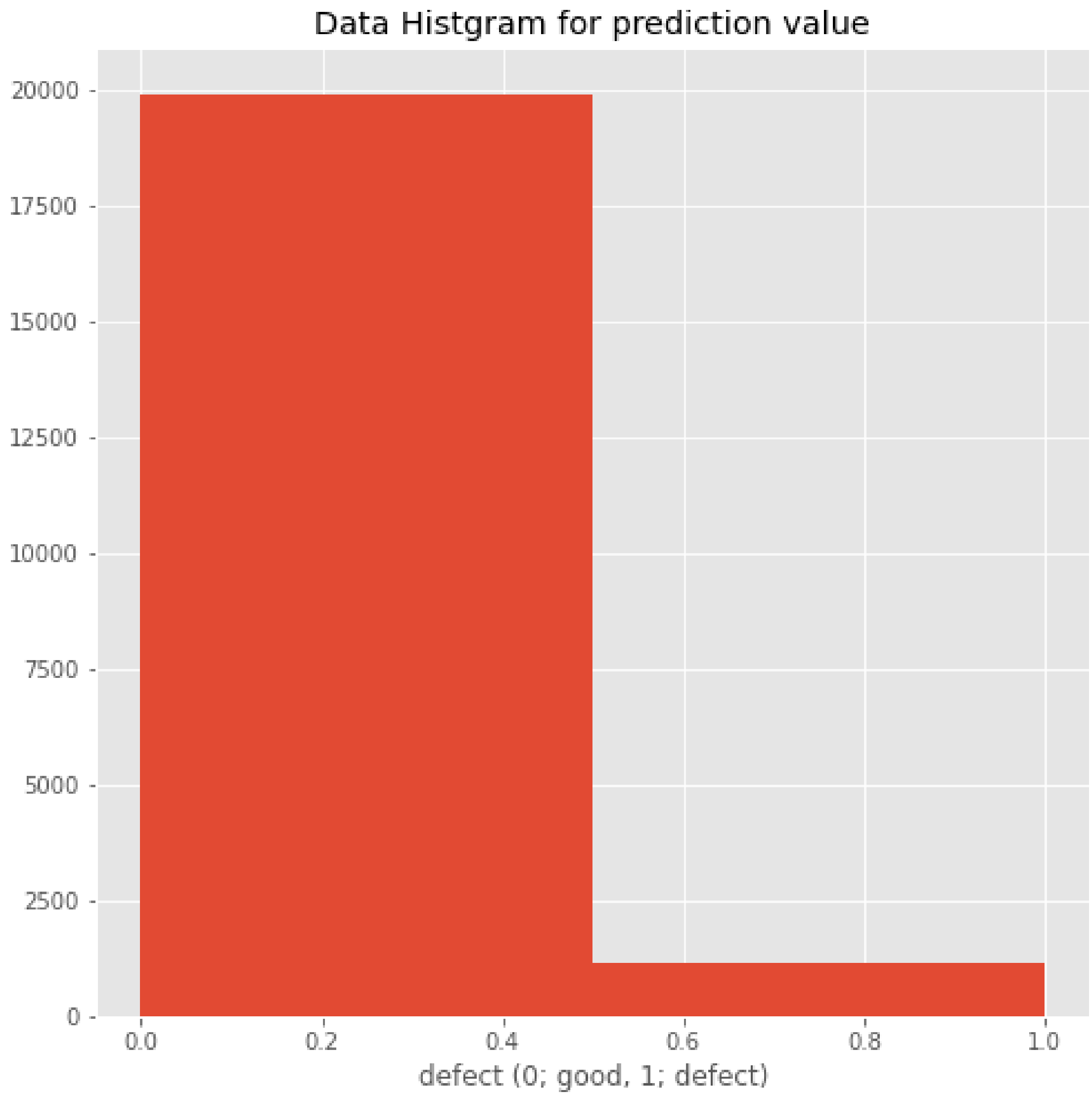
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
import seaborn as sns
```

```
df = pd.read_csv('sample_data.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: 21041 entries, 1 to 23985
Data columns (total 19 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   sample_id                      21041 non-null  float64
 1   defect                         21041 non-null  float64
 2   time                           21041 non-null  float64
 3   rpm                            21041 non-null  float64
 4   temp                           21041 non-null  float64
 5   power                          21041 non-null  float64
 6   cum_rpm                        21041 non-null  float64
 7   cum_temp                       21041 non-null  float64
 8   cum_power                      21041 non-null  float64
 9   RpmSpec                        21041 non-null  float64
 10  PowerSpec                      21041 non-null  float64
 11  DiameterSpec                   21041 non-null  float64
 12  ThicknessSpec                  21041 non-null  float64
 13  QSpec                          21041 non-null  float64
 14  PSpec                          21041 non-null  float64
 15  LifeSpec                       21041 non-null  float64
 16  cumurated_life_impact_factor   21041 non-null  float64
 17  death                          21041 non-null  float64
 18  remaining_life                 21041 non-null  float64
dtypes: float64(19)
memory usage: 3.2 MB
```

[10]:

```
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()
```



[11]:

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

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

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

# train      test
#          20%
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)
```

```
(21041, 16)
(21041,)
(16832, 16)
(4209, 16)
```

[12]:

```
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
# 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')])
```

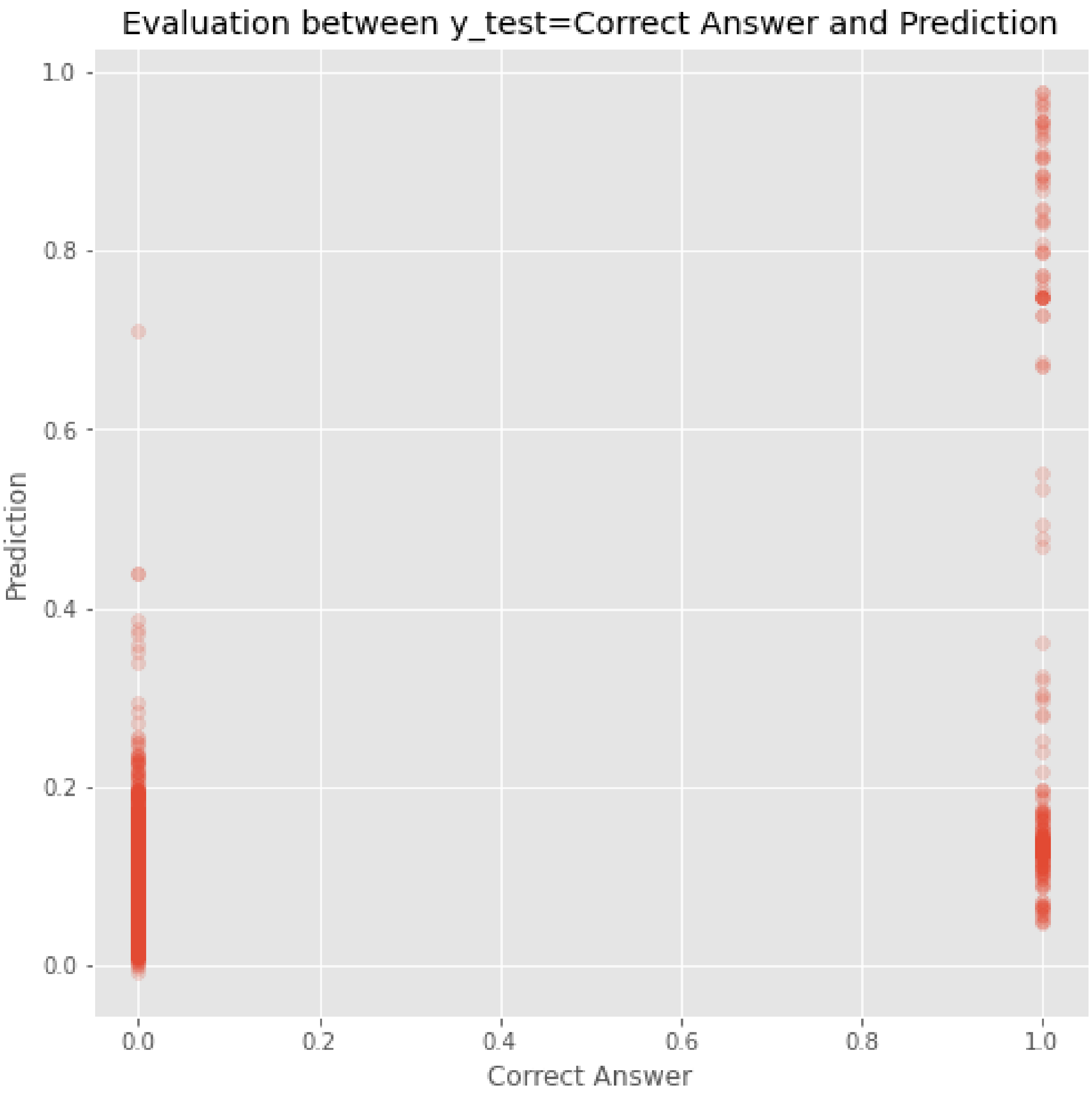
```
[09:07:16] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now
deprecated in favor of reg:squarederror.
[09:07:16] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.4.0/src/learner.cc:573:
Parameters: { "silent" } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
```

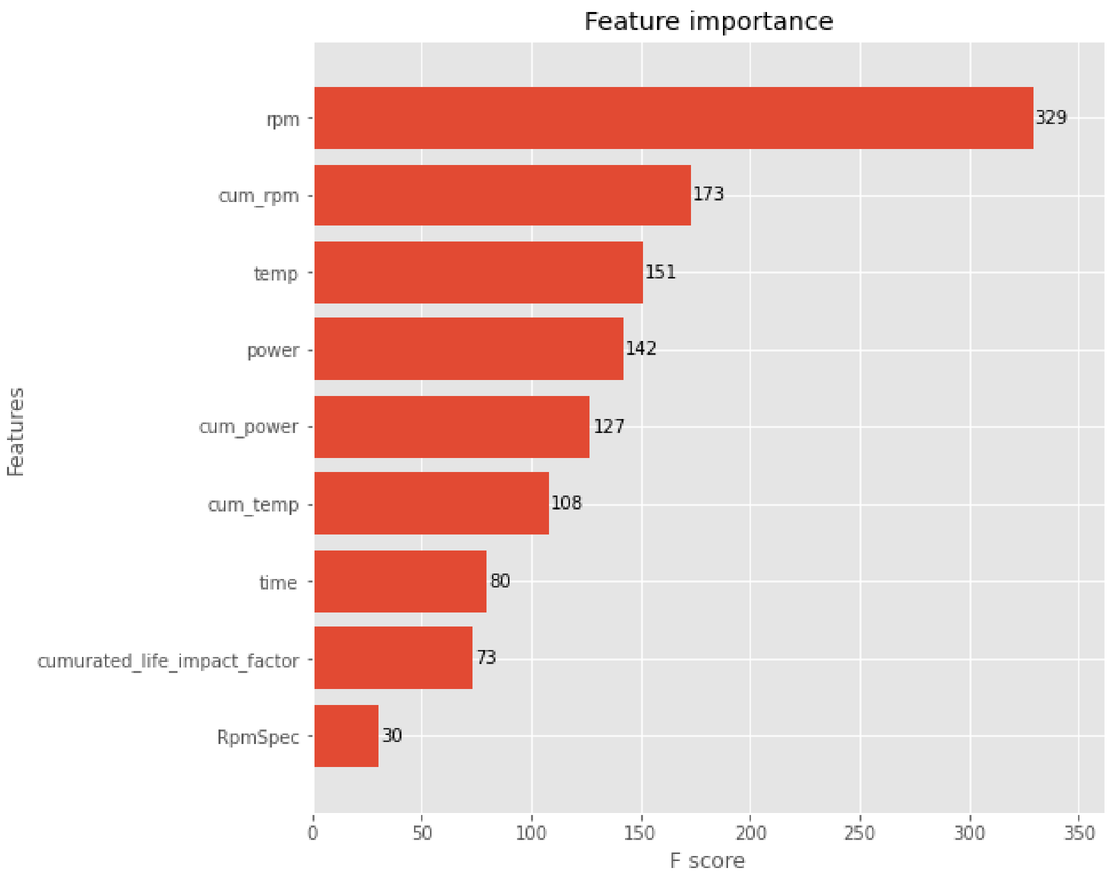
```
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


[0]     test-rmse:0.45762
[1]     test-rmse:0.42024
[2]     test-rmse:0.38718
[3]     test-rmse:0.35830
[4]     test-rmse:0.33291
[5]     test-rmse:0.31060
[6]     test-rmse:0.29151
[7]     test-rmse:0.27465
[8]     test-rmse:0.26037
[9]     test-rmse:0.24807
[10]    test-rmse:0.23767
[11]    test-rmse:0.22901
[12]    test-rmse:0.22181
[13]    test-rmse:0.21548
[14]    test-rmse:0.21042
[15]    test-rmse:0.20578
[16]    test-rmse:0.20226
[17]    test-rmse:0.19900
[18]    test-rmse:0.19672
[19]    test-rmse:0.19479
[20]    test-rmse:0.19304
[21]    test-rmse:0.19179
[22]    test-rmse:0.19074
[23]    test-rmse:0.18996
[24]    test-rmse:0.18924
[25]    test-rmse:0.18862
[26]    test-rmse:0.18809
[27]    test-rmse:0.18770
[28]    test-rmse:0.18738
[29]    test-rmse:0.18721
[30]    test-rmse:0.18701
[31]    test-rmse:0.18683
[32]    test-rmse:0.18686
[33]    test-rmse:0.18679
[34]    test-rmse:0.18656
[35]    test-rmse:0.18662
[36]    test-rmse:0.18658
[37]    test-rmse:0.18662
[38]    test-rmse:0.18662
```

[13]:

```
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 0x000001A25E5AE130>
[09:07:17] 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(
```


Evaluation between y_test=Correct Answer and Prediction
Prediction
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Correct Answer

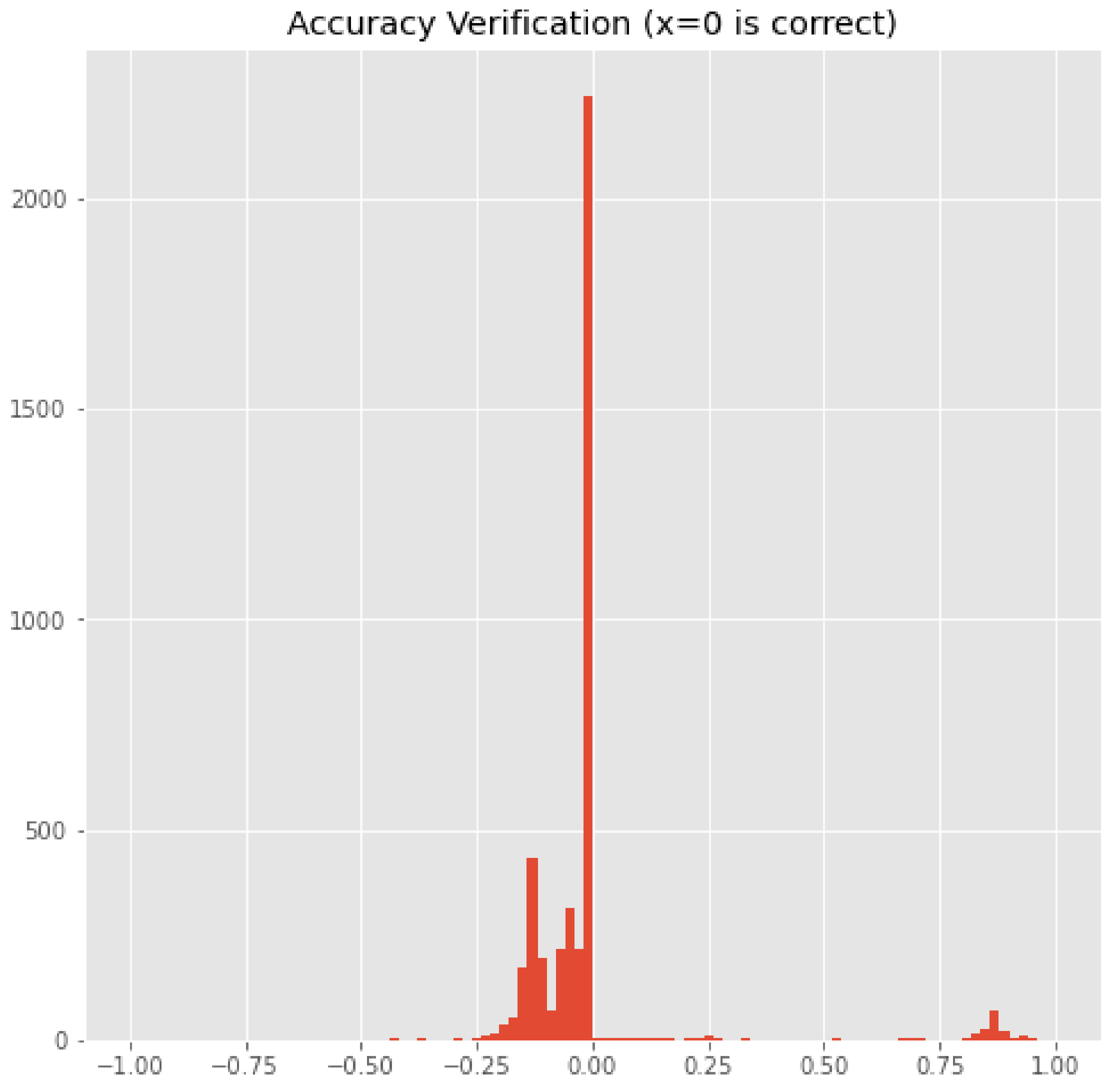


[14]:

```
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     23242     0.0
1     12692     0.0
2      4334     0.0
3     11191     0.0
4     20434     0.0
…       …       …
4204  11936     0.0
4205   3946     0.0
4206   5034     0.0
4207   1578     0.0
4208  17770     0.0

[4209 rows x 2 columns]
[0.0, 0.0, 0.0, 0.0, 0.0] 4209
[0.03570520877838135, 0.013064580038189888, 0.1945408284664154,
0.013400504365563393, 0.015158231370151043] 4209
[-0.03570520877838135, -0.013064580038189888, -0.1945408284664154,
-0.013400504365563393, -0.015158231370151043]
      defference_rate
0               -0.04
```

```
1               -0.01
2               -0.19
3               -0.01
4               -0.02
…               …
4204            -0.13
4205            -0.06
4206            -0.01
4207            -0.01
4208            -0.05

[4209 rows x 1 columns]

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Data columns (total 1 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   defference_rate  4209 non-null   float64
dtypes: float64(1)
memory usage: 33.0 KB
difference_df.info()==> None
difference_df.describe()==>        defference_rate
count          4209.00
mean             -0.01
std               0.19
min              -0.71
25%              -0.07
50%              -0.02
75%              -0.01
max               0.95
length comparison==> 4209 4209 4209 4209
```



```
[15]: print('Completed!!Completed!!!Completed!!!!Completed!!!!Completed!!!!Completed!!
       ↪!!Completed!!!')
```

```
Completed!!Completed!!!Completed!!!!Completed!!!!Completed!!!!Completed!!!!Compl
eted!!!
```

```
[ ]:
```

```
[ ]:
```