Feature Engineering (Part-2)

(continue from FE_part_1)

The goal of this case study is to build a predictive model that a machine will fail in the next 24 hours due to a certain component failure (component 1, 2, 3, or 4) or not.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Importing libraries
import os
import sys
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
matplotlib.style.use("Solarize_Light2")
%matplotlib inline
```

```
#Loading all the dataset using Pandas library from gdrive
import pandas as pd

telemetry = pd.read_csv('/content/drive/MyDrive/AAIC/Case_study_1/PdM_telemetry.csv')
errors = pd.read_csv('/content/drive/MyDrive/AAIC/Case_study_1/PdM_errors.csv')
maint = pd.read_csv('/content/drive/MyDrive/AAIC/Case_study_1/PdM_maint.csv')
failures = pd.read_csv('/content/drive/MyDrive/AAIC/Case_study_1/PdM_failures.csv')
machines = pd.read_csv('/content/drive/MyDrive/AAIC/Case_study_1/PdM_machines.csv')
```

```
# #Loading all the datasets using Pandas library from local PC
# import pandas as pd

# telemetry = pd.read_csv('PdM_telemetry.csv')
# errors = pd.read_csv('PdM_errors.csv')
# maint = pd.read_csv('PdM_maint.csv')
# failures = pd.read_csv('PdM_failures.csv')
# machines = pd.read_csv('PdM_machines.csv')
```

```
# Formating datetime field.
telemetry['datetime'] = pd.to_datetime(telemetry['datetime'], format="%Y-%m-%d %H:%M:%S")

errors['datetime'] = pd.to_datetime(errors['datetime'], format="%Y-%m-%d %H:%M:%S")
errors['errorID'] = errors['errorID'].astype('category')

maint['datetime'] = pd.to_datetime(maint['datetime'], format="%Y-%m-%d %H:%M:%S")
maint['comp'] = maint['comp'].astype('category')

machines['model'] = machines['model'].astype('category')

failures['datetime'] = pd.to_datetime(failures['datetime'], format="%Y-%m-%d %H:%M:%S")
failures['failure'] = failures['failure'].astype('category')
```

Lag Features from Telemetry data

Lag features are the classical way that time series forecasting problems are transformed into supervised learning problems.

Telemetry data comes with time-stamps which makes it suitable for calculating lagging features. A common method is to pick a window size for the lag features to be created and compute rolling aggregate measures such as mean, standard deviation, minimum, maximum, etc. to represent the short term history of the telemetry over the lag window. In the following, pandas function "resample" on min, max, mean and standard deviation of the telemetry data over the last 3 hour lag window will be calculated for every 3 hours.

- Reference: https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/
  #Reference: https://stackoverflow.com/questions/45370666/what-are-pandas-expanding-window-functions
  #Reference: https://towardsdatascience.com/using-the-pandas-resample-function-a231144194c4

```
# Calculate "resample min values" over the last 3 hour lag window for telemetry features.
temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
    temp.append(pd.pivot_table(telemetry,
                               index='datetime',
                               columns='machineID',
                        values=col).resample('3H', closed='left', label='right').min().unstack())

telemetry_min_3h = pd.concat(temp, axis=1)
telemetry_min_3h.columns = [i + '_min_3h' for i in fields]
telemetry_min_3h.reset_index(inplace=True)

telemetry_min_3h.head()
```

	machineID	datetime	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h
0	1	2015-01-01 09:00:00	162.879223	402.747490	75.237905	34.178847
1	1	2015-01-01 12:00:00	157.610021	346.149335	95.927042	25.990511
2	1	2015-01-01 15:00:00	156.556031	398.648781	101.001083	35.482009
3	1	2015-01-01 18:00:00	160.263954	382.483543	96.480976	38.543681
4	1	2015-01-01 21:00:00	153.353492	402.461187	86.012440	39.739883

```
# Calculate "resample max values" over the last 3 hour lag window for telemetry features.
temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
    temp.append(pd.pivot_table(telemetry,
                               index='datetime',
                               columns='machineID',
                        values=col).resample('3H', closed='left', label='right').max().unstack())

telemetry_max_3h = pd.concat(temp, axis=1)
telemetry_max_3h.columns = [i + '_max_3h' for i in fields]
telemetry_max_3h.reset_index(inplace=True)

telemetry_max_3h.head()
```

	machineID	datetime	volt_max_3h	rotate_max_3h	pressure_max_3h	vibration_max_3h
0	1	2015-01-01 09:00:00	176.217853	527.349825	113.077935	45.087686
1	1	2015-01-01 12:00:00	172.504839	435.376873	111.886648	41.122144
2	1	2015-01-01 15:00:00	175.324524	499.071623	111.755684	45.482287
3	1	2015-01-01 18:00:00	169.218423	460.850670	104.848230	42.675800
4	1	2015-01-01 21:00:00	182.739113	490.672921	93.484954	44.108554

```
# Calculate "resample mean values" over the last 3 hour lag window for telemetry features.
temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
    temp.append(pd.pivot_table(telemetry,
                               index='datetime',
                               columns='machineID',
                        values=col).resample('3H', closed='left', label='right').mean().unstack())

telemetry_mean_3h = pd.concat(temp, axis=1)
telemetry_mean_3h.columns = [i + '_mean_3h' for i in fields]
telemetry_mean_3h.reset_index(inplace=True)
```

```
telemetry_mean_3h.head()
```

	machineID	datetime	volt_mean_3h	rotate_mean_3h	pressure_mean_3h	vibration_mean_3h
0	1	2015-01-01 09:00:00	170.028993	449.533798	94.592122	40.893502
1	1	2015-01-01 12:00:00	164.192565	403.949857	105.687417	34.255891
2	1	2015-01-01 15:00:00	168.134445	435.781707	107.793709	41.239405
3	1	2015-01-01 18:00:00	165.514453	430.472823	101.703289	40.373739
4	1	2015-01-01 21:00:00	168.809347	437.111120	90.911060	41.738542

```
# Calculate "resample standard deviation" over the last 3 hour lag window for telemetry features.
temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
    temp.append(pd.pivot_table(telemetry,
                               index='datetime',
                               columns='machineID',
                        values=col).resample('3H', closed='left', label='right').std().unstack())

telemetry_sd_3h = pd.concat(temp, axis=1)
telemetry_sd_3h.columns = [i + '_sd_3h' for i in fields]
telemetry_sd_3h.reset_index(inplace=True)

telemetry_sd_3h.head()
```

	machineID	datetime	volt_sd_3h	rotate_sd_3h	pressure_sd_3h	vibration_sd_3h
0	1	2015-01-01 09:00:00	6.721032	67.849599	18.934956	5.874970
1	1	2015-01-01 12:00:00	7.596570	50.120452	8.555032	7.662229
2	1	2015-01-01 15:00:00	10.124584	55.084734	5.909721	5.169304
3	1	2015-01-01 18:00:00	4.673269	42.047278	4.554047	2.106108
4	1	2015-01-01 21:00:00	14.752132	47.048609	4.244158	2.207884

For capturing a longer term effect, 24 hour lag features will be calculated as below.

```
#Calculate "rolling min" over the last 24 hour lag window for telemetry features.

temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
        temp.append(pd.pivot_table(telemetry,  index='datetime',
                                               columns='machineID',
                                   values=col).rolling(window=24,center=False).min().resample('3H',
                                                                                              closed='left',
                                                                                    label='right').first().unstack())


telemetry_min_24h = pd.concat(temp, axis=1)
telemetry_min_24h.columns = [i + '_min_24h' for i in fields]
telemetry_min_24h.reset_index(inplace=True)
telemetry_min_24h = telemetry_min_24h.loc[-telemetry_min_24h['volt_min_24h'].isnull()]

# Notice that a 24h rolling min is not available at the earliest timepoints
telemetry_min_24h.head(10)
```

	machineID	datetime	volt_min_24h	rotate_min_24h	pressure_min_24h	vibration_min_24h
7	1	2015-01-02 06:00:00	151.335682	346.149335	75.237905	25.990511
8	1	2015-01-02 09:00:00	151.335682	346.149335	75.237905	25.990511
9	1	2015-01-02 12:00:00	147.300678	382.483543	78.880780	25.990511
10	1	2015-01-02 15:00:00	147.300678	382.483543	78.880780	29.527665

```
#Calculate "rolling max" over the last 24 hour lag window for telemetry features.

temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
        temp.append(pd.pivot_table(telemetry,  index='datetime',
                                               columns='machineID',
                                   values=col).rolling(window=24,center=False).max().resample('3H',
                                                                                              closed='left',
                                                                                    label='right').first().unstack())


telemetry_max_24h = pd.concat(temp, axis=1)
telemetry_max_24h.columns = [i + '_max_24h' for i in fields]
telemetry_max_24h.reset_index(inplace=True)
telemetry_max_24h = telemetry_max_24h.loc[-telemetry_max_24h['volt_max_24h'].isnull()]

# Notice that a 24h rolling max is not available at the earliest timepoints
telemetry_max_24h.head(10)
```

	machineID	datetime	volt_max_24h	rotate_max_24h	pressure_max_24h	vibration_max_24h
7	1	2015-01-02 06:00:00	200.872430	527.349825	113.077935	52.355876
8	1	2015-01-02 09:00:00	200.872430	527.349825	114.342061	52.355876
9	1	2015-01-02 12:00:00	200.872430	519.452812	114.342061	52.355876
10	1	2015-01-02 15:00:00	200.872430	519.452812	114.342061	52.355876
11	1	2015-01-02 18:00:00	200.872430	519.452812	114.342061	52.355876
12	1	2015-01-02 21:00:00	200.872430	519.452812	127.014498	52.355876
13	1	2015-01-03 00:00:00	200.872430	519.452812	127.014498	52.355876
14	1	2015-01-03 03:00:00	200.872430	519.452812	127.014498	52.355876
15	1	2015-01-03 06:00:00	200.872430	521.837936	127.014498	52.355876
16	1	2015-01-03 09:00:00	194.942847	521.837936	127.014498	52.997327

```
#Calculate "rolling mean" over the last 24 hour lag window for telemetry features.

temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
        temp.append(pd.pivot_table(telemetry,  index='datetime',
                                               columns='machineID',
                                     values=col).rolling(window=24,center=False).mean().resample('3H',
                                                                                                 closed='left',
                                                                                       label='right').first().unstack())


telemetry_mean_24h = pd.concat(temp, axis=1)
telemetry_mean_24h.columns = [i + '_mean_24h' for i in fields]
telemetry_mean_24h.reset_index(inplace=True)
telemetry_mean_24h = telemetry_mean_24h.loc[-telemetry_mean_24h['volt_mean_24h'].isnull()]

# Notice that a 24h rolling average is not available at the earliest timepoints
telemetry_mean_24h.head(10)
```

	machineID	datetime	volt_mean_24h	rotate_mean_24h	pressure_mean_24h	vibration_mean_24h
7	1	2015-01-02 06:00:00	169.733809	445.179865	96.797113	40.385160
8	1	2015-01-02 09:00:00	170.614862	446.364859	96.849785	39.736826
9	1	2015-01-02 12:00:00	169.893965	447.009407	97.715600	39.498374
10	1	2015-01-02 15:00:00	171.243444	444.233563	96.666060	40.229370
11	1	2015-01-02 18:00:00	170.792486	448.440437	95.766838	40.055214
12	1	2015-01-02 21:00:00	170.556674	452.267095	98.065860	40.033247
13	1	2015-01-03 00:00:00	168.460525	451.031783	99.273286	38.903462
14	1	2015-01-03 03:00:00	169.772951	447.502464	99.005946	39.389725

```
#Calculate "rolling standard deviation" over the last 24 hour lag window for telemetry features.

temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
        temp.append(pd.pivot_table(telemetry,  index='datetime',
                                             columns='machineID',
                           values=col).rolling(window=24,center=False).std().resample('3H',
                                              closed='left', label='right').first().unstack())

telemetry_sd_24h = pd.concat(temp, axis=1)
telemetry_sd_24h.columns = [i + '_sd_24h' for i in fields]
telemetry_sd_24h.reset_index(inplace=True)
telemetry_sd_24h = telemetry_sd_24h.loc[-telemetry_sd_24h['volt_sd_24h'].isnull()]

# Notice that a 24h rolling std. deviation is not available at the earliest timepoints
telemetry_sd_24h.head(10)
```

	machineID	datetime	volt_sd_24h	rotate_sd_24h	pressure_sd_24h	vibration_sd_24h
7	1	2015-01-02 06:00:00	11.233120	48.717395	10.079880	5.853209
8	1	2015-01-02 09:00:00	12.519402	48.385076	10.171540	6.163231
9	1	2015-01-02 12:00:00	13.370357	42.432317	9.471669	6.195076
10	1	2015-01-02 15:00:00	13.299281	41.346121	8.731229	5.687944
11	1	2015-01-02 18:00:00	13.954518	43.490234	8.061653	5.898069
12	1	2015-01-02 21:00:00	14.402740	42.626186	10.408012	5.941890
13	1	2015-01-03 00:00:00	15.513819	40.395881	10.833294	5.737671
14	1	2015-01-03 03:00:00	15.726970	39.648116	11.904700	5.601191
15	1	2015-01-03 06:00:00	15.635083	41.828592	11.326412	5.583521
16	1	2015-01-03 09:00:00	13.995465	40.843882	11.036546	5.561553

Now, the above new columns of the feature datasets will be merged below to create the final features set from telemetry.

```
# Merge columns of feature sets created earlier
telemetry_feat = pd.concat([telemetry_min_3h,
                            telemetry_max_3h.iloc[:, 2:6],
                            telemetry_mean_3h.iloc[:, 2:6],
                            telemetry_sd_3h.iloc[:, 2:6],
                            telemetry_min_24h.iloc[:, 2:6],
                            telemetry_max_24h.iloc[:, 2:6],
                            telemetry_mean_24h.iloc[:, 2:6],
                            telemetry_sd_24h.iloc[:, 2:6]], axis=1).dropna()

telemetry_feat.head()
```

	machineID	datetime	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max_3h
7	1	2015-01-02 06:00:00	158.271400	403.235951	92.439132	32.516838	200.872430	495.777958
8	1	2015-01-02 09:00:00	160.528861	384.645962	86.944273	29.527665	197.363125	486.459056
9	1	2015-01-02 12:00:00	147.300678	412.965696	90.711354	34.203042	173.394523	439.579460
10	1	2015-01-02 15:00:00	152.420775	385.354924	99.506819	30.665184	185.205355	497.840620
11	1	2015-01-02 18:00:00	145.248486	424.542633	93.743827	37.422272	180.030715	495.376449

```
telemetry_feat.describe()
```

	machineID	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max_3h
count	291300.00000	291300.000000	291300.000000	291300.000000	291300.000000	291300.000000	291300.000000
mean	50.50000	158.071254	404.140202	92.378383	36.147281	183.471584	489.049230
std	28.86612	11.871003	40.811111	8.775150	4.206936	11.902916	40.782581
min	1.00000	97.333604	138.432075	51.237106	14.877054	134.886496	237.641009
25%	25.75000	150.363984	379.505260	86.862285	33.457488	175.270663	463.345990
50%	50.50000	158.162633	406.887414	92.115722	36.088251	182.767736	489.171998
75%	75.25000	165.835015	432.057742	97.273981	38.666539	190.890580	515.381821
max	100.00000	235.726785	565.962115	160.026994	68.001841	255.124717	695.020984

8 rows × 33 columns

Lag Features from Errors dataset

Like telemetry data, errors data set comes with timestamps. An important difference is that the error IDs are categorical values and should not be averaged over time intervals like the telemetry measurements. Instead, we count the number of errors of each type in a lagging window. We begin by reformatting the error data to have one entry per machine per time at which at least one error occurred:

```
# Create a column for each error type
error_count = pd.get_dummies(errors.set_index('datetime')).reset_index()
error_count.columns = ['datetime', 'machineID', 'error1', 'error2', 'error3', 'error4', 'error5']

# Combine errors for a given machine in a given hour
error_count = error_count.groupby(['machineID', 'datetime']).sum().reset_index()
error_count.head()
```

	machineID	datetime	error1	error2	error3	error4	error5
0	1	2015-01-03 07:00:00	1	0	0	0	0
1	1	2015-01-03 20:00:00	0	0	1	0	0
2	1	2015-01-04 06:00:00	0	0	0	0	1
3	1	2015-01-10 15:00:00	0	0	0	1	0
4	1	2015-01-22 10:00:00	0	0	0	1	0

Now, we will add feature 'datetime' and 'machineID' from telemetry data set and add blank entries for all other hourly timepoints (since no errors occurred at those times):

```
error_count = telemetry[['datetime', 'machineID']].merge(error_count, on=['machineID', 'datetime'],
                                                         how='left').fillna(0.0)
```

```
error_count.head()
```

	datetime	machineID	error1	error2	error3	error4	error5
0	2015-01-01 06:00:00	1	0.0	0.0	0.0	0.0	0.0
1	2015-01-01 07:00:00	1	0.0	0.0	0.0	0.0	0.0
2	2015-01-01 08:00:00	1	0.0	0.0	0.0	0.0	0.0
3	2015-01-01 09:00:00	1	0.0	0.0	0.0	0.0	0.0
4	2015-01-01 10:00:00	1	0.0	0.0	0.0	0.0	0.0

```
error_count.describe()
```

	machineID	error1	error2	error3	error4	error5
count	876100.000000	876100.000000	876100.000000	876100.000000	876100.000000	876100.000000
mean	50.500000	0.001153	0.001128	0.000957	0.000830	0.000406
std	28.866087	0.033934	0.033563	0.030913	0.028795	0.020154
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	25.750000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	50.500000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	75.250000	0.000000	0.000000	0.000000	0.000000	0.000000
max	100.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Finally, we can compute the total number of errors of each type over the last 24 hours, for timepoints taken every 3 hours:

```
temp = []
fields = ['error%d' % i for i in range(1,6)]
for col in fields:
    temp.append(pd.pivot_table(error_count,
                                           index='datetime',
                                           columns='machineID',
                                    values=col).rolling(window=24).sum().resample('3H',
                                              closed='left', label='right').first().unstack())
```

```
error_count = pd.concat(temp, axis=1)
error_count.columns = [i + 'count' for i in fields]
# error_count.reset_index(inplace=True)#To be activate
error_count = error_count.dropna()
error_count.head()
```

		error1count	error2count	error3count	error4count	error5count
machineID	**datetime**					
1	**2015-01-02 06:00:00**	0.0	0.0	0.0	0.0	0.0
	2015-01-02 09:00:00	0.0	0.0	0.0	0.0	0.0
	2015-01-02 12:00:00	0.0	0.0	0.0	0.0	0.0
	2015-01-02 15:00:00	0.0	0.0	0.0	0.0	0.0
	2015-01-02 18:00:00	0.0	0.0	0.0	0.0	0.0

```
error count.describe()
```

	error1count	error2count	error3count	error4count	error5count
count	291400.000000	291400.000000	291400.000000	291400.000000	291400.000000
mean	0.027649	0.027069	0.022907	0.019904	0.009753
std	0.166273	0.164429	0.151453	0.140820	0.098797
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000
max	2.000000	2.000000	2.000000	2.000000	2.000000

Count nos of Days Since Last Replacement of component from Maintenance data

A crucial data set in this example is the maintenance records which contain the information of component replacement records. Possible features from this data set can be, for example, the number of replacements of each component in the last 3 months to incorporate the frequency of replacements. However, more relevent information would be to calculate how long it has been since a component is last replaced as that would be expected to correlate better with component failures since the longer a component is used, the more degradation should be expected.

As a side note, creating lagging features from maintenance data is not as straightforward as for telemetry and errors, so the features from this data are generated in a more custom way. This type of ad-hoc feature engineering is very common in predictive maintenance. In the following step, the numbers of days since last component replacement are calculated for each component type as features from the maintenance data.

```
import numpy as np
from tqdm.notebook import tqdm_notebook

# Create a column for each error type
comp_rep = pd.get_dummies(maint.set_index('datetime')).reset_index()
comp_rep.columns = ['datetime', 'machineID', 'comp1', 'comp2', 'comp3', 'comp4']

# Combine repairs for a given machine in a given hour
comp_rep = comp_rep.groupby(['machineID', 'datetime']).sum().reset_index()

# Add timepoints where no components were replaced
comp_rep = telemetry[['datetime', 'machineID']].merge(comp_rep,
                                                      on=['datetime', 'machineID'],
                                      how='outer').fillna(0).sort_values(by=['machineID', 'datetime'])


components = ['comp1', 'comp2', 'comp3', 'comp4']
for comp in tqdm_notebook(components):
    # Convert indicator to most recent date of component change
    comp_rep.loc[comp_rep[comp] < 1, comp] = None
    comp_rep.loc[-comp_rep[comp].isnull(), comp] = comp_rep.loc[-comp_rep[comp].isnull(),'datetime']

    # Forward-fill the most-recent date of component change
    comp_rep[comp] = comp_rep[comp].fillna(method='ffill')

# Remove dates in 2014 (may have NaN or future component change dates)
comp_rep = comp_rep.loc[comp_rep['datetime'] > pd.to_datetime('2015-01-01')]

# Replace dates of most recent component change with days since most recent component change
for comp in tqdm_notebook(components):
    comp_rep[comp] = (comp_rep['datetime'] - comp_rep[comp]) / np.timedelta64(1, 'D')

comp_rep.describe()
```

100% 4/4 [00:02<00:00, 1.35it/s]

100% 4/4 [00:00<00:00, 29.35it/s]

	machineID	comp1	comp2	comp3	comp4
count	876100.000000	876100.000000	876100.000000	876100.000000	876100.000000
mean	50.500000	53.525185	51.540806	52.725962	53.834191
std	28.866087	62.491679	59.269254	58.873114	59.707978
min	1.000000	0.000000	0.000000	0.000000	0.000000
25%	25.750000	13.291667	12.125000	13.125000	13.000000

```
comp_rep.head()
```

	datetime	machineID	comp1	comp2	comp3	comp4
0	2015-01-01 06:00:00	1	19.000000	214.000000	154.000000	169.000000
1	2015-01-01 07:00:00	1	19.041667	214.041667	154.041667	169.041667
2	2015-01-01 08:00:00	1	19.083333	214.083333	154.083333	169.083333
3	2015-01-01 09:00:00	1	19.125000	214.125000	154.125000	169.125000
4	2015-01-01 10:00:00	1	19.166667	214.166667	154.166667	169.166667

Machine Features

The machine features can be used without further modification. These include descriptive information about the type of each machine and its age (number of years in service). If the age information had been recorded as a "first use date" for each machine, a transformation would have been necessary to turn those into a numeric values indicating the years in service.

Lastly, we merge all the feature data sets we created above to get the final feature matrix.

```
final_feat = telemetry_feat.merge(error_count, on=['datetime', 'machineID'], how='left')
final_feat = final_feat.merge(comp_rep, on=['datetime', 'machineID'], how='left')
final_feat = final_feat.merge(machines, on=['machineID'], how='left')

print(final_feat.head())
final_feat.describe()
```

```
   machineID            datetime  volt_min_3h  rotate_min_3h  pressure_min_3h  \
0          1 2015-01-02 06:00:00   158.271400     403.235951        92.439132
1          1 2015-01-02 09:00:00   160.528861     384.645962        86.944273
2          1 2015-01-02 12:00:00   147.300678     412.965696        90.711354
3          1 2015-01-02 15:00:00   152.420775     385.354924        99.506819
4          1 2015-01-02 18:00:00   145.248486     424.542633        93.743827

   vibration_min_3h  volt_max_3h  rotate_max_3h  pressure_max_3h  \
0         32.516838   200.872430     495.777958        96.535487
1         29.527665   197.363125     486.459056       114.342061
2         34.203042   173.394523     439.579460       110.408985
3         30.665184   185.205355     497.840620       105.993247
4         37.422272   180.030715     495.376449       111.950587

   vibration_max_3h  ...  error2count  error3count  error4count  error5count  \
0         52.355876  ...          0.0          0.0          0.0          0.0
1         42.992509  ...          0.0          0.0          0.0          0.0
2         37.117103  ...          0.0          0.0          0.0          0.0
3         47.862484  ...          0.0          0.0          0.0          0.0
4         43.099758  ...          0.0          0.0          0.0          0.0
```

Label Construction

When using multi-class classification for predicting failure due to a problem, labelling is done by taking a time window prior to the failure of an asset and labelling the feature records that fall into that window as "about to fail due to a problem" while labelling all other records as "normal". This time window should be picked according to the business case: in some situations it may be enough to predict failures hours in advance, while in others days or weeks may be needed to allow e.g. for arrival of replacement parts.

The prediction problem for this example scenario is to predict that a machine will fail in the near future due to a failure of a certain component or not. More specifically, **the goal is to predict that a machine will fail in the next 24 hours due to a certain component failure (component 1, 2, 3, or 4)** or not. Below, a categorical `failure` feature is created to serve as the label. All records within a 24 hour window before a failure of component 1 have `failure=comp1`, and so on for components 2, 3, and 4; all records not within 24 hours of a component failure have `failure=none`.

```
labeled_features = final_feat.merge(failures, on=['datetime', 'machineID'], how='left')
labeled_features = labeled_features.fillna(method='bfill', limit=7) # fill backward up to 24h
labeled_features['failure'] = labeled_features['failure'].astype('str')
labeled_features.replace({'nan': "none"}, inplace= True)
```

8 rows × 45 columns

```
labeled_features.head(2)
```

	machineID	datetime	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max_3h	p
0	1	2015-01-02 06:00:00	158.271400	403.235951	92.439132	32.516838	200.872430	495.777958	
1	1	2015-01-02 09:00:00	160.528861	384.645962	86.944273	29.527665	197.363125	486.459056	

2 rows × 46 columns

Below is an example of records that are labeled as failure=comp4 in the failure column. Notice that the first 8 records all occur in the 24-hour window before the first recorded failure of component 4. The next 8 records are within the 24 hour window before another failure of component 4.

```
labeled_features.loc[labeled_features['failure'] == 'comp4'][:16]
```

	machineID	datetime	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max
17	1	2015-01-04 09:00:00	142.666469	433.279499	97.709630	48.238941	191.168936	479.615
18	1	2015-01-04 12:00:00	153.143558	438.091311	94.524894	51.647981	215.656488	458.097
19	1	2015-01-04 15:00:00	129.016707	421.728389	91.675576	45.951349	173.525320	479.457
20	1	2015-01-04 18:00:00	168.503141	365.213804	82.400818	43.917862	184.640476	517.348
21	1	2015-01-04 21:00:00	183.684832	414.481164	103.159963	41.674887	199.755983	427.763
22	1	2015-01-05 00:00:00	162.368945	447.101400	89.260131	50.240045	180.562703	471.194
23	1	2015-01-05 03:00:00	127.163620	376.719605	89.969588	46.845600	161.928938	430.475
24	1	2015-01-05 06:00:00	177.317220	387.005318	94.686208	45.202347	202.520488	469.787
1337	1	2015-06-18 09:00:00	142.165191	417.834555	96.780895	51.105583	198.380679	500.852
1338	1	2015-06-18 12:00:00	184.681384	387.342414	91.050336	40.747029	197.240367	474.031
1339	1	2015-06-18 15:00:00	143.320854	402.864601	86.351078	39.927737	178.305492	463.301
1340	1	2015-06-18 18:00:00	176.531054	408.749781	107.166360	47.609185	180.957236	442.629
1341	1	2015-06-18 21:00:00	142.194697	437.599207	96.911121	39.065462	169.116734	590.323
1342	1	2015-06-19 00:00:00	147.914394	432.857174	94.930887	47.202762	171.499274	491.996
		2015-06-						

```
# #·#Save·pre-processed·data·in·CSV.
#·labeled_features.to_csv('/content/drive/MyDrive/AAIC/Case_study_1/preprocessed_2.csv',·encoding='utf-8',·index=False)
```

Modeling

After the feature engineering and labeling steps, below, modeling process is being described.

Training, Validation and Testing

When working with time-stamped data as in this example, record partitioning into training, validation, and test sets should be performed carefully to prevent overestimating the performance of the models. In predictive maintenance, the features are usually

generated using lagging aggregates: records in the same time window will likely have identical labels and similar feature values. These correlations can give a model an "unfair advantage" when predicting on a test set record that shares its time window with a training set record. We therefore partition records into training, validation, and test sets in large chunks, to minimize the number of time intervals shared between them.

Predictive models have no advance knowledge of future chronological trends: in practice, such trends are likely to exist and to adversely impact the model's performance. To obtain an accurate assessment of a predictive model's performance, it is recommended to train on older records and validating/testing using newer records.

For both of these reasons, a time-dependent record splitting strategy is an excellent choice for predictive maintenance models. The featured dataset has been splitted into three dataset "Training", "Cross Validation and "Test" dataset with time.

All the featured dataset has been saved in "preprocessed.csv" for ease of analysis.

```
# #Importing the "preprocessed.csv".
import pandas as pd
labeled_features= pd.read_csv("/content/drive/MyDrive/AAIC/Case_study_1/preprocessed_2.csv")
# Format datetime field which comes in as string

labeled_features['datetime'] = pd.to_datetime(labeled_features['datetime'], format="%Y-%m-%d %H:%M:%S")
```

```
labeled_features.head(2)
```

	machineID	datetime	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max_3h	p
0	1	2015-01-02 06:00:00	158.271400	403.235951	92.439132	32.516838	200.872430	495.777958	
1	1	2015-01-02 09:00:00	160.528861	384.645962	86.944273	29.527665	197.363125	486.459056	

2 rows × 46 columns

```
labeled_features.tail(2)
```

	machineID	datetime	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max_
291339	100	2016-01-01 03:00:00	162.742669	395.222827	101.589735	44.382754	179.438162	481.253
291340	100	2016-01-01 06:00:00	165.475310	413.771670	94.132837	35.123072	192.483414	447.816

2 rows × 46 columns

```
labeled_features.describe()
```

	machineID	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max_3h
count	291341.000000	291341.000000	291341.000000	291341.000000	291341.000000	291341.000000	291341.000000
mean	50.499243	158.072427	404.132559	92.380133	36.147905	183.473297	489.041432
std	28.866522	11.872268	40.817688	8.777961	4.207675	11.904876	40.789312
min	1.000000	97.333604	138.432075	51.237106	14.877054	134.886496	237.641009
25%	25.000000	150.364806	379.495414	86.862934	33.457547	175.271243	463.339007
50%	50.000000	158.163452	406.881373	92.116192	36.088485	182.768621	489.167126

```
#https://towardsdatascience.com/time-based-cross-validation-d259b13d42b8
#https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/
import numpy as np
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
```

Split the "preprocessed.csv" with Sklearn "train_test_split" function with "shuffle=False".

```
X = labeled_features.drop(['datetime', 'machineID', 'failure'], 1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: In a future version of pandas all arg
  """Entry point for launching an IPython kernel.
```

```
X.head(2)
```

	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max_3h	pressure_max_3h	vibra
0	158.271400	403.235951	92.439132	32.516838	200.872430	495.777958	96.535487	
1	160.528861	384.645962	86.944273	29.527665	197.363125	486.459056	114.342061	

2 rows × 43 columns

```
X_final = pd.get_dummies(X)
X_final.head(2)
```

	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max_3h	pressure_max_3h	vibra
0	158.271400	403.235951	92.439132	32.516838	200.872430	495.777958	96.535487	
1	160.528861	384.645962	86.944273	29.527665	197.363125	486.459056	114.342061	

2 rows × 46 columns

```
X_final.describe()
```

	volt_min_3h	rotate_min_3h	pressure_min_3h	vibration_min_3h	volt_max_3h	rotate_max_3h	pressure_max_3
count	291341.000000	291341.000000	291341.000000	291341.000000	291341.000000	291341.000000	291341.00000
mean	158.072427	404.132559	92.380133	36.147905	183.473297	489.041432	109.34874

```
X_final_train = X_final.values
X_final_train[1]
```

```
array([160.52886052, 384.64596164,  86.94427269,  29.52766452,
       197.36312454, 486.45905612, 114.34206081,  42.99250944,
       176.36429322, 439.34965502, 101.55320892,  36.10558003,
        18.95221004,  51.32963577,  13.78927949,   6.73773919,
       151.33568223, 346.14933504,  75.23790486,  25.990511  ,
       200.87242982, 527.34982545, 114.34206081,  52.35587614,
       170.61486188, 446.36485915,  96.84978485,  39.73682577,
        12.51940225,  48.38507588,  10.17153979,   6.16323082,
         0.        ,   0.        ,   0.        ,   0.        ,
         0.        ,  20.125     , 215.125     , 155.125     ,
       170.125     ,  18.        ,   0.        ,   0.        ,
         1.        ,   0.        ])
```

```
y_final=labeled_features['failure']
y_final.head(2)
```

```
0    none
1    none
Name: failure, dtype: object
```

```
y_final_train = y_final.values
y_final_train[1]
```

```
'none'
```

```
X_train, X_test, y_train, y_test = train_test_split(X_final_train, y_final_train,test_size=0.20, shuffle=False)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.20, shuffle=False)
```

```
print('X_train Observations: %d' % (len(X_train)))
print('y_train Observations: %d' % (len(y_train)))

print('X_cv Observations: %d' % (len(X_cv)))
print('y_cv Observations: %d' % (len(y_cv)))

print('X_test Observations: %d' % (len(X_test)))
print('y_test Observations: %d' % (len(y_test)))
```

```
X_train Observations: 186457
y_train Observations: 186457
X_cv Observations: 46615
y_cv Observations: 46615
X_test Observations: 58269
y_test Observations: 58269
```

```
#Reference: AAIC Case_study_2.
# This function plots the confusion matrices given y_i, y_i_hat.
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    A =(((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))

    labels = ['comp1', 'comp2', 'comp3', 'comp4', 'none']
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
```

```
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

Apply model XGBClassifier

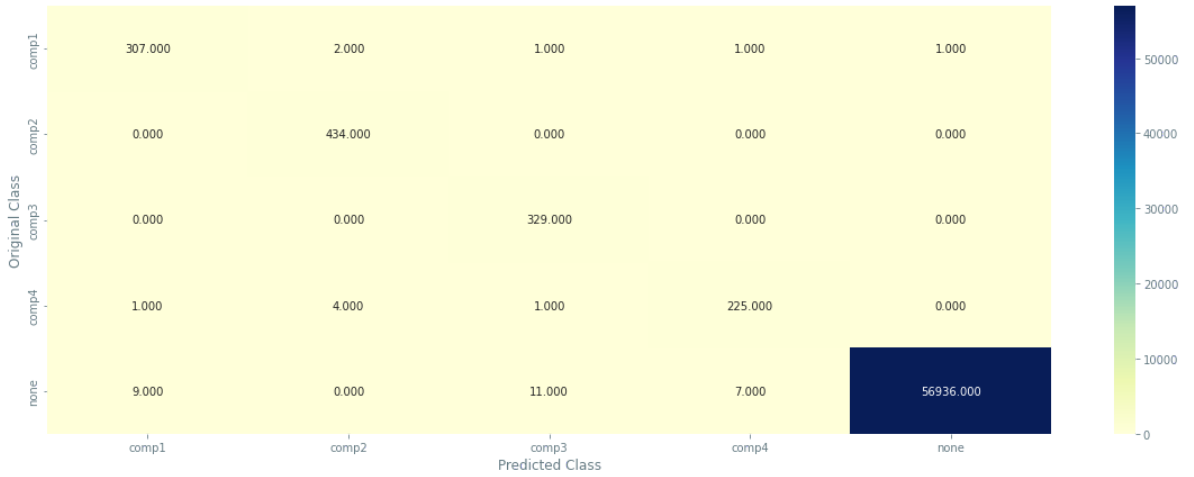
```
from xgboost import XGBClassifier

x_cfl=XGBClassifier()
x_cfl.fit(X_train,y_train)
```

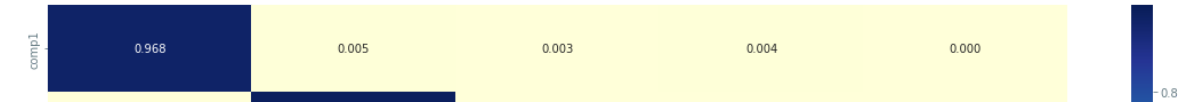
```
     XGBClassifier(objective='multi:softprob')
```

```
plot_confusion_matrix(y_test, x_cfl.predict(X_test))
```

```
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
from prettytable import PrettyTable

# Specify the Column Names
myTable = PrettyTable(["Model name (Recall score )", "comp1", "comp2", "comp3", 'comp4', 'none(no fail)'])

# Add rows

myTable.add_row(["Xgboost with available 19 nos of features",
                 "0.206", "0.366", "0.639", "0.556", "1"])

myTable.add_row(["Xgboost with new 22 nos of features",
                 "0.818", "0.982", "0.941", "0.903", "1"])

myTable.add_row(["Xgboost with new 46 nos of features",
                 "0.984", "1", "1", "0.974", "1"])

print(myTable)
```

```
+-----------------------------------------+-------+-------+-------+-------+---------------+
|         Model name (Recall score )      | comp1 | comp2 | comp3 | comp4 | none(no fail) |
+-----------------------------------------+-------+-------+-------+-------+---------------+
| Xgboost with available 19 nos of features | 0.206 | 0.366 | 0.639 | 0.556 |       1       |
|    Xgboost with new 22 nos of features    | 0.818 | 0.982 | 0.941 | 0.903 |       1       |
|    Xgboost with new 46 nos of features    | 0.984 |   1   |   1   | 0.974 |       1       |
+-----------------------------------------+-------+-------+-------+-------+---------------+
```

Observation on the above approach:

From the above table, it is observed recall value of all the failure components has increased a lot after creating new features (without performing hyper-parameter tuning). This predictive model can predict the failure due to comp2 and comp3 perfectly. However, this model can predict around 98% correctly for comp2 and comp3.

Next step:

1. Calibration method will be computed.
2. Hyperparameter tuning will be computed.
3. Other Machine Learning Model will be trained.

>>>>>>>> End of "FE_Case_study_1_part_2" <<<<<<<<