(https://www.i2a2.academy/pt)

Regras do jogo

Dataset

- https://www.kaggle.com/datasets/nphantawee/pump-sensor-data

Atividades

- Fazer a carga em um notebook (Google Colab ou Jupyter local)
- Realizar a análise dos dados (EDA)
- Investigar o código e discussões disponibilizados na página do dataset
- Montar um relatório com suas conclusões sobre o dataset (seja detalhista)
- Entregar o código fonte gerado e o relatório. Enviar e mail para XXXXX@i2a2.academy com o título “I2A2 ABDI Desafio 2”.
- Data de entrega: 27/02/2023 às 23h59 BRT.

Opcionais:

- Ler o artigo " Anomaly Detection in Time Series Sensor Data" https://towardsdatascience.com/anomaly-detection-in-time-series-sensor-data-86fd52e62538
- Refatorar o código utilizando algum LLM Large Language Model (ex. ChatGPT).

O problema

Tenho um amigo que trabalha em uma pequena equipe que cuida da bomba d'água de uma pequena área longe da cidade grande, houve 7 falhas no sistema no ano passado. Essas falhas causam grandes problemas para muitas pessoas e também levam a alguns sérios problemas de vida de algumas famílias. A equipe não consegue ver nenhum padrão nos dados quando o sistema cai, então eles não sabem onde colocar mais atenção.

Como acredito no uso de dados para resolver problemas, peço a ele que forneça os dados do sensor disponíveis e espero que alguém aqui possa ajudar.

Insights

- Em quais datas o sistema esteve um falha?
- A falha foi total?
- Porque nenhum padrão é vísivel nos dados?
- Em quais sensores devem-se por maior atenção?

Carregamento de bibliotecas e conjunto de dados

In [1]:

```
# Carregamento de bibliotecas
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import os


# Utilizar códigos abaixo somente se necessário para ignorar avisos 'warnings'
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
# Carrega os dados do arquivo CSV no dataframe: df
df = pd.read_csv(os.getcwd() + '\\sensor.csv', index_col=0)
```

Exploratory Analysis Data - EDA

Sistema de bombeamento de água

In [3]:

```
# Visualizar os dados carregados
df
```

Out[3]:

	timestamp	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sensor_05	se
0	2018-04-01 00:00:00	2.465394	47.09201	53.211800	46.310760	634.375000	76.45975	1
1	2018-04-01 00:01:00	2.465394	47.09201	53.211800	46.310760	634.375000	76.45975	1
2	2018-04-01 00:02:00	2.444734	47.35243	53.211800	46.397570	638.888900	73.54598	1
3	2018-04-01 00:03:00	2.460474	47.09201	53.168400	46.397568	628.125000	76.98898	1
4	2018-04-01 00:04:00	2.445718	47.13541	53.211800	46.397568	636.458300	76.58897	1
...	...	...	...	...	...	...	...	
220315	2018-08-31 23:55:00	2.407350	47.69965	50.520830	43.142361	634.722229	64.59095	1
220316	2018-08-31 23:56:00	2.400463	47.69965	50.564240	43.142361	630.902771	65.83363	1
220317	2018-08-31 23:57:00	2.396528	47.69965	50.520830	43.142361	625.925903	67.29445	1
220318	2018-08-31 23:58:00	2.406366	47.69965	50.520832	43.142361	635.648100	65.09175	1
220319	2018-08-31 23:59:00	2.396528	47.69965	50.520832	43.142361	639.814800	65.45634	1

220320 rows × 54 columns

Insight - A leitura do dataframe(df) os sensores possuem característica de mensuração de **RPM e Volume**.

In [4]:

```
# Analise de valores estatísticos padrão
df.describe().T.style.background_gradient(cmap='YlOrRd')
```

Out[4]:

	count	mean	std	min	25%	50%	
sensor_00	210112.000000	2.372221	0.412227	0.000000	2.438831	2.456539	2.4
sensor_01	219951.000000	47.591611	3.296666	0.000000	46.310760	48.133678	49.4
sensor_02	220301.000000	50.867392	3.666820	33.159720	50.390620	51.649300	52.7
sensor_03	220301.000000	43.752481	2.418887	31.640620	42.838539	44.227428	45.3
sensor_04	220301.000000	590.673936	144.023912	2.798032	626.620400	632.638916	637.6
sensor_05	220301.000000	73.396414	17.298247	0.000000	69.976260	75.576790	80.9
sensor_06	215522.000000	13.501537	2.163736	0.014468	13.346350	13.642940	14.5
sensor_07	214869.000000	15.843152	2.201155	0.000000	15.907120	16.167530	16.4
sensor_08	215213.000000	15.200721	2.037390	0.028935	15.183740	15.494790	15.6
sensor_09	215725.000000	14.799210	2.091963	0.000000	15.053530	15.082470	15.1
sensor_10	220301.000000	41.470339	12.093519	0.000000	40.705260	44.291340	47.4
sensor_11	220301.000000	41.918319	13.056425	0.000000	38.856420	45.363140	49.6
sensor_12	220301.000000	29.136975	10.113935	0.000000	28.686810	32.515830	34.9
sensor_13	220301.000000	7.078858	6.901755	0.000000	1.538516	2.929809	12.8
sensor_14	220299.000000	376.860041	113.206382	32.409550	418.103250	420.106200	420.9
sensor_15	0.000000	nan	nan	nan	nan	nan	
sensor_16	220289.000000	416.472892	126.072642	0.000000	459.453400	462.856100	464.3
sensor_17	220274.000000	421.127517	129.156175	0.000000	454.138825	462.020250	466.8
sensor_18	220274.000000	2.303785	0.765883	0.000000	2.447542	2.533704	2.5
sensor_19	220304.000000	590.829775	199.345820	0.000000	662.768975	665.672400	667.1
sensor_20	220304.000000	360.805165	101.974118	0.000000	398.021500	399.367000	400.0
sensor_21	220304.000000	796.225942	226.679317	95.527660	875.464400	879.697600	882.1
sensor_22	220279.000000	459.792815	154.528337	0.000000	478.962600	531.855900	534.2
sensor_23	220304.000000	922.609264	291.835280	0.000000	950.922400	981.925000	1090.8
sensor_24	220304.000000	556.235397	182.297979	0.000000	601.151050	625.873500	628.6
sensor_25	220284.000000	649.144799	220.865166	0.000000	693.957800	740.203500	750.3
sensor_26	220300.000000	786.411781	246.663608	43.154790	790.489575	861.869600	919.1
sensor_27	220304.000000	501.506589	169.823173	0.000000	448.297950	494.468450	536.2
sensor_28	220304.000000	851.690339	313.074032	4.319347	782.682625	967.279850	1043.9
sensor_29	220248.000000	576.195305	225.764091	0.636574	518.947225	564.872500	744.0
sensor_30	220059.000000	614.596442	195.726872	0.000000	627.777800	668.981400	697.2
sensor_31	220304.000000	863.323100	283.544760	23.958330	839.062400	917.708300	981.2
sensor_32	220252.000000	804.283915	260.602361	0.240716	760.607475	878.850750	943.8
sensor_33	220304.000000	486.405980	150.751836	6.460602	489.761075	512.271750	555.1
sensor_34	220304.000000	234.971776	88.376065	54.882370	172.486300	226.356050	316.8
sensor_35	220304.000000	427.129817	141.772519	0.000000	353.176625	473.349350	528.8
sensor_36	220304.000000	593.033876	289.385511	2.260970	288.547575	709.668050	837.3

	count	mean	std	min	25%	50%	
sensor_37	220304.000000	60.787360	37.604883	0.000000	28.799220	64.295485	90.8
sensor_38	220293.000000	49.655946	10.540397	24.479166	45.572910	49.479160	53.6
sensor_39	220293.000000	36.610444	15.613723	19.270830	32.552080	35.416660	39.0
sensor_40	220293.000000	68.844530	21.371139	23.437500	57.812500	66.406250	77.8
sensor_41	220293.000000	35.365126	7.898665	20.833330	32.552080	34.895832	37.7
sensor_42	220293.000000	35.453455	10.259521	22.135416	32.812500	35.156250	36.9
sensor_43	220293.000000	43.879591	11.044404	24.479166	39.583330	42.968750	46.6
sensor_44	220293.000000	42.656877	11.576355	25.752316	36.747684	40.509260	45.1
sensor_45	220293.000000	43.094984	12.837520	26.331018	36.747684	40.219910	44.8
sensor_46	220293.000000	48.018585	15.641284	26.331018	40.509258	44.849540	51.2
sensor_47	220293.000000	44.340903	10.442437	27.199070	39.062500	42.534720	46.5
sensor_48	220293.000000	150.889044	82.244957	26.331018	83.912030	138.020800	208.3
sensor_49	220293.000000	57.119968	19.143598	26.620370	47.743060	52.662040	60.7
sensor_50	143303.000000	183.049260	65.258650	27.488426	167.534700	193.865700	219.9
sensor_51	204937.000000	202.699667	109.588607	27.777779	179.108800	197.338000	216.7

Atenção - A análise do objeto (df_mean_ascedenting) pode ser observado defeito no sensor:

Observe - O gráfico utliza os dados sem EDA - Análise Exploratória de dados.

In [5]:

```
# Amostragem gráfica de linhas do comportamento de máquinas por sensores
df.plot.line(figsize=(25,10)).set_title('Processo por todos os sensores', fontsize=16)
```

Out[5]:

```
Text(0.5, 1.0, 'Processo por todos os sensores')
```

IMPRESSÃO - Sistema gráfico difuso do dataset original.

In [6]:

```
# Carrega a média do dataset original
df_mean = df.mean()
```

In [7]:

```
# Carrega o máximo do dataset original
df_max_compare = df.max(numeric_only=True)
```

In [8]:

```
# Carrega a mínimo do dataset original
df_min_compare = df.min(numeric_only=True)
```

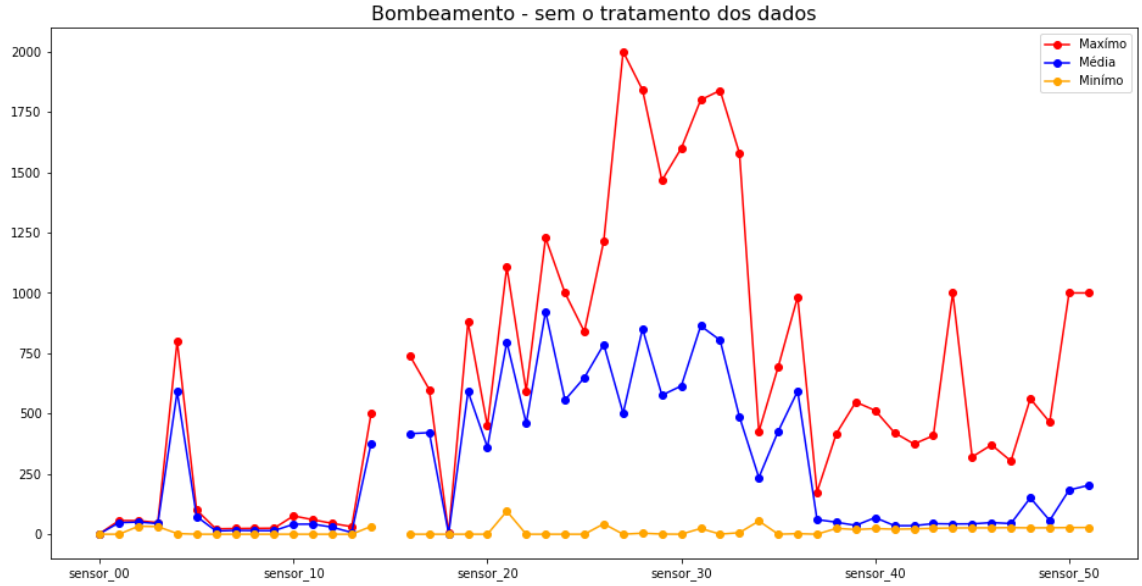
In [9]:

```
# Gráfico do dataset sem tratamento.
df_max_compare.plot.line(figsize=(16,8), marker='o', color='red', label='Maxímo').set_ti
df_mean.plot.line(marker='o', color='blue', label='Média')
df_min_compare.plot.line(marker='o', color='orange', label='Minímo')

plt.legend()
```

Out[9]:

```
<matplotlib.legend.Legend at 0x1e280dad6d0>
```



IMPRESSÃO - Os sensores mesuram rotação (RPM - Rotação por minuto)

Insight - Modelo de gráfico para o funcionamento de uma sistema de bombeamento de água

Resultado - Apos a EDA houve necessidade de reorganizar as TAGs dos sensores.

In [10]:

```
# Reorganizar as TAGs dos sensores
df_new = df.rename(columns={'sensor_18': 'S00', 'sensor_00': 'S01', 'sensor_13':'S02', '
                            'sensor_08':'S05','sensor_07':'S06', 'sensor_12':'S07','sens
                            'sensor_39':'S10','sensor_10':'S11','sensor_11':'S12','senso
                            'sensor_03':'S15','sensor_43':'S16','sensor_47':'S17','senso
                            'sensor_38':'S20','sensor_02':'S21','sensor_49':'S22','senso
                            'sensor_05':'S25','sensor_48':'S26','sensor_50':'S27','senso
                            'sensor_20':'S30','sensor_14':'S31','sensor_16':'S32','senso
                            'sensor_22':'S35','sensor_33':'S36','sensor_27':'S37','senso
                            'sensor_04':'S40','sensor_19':'S41','sensor_36':'S42','senso
                            'sensor_26':'S45','sensor_21':'S46','sensor_32':'S47','senso
                            'sensor_23':'S50','sensor_15':'S51'})
```

In [11]:

```
df_new_tag = df_new.reindex(sorted(df_new.columns), axis=1)
```

In [12]:

```
df_new_tag
```

Out[12]:

	S00	S01	S02	S03	S04	S05	S06	S07	
0	2.565284	2.465394	1.681353	13.41146	15.05353	15.56713	16.13136	31.11716	30.
1	2.565284	2.465394	1.681353	13.41146	15.05353	15.56713	16.13136	31.11716	30.
2	2.500062	2.444734	1.708474	13.32465	15.01013	15.61777	16.03733	32.08894	30.
3	2.509521	2.460474	1.579427	13.31742	15.08247	15.69734	16.24711	31.67221	30.
4	2.604785	2.445718	1.683831	13.35359	15.08247	15.69734	16.21094	31.95202	30.
...	...	...	...	...	...	...	...	...	
220315	2.499117	2.407350	13.265320	15.11863	15.16204	15.65393	16.65220	38.05424	30.
220316	2.618476	2.400463	13.242270	15.15480	15.11863	15.65393	16.70284	38.53485	30.
220317	2.620500	2.396528	13.188660	15.08970	15.11863	15.69734	16.70284	38.52678	29.
220318	2.514596	2.406366	13.173460	15.11863	15.11863	15.74074	16.56539	38.89159	29.
220319	2.487299	2.396528	13.125930	15.11863	15.01013	15.65393	16.65220	39.40957	29.

220320 rows × 54 columns

In [13]:

```
# Analise de valores estatísticos padrão: média
df_mean_new_tag = df_new_tag.mean()
```

In [14]:

```
df_mean_new_tag
```

Out[14]:

```
S00      2.303785
S01      2.372221
S02      7.078858
S03     13.501537
S04     14.799210
S05     15.200721
S06     15.843152
S07     29.136975
S08     35.365126
S09     35.453455
S10     36.610444
S11     41.470339
S12     41.918319
S13     42.656877
S14     43.094984
S15     43.752481
S16     43.879591
S17     44.340903
S18     47.591611
S19     48.018585
S20     49.655946
S21     50.867392
S22     57.119968
S23     60.787360
S24     68.844530
S25     73.396414
S26    150.889044
S27    183.049260
S28    202.699667
S29    234.971776
S30    360.805165
S31    376.860041
S32    416.472892
S33    421.127517
S34    427.129817
S35    459.792815
S36    486.405980
S37    501.506589
S38    556.235397
S39    576.195305
S40    590.673936
S41    590.829775
S42    593.033876
S43    614.596442
S44    649.144799
S45    786.411781
S46    796.225942
S47    804.283915
S48    851.690339
S49    863.323100
S50    922.609264
S51           NaN
dtype: float64
```

Atenção - Os sensores S00, S01 e S02, por seus valores, indicam falha, pois estás rotações não são suficientes para iniciar o bombeamento como também os sensores de S03 a S25.

In [15]:

```
# Default sort ascedenting
df_mean_ascedenting= df_mean.sort_values()
```

In [16]:

```
df_mean_ascedenting
```

Out[16]:

```
sensor_18      2.303785
sensor_00      2.372221
sensor_13      7.078858
sensor_06     13.501537
sensor_09     14.799210
sensor_08     15.200721
sensor_07     15.843152
sensor_12     29.136975
sensor_41     35.365126
sensor_42     35.453455
sensor_39     36.610444
sensor_10     41.470339
sensor_11     41.918319
sensor_44     42.656877
sensor_45     43.094984
sensor_03     43.752481
sensor_43     43.879591
sensor_47     44.340903
sensor_01     47.591611
sensor_46     48.018585
sensor_38     49.655946
sensor_02     50.867392
sensor_49     57.119968
sensor_37     60.787360
sensor_40     68.844530
sensor_05     73.396414
sensor_48    150.889044
sensor_50    183.049260
sensor_51    202.699667
sensor_34    234.971776
sensor_20    360.805165
sensor_14    376.860041
sensor_16    416.472892
sensor_17    421.127517
sensor_35    427.129817
sensor_22    459.792815
sensor_33    486.405980
sensor_27    501.506589
sensor_24    556.235397
sensor_29    576.195305
sensor_04    590.673936
sensor_19    590.829775
sensor_36    593.033876
sensor_30    614.596442
sensor_25    649.144799
sensor_26    786.411781
sensor_21    796.225942
sensor_32    804.283915
sensor_28    851.690339
sensor_31    863.323100
sensor_23    922.609264
sensor_15           NaN
dtype: float64
```

In [17]:

```
# Analise de valores estatísticos padrão: maxímo
df_max = df_new_tag.max(numeric_only=True)
```

In [18]:

```
df_max
```

Out[18]:

```
S00       4.873250
S01       2.549016
S02      31.187550
S03      22.251160
S04      25.000000
S05      24.348960
S06      23.596640
S07      45.000000
S08     420.312500
S09     374.218800
S10     547.916600
S11      76.106860
S12      60.000000
S13    1000.000000
S14     320.312500
S15      48.220490
S16     408.593700
S17     303.530100
S18      56.727430
S19     370.370400
S20     417.708300
S21      56.032990
S22     464.409700
S23     174.901200
S24     512.760400
S25      99.999880
S26     561.632000
S27    1000.000000
S28    1000.000000
S29     425.549800
S30     448.907900
S31     500.000000
S32     739.741500
S33     599.999939
S34     694.479126
S35     594.061100
S36    1578.600000
S37    2000.000000
S38    1000.000000
S39    1466.281000
S40     800.000000
S41     878.917900
S42     984.060700
S43    1600.000000
S44     839.575000
S45    1214.420000
S46    1107.526000
S47    1839.211000
S48    1841.146000
S49    1800.000000
S50    1227.564000
S51            NaN
dtype: float64
```

In [19]:

```
# Análise de valores estatísticos padrão: mínimo
df_min = df_new_tag.min(numeric_only=True)
```

In [20]:

```
df_min
```

Out[20]:

```
S00     0.000000
S01     0.000000
S02     0.000000
S03     0.014468
S04     0.000000
S05     0.028935
S06     0.000000
S07     0.000000
S08    20.833330
S09    22.135416
S10    19.270830
S11     0.000000
S12     0.000000
S13    25.752316
S14    26.331018
S15    31.640620
S16    24.479166
S17    27.199070
S18     0.000000
S19    26.331018
S20    24.479166
S21    33.159720
S22    26.620370
S23     0.000000
S24    23.437500
S25     0.000000
S26    26.331018
S27    27.488426
S28    27.777779
S29    54.882370
S30     0.000000
S31    32.409550
S32     0.000000
S33     0.000000
S34     0.000000
S35     0.000000
S36     6.460602
S37     0.000000
S38     0.000000
S39     0.636574
S40     2.798032
S41     0.000000
S42     2.260970
S43     0.000000
S44     0.000000
S45    43.154790
S46    95.527660
S47     0.240716
S48     4.319347
S49    23.958330
S50     0.000000
S51          NaN
dtype: float64
```

In [21]:

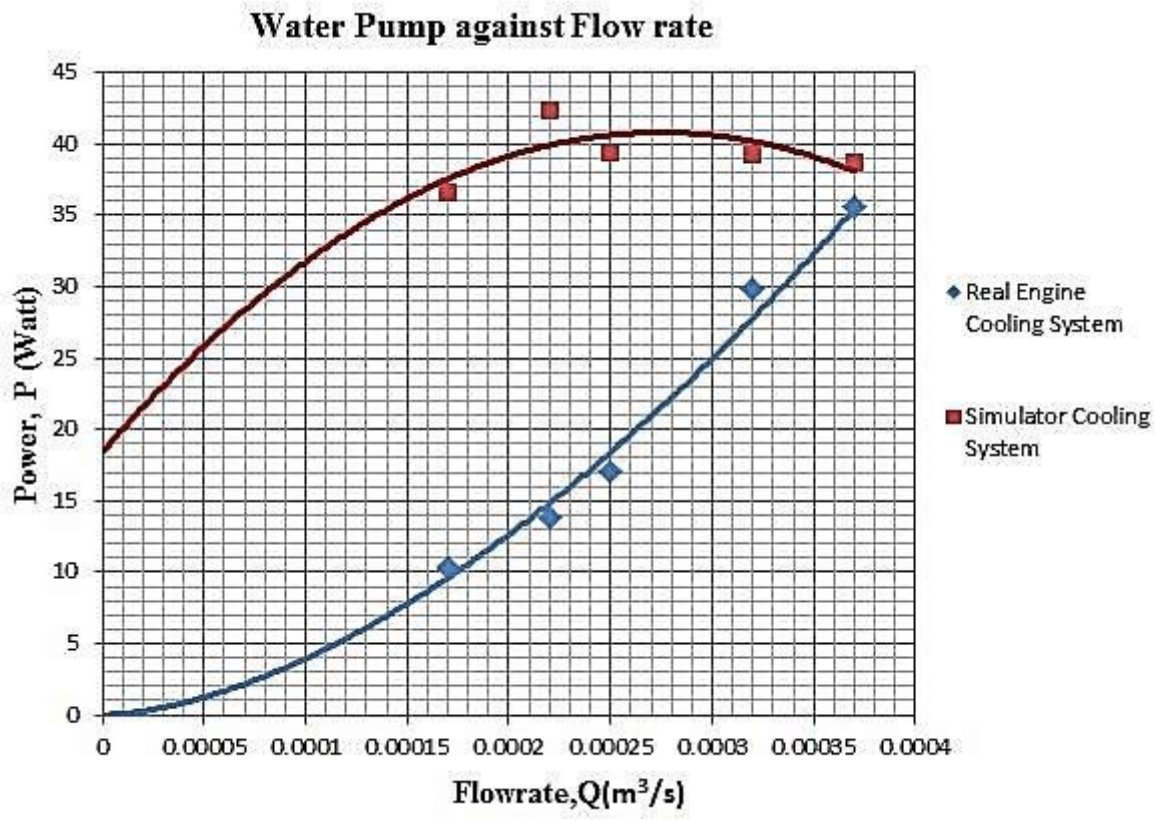
```
# Valor médio acrescido em 20% uma excelente taxa de sobrecarga.
df_mean_high = df_mean_ascedenting*1.2
```

In [22]:

```
# Valor médio com reução de 20% uma oscilação aceitavél.
df_mean_low = df_mean_ascedenting*0.8
```

Atenção - Os nomes dos sensores não seguem a ordem de funcionamento do sistema de bombeamento de água.

Gráfico exemplifica o funcionamento de uma bomba de água.



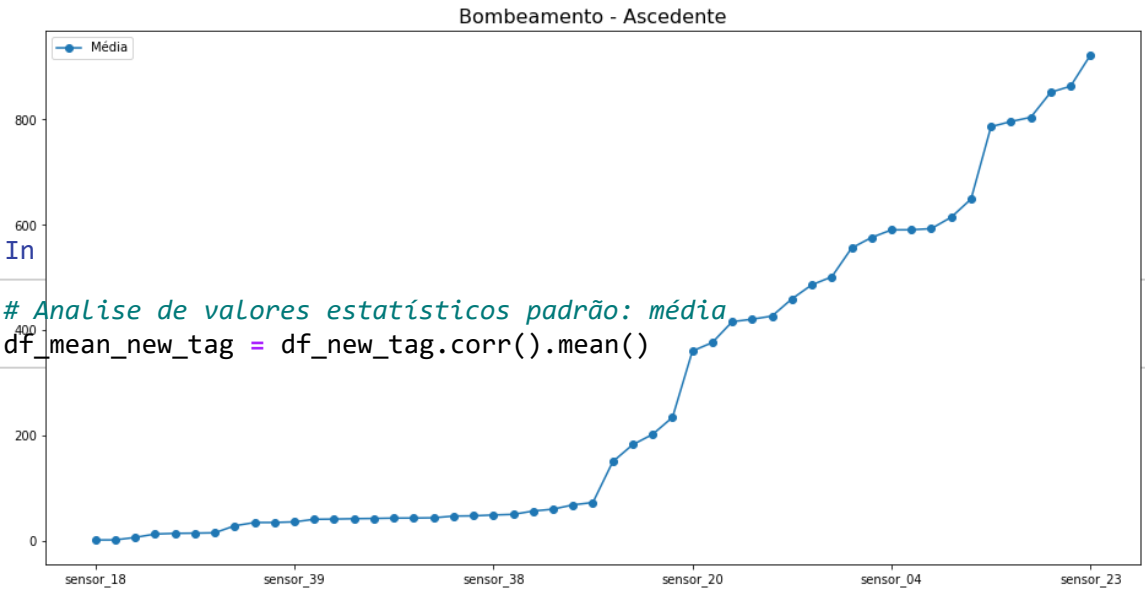
[(https://www.researchgate.net/figure/Graph-of-power-of-water-pump-against-flow-rate-for-the-real-engine-and-simulator-cooling_fig7_309439547)](https://www.researchgate.net/figure/Graph-of-power-of-water-pump-against-flow-rate-for-the-real-engine-and-simulator-cooling_fig7_309439547)

In [23]:

```
# Comportamento esperado do sistema de bombeamento de água - de lugar baixo para alto
df_mean_ascedenting.plot.line(figsize=(16,8), marker='o', label='Média').set_title('Bomb
plt.legend()
```

Out[23]:

```
<matplotlib.legend.Legend at 0x1e283b4a280>
```



In

```
# Analise de valores estatísticos padrão: média
df_mean_new_tag = df_new_tag.corr().mean()
```

In [25]:

```
df_mean_new_tag
```

Out[25]:

```
S00    0.356792
S01    0.197854
S02    0.206399
S03    0.259175
S04    0.200614
S05    0.206548
S06    0.216644
S07    0.212476
S08    0.105941
S09    0.089941
S10    0.087508
S11    0.246560
S12    0.241984
S13    0.176710
S14    0.151886
S15    0.191047
S16    0.137118
S17    0.164254
S18    0.217112
S19    0.165226
S20    0.152187
S21    0.192684
S22    0.187204
S23   -0.118868
S24    0.205414
S25    0.105771
S26    0.248378
S27    0.208832
S28    0.050874
S29    0.273188
S30    0.372886
S31    0.364419
S32    0.368059
S33    0.360259
S34    0.307306
S35    0.375650
S36    0.331085
S37    0.198864
S38    0.366508
S39    0.258495
S40    0.229756
S41    0.372495
S42    0.274812
S43    0.334485
S44    0.372199
S45    0.369639
S46    0.374855
S47    0.322261
S48    0.262272
S49    0.298862
S50    0.377445
S51         NaN
dtype: float64
```

In [26]:

```
df_corr = df.corr().mean()
```

In [27]:

```
df_corr_sorted = df_corr.sort_values()
```

In [28]:

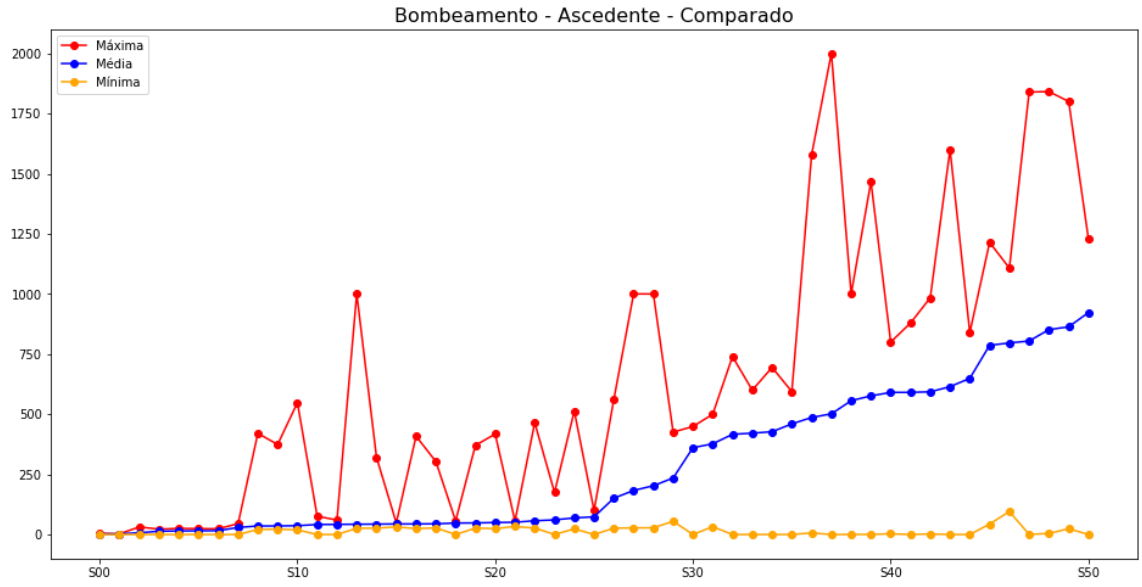
```
# Comportamento esperado do sistema de bombeamento de água - de lugar baixo para alto
df_max.plot.line(figsize=(16,8), marker='o', color='red', label='Máxima').set_title('Bom
df_mean_ascedenting.plot.line( marker='o', label='Média', color='blue')
df_min.plot.line(marker='o', color='orange', label='Mínima')

#df_mean_high.plot.line(marker='o', color='black', label='Média + 20%')
#df_mean_low.plot.line(marker='o', color='yellow', label='Média = 80%')

plt.legend()
```

Out[28]:

```
<matplotlib.legend.Legend at 0x1e2b84d1130>
```



IMPRESSÃO - Os 30 primeiros sensores (S00 a S29) indicam que o sistema de bombeamento é inoperante ou todos estes sensores estão em falha.

IMPRESSÃO - Os picos vermelhos indicam cavitação, no entanto, que o sistema suporta está rotação.

Insight - A leitura do dataframe(df) os sensores possuem característica de mensuração em correlação.

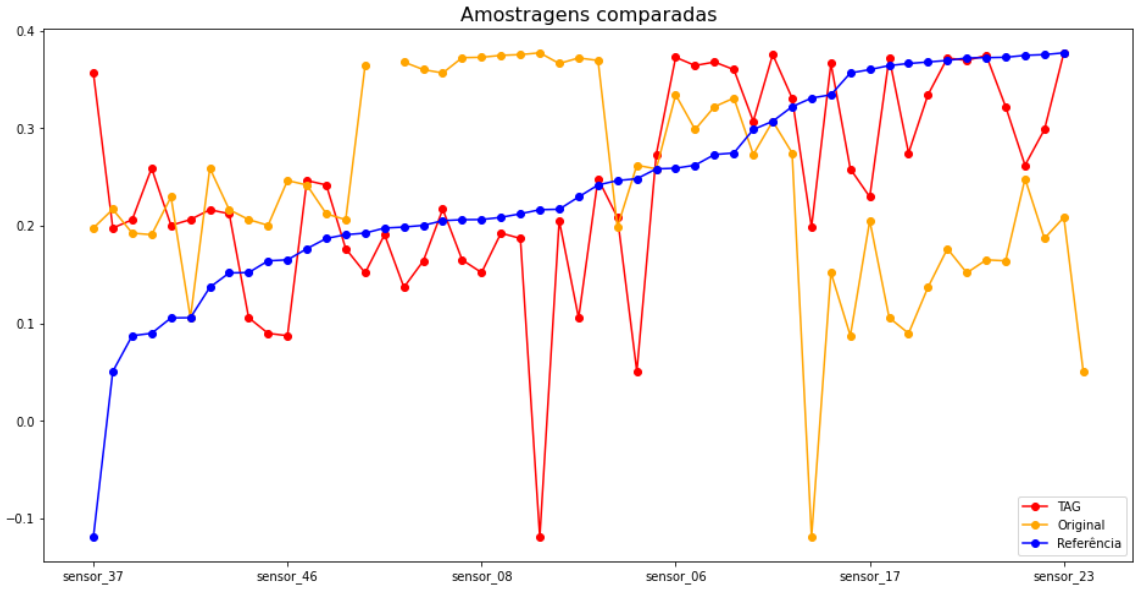
In [29]:

```
# Comportamento esperado do sistema de bombeamento de água - de lugar baixo para alto
df_mean_new_tag.plot.line(figsize=(16,8), marker='o', color='red', label='TAG').set_titl
df_corr.plot.line(marker='o', color='orange', label='Original')
df_corr_sorted.plot.line(marker='o', color='blue', label='Referência')

plt.legend()
```

Out[29]:

```
<matplotlib.legend.Legend at 0x1e283c2b460>
```



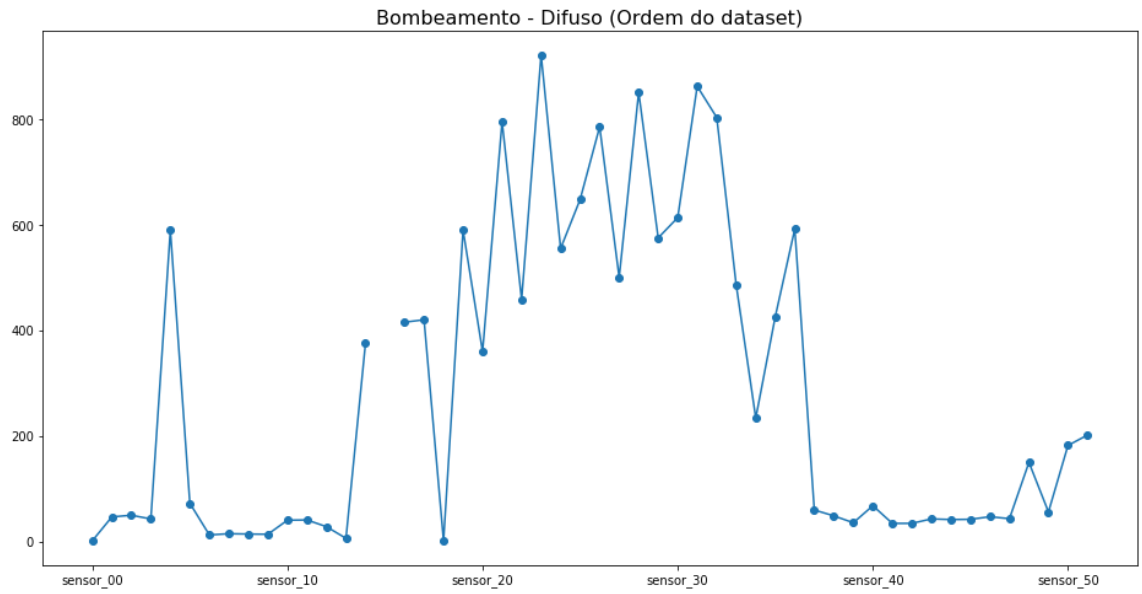
IMPRESSÃO - A mensuração deveria ter o modelo de Referência (em azul) como modelo no entanto o modelo amostral (TAG - Vermelho) é difuso. O modelo de Referência não está correlacionado com o eixo x somente em posicionamento.

In [30]:

```
# Erro de documentação, anotação ou de etiquetamento(TAG's)
plt.figure(figsize=(16,8))
df_mean.plot.line(marker='o').set_title('Bombeamento - Difuso (Ordem do dataset)', fonts
```

Out[30]:

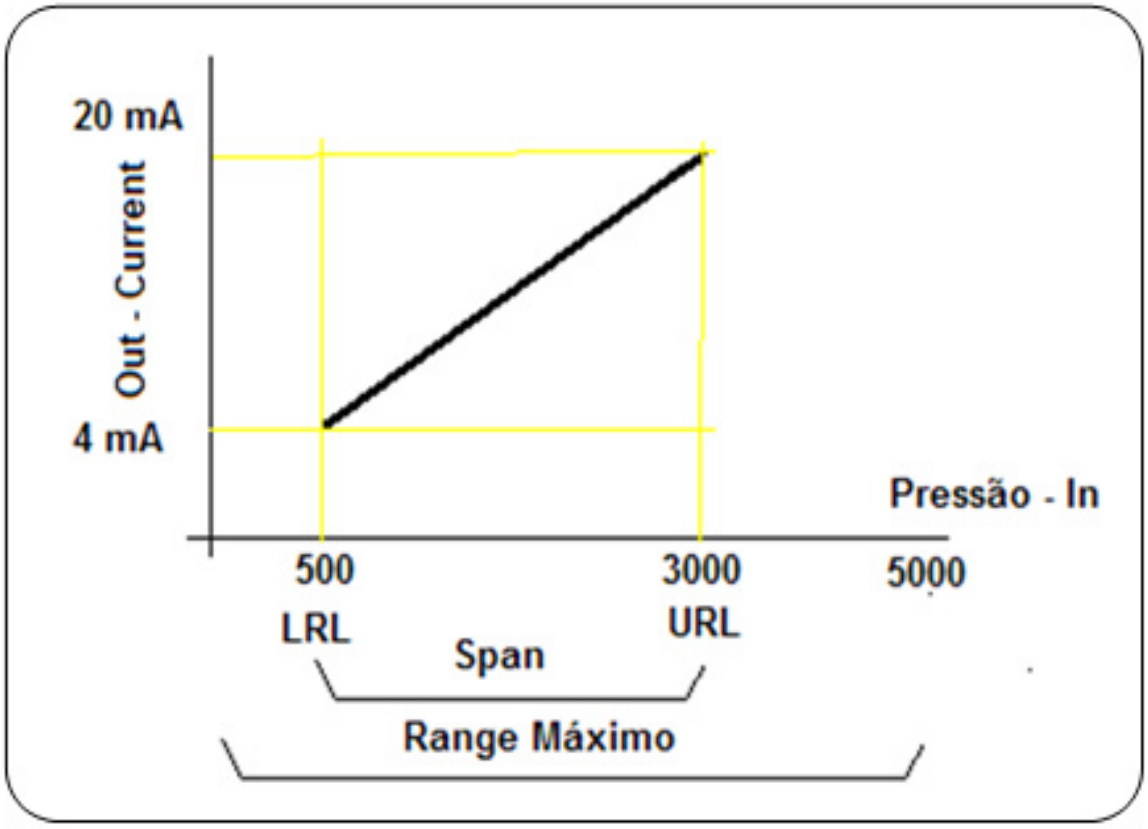
```
Text(0.5, 1.0, 'Bombeamento - Difuso (Ordem do dataset)')
```



Exploratory Analysis Data - EDA - Sensores

Observe - Análise de valores mínimo (zero) - Para este sistema o não fucionamento deve estar parametrizado em 0.00 qualquer valor acima é um possível descalibração, falha ou defeito

Gráfico exemplifica o funcionamento de um sensor. (zero, range e span)



(https://www.smar.com/en/technical-article/a-few-important-concepts-about-pressure-transmitters)

In [31]:

```
# Análise de valores estatísticos padrão: Mínimo
df_T = df.describe().T
```

In [32]:

```
df_T.sort_values(by=['min']).style.background_gradient(cmap='YlOrRd')
```

Out[32]:

	count	mean	std	min	25%	50%	
sensor_00	210112.000000	2.372221	0.412227	0.000000	2.438831	2.456539	2.4
sensor_37	220304.000000	60.787360	37.604883	0.000000	28.799220	64.295485	90.8
sensor_35	220304.000000	427.129817	141.772519	0.000000	353.176625	473.349350	528.8
sensor_30	220059.000000	614.596442	195.726872	0.000000	627.777800	668.981400	697.2
sensor_27	220304.000000	501.506589	169.823173	0.000000	448.297950	494.468450	536.2
sensor_25	220284.000000	649.144799	220.865166	0.000000	693.957800	740.203500	750.3
sensor_24	220304.000000	556.235397	182.297979	0.000000	601.151050	625.873500	628.6
sensor_23	220304.000000	922.609264	291.835280	0.000000	950.922400	981.925000	1090.8
sensor_20	220304.000000	360.805165	101.974118	0.000000	398.021500	399.367000	400.0
sensor_19	220304.000000	590.829775	199.345820	0.000000	662.768975	665.672400	667.1
sensor_18	220274.000000	2.303785	0.765883	0.000000	2.447542	2.533704	2.5
sensor_17	220274.000000	421.127517	129.156175	0.000000	454.138825	462.020250	466.8
sensor_16	220289.000000	416.472892	126.072642	0.000000	459.453400	462.856100	464.3
sensor_22	220279.000000	459.792815	154.528337	0.000000	478.962600	531.855900	534.2
sensor_01	219951.000000	47.591611	3.296666	0.000000	46.310760	48.133678	49.4
sensor_12	220301.000000	29.136975	10.113935	0.000000	28.686810	32.515830	34.9
sensor_11	220301.000000	41.918319	13.056425	0.000000	38.856420	45.363140	49.6
sensor_10	220301.000000	41.470339	12.093519	0.000000	40.705260	44.291340	47.4
sensor_09	215725.000000	14.799210	2.091963	0.000000	15.053530	15.082470	15.1
sensor_13	220301.000000	7.078858	6.901755	0.000000	1.538516	2.929809	12.8
sensor_07	214869.000000	15.843152	2.201155	0.000000	15.907120	16.167530	16.4
sensor_05	220301.000000	73.396414	17.298247	0.000000	69.976260	75.576790	80.9
sensor_06	215522.000000	13.501537	2.163736	0.014468	13.346350	13.642940	14.5
sensor_08	215213.000000	15.200721	2.037390	0.028935	15.183740	15.494790	15.6
sensor_32	220252.000000	804.283915	260.602361	0.240716	760.607475	878.850750	943.8
sensor_29	220248.000000	576.195305	225.764091	0.636574	518.947225	564.872500	744.0
sensor_36	220304.000000	593.033876	289.385511	2.260970	288.547575	709.668050	837.3
sensor_04	220301.000000	590.673936	144.023912	2.798032	626.620400	632.638916	637.6
sensor_28	220304.000000	851.690339	313.074032	4.319347	782.682625	967.279850	1043.9
sensor_33	220304.000000	486.405980	150.751836	6.460602	489.761075	512.271750	555.1
sensor_39	220293.000000	36.610444	15.613723	19.270830	32.552080	35.416660	39.0
sensor_41	220293.000000	35.365126	7.898665	20.833330	32.552080	34.895832	37.7
sensor_42	220293.000000	35.453455	10.259521	22.135416	32.812500	35.156250	36.9
sensor_40	220293.000000	68.844530	21.371139	23.437500	57.812500	66.406250	77.8
sensor_31	220304.000000	863.323100	283.544760	23.958330	839.062400	917.708300	981.2
sensor_43	220293.000000	43.879591	11.044404	24.479166	39.583330	42.968750	46.6
sensor_38	220293.000000	49.655946	10.540397	24.479166	45.572910	49.479160	53.6

	count	mean	std	min	25%	50%	
sensor_44	220293.000000	42.656877	11.576355	25.752316	36.747684	40.509260	45.1
sensor_45	220293.000000	43.094984	12.837520	26.331018	36.747684	40.219910	44.8
sensor_48	220293.000000	150.889044	82.244957	26.331018	83.912030	138.020800	208.3
sensor_46	220293.000000	48.018585	15.641284	26.331018	40.509258	44.849540	51.2
sensor_49	220293.000000	57.119968	19.143598	26.620370	47.743060	52.662040	60.7
sensor_47	220293.000000	44.340903	10.442437	27.199070	39.062500	42.534720	46.5
sensor_50	143303.000000	183.049260	65.258650	27.488426	167.534700	193.865700	219.9
sensor_51	204937.000000	202.699667	109.588607	27.777779	179.108800	197.338000	216.7
sensor_03	220301.000000	43.752481	2.418887	31.640620	42.838539	44.227428	45.3
sensor_14	220299.000000	376.860041	113.206382	32.409550	418.103250	420.106200	420.9
sensor_02	220301.000000	50.867392	3.666820	33.159720	50.390620	51.649300	52.7
sensor_26	220300.000000	786.411781	246.663608	43.154790	790.489575	861.869600	919.1
sensor_34	220304.000000	234.971776	88.376065	54.882370	172.486300	226.356050	316.8
sensor_21	220304.000000	796.225942	226.679317	95.527660	875.464400	879.697600	882.1
sensor_15	0.000000	nan	nan	nan	nan	nan	

In [33]:

```
df_min = df_T.sort_values(by=['min'])
```

In [34]:

```
df_zero = df_min['min'].apply(lambda x: 'zero' if x==0 else 'manutencao' )
```

IMPRESSÃO - Classificação de sensores que possuem o zero e os que não possuem e necessitam de manutenção do seu respectivo sensor ou no sistema de entrada de sensoreamento (INPUT).

In [35]:

```
df_zero
```

Out[35]:

```
sensor_00          zero
sensor_37          zero
sensor_35          zero
sensor_30          zero
sensor_27          zero
sensor_25          zero
sensor_24          zero
sensor_23          zero
sensor_20          zero
sensor_19          zero
sensor_18          zero
sensor_17          zero
sensor_16          zero
sensor_22          zero
sensor_01          zero
sensor_12          zero
sensor_11          zero
sensor_10          zero
sensor_09          zero
sensor_13          zero
sensor_07          zero
sensor_05          zero
sensor_06    manutencao
sensor_08    manutencao
sensor_32    manutencao
sensor_29    manutencao
sensor_36    manutencao
sensor_04    manutencao
sensor_28    manutencao
sensor_33    manutencao
sensor_39    manutencao
sensor_41    manutencao
sensor_42    manutencao
sensor_40    manutencao
sensor_31    manutencao
sensor_43    manutencao
sensor_38    manutencao
sensor_44    manutencao
sensor_45    manutencao
sensor_48    manutencao
sensor_46    manutencao
sensor_49    manutencao
sensor_47    manutencao
sensor_50    manutencao
sensor_51    manutencao
sensor_03    manutencao
sensor_14    manutencao
sensor_02    manutencao
sensor_26    manutencao
sensor_34    manutencao
sensor_21    manutencao
sensor_15    manutencao
Name: min, dtype: object
```

Atenção - 30 sensores necessitam de manutenção. Pois estes nunca chegam ao zero mesmo quando quebrados ou danificados ou em manutenção.

In [36]:

```
# Transforma o objeto df_zero em dataframe
df_manutencao = pd.DataFrame(data=df_zero)
```

In [37]:

```
# Deleta as linha que contem 'zero'.
df_manutencao_zero = df_manutencao[df_manutencao["min"].str.contains("zero") == False]
```

In [38]:

```
# Lista de sensores que necesitam de manutenção.
df_manutencao_zero
```

Out[38]:

	min
sensor_06	manutencao
sensor_08	manutencao
sensor_32	manutencao
sensor_29	manutencao
sensor_36	manutencao
sensor_04	manutencao
sensor_28	manutencao
sensor_33	manutencao
sensor_39	manutencao
sensor_41	manutencao
sensor_42	manutencao
sensor_40	manutencao
sensor_31	manutencao
sensor_43	manutencao
sensor_38	manutencao
sensor_44	manutencao
sensor_45	manutencao
sensor_48	manutencao
sensor_46	manutencao
sensor_49	manutencao
sensor_47	manutencao
sensor_50	manutencao
sensor_51	manutencao
sensor_03	manutencao
sensor_14	manutencao
sensor_02	manutencao
sensor_26	manutencao
sensor_34	manutencao
sensor_21	manutencao
sensor_15	manutencao

Exploratory Analysis Data - EDA - Análise de processo

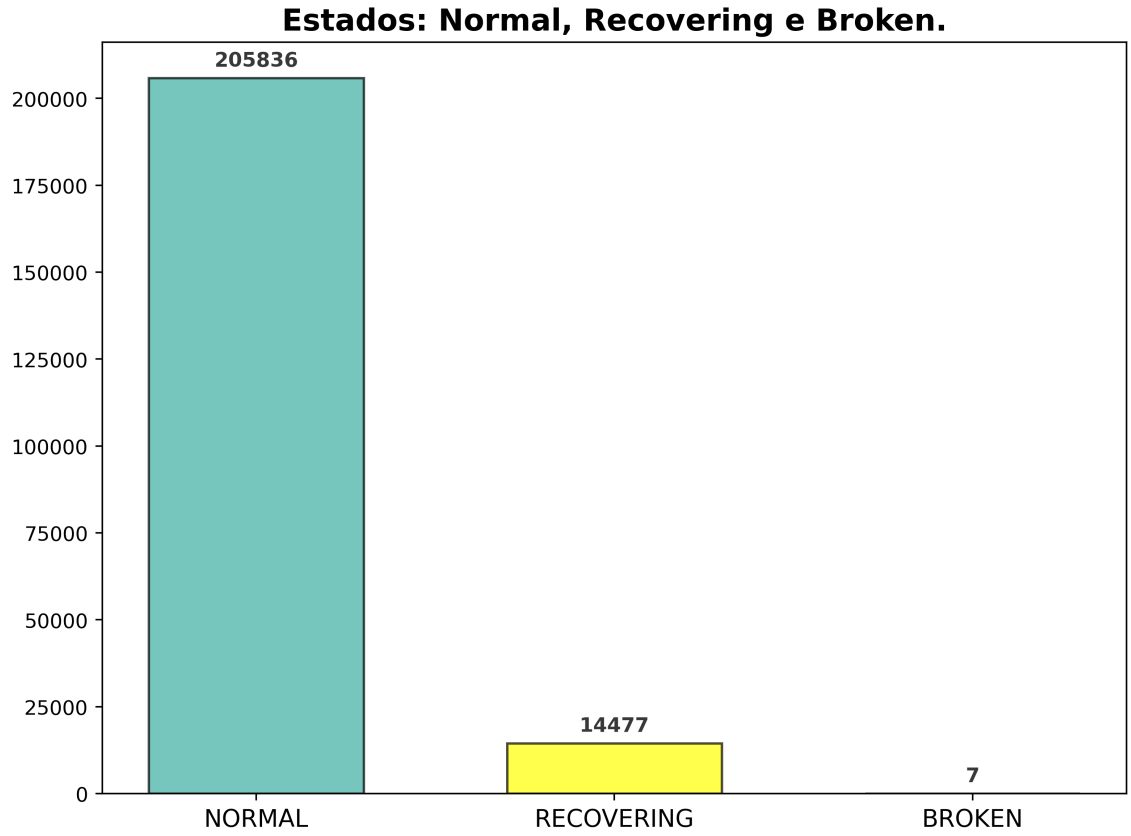
In [39]:

```
# label check (extremely imbalanced label, real world)
label = df['machine_status'].value_counts().sort_values(ascending=False)
fig, ax = plt.subplots(1, 1, figsize=(8, 6), dpi=400)
color_map = ['yellow' for _ in range(len(df['machine_status'].value_counts()))]
color_map[0] = "#3caea3"

ax.bar(label.index, label, alpha=0.7, color=color_map, width=0.6, edgecolor='black', lin
ax.set_title('Estados: Normal, Recovering e Broken.', fontsize=15, fontweight='bold', po

for i in label.index:
    ax.annotate(f"{label[i]}",
                xy=(i, label[i] + 5000),
                va='center', ha='center', fontweight='bold', color='#383838')

ax.set_xticklabels(label.index, fontsize=12, rotation=0)
plt.tight_layout()
plt.show()
```



IMPRESSÃO - Quantificação dos estados de funcionamento: NORMAL, RECOVERING (Recuperando) e BROKEN (Quebrado).

In [40]:

```
df_new_tag
```

Out[40]:

	S00	S01	S02	S03	S04	S05	S06	S07	
0	2.565284	2.465394	1.681353	13.41146	15.05353	15.56713	16.13136	31.11716	30.
1	2.565284	2.465394	1.681353	13.41146	15.05353	15.56713	16.13136	31.11716	30.
2	2.500062	2.444734	1.708474	13.32465	15.01013	15.61777	16.03733	32.08894	30.
3	2.509521	2.460474	1.579427	13.31742	15.08247	15.69734	16.24711	31.67221	30.
4	2.604785	2.445718	1.683831	13.35359	15.08247	15.69734	16.21094	31.95202	30.
...	...	...	...	...	...	...	...	...	
220315	2.499117	2.407350	13.265320	15.11863	15.16204	15.65393	16.65220	38.05424	30.
220316	2.618476	2.400463	13.242270	15.15480	15.11863	15.65393	16.70284	38.53485	30.
220317	2.620500	2.396528	13.188660	15.08970	15.11863	15.69734	16.70284	38.52678	29.
220318	2.514596	2.406366	13.173460	15.11863	15.11863	15.74074	16.56539	38.89159	29.
220319	2.487299	2.396528	13.125930	15.11863	15.01013	15.65393	16.65220	39.40957	29.

220320 rows × 54 columns

AÇÃO - Criação de um dataset de Referência baseado no valor médio, com finalidade de observar o processo.

In [41]:

```
# Carrega os dados do arquivo CSV no dataframe: df
df_ideal = pd.read_csv(os.getcwd() + '\\ideal_sensor.csv', index_col=0)
```

In [42]:

```
# Reorganizar as TAGs dos sensores
df_new_ideal = df_ideal.rename(columns={'sensor_18': 'S00', 'sensor_00': 'S01', 'sensor_
                               'sensor_08':'S05','sensor_07':'S06', 'sensor_12':'S07','sens
                               'sensor_39':'S10','sensor_10':'S11','sensor_11':'S12','senso
                               'sensor_03':'S15','sensor_43':'S16','sensor_47':'S17','senso
                               'sensor_38':'S20','sensor_02':'S21','sensor_49':'S22','senso
                               'sensor_05':'S25','sensor_48':'S26','sensor_50':'S27','senso
                               'sensor_20':'S30','sensor_14':'S31','sensor_16':'S32','senso
                               'sensor_22':'S35','sensor_33':'S36','sensor_27':'S37','senso
                               'sensor_04':'S40','sensor_19':'S41','sensor_36':'S42','senso
                               'sensor_26':'S45','sensor_21':'S46','sensor_32':'S47','senso
                               'sensor_23':'S50','sensor_15':'S51'})
```

In [43]:

```
df_ideal_tag = df_new_ideal.reindex(sorted(df_new_ideal.columns), axis=1)
```

In [44]:

```
df_ideal_tag
```

Out[44]:

Column1	S00	S01	S02	S03	S04	S05	S06	S07
0	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697
1	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697
2	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697
3	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697
4	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697
...	...	...	...	...	...	...	...	..
220315	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697
220316	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697
220317	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697
220318	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697
220319	2.303785	2.372221	7.078858	13.501537	14.79921	15.200721	15.843152	29.13697

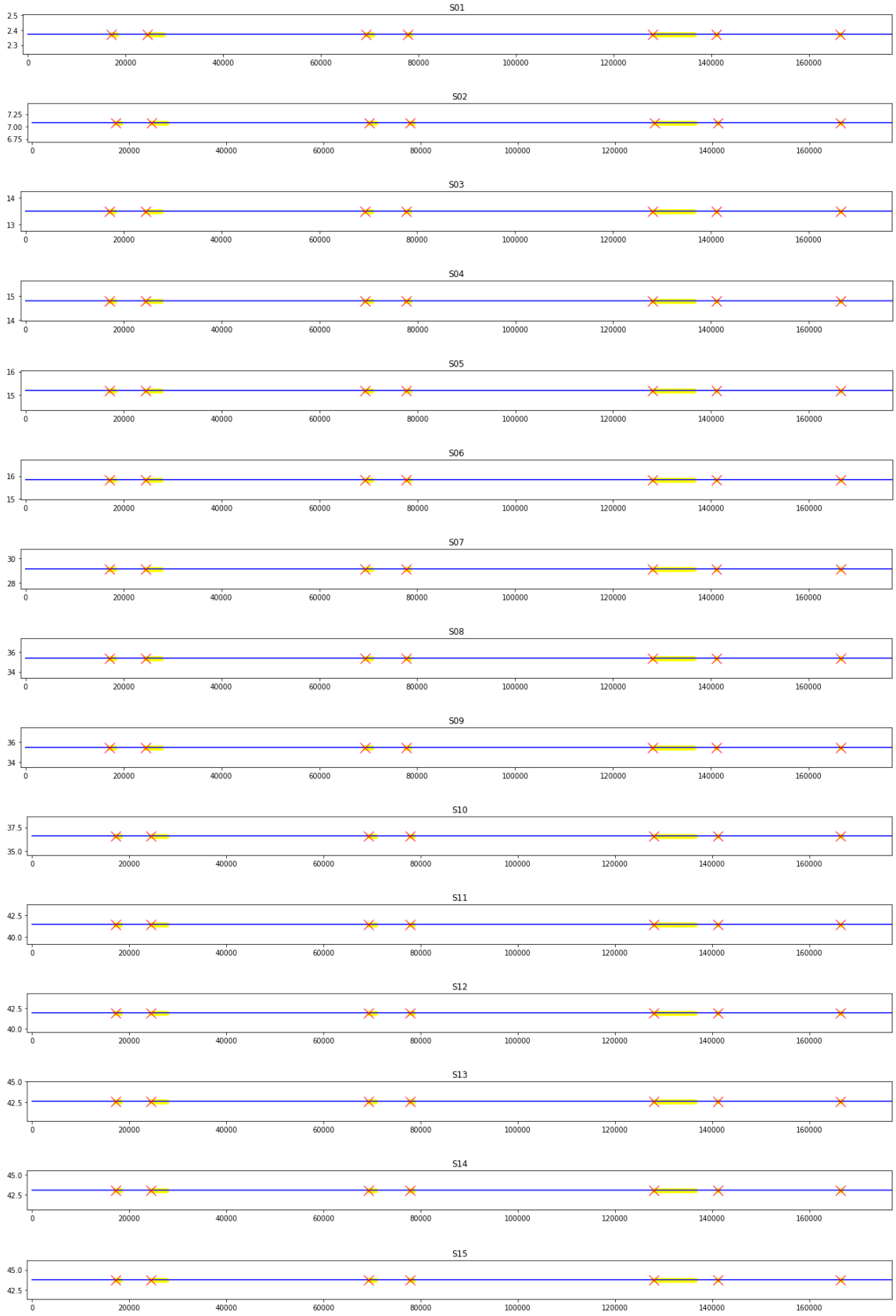
220320 rows × 54 columns

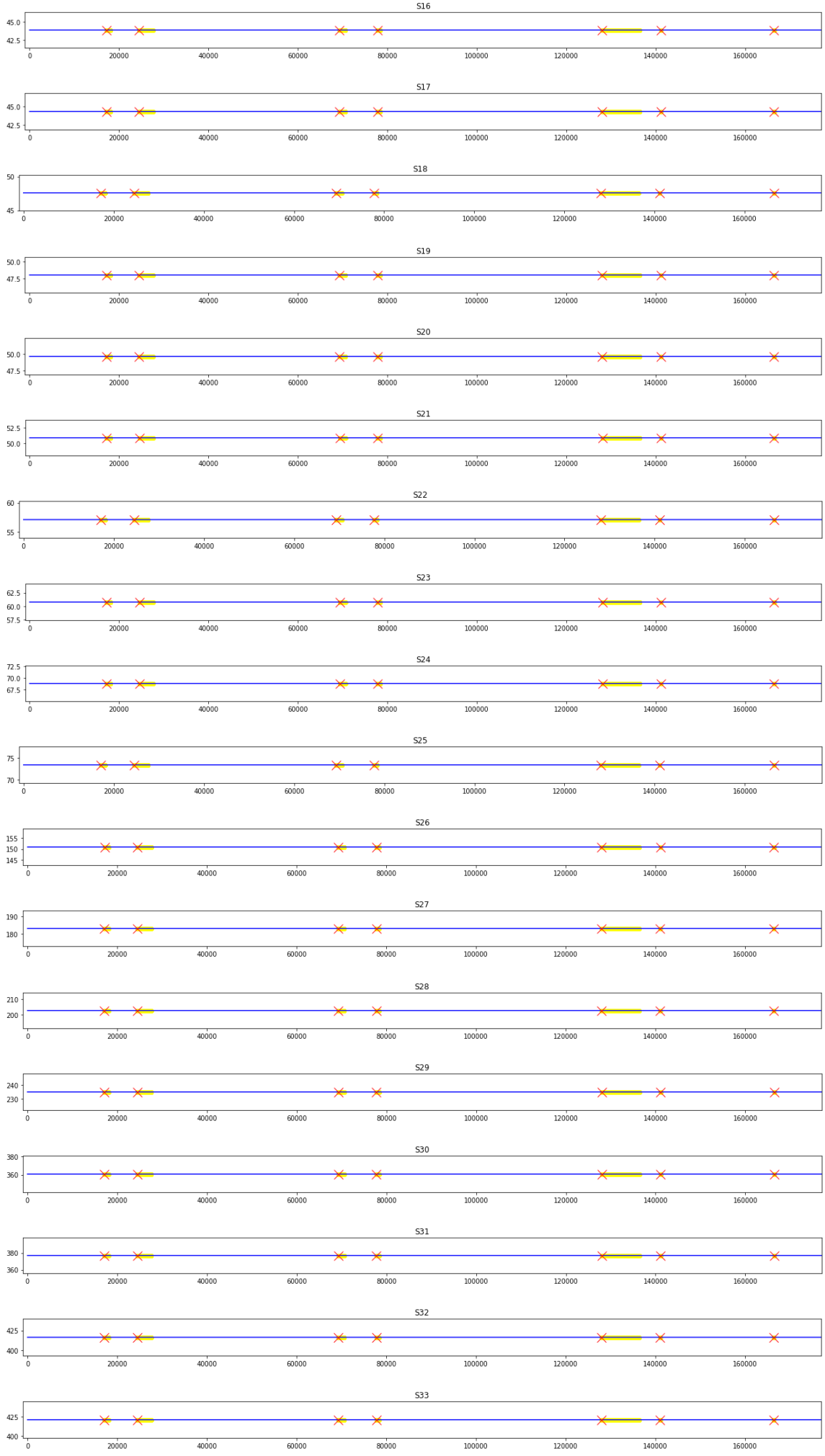
In [81]:

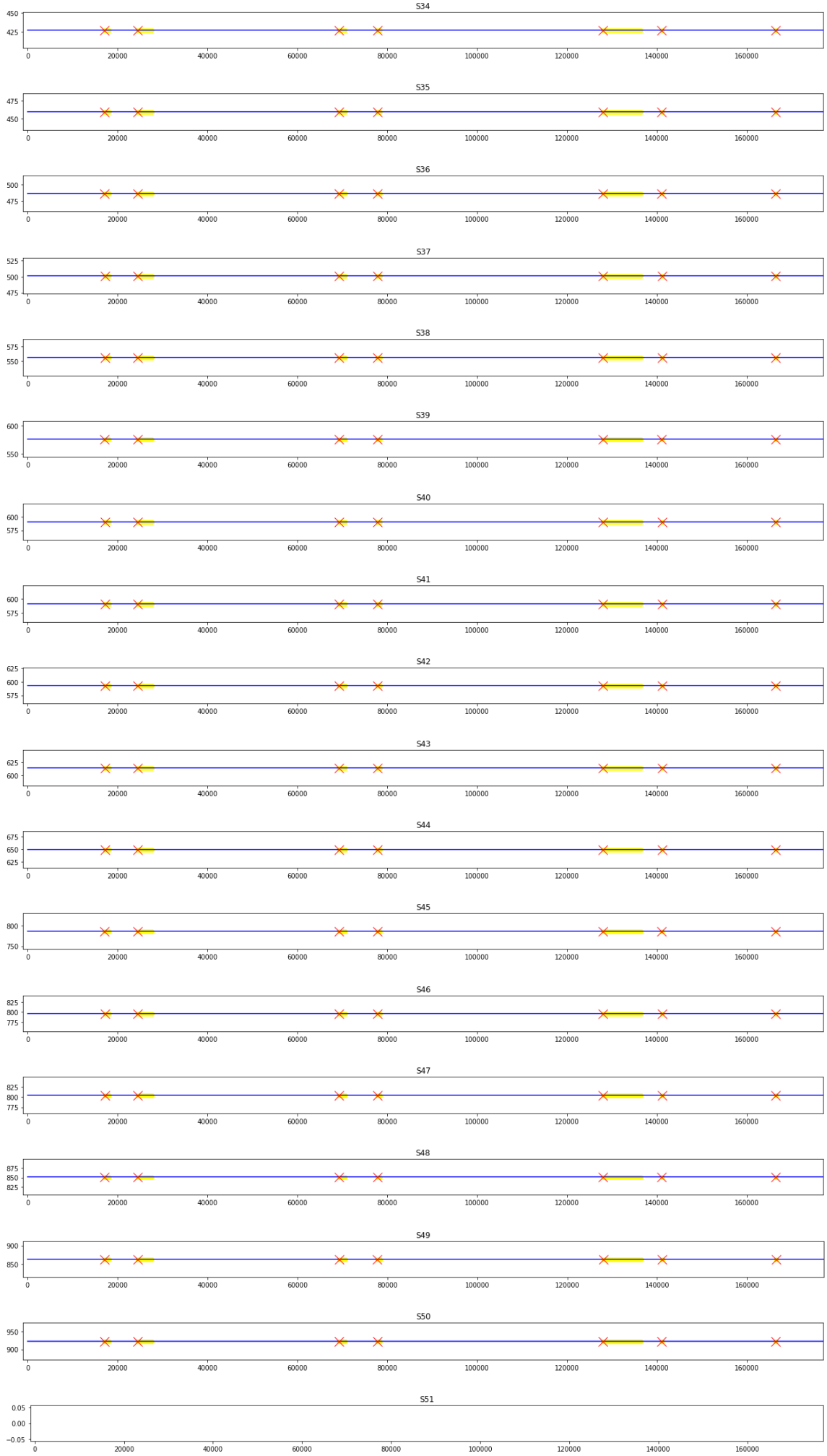
```
# get useful columns and rows
sensor_cols = df_ideal_tag.iloc[:,1:54]
broken_rows = df_ideal_tag[df_ideal_tag['machine_status']=='BROKEN']
recovery_rows = df_ideal_tag[df_ideal_tag['machine_status']=='RECOVERING']
normal_rows = df_ideal_tag[df_ideal_tag['machine_status']=='NORMAL']
machine_status_col = df_ideal_tag['machine_status']
```

In [*]:

```
for sensor in sensor_cols:
    plot = plt.figure(figsize=(22,1))
    plot = plt.plot(recovery_rows[sensor], linestyle='none', marker='o', color='yellow',
    plot = plt.plot(df_ideal_tag[sensor], color='blue')
    plot = plt.plot(broken_rows[sensor], linestyle='none', marker='x', color='red', mark
    plot = plt.title(sensor)
    plt.xlim((-1000,177000))
    plt.show();
```




S16
S17
S18
S19
S20
S21
S22
S23
S24
S25
S26
S27
S28
S29
S30
S31
S32
S33


S34
S35
S36
S37
S38
S39
S40
S41
S42
S43
S44
S45
S46
S47
S48
S49
S50
S51

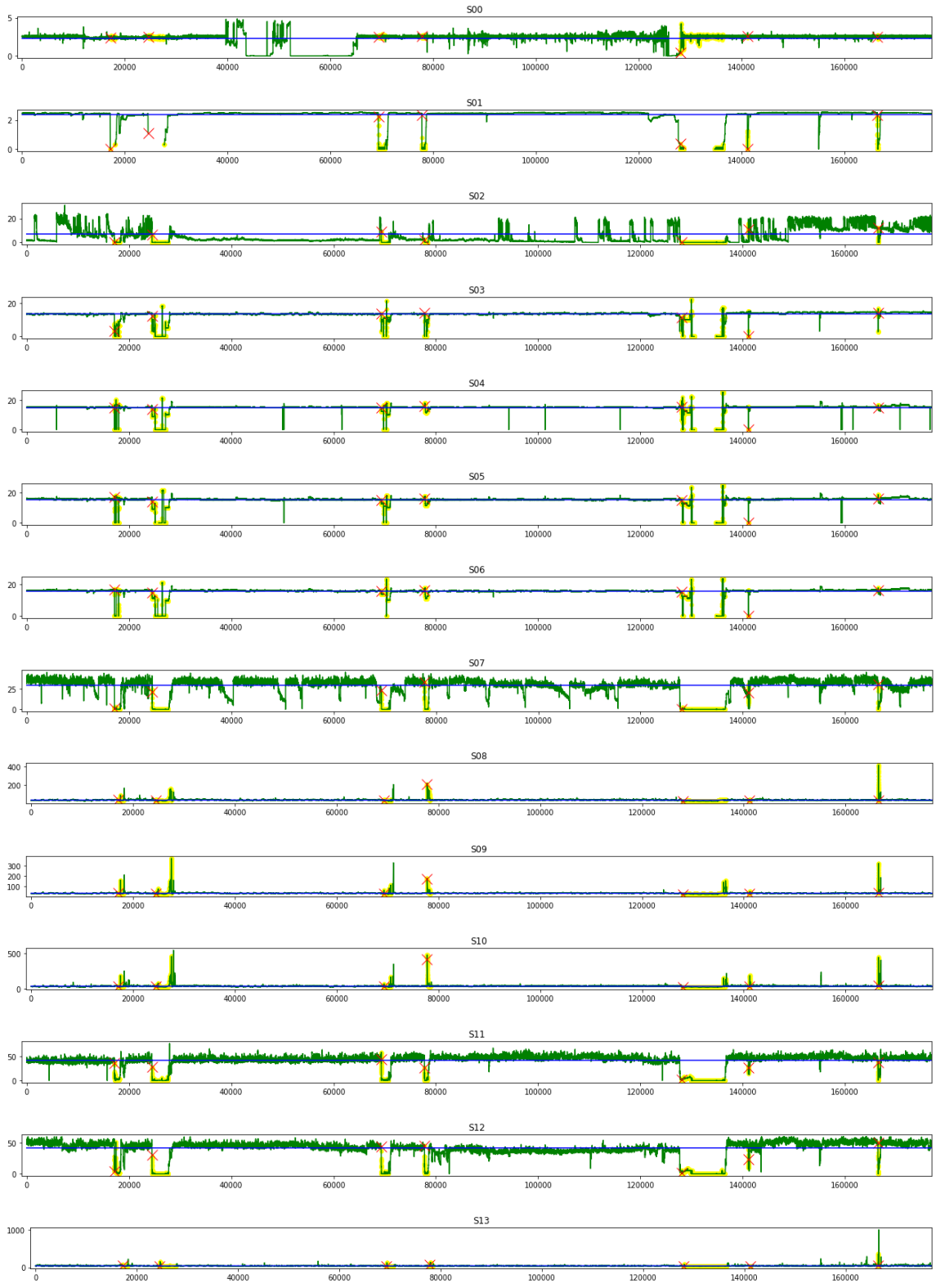
OBSERVE a visualização do processo em função do sensoreamento e estados de processos.

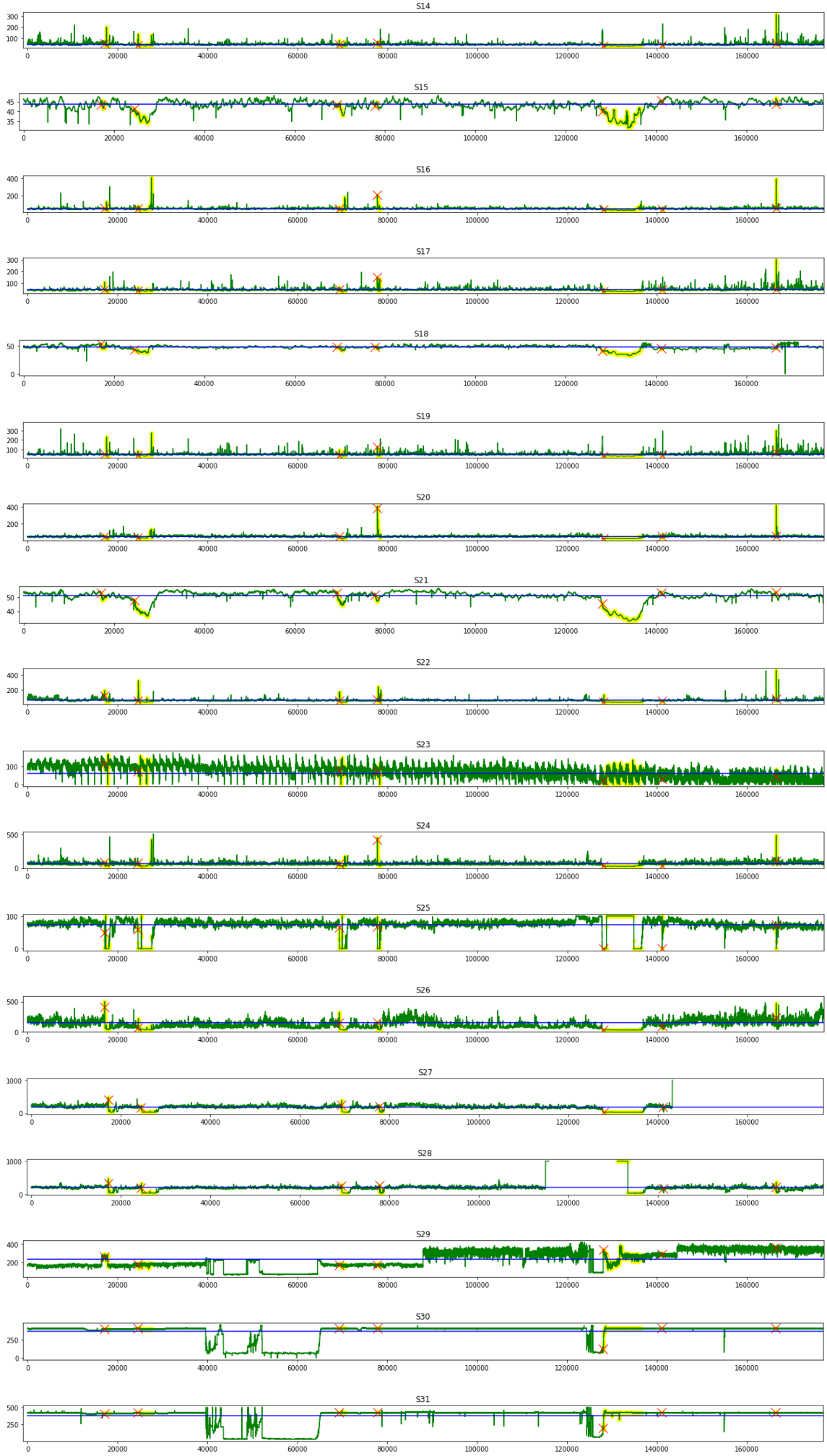
In [47]:

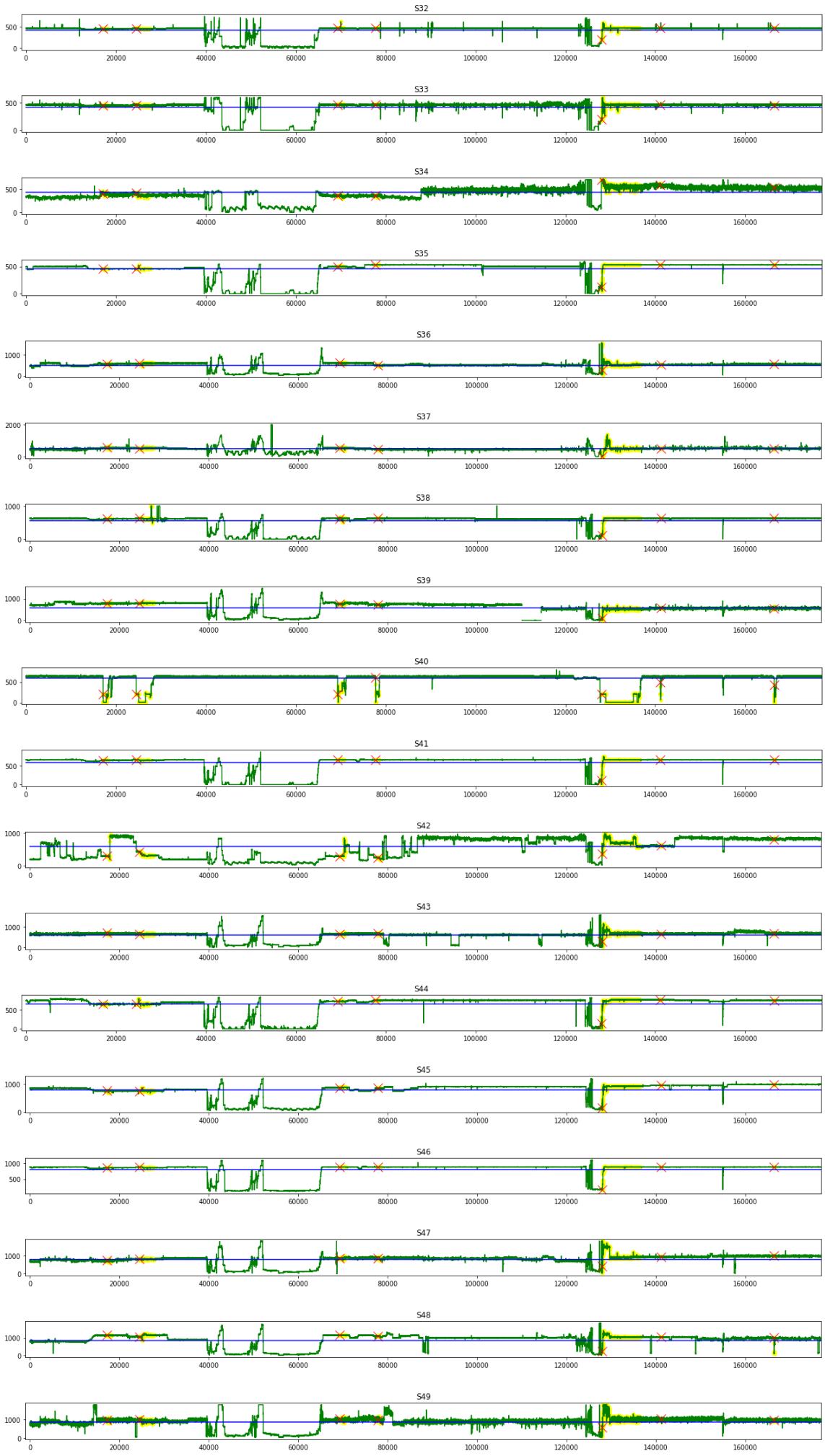
```
# get useful columns and rows
sensor_cols = df_new_tag.iloc[:,0:53]
broken_rows = df_new_tag[df_new_tag['machine_status']=='BROKEN']
recovery_rows = df_new_tag[df_new_tag['machine_status']=='RECOVERING']
normal_rows = df_new[df_new_tag['machine_status']=='NORMAL']
machine_status_col = df_new_tag['machine_status']
```

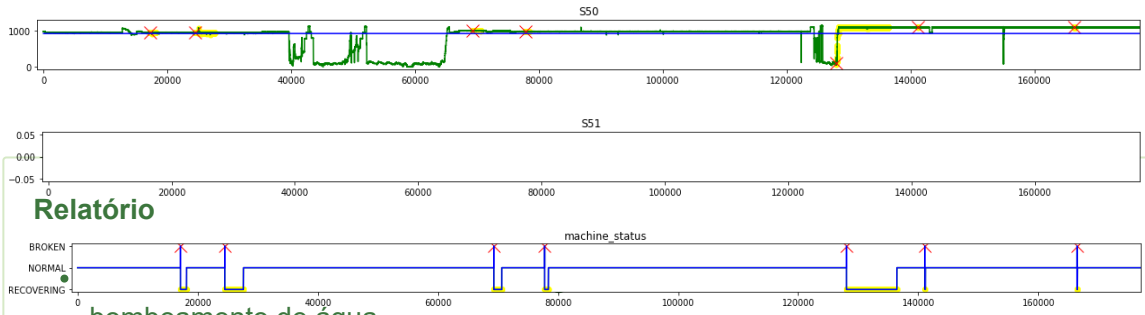
In [48]:

```
for sensor in sensor_cols:
    plot = plt.figure(figsize=(22,1))
    plot = plt.plot(recovery_rows[sensor], linestyle='none', marker='o', color='yellow',
    plot = plt.plot(df_new_tag[sensor], color='green')
    plot = plt.plot(broken_rows[sensor], linestyle='none', marker='x', color='red', mark
    plot = plt.title(sensor)
    plot = plt.plot(df_ideal_tag[sensor], color='blue')
    plt.xlim((-1000,177000))
    plt.show();
```




S14
S15
S16
S17
S18
S19
S20
S21
S22
S23
S24
S25
S26
S27
S28
S29
S30
S31


S32
S33
S34
S35
S36
S37
S38
S39
S40
S41
S42
S43
S44
S45
S46
S47
S48
S49



Relatório

sistema de bombeamento de água.

- Existem sensores que nunca chegam no zero.
- Existem sensores que não foram substituídos.
- Existem sensores que não funcionaram em todos o período amostral.
- Existem sensores que sua amostragem não condiz com o real funcionamento.
- Existe desbalanceamento no sistema alguns pontos de sobrecargas e outros em baixa eficiência.
- O sistema é de acionamento eletromecânico
- É observável, picos causados devido à cavitação
- É necessário uma auditória de serviços.
- É necessário reavaliação da atuação da equipe de manutenção.
- Várias manutenções (Recovering) não foram realmente realizadas.
- Diversas falhas com perda de sinal não foram notificadas.
- É necessário realizar inspeção de engenharia especializada em eletro-instrumentação para a revisão completa do sistema
- É necessário um retrofitting em todo o sistema

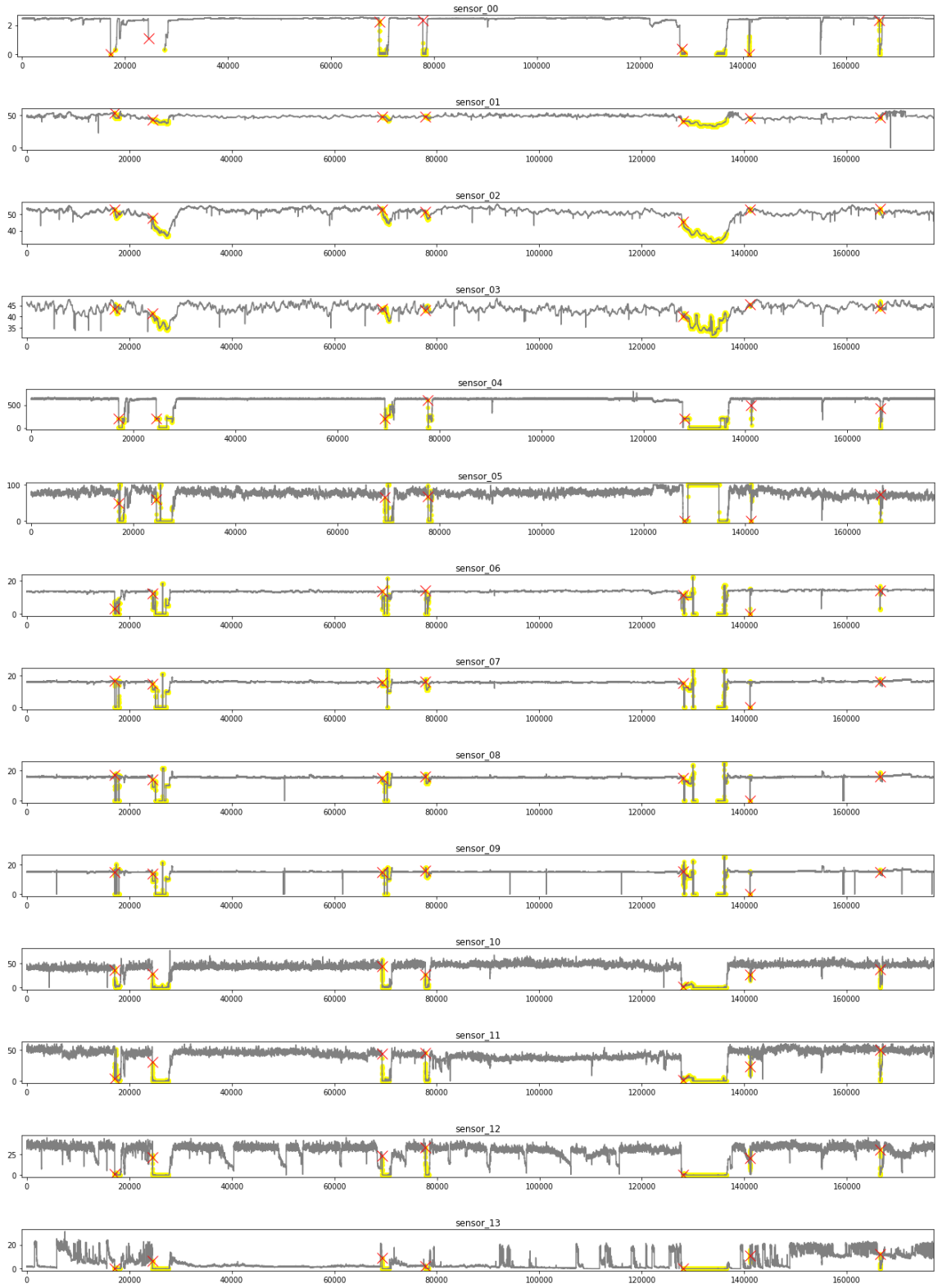
Atenção - A aplicação da predição de falha por Machine Learning - ML, por métodos diretos sem a aplicação das correções necessárias, por mais acurado que seja estas análises todas são erradas. Se não for considerado os seguintes critérios: Erro de processo, ineficiência do processo, revisão de engenharia da planta, ineficiência de mão de obra de manutenção e apuração de fraude em manutenção.

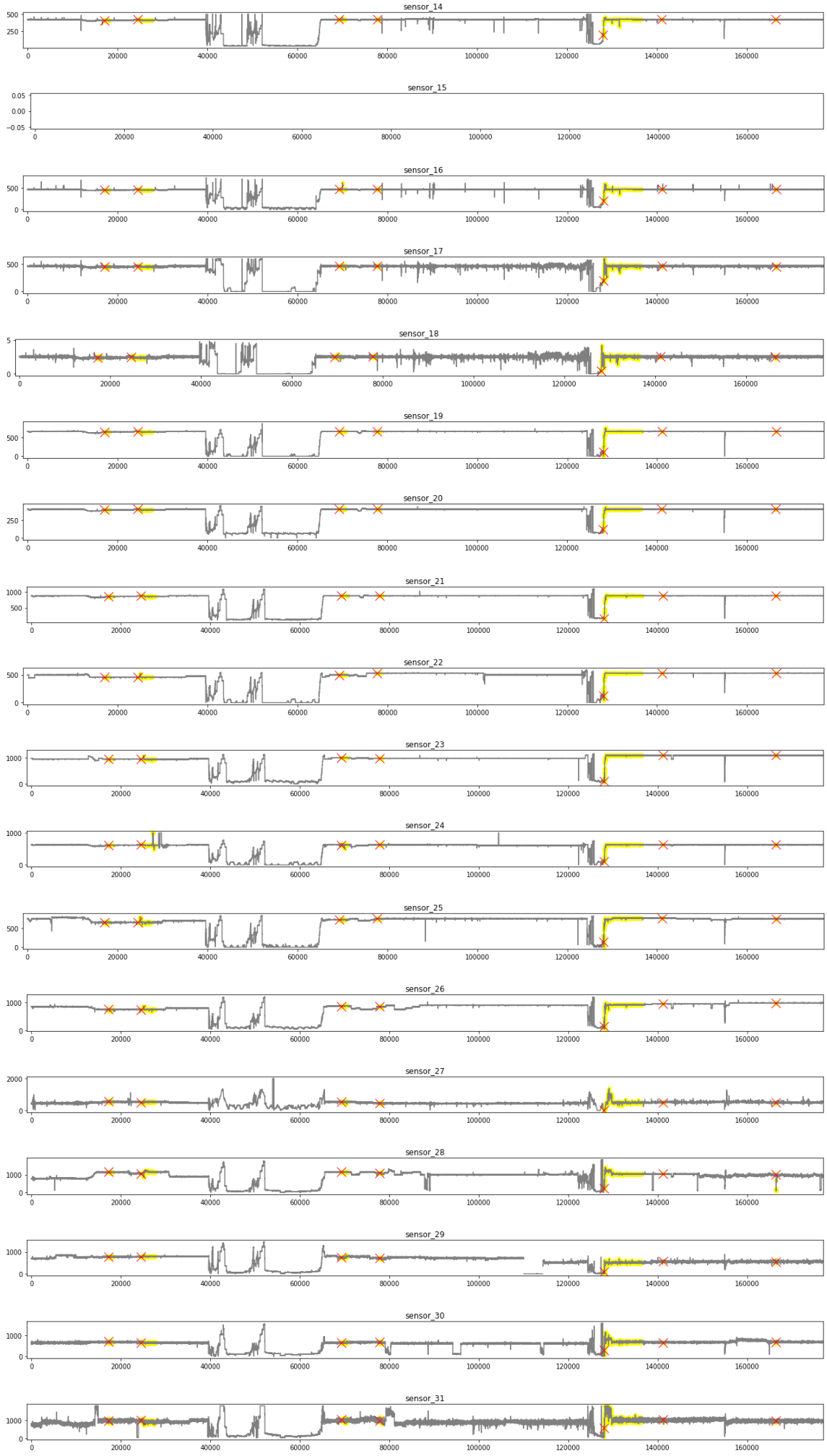
In [49]:

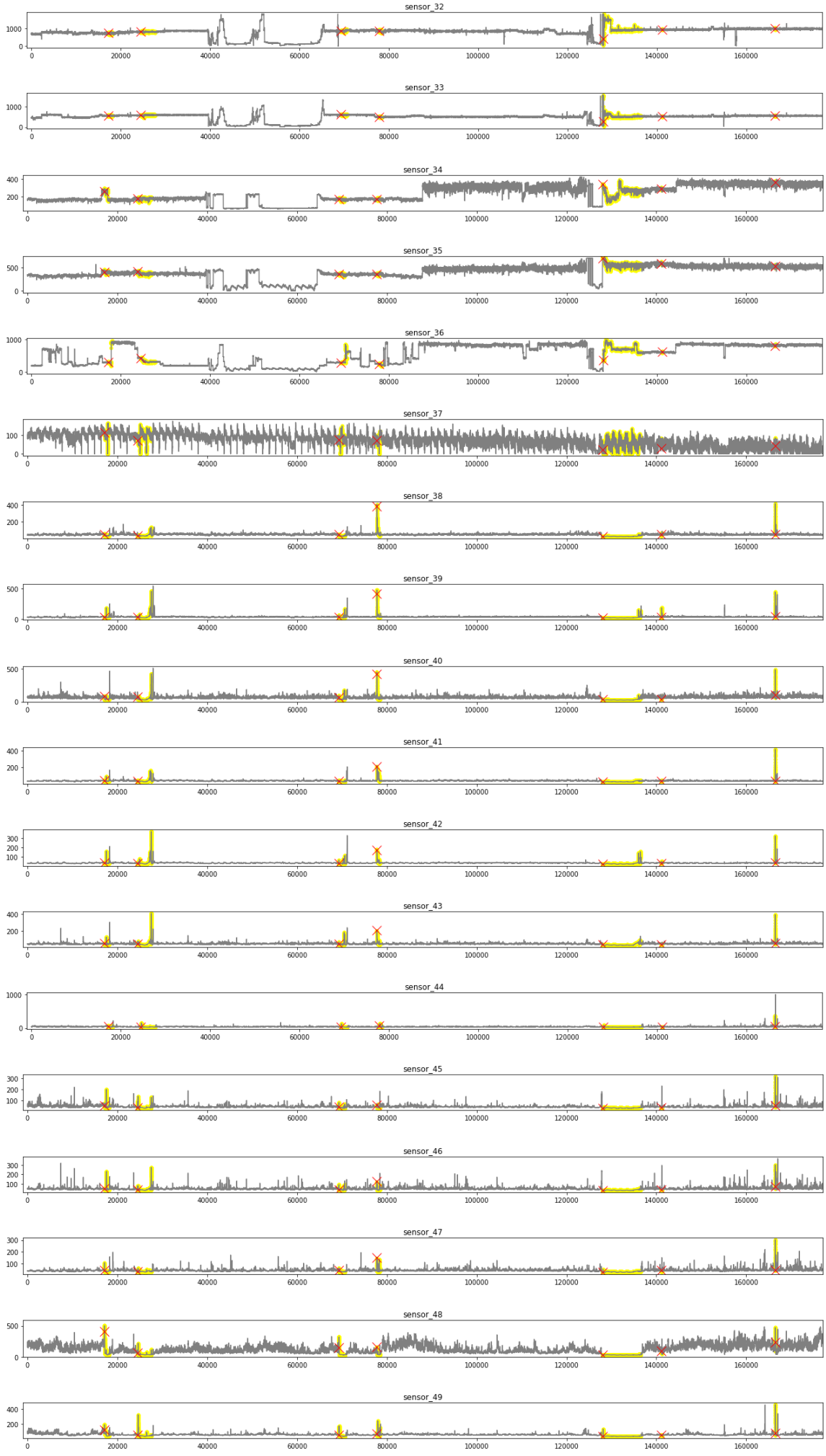
```
# get useful columns and rows
sensor_cols = df.iloc[:,1:54]
broken_rows = df[df['machine_status']=='BROKEN']
recovery_rows = df[df['machine_status']=='RECOVERING']
normal_rows = df[df['machine_status']=='NORMAL']
machine_status_col = df['machine_status']
```

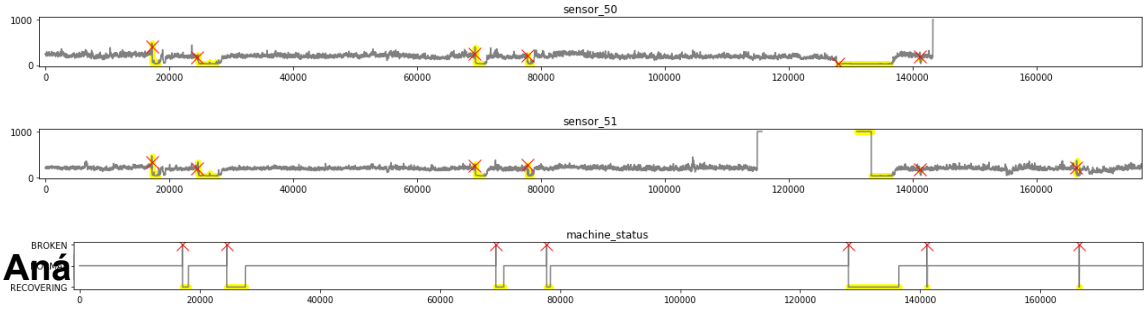
In [50]:

```
for sensor in sensor_cols:
    plot = plt.figure(figsize=(22,1))
    plot = plt.plot(recovery_rows[sensor], linestyle='none', marker='o', color='yellow',
    plot = plt.plot(df[sensor], color='grey')
    #plot = plt.plot(df_ideal[sensor]*1.2, color='green')
    #plot = plt.plot(df_ideal[sensor]*0.8, color='green')
    #plot = plt.plot(df_ideal[sensor]*0, color='black')
    plot = plt.plot(broken_rows[sensor], linestyle='none', marker='x', color='red', mark
    plot = plt.title(sensor)
    plt.xlim((-1000,177000))
    plt.show();
```




sensor_14
sensor_15
sensor_16
sensor_17
sensor_18
sensor_19
sensor_20
sensor_21
sensor_22
sensor_23
sensor_24
sensor_25
sensor_26
sensor_27
sensor_28
sensor_29
sensor_30
sensor_31


sensor_32
sensor_33
sensor_34
sensor_35
sensor_36
sensor_37
sensor_38
sensor_39
sensor_40
sensor_41
sensor_42
sensor_43
sensor_44
sensor_45
sensor_46
sensor_47
sensor_48
sensor_49

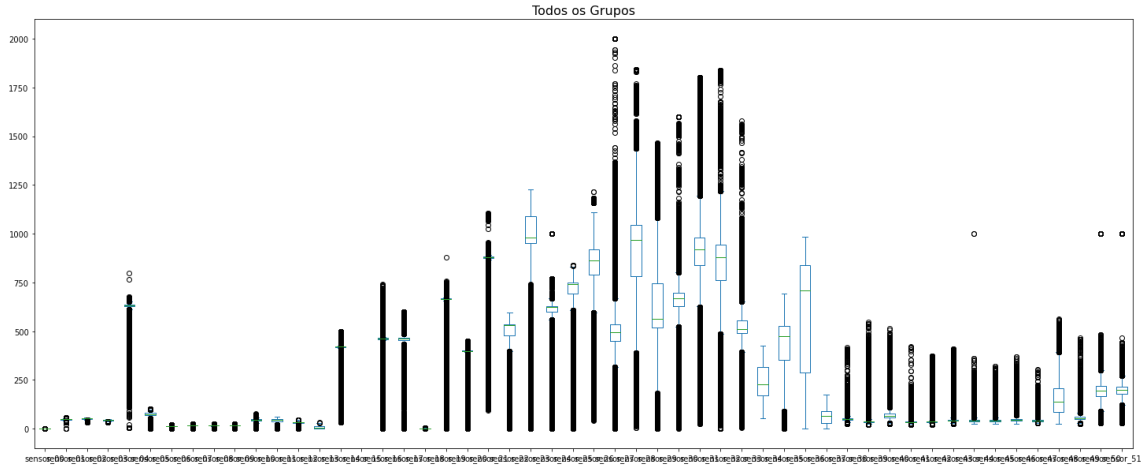


In [51]:

```
# Verificação gráfico de caixa 'Box' por quartis e pontos fora de padrão(outliers)
df.plot(kind = 'box', figsize=(25,10)).set_title('Todos os Grupos', fontsize=16)
```

Out[51]:

```
Text(0.5, 1.0, 'Todos os Grupos')
```

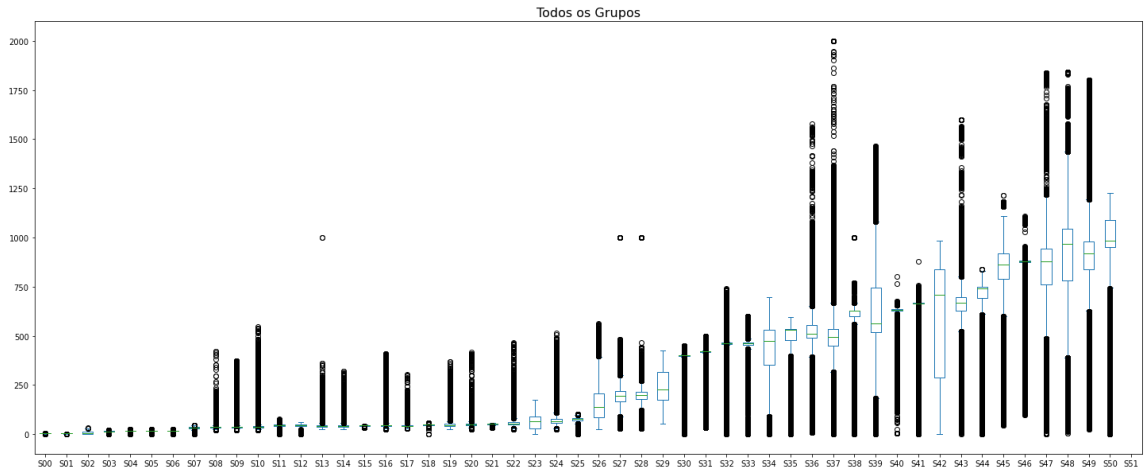


In [52]:

```
# Verificação gráfico de caixa 'Box' por quartis e pontos fora de padrão(outliers)
df_new_tag.plot(kind = 'box', figsize=(25,10)).set_title('Todos os Grupos', fontsize=16)
```

Out[52]:

```
Text(0.5, 1.0, 'Todos os Grupos')
```

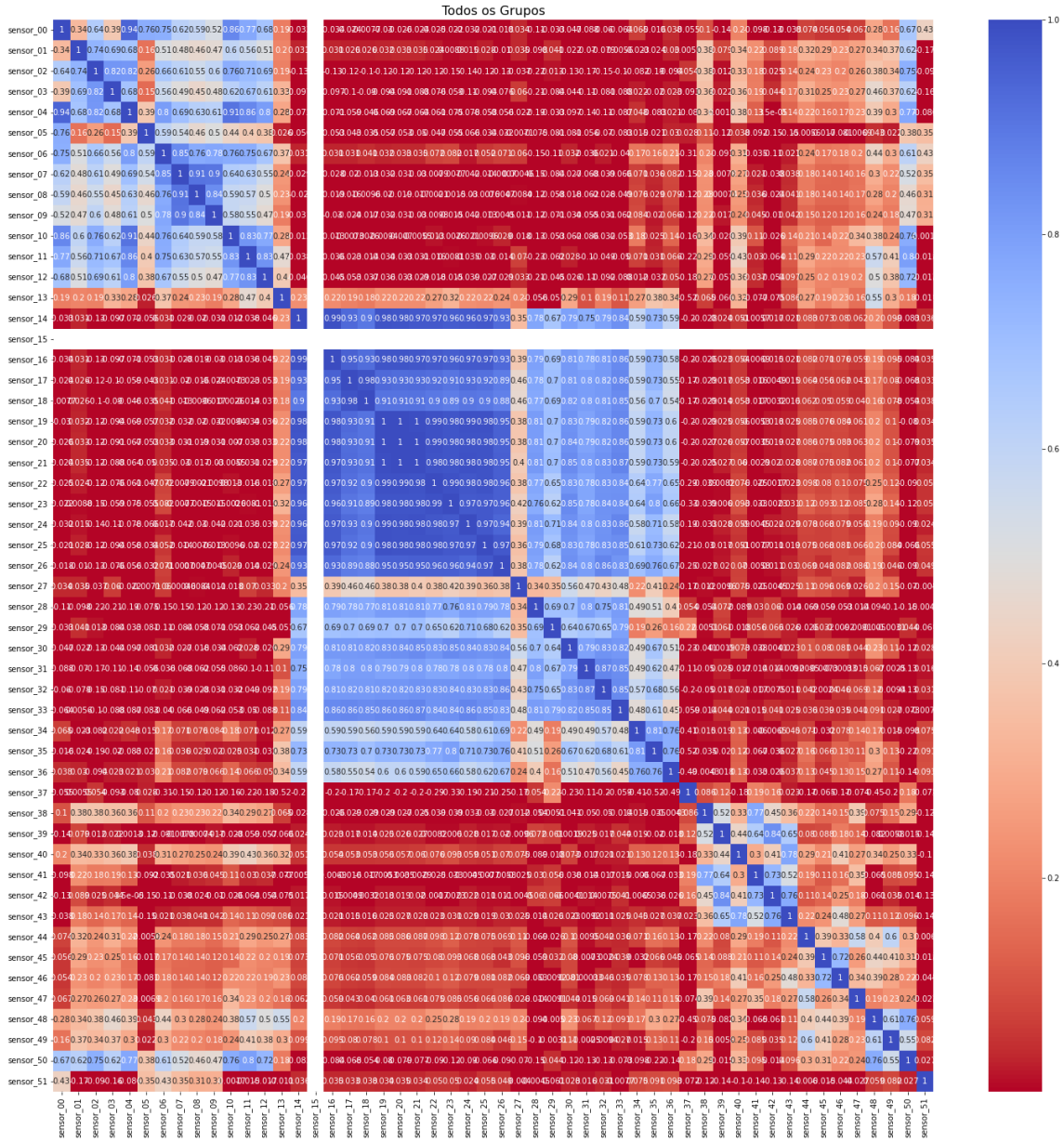


In [53]:

```
# Heatmap todo o conjunto de dados
plt.figure(figsize=(25,25))
sns.heatmap(df.corr(),annot=True, vmin=.001, vmax=1, cmap="coolwarm_r").set_title('Todos
```

Out[53]:

```
Text(0.5, 1.0, 'Todos os Grupos')
```

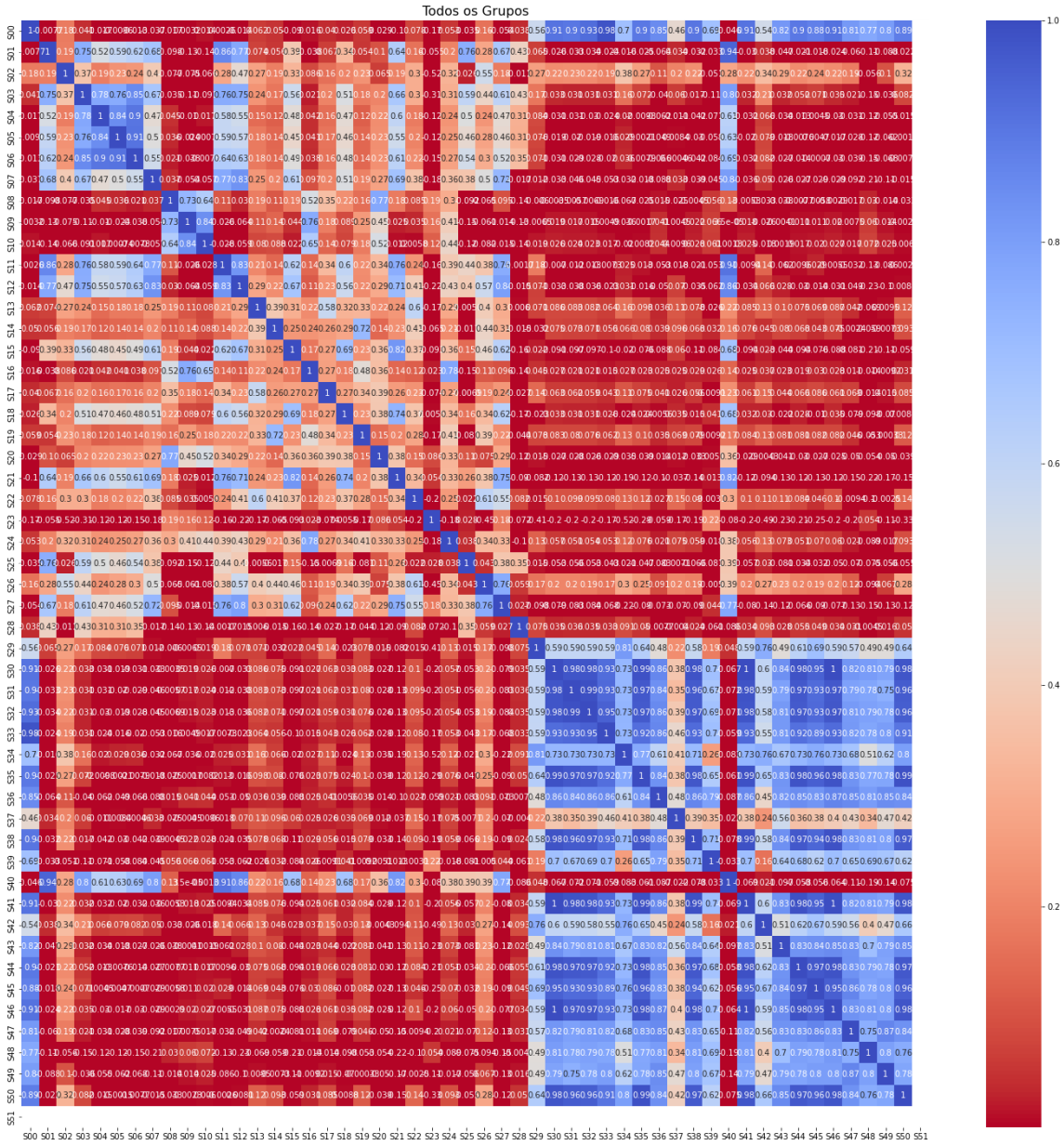


In [54]:

```
# Heatmap todo o conjunto de dados
plt.figure(figsize=(25,25))
sns.heatmap(df_new_tag.corr(),annot=True, vmin=.001, vmax=1, cmap="coolwarm_r").set_titl
```

Out[54]:

```
Text(0.5, 1.0, 'Todos os Grupos')
```



Machine Learning

K-Nearest Neighbors - (K-ésimo Vizinho mais Próximo)

In [55]:

```
pip install --upgrade pip
```

```
Requirement already satisfied: pip in c:\users\mecha\appdata\local\program
s\python\python39\lib\site-packages (23.0.1)
Note: you may need to restart the kernel to use updated packages.
```

In [56]:

```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\mecha\appdata\roam
ing\python\python39\site-packages (1.2.1)
Requirement already satisfied: scipy>=1.3.2 in c:\users\mecha\appdata\loca
l\programs\python\python39\lib\site-packages (from scikit-learn) (1.8.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\mecha\appdata\loc
al\programs\python\python39\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\mecha\appd
ata\local\programs\python\python39\lib\site-packages (from scikit-learn)
(3.1.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\mecha\appdata\loc
al\programs\python\python39\lib\site-packages (from scikit-learn) (1.22.3)
Note: you may need to restart the kernel to use updated packages.
```

In [57]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import preprocessing
%matplotlib inline
```

In [58]:

```
df_new_tag
```

Out[58]:

	S00	S01	S02	S03	S04	S05	S06	S07	
0	2.565284	2.465394	1.681353	13.41146	15.05353	15.56713	16.13136	31.11716	30.
1	2.565284	2.465394	1.681353	13.41146	15.05353	15.56713	16.13136	31.11716	30.
2	2.500062	2.444734	1.708474	13.32465	15.01013	15.61777	16.03733	32.08894	30.
3	2.509521	2.460474	1.579427	13.31742	15.08247	15.69734	16.24711	31.67221	30.
4	2.604785	2.445718	1.683831	13.35359	15.08247	15.69734	16.21094	31.95202	30.
...	...	...	...	...	...	...	...	...	
220315	2.499117	2.407350	13.265320	15.11863	15.16204	15.65393	16.65220	38.05424	30.
220316	2.618476	2.400463	13.242270	15.15480	15.11863	15.65393	16.70284	38.53485	30.
220317	2.620500	2.396528	13.188660	15.08970	15.11863	15.69734	16.70284	38.52678	29.
220318	2.514596	2.406366	13.173460	15.11863	15.11863	15.74074	16.56539	38.89159	29.
220319	2.487299	2.396528	13.125930	15.11863	15.01013	15.65393	16.65220	39.40957	29.

220320 rows × 54 columns

In [59]:

```
df_new_tag.fillna(value=df_mean_new_tag, inplace=True)
```

In [60]:

```
df_new_tag.isna().sum()
```

Out[60]:

```
S00                  0
S01                  0
S02                  0
S03                  0
S04                  0
S05                  0
S06                  0
S07                  0
S08                  0
S09                  0
S10                  0
S11                  0
S12                  0
S13                  0
S14                  0
S15                  0
S16                  0
S17                  0
S18                  0
S19                  0
S20                  0
S21                  0
S22                  0
S23                  0
S24                  0
S25                  0
S26                  0
S27                  0
S28                  0
S29                  0
S30                  0
S31                  0
S32                  0
S33                  0
S34                  0
S35                  0
S36                  0
S37                  0
S38                  0
S39                  0
S40                  0
S41                  0
S42                  0
S43                  0
S44                  0
S45                  0
S46                  0
S47                  0
S48                  0
S49                  0
S50                  0
S51             220320
machine_status       0
timestamp            0
dtype: int64
```

In [61]:

```
df_new_tag.drop(["S51"], axis=1, inplace=True)
```

In [62]:

```
df_new_tag.isna().sum()
```

Out[62]:

```
S00              0
S01              0
S02              0
S03              0
S04              0
S05              0
S06              0
S07              0
S08              0
S09              0
S10              0
S11              0
S12              0
S13              0
S14              0
S15              0
S16              0
S17              0
S18              0
S19              0
S20              0
S21              0
S22              0
S23              0
S24              0
S25              0
S26              0
S27              0
S28              0
S29              0
S30              0
S31              0
S32              0
S33              0
S34              0
S35              0
S36              0
S37              0
S38              0
S39              0
S40              0
S41              0
S42              0
S43              0
S44              0
S45              0
S46              0
S47              0
S48              0
S49              0
S50              0
machine_status   0
timestamp        0
dtype: int64
```

In [63]:

```
df_new_tag.isnull().sum()
```

Out[63]:

```
S00              0
S01              0
S02              0
S03              0
S04              0
S05              0
S06              0
S07              0
S08              0
S09              0
S10              0
S11              0
S12              0
S13              0
S14              0
S15              0
S16              0
S17              0
S18              0
S19              0
S20              0
S21              0
S22              0
S23              0
S24              0
S25              0
S26              0
S27              0
S28              0
S29              0
S30              0
S31              0
S32              0
S33              0
S34              0
S35              0
S36              0
S37              0
S38              0
S39              0
S40              0
S41              0
S42              0
S43              0
S44              0
S45              0
S46              0
S47              0
S48              0
S49              0
S50              0
machine_status   0
timestamp        0
dtype: int64
```

Feature set

In [64]:

```
y = df_new_tag['machine_status'].values
y[0:5]
```

Out[64]:

```
array(['NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL'], dtype=object)
```

In [65]:

```
X = df_new_tag[['S00','S01','S02','S03','S04','S05','S06','S07','S08','S09',
                'S10','S11','S12','S13','S14','S15','S16','S17','S18','S19',
                'S20','S21','S22','S23','S24','S25','S26','S27','S28','S29',
                'S30','S31','S32','S33','S34','S35','S36','S37','S38','S39',
                'S40','S41','S42','S43','S44','S45','S46','S47','S48','S49',
                'S50']].values  #.astype(float)
X[0:5]
```

Out[65]:

```
array([[  2.565284  ,   2.465394  ,   1.681353  ,  13.41146   ,
         15.05353   ,  15.56713   ,  16.13136   ,  31.11716   ,
         30.98958   ,  31.77083206,  31.51042   ,  37.2274    ,
         47.52422   ,  39.6412    ,  65.68287   ,  46.31076   ,
         41.92708   ,  38.19444   ,  47.09201   ,  50.92593   ,
         40.36458   ,  53.2118    ,  67.70834   ,  90.32386   ,
         70.57291   ,  76.45975   , 157.9861    , 243.0556    ,
        201.3889    , 171.9375    , 398.9862    , 419.5747    ,
        461.8781    , 466.3284    , 341.9039    , 498.8926    ,
        433.7037    , 429.0377    , 627.674     , 684.9443    ,
        634.375     , 665.3993    , 195.0655    , 594.4445    ,
        741.7151    , 848.0708    , 880.0001    , 680.4416    ,
        785.1935    , 682.8125    , 975.9409    ],
       [  2.565284  ,   2.465394  ,   1.681353  ,  13.41146   ,
         15.05353   ,  15.56713   ,  16.13136   ,  31.11716   ,
         30.98958   ,  31.77083206,  31.51042   ,  37.2274    ,
         47.52422   ,  39.6412    ,  65.68287   ,  46.31076   ,
         41.92708   ,  38.19444   ,  47.09201   ,  50.92593   ,
         40.36458   ,  53.2118    ,  67.70834   ,  90.32386   ,
         70.57291   ,  76.45975   , 157.9861    , 243.0556    ,
        201.3889    , 171.9375    , 398.9862    , 419.5747    ,
        461.8781    , 466.3284    , 341.9039    , 498.8926    ,
        433.7037    , 429.0377    , 627.674     , 684.9443    ,
        634.375     , 665.3993    , 195.0655    , 594.4445    ,
        741.7151    , 848.0708    , 880.0001    , 680.4416    ,
        785.1935    , 682.8125    , 975.9409    ],
       [  2.500062  ,   2.444734  ,   1.708474  ,  13.32465   ,
         15.01013   ,  15.61777   ,  16.03733   ,  32.08894   ,
         30.46875   ,  31.77083   ,  31.25      ,  37.86777   ,
         48.17723   ,  39.35185242,  65.39352   ,  46.39757   ,
         41.66666   ,  38.19444275,  47.35243   ,  51.21528   ,
         41.40625   ,  53.2118    ,  67.12963   ,  93.90508   ,
         69.53125   ,  73.54598   , 155.9606    , 241.3194    ,
        203.7037    , 169.982     , 399.9418    , 420.848     ,
        462.7798    , 459.6364    , 343.1955    , 501.3617    ,
        441.2635    , 454.239     , 631.1326    , 715.6266    ,
        638.8889    , 666.2234    , 200.9694    , 661.574     ,
        740.8031    , 849.8997    , 880.4237    , 694.7721    ,
        778.5734    , 721.875     , 982.7342    ],
       [  2.509521  ,   2.460474  ,   1.579427  ,  13.31742   ,
         15.08247   ,  15.69734   ,  16.24711   ,  31.67221   ,
         30.46875   ,  31.51042   ,  31.51042   ,  38.57977   ,
         48.65607   ,  39.0625    ,  64.81481   ,  46.39756775,
         40.88541   ,  38.19444   ,  47.09201   ,  51.21528   ,
         41.92708   ,  53.1684    ,  66.84028   , 101.0406    ,
         72.13541   ,  76.98898   , 155.9606    , 240.4514    ,
        203.125     , 166.4987    , 399.1046    , 420.7494    ,
        462.898     , 460.8858    , 343.9586    , 499.043     ,
        446.2493    , 474.8731    , 625.4076    , 690.4011    ,
        628.125     , 666.0114    , 193.1689    , 686.1111    ,
        739.2722    , 847.7579    , 878.8917    , 683.3831    ,
        779.5091    , 754.6875    , 977.752     ],
       [  2.604785  ,   2.445718  ,   1.683831  ,  13.35359   ,
         15.08247   ,  15.69734   ,  16.21094   ,  31.95202   ,
         30.98958   ,  31.51042   ,  31.51042   ,  39.48939   ,
         49.06298   ,  38.77315   ,  65.10416   ,  46.39756775,
         41.40625   ,  38.77315   ,  47.13541   ,  51.79398   ,
         42.70833   ,  53.2118    ,  66.55093   , 101.7038    ,
         76.82291   ,  76.58897   , 158.2755    , 242.1875    ,
        201.3889    , 164.7498    , 400.5426    , 419.8926    ,
        461.4906    , 468.2206    , 339.963     , 498.5383    ,
```

```
       433.9081    , 408.8159    , 627.183     , 704.6937    ,
       636.4583    , 663.2111    , 193.877     , 631.4814    ,
       737.6033    , 846.9182    , 882.5874    , 702.4431    ,
       785.2307    , 766.1458    , 979.5755    ]])
```

Normalize Data

In [66]:

```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

Out[66]:

```
array([[ 0.34177145,  0.31838419, -0.78196435,  0.06876917,  0.19013546,
         0.23609198,  0.2071035 ,  0.19597424, -0.5527738 , -0.35828653,
        -0.326262  , -0.35039048,  0.42946569, -0.25985563,  1.75883723,
         1.04471876, -0.17614398, -0.58747931, -0.10997716,  0.18615638,
        -0.87979269,  0.63536701,  0.55320239,  0.78551632,  0.08122195,
         0.17732563,  0.08650387,  1.21684351,  0.10913687, -0.71290266,
         0.37452016,  0.37745365,  0.36036568,  0.35030044, -0.6007524 ,
         0.25339575, -0.34924378, -0.42639864,  0.39198245,  0.48209432,
         0.30357664,  0.37418153, -1.37491358, -0.09872644,  0.41934555,
         0.25015803,  0.36967506, -0.47364261, -0.21215345, -0.63621072,
         0.18291591],
       [ 0.34177145,  0.31838419, -0.78196435,  0.06876917,  0.19013546,
         0.23609198,  0.2071035 ,  0.19597424, -0.5527738 , -0.35828653,
        -0.326262  , -0.35039048,  0.42946569, -0.25985563,  1.75883723,
         1.04471876, -0.17614398, -0.58747931, -0.10997716,  0.18615638,
        -0.87979269,  0.63536701,  0.55320239,  0.78551632,  0.08122195,
         0.17732563,  0.08650387,  1.21684351,  0.10913687, -0.71290266,
         0.37452016,  0.37745365,  0.36036568,  0.35030044, -0.6007524 ,
         0.25339575, -0.34924378, -0.42639864,  0.39198245,  0.48209432,
         0.30357664,  0.37418153, -1.37491358, -0.09872644,  0.41934555,
         0.25015803,  0.36967506, -0.47364261, -0.21215345, -0.63621072,
         0.18291591],
       [ 0.25666056,  0.28446345, -0.77803476,  0.03866508,  0.17536803,
         0.2528373 ,  0.17824586,  0.29202801, -0.61863661, -0.35828673,
        -0.34293638, -0.29746308,  0.47946046, -0.28483133,  1.73631183,
         1.08011718, -0.1997022 , -0.58747904, -0.04182765,  0.20464608,
        -0.78109327,  0.63536701,  0.52298689,  0.88074376,  0.0325082 ,
         0.00900514,  0.06187971,  1.19979497,  0.12881455, -0.73502488,
         0.38388728,  0.38869588,  0.36751295,  0.29853899, -0.59164467,
         0.26936245, -0.29911345, -0.27804255,  0.41094901,  0.61787678,
         0.33489669,  0.3783144 , -1.3545144 ,  0.24246831,  0.4152189 ,
         0.25756952,  0.37154302, -0.41872468, -0.23329406, -0.49848688,
         0.20618623],
       [ 0.269004  ,  0.31030626, -0.79673252,  0.03615785,  0.19998268,
         0.27914902,  0.24262697,  0.25083712, -0.61863661, -0.38365216,
        -0.326262  , -0.23861538,  0.51612068, -0.30980744,  1.69126027,
         1.08011627, -0.27037598, -0.58747931, -0.10997716,  0.20464608,
        -0.73174404,  0.6236269 ,  0.5078794 ,  1.07048284,  0.15429305,
         0.20789779,  0.06187971,  1.19127168,  0.12389513, -0.77443083,
         0.37568076,  0.38782532,  0.36844985,  0.30820288, -0.58626367,
         0.25436833, -0.26605175, -0.15657285,  0.37955377,  0.50624302,
         0.26021051,  0.37725122, -1.38146672,  0.36718144,  0.40829182,
         0.24889002,  0.36478733, -0.46237007, -0.230306  , -0.38279886,
         0.1891198 ],
       [ 0.39331795,  0.28607903, -0.78160531,  0.04870093,  0.19998268,
         0.27914902,  0.23152646,  0.27849441, -0.5527738 , -0.38365216,
        -0.326262  , -0.16343413,  0.54727391, -0.33478334,  1.71378566,
         1.08011627, -0.22325953, -0.53211741, -0.09861978,  0.24162547,
        -0.65771972,  0.63536701,  0.49277191,  1.08811785,  0.37350634,
         0.18479031,  0.09002213,  1.20831924,  0.10913687, -0.79421582,
         0.38977653,  0.38026045,  0.35729418,  0.36493627, -0.61443866,
         0.25110463, -0.34788837, -0.54544119,  0.38928986,  0.56949396,
         0.31803178,  0.3632077 , -1.37902009,  0.08951861,  0.40074031,
         0.24548719,  0.38108432, -0.38932757, -0.21203465, -0.34239998,
         0.19536617]])
```

Train Test Split

In [67]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (176256, 51) (176256,)
Test set: (44064, 51) (44064,)
```

Classification

K nearest neighbor (KNN)

Import library

In [68]:

```
from sklearn.neighbors import KNeighborsClassifier
```

Training

In [69]:

```
k = 5
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[69]:

```
KNeighborsClassifier
KNeighborsClassifier()
```

Predicting

In [70]:

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[70]:

```
array(['NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL'], dtype=object)
```

Accuracy evaluation

In [71]:

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.9998978758169934
Test set Accuracy:  0.9997957516339869
```

Practice

In [73]:

```
k = 6
neigh6 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat6 = neigh6.predict(X_test)
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh6.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat6))
```

```
Train set Accuracy:  0.9998751815541032
Test set Accuracy:  0.9997730573710966
```

In [74]:

```
Ks = 17
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

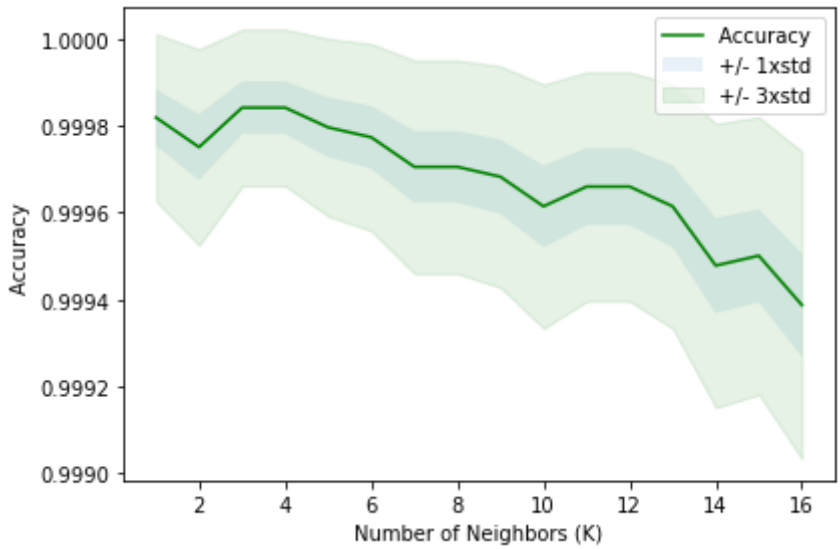
Out[74]:

```
array([0.99981845, 0.99975036, 0.99984114, 0.99984114, 0.99979575,
       0.99977306, 0.99970497, 0.99970497, 0.99968228, 0.9996142 ,
       0.99965959, 0.99965959, 0.9996142 , 0.99947803, 0.99950073,
       0.99938725])
```

Plot the model accuracy for a different number of neighbors.

In [75]:

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.10,c
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [76]:

```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.9998411401597677 with k= 3
```

In [ ]:

```

```