1. Exploração Preliminar dos Dados

```
# Importando bibliotecas necessárias

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()

from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
from lifelines import KaplanMeierFitter, CoxTimeVaryingFitter

# Apenas para ignorar os alertas do python
import warnings
warnings.filterwarnings("ignore")

# Definindo caminho dos dados
dir_path = './CMAPSSData/'

# Definindo o nome das colunas para facilitar a exploração dos dados
index_names = ['unidade', 'ciclo_tempo']
setting_names = ['config_1', 'config_2', 'config_3']
sensor_names = ['s_{}'.format(i) for i in range(1,22)]
col_names = index_names + setting_names + sensor_names

# Lendo os dados
train = pd.read_csv((dir_path+'train_FD001.txt'), sep='\s+',
header=None, names=col_names)
test = pd.read_csv((dir_path+'test_FD001.txt'), sep='\s+',
header=None, names=col_names)
y_test = pd.read_csv((dir_path+'RUL_FD001.txt'), sep='\s+',
header=None, names=['RUL'])

# Analisar as primeiras linhas da nossa base de dados
train.head()
```

```
   unidade  ciclo_tempo  config_1  config_2  config_3     s_1     s_2
\
0        1            1   -0.0007   -0.0004     100.0  518.67  641.82

1        1            2    0.0019   -0.0003     100.0  518.67  642.15

2        1            3   -0.0043    0.0003     100.0  518.67  642.35

3        1            4    0.0007    0.0000     100.0  518.67  642.35

4        1            5   -0.0019   -0.0002     100.0  518.67  642.37
```

```
        s_3      s_4    s_5  ...    s_12     s_13     s_14    s_15
s_16  s_17  \
0  1589.70  1400.60  14.62  ...  521.66  2388.02  8138.62  8.4195
0.03   392
1  1591.82  1403.14  14.62  ...  522.28  2388.07  8131.49  8.4318
0.03   392
2  1587.99  1404.20  14.62  ...  522.42  2388.03  8133.23  8.4178
0.03   390
3  1582.79  1401.87  14.62  ...  522.86  2388.08  8133.83  8.3682
0.03   392
4  1582.85  1406.22  14.62  ...  522.19  2388.04  8133.80  8.4294
0.03   393

   s_18   s_19   s_20     s_21
0  2388  100.0  39.06  23.4190
1  2388  100.0  39.00  23.4236
2  2388  100.0  38.95  23.3442
3  2388  100.0  38.88  23.3739
4  2388  100.0  38.90  23.4044

[5 rows x 26 columns]
```

```
# Analisar o conjunto de treino
train[index_names].describe()
```

```
            unidade   ciclo_tempo
count  20631.000000  20631.000000
mean      51.506568    108.807862
std       29.227633     68.880990
min        1.000000      1.000000
25%       26.000000     52.000000
50%       52.000000    104.000000
75%       77.000000    156.000000
max      100.000000    362.000000
```

```
# Analisar os ciclos temporais
train[index_names].groupby('unidade').max().describe()
```

```
       ciclo_tempo
count   100.000000
mean    206.310000
std      46.342749
min     128.000000
25%     177.000000
50%     199.000000
75%     229.250000
max     362.000000
```

```
# Analisar as configuracoes de operacao
train[setting_names].describe()
```

```
           config_1      config_2  config_3
count  20631.000000  20631.000000   20631.0
mean      -0.000009      0.000002     100.0
std        0.002187      0.000293       0.0
min       -0.008700     -0.000600     100.0
25%       -0.001500     -0.000200     100.0
50%        0.000000      0.000000     100.0
75%        0.001500      0.000300     100.0
max        0.008700      0.000600     100.0
```

```
# Analisar os valores dos sinais
train[sensor_names].describe().transpose()
```

```
         count         mean           std        min        25%
50%  \
s_1   20631.0   518.670000  6.537152e-11   518.6700   518.6700
518.6700
s_2   20631.0   642.680934  5.000533e-01   641.2100   642.3250
642.6400
s_3   20631.0  1590.523119  6.131150e+00  1571.0400  1586.2600
1590.1000
s_4   20631.0  1408.933782  9.000605e+00  1382.2500  1402.3600
1408.0400
s_5   20631.0    14.620000  3.394700e-12    14.6200    14.6200
14.6200
s_6   20631.0    21.609803  1.388985e-03    21.6000    21.6100
21.6100
s_7   20631.0   553.367711  8.850923e-01   549.8500   552.8100
553.4400
s_8   20631.0  2388.096652  7.098548e-02  2387.9000  2388.0500
2388.0900
s_9   20631.0  9065.242941  2.208288e+01  9021.7300  9053.1000
9060.6600
s_10  20631.0     1.300000  4.660829e-13     1.3000     1.3000
1.3000
s_11  20631.0    47.541168  2.670874e-01    46.8500    47.3500
47.5100
s_12  20631.0   521.413470  7.375534e-01   518.6900   520.9600
521.4800
s_13  20631.0  2388.096152  7.191892e-02  2387.8800  2388.0400
2388.0900
s_14  20631.0  8143.752722  1.907618e+01  8099.9400  8133.2450
8140.5400
s_15  20631.0     8.442146  3.750504e-02     8.3249     8.4149
8.4389
s_16  20631.0     0.030000  1.556432e-14     0.0300     0.0300
0.0300
```

```
s_17  20631.0   393.210654  1.548763e+00   388.0000   392.0000
393.0000
s_18  20631.0  2388.000000  0.000000e+00  2388.0000  2388.0000
2388.0000
s_19  20631.0   100.000000  0.000000e+00   100.0000   100.0000
100.0000
s_20  20631.0    38.816271  1.807464e-01    38.1400    38.7000
38.8300
s_21  20631.0    23.289705  1.082509e-01    22.8942    23.2218
23.2979

            75%        max
s_1    518.6700   518.6700
s_2    643.0000   644.5300
s_3   1594.3800  1616.9100
s_4   1414.5550  1441.4900
s_5     14.6200    14.6200
s_6     21.6100    21.6100
s_7    554.0100   556.0600
s_8   2388.1400  2388.5600
s_9   9069.4200  9244.5900
s_10     1.3000     1.3000
s_11    47.7000    48.5300
s_12   521.9500   523.3800
s_13  2388.1400  2388.5600
s_14  8148.3100  8293.7200
s_15     8.4656     8.5848
s_16     0.0300     0.0300
s_17   394.0000   400.0000
s_18  2388.0000  2388.0000
s_19   100.0000   100.0000
s_20    38.9500    39.4300
s_21    23.3668    23.6184
```

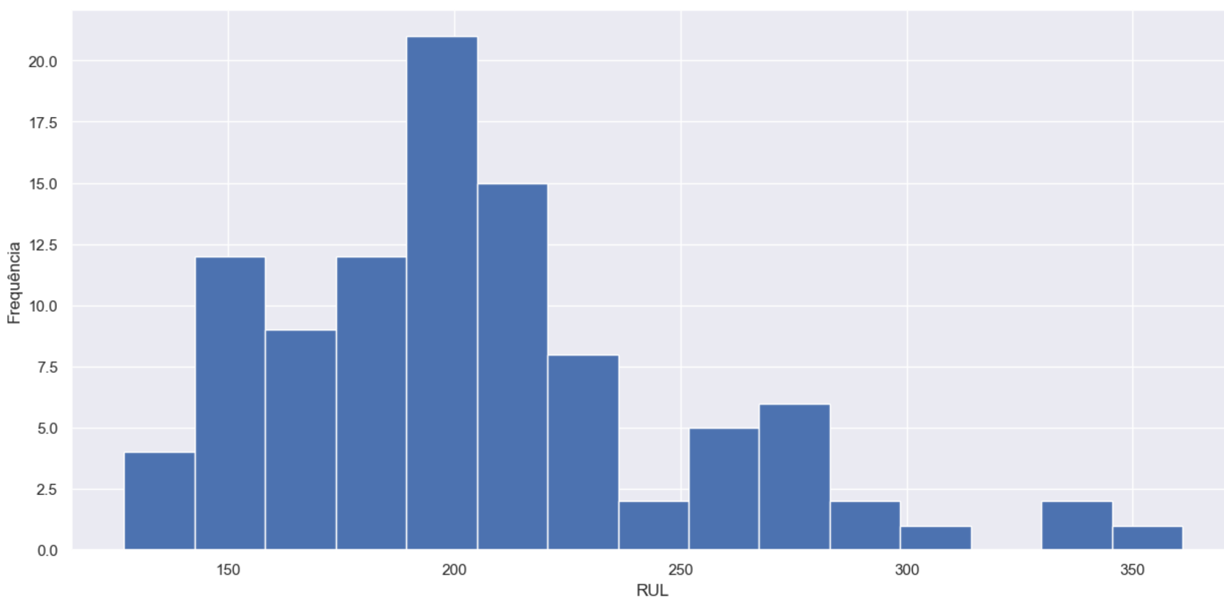
2. Computando o RUL

```
def add_RUL(df):
    # Obter o numero total de ciclos para cada unidade
    grupo_unidade = df.groupby(by="unidade")
    ciclo_max = grupo_unidade["ciclo_tempo"].max()

    # Mesclar o valor do ciclo maximo no dataframe de origem
    df_resultado = df.merge(ciclo_max.to_frame(name='ciclo_max'),
left_on='unidade', right_index=True)

    # Calcular o RUL para cada linha
    vida_restante = df_resultado["ciclo_max"] -
df_resultado["ciclo_tempo"]
```

```
    df_resultado["RUL"] = vida_restante

    # Remover o valor do ciclo maximo, que nao e mais necessario
    df_resultado = df_resultado.drop("ciclo_max", axis=1)
    return df_resultado

train = add_RUL(train)
train[index_names+['RUL']].head()
```

	unidade	ciclo_tempo	RUL
0	1	1	191
1	1	2	190
2	1	3	189
3	1	4	188
4	1	5	187

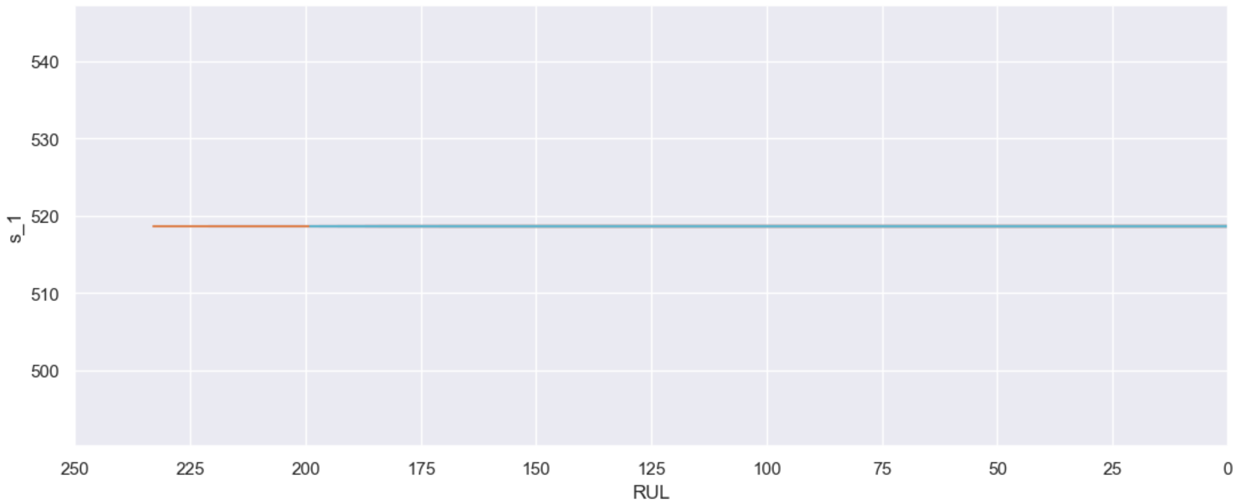
```
# Distribuicao do RUL. Similar a funcao descricao feita acima na
coluna ciclo_tempo, mas visual
df_max_rul = train[['unidade',
'RUL']].groupby('unidade').max().reset_index()
df_max_rul['RUL'].hist(bins=15, figsize=(15,7))
plt.xlabel('RUL')
plt.ylabel('Frequência')
plt.show()
```

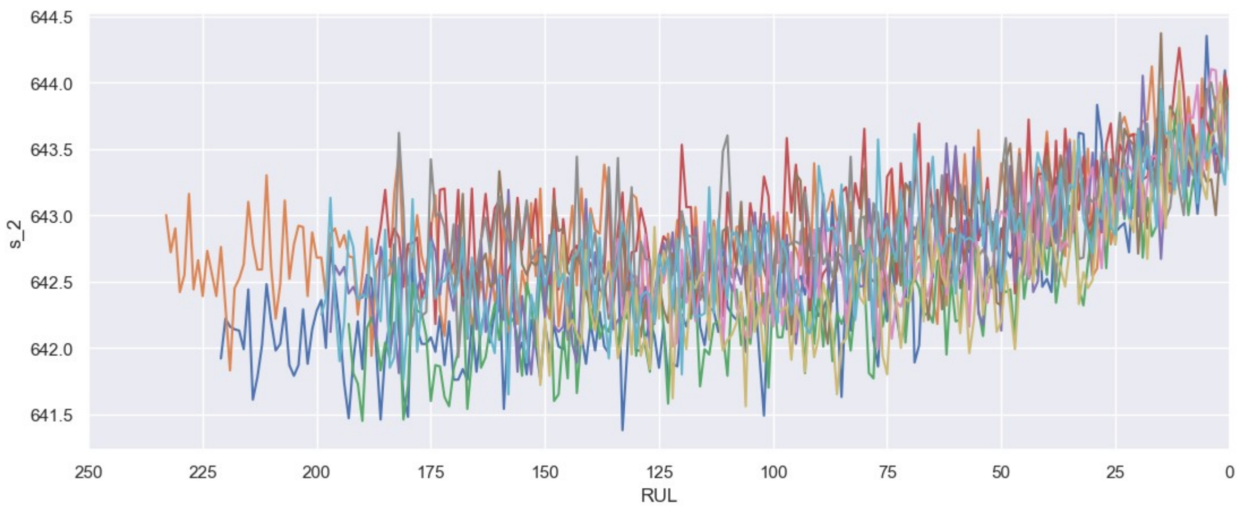


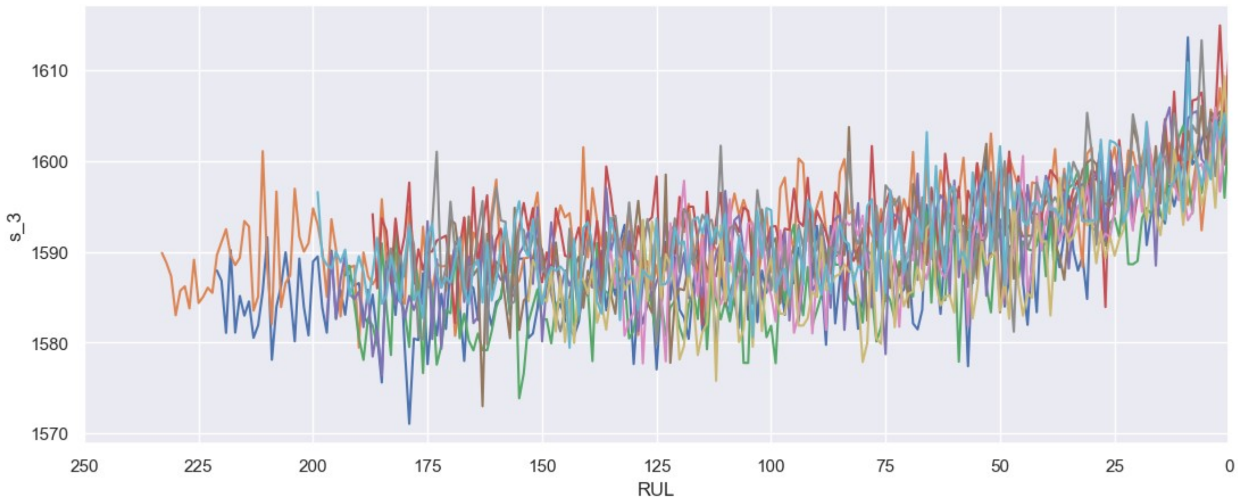
3. Grafico dos sensores

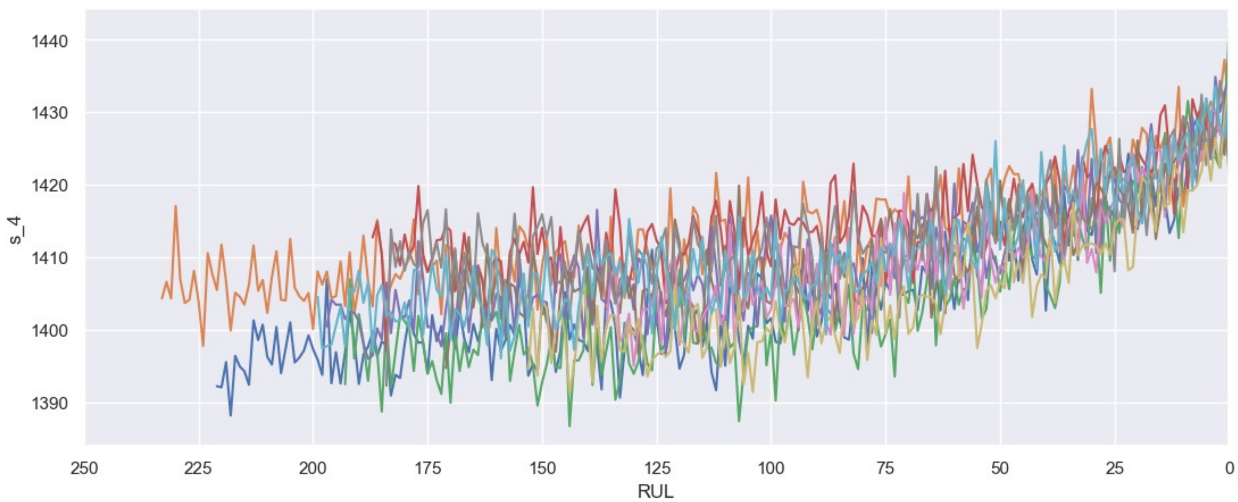
```
def plot_sinal(sensor):
    plt.figure(figsize=(13,5))
```

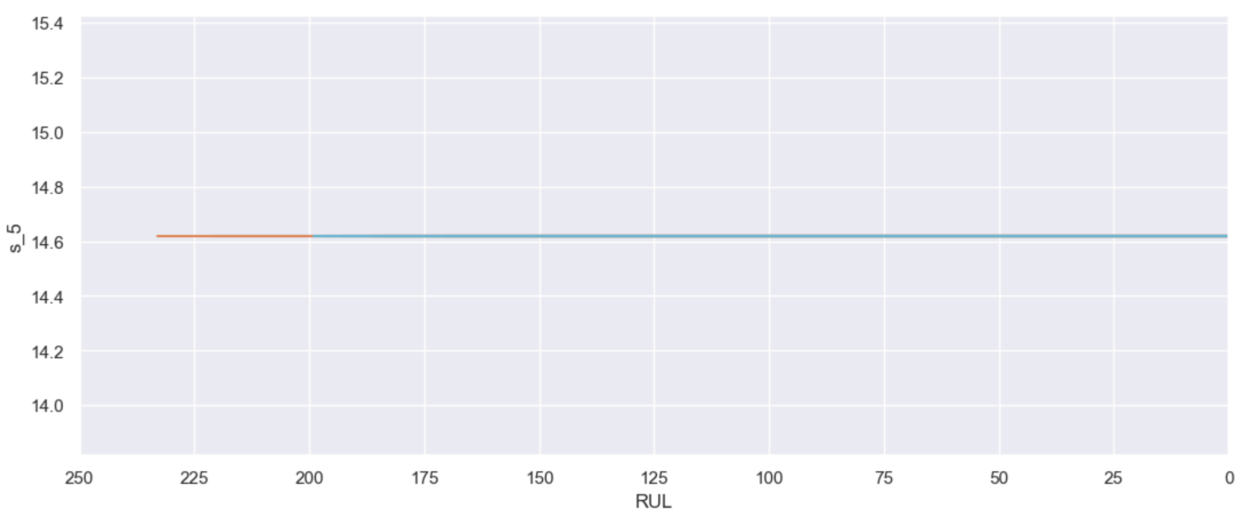
```
    for i in train['unidade'].unique():
        if (i % 10 == 0):  # grafico a cada 10 unidades
            plt.plot('RUL', sensor,
                     data=train[train['unidade']==i])
    plt.xlim(250, 0)  # reverte o eixo x para que o RUL va do maximo
ate zero
    plt.xticks(np.arange(0, 275, 25))
    plt.ylabel(sensor)
    plt.xlabel('RUL')
    plt.show()

for sensor in sensor_names:
    plot_sinal(sensor)
```

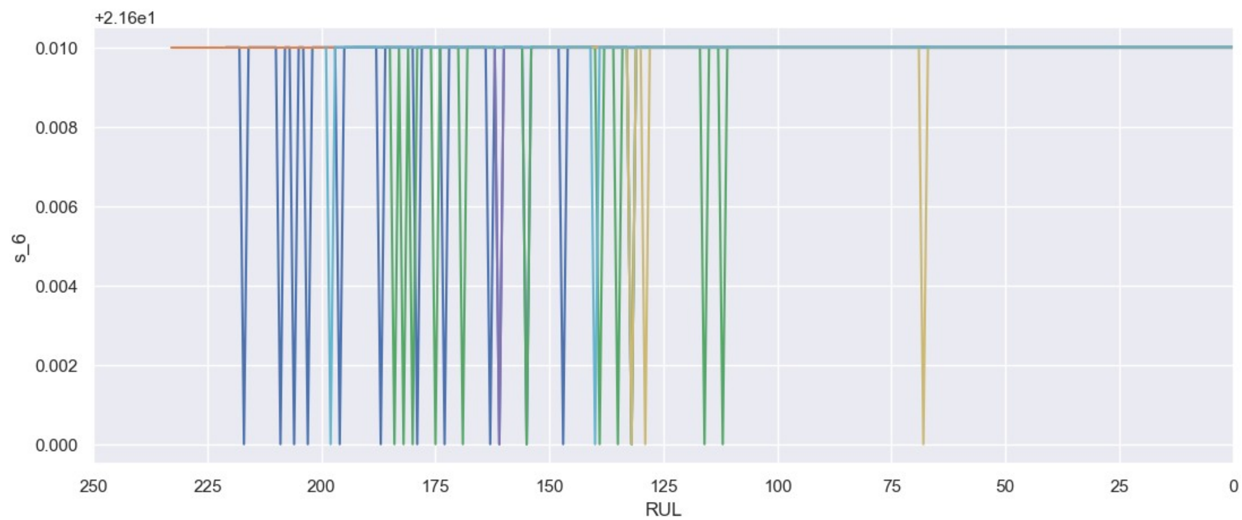


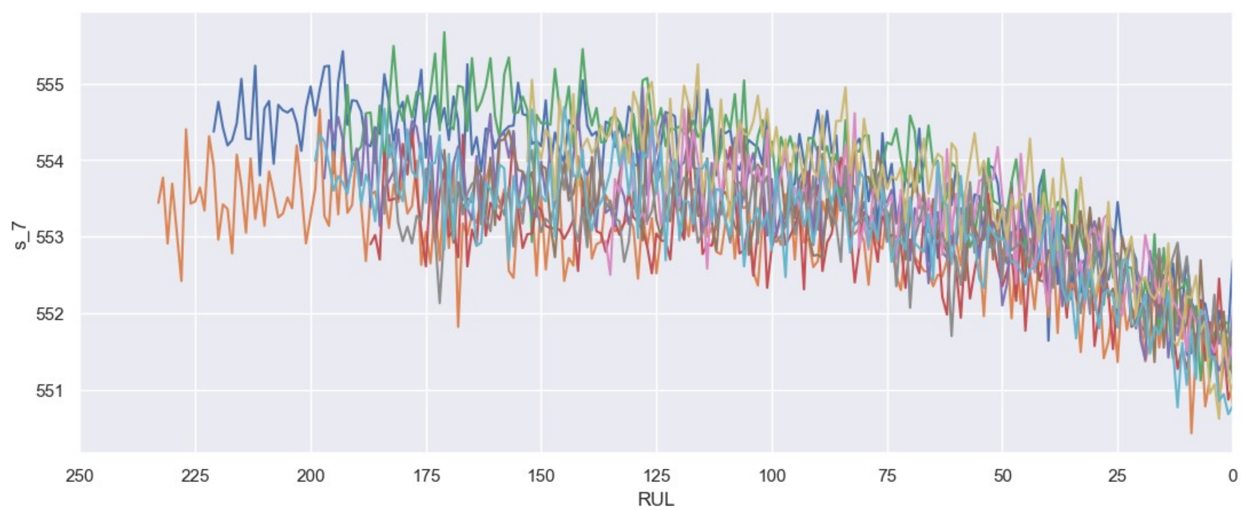


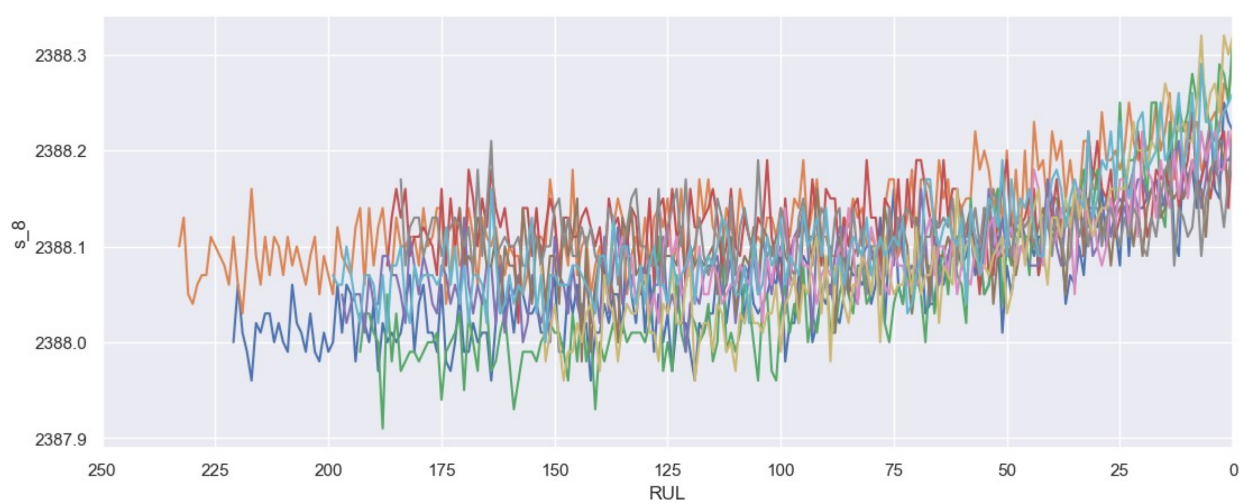

s_3
1610
1600
1590
1580
1570
250
225
200
175
150
125
100
75
50
25
0
RUL

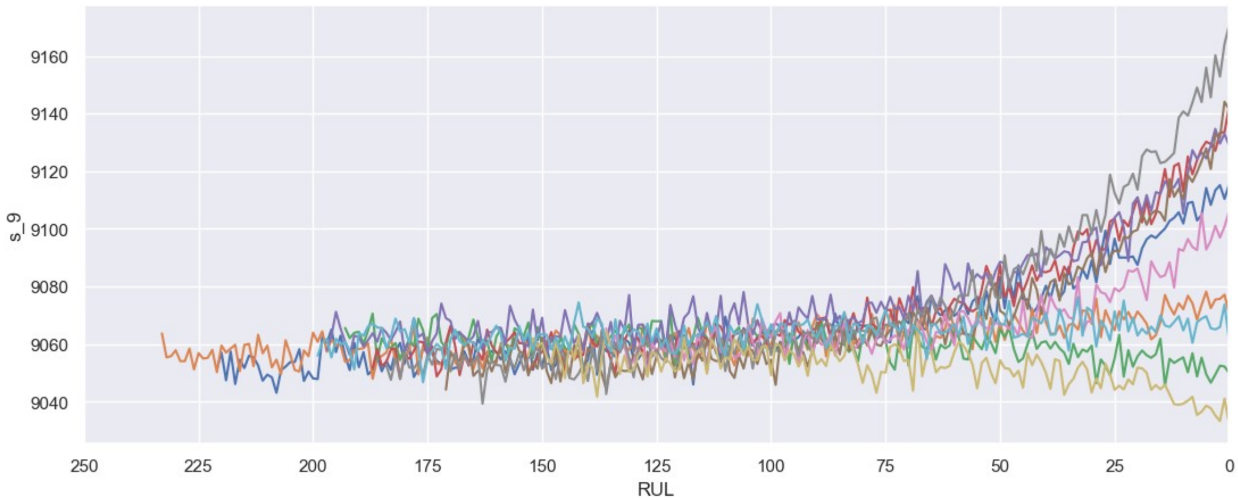

s_4
1440
1430
1420
1410
1400
1390
250
225
200
175
150
125
100
75
50
25
0
RUL


s_5
15.4
15.2
15.0
14.8
14.6
14.4
14.2
14.0
250
225
200
175
150
125
100
75
50
25
0
RUL

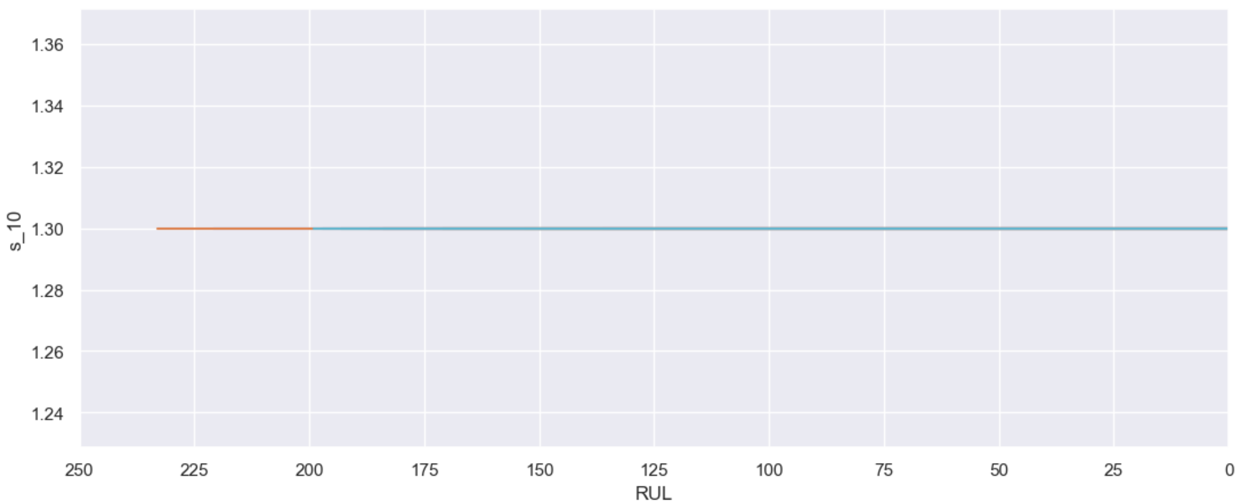

+2.16e1
0.010
0.008
0.006
0.004
0.002
0.000
s_6
250
225
200
175
150
125
100
75
50
25
0
RUL


555
554
553
552
551
s_7
250
225
200
175
150
125
100
75
50
25
0
RUL

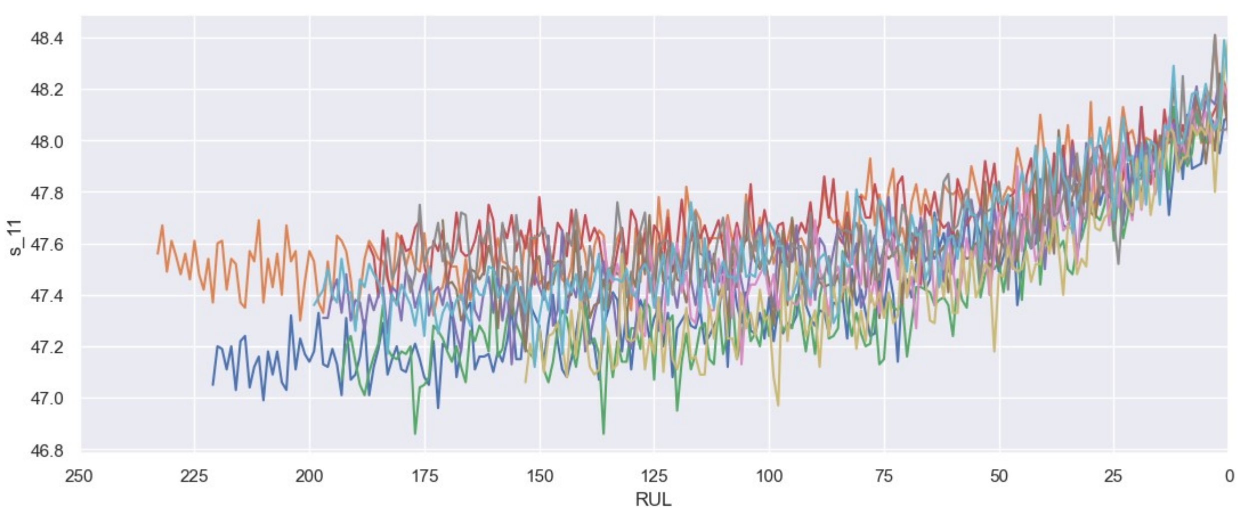

2388.3
2388.2
2388.1
2388.0
2387.9
s_8
250
225
200
175
150
125
100
75
50
25
0
RUL



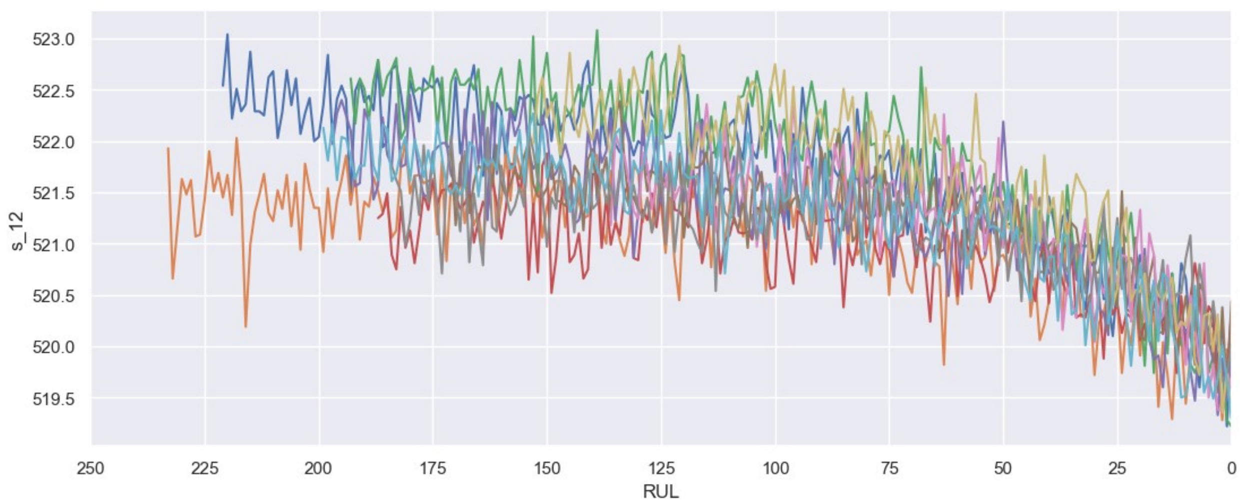
9160
9140
9120
9100
9080
9060
9040
s_9
250
225
200
175
150
125
100
75
50
25
0
RUL



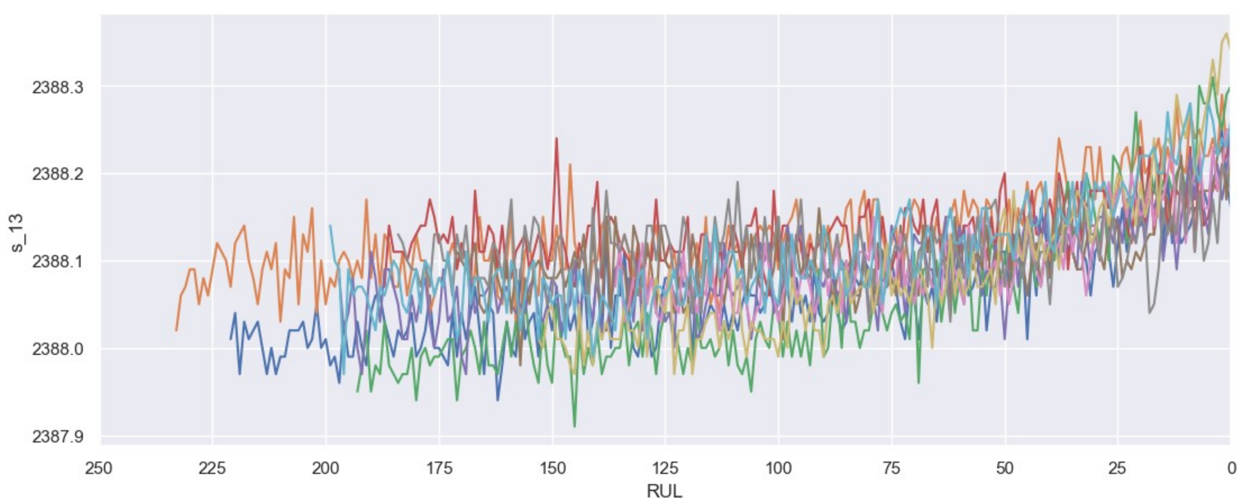
1.36
1.34
1.32
1.30
1.28
1.26
1.24
s_10
250
225
200
175
150
125
100
75
50
25
0
RUL



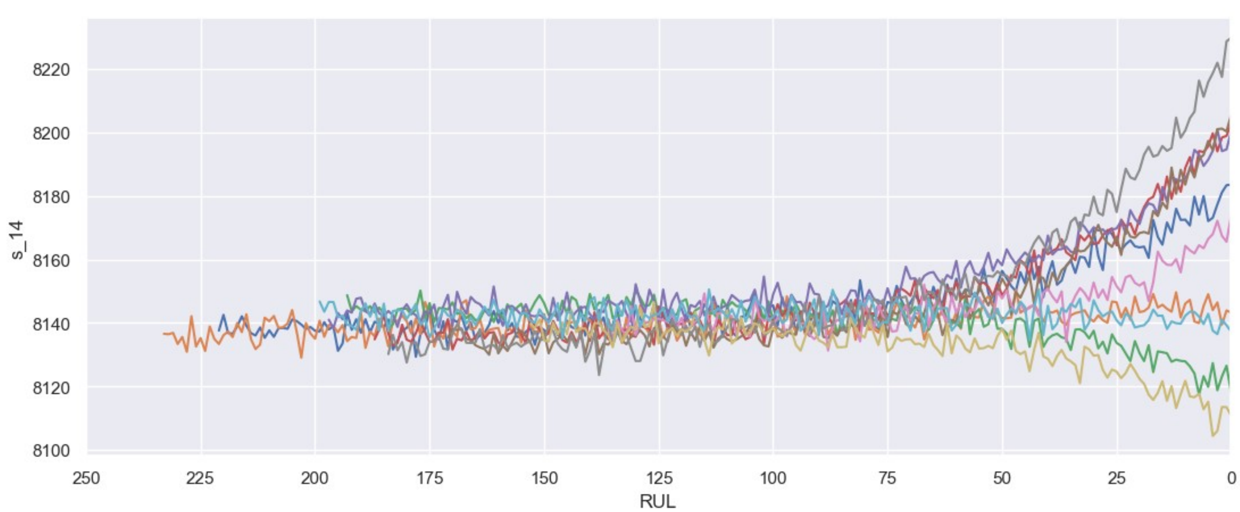
48.4
48.2
48.0
47.8
47.6
47.4
47.2
47.0
46.8
s_11
250
225
200
175
150
125
100
75
50
25
0
RUL



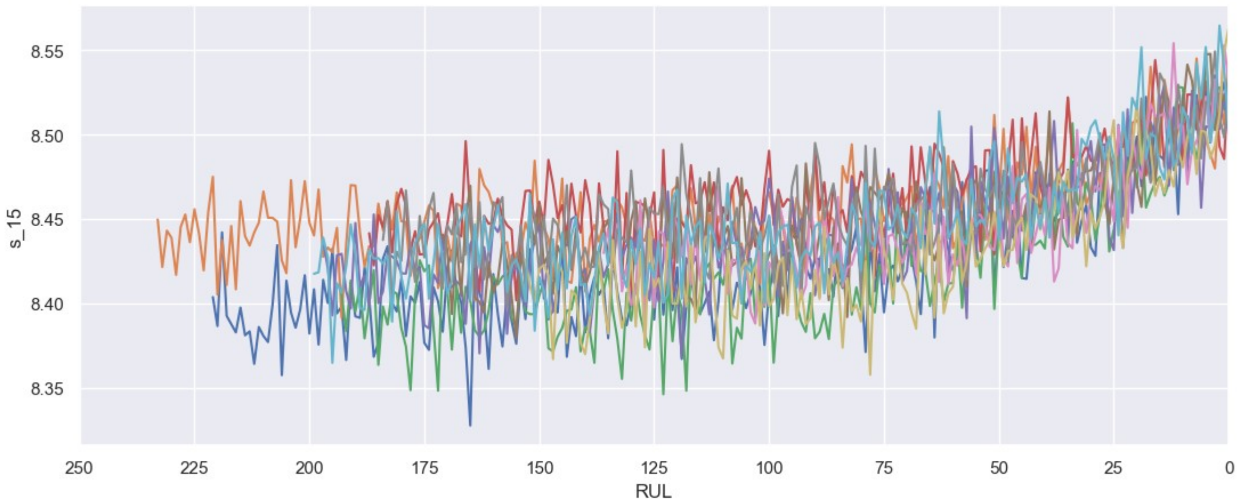
523.0
522.5
522.0
521.5
521.0
520.5
520.0
519.5
s_12
250
225
200
175
150
125
100
75
50
25
0
RUL



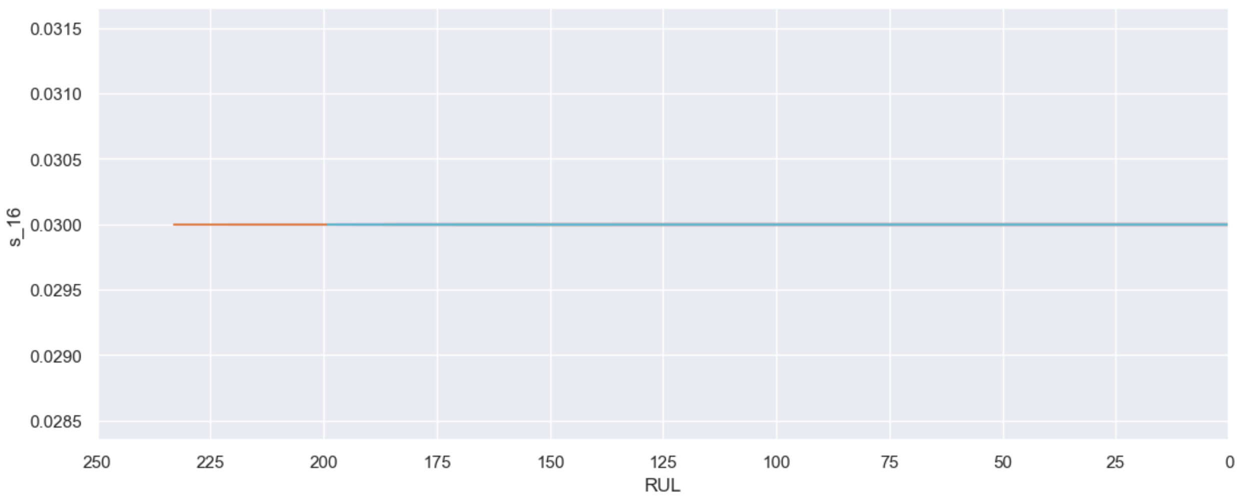
2388.3
2388.2
2388.1
2388.0
2387.9
s_13
250
225
200
175
150
125
100
75
50
25
0
RUL



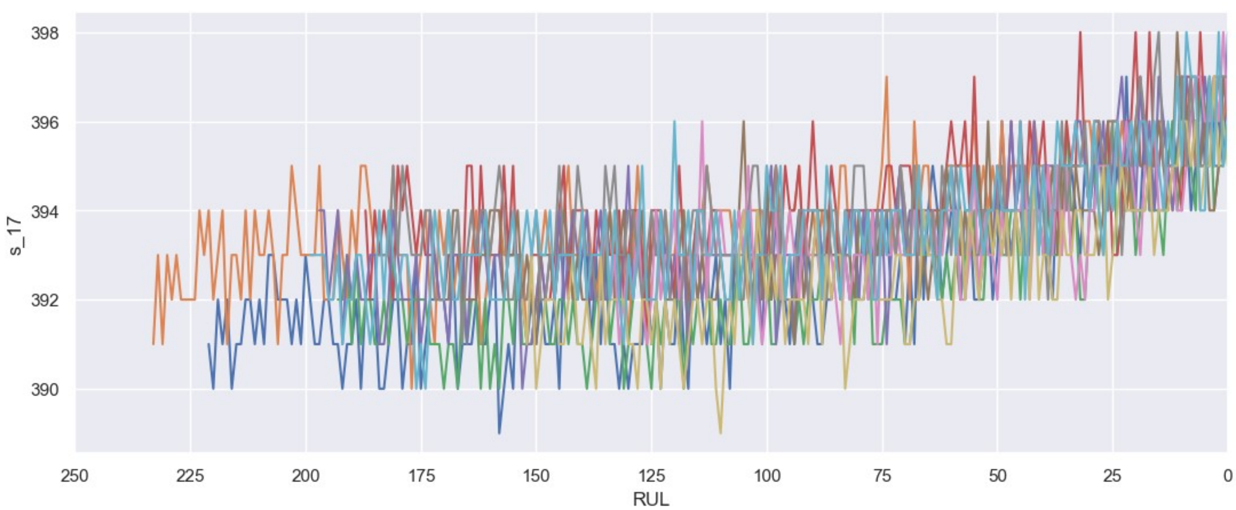
8220
8200
8180
8160
8140
8120
8100
s_14
250
225
200
175
150
125
100
75
50
25
0
RUL



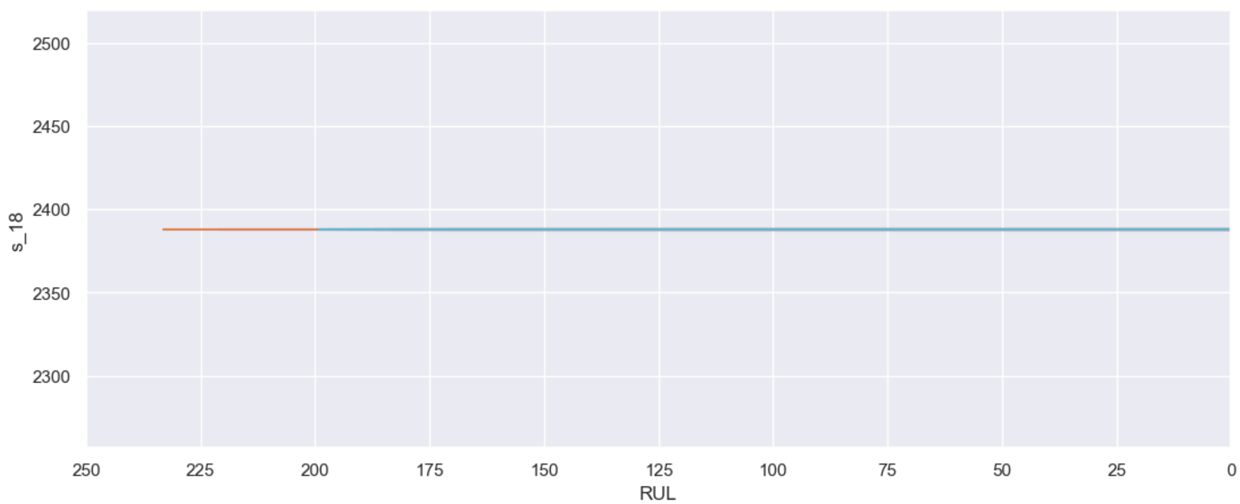
s_15
RUL



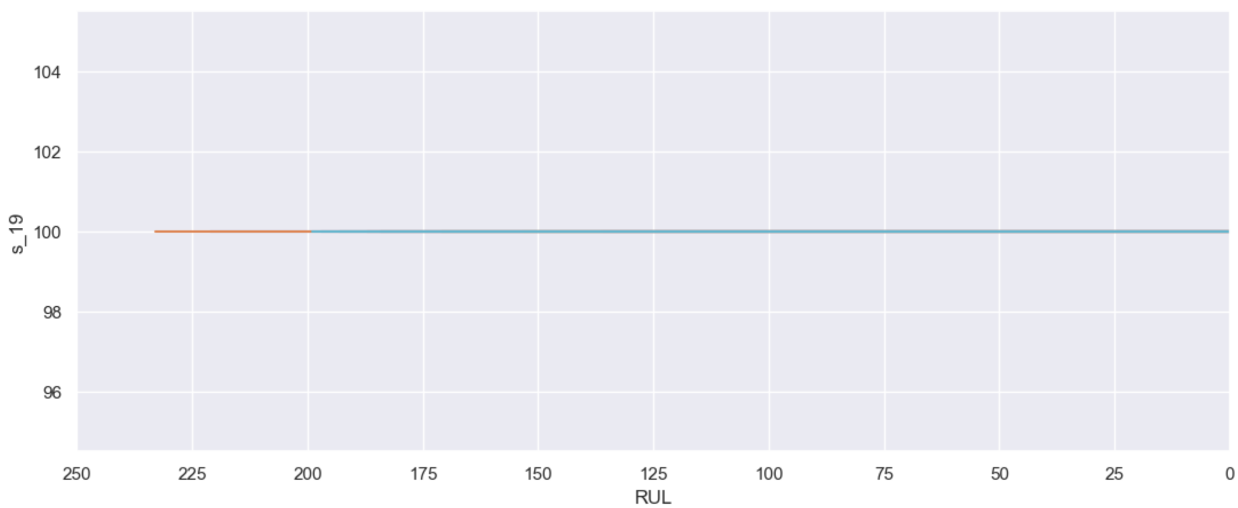
s_16
RUL



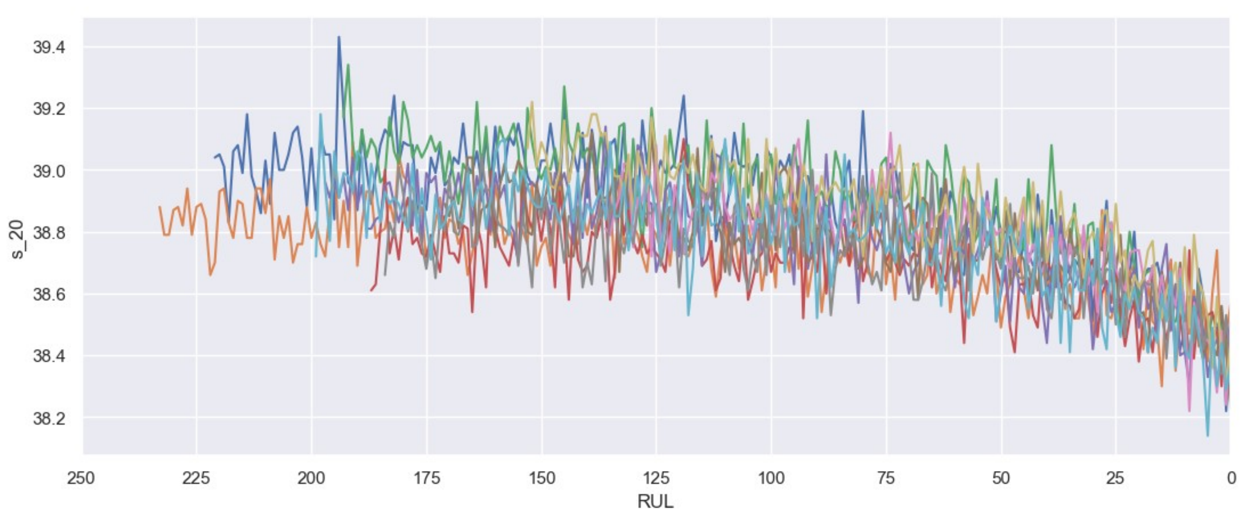
s_17
RUL



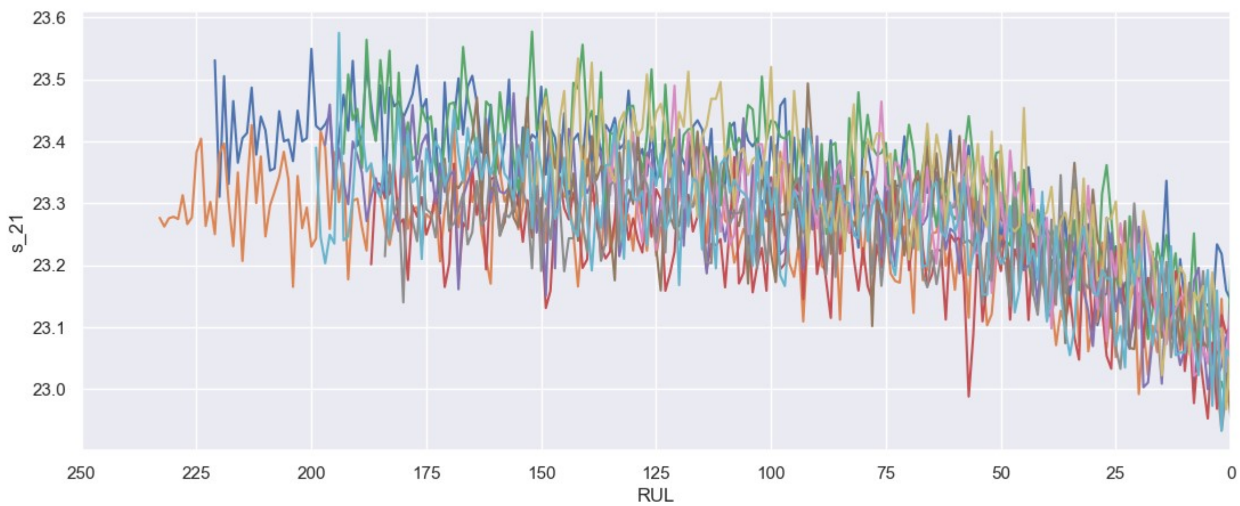
2500
2450
2400
2350
2300
s_18
250
225
200
175
150
125
100
75
50
25
0
RUL



104
102
100
98
96
s_19
250
225
200
175
150
125
100
75
50
25
0
RUL



39.4
39.2
39.0
38.8
38.6
38.4
38.2
s_20
250
225
200
175
150
125
100
75
50
25
0
RUL



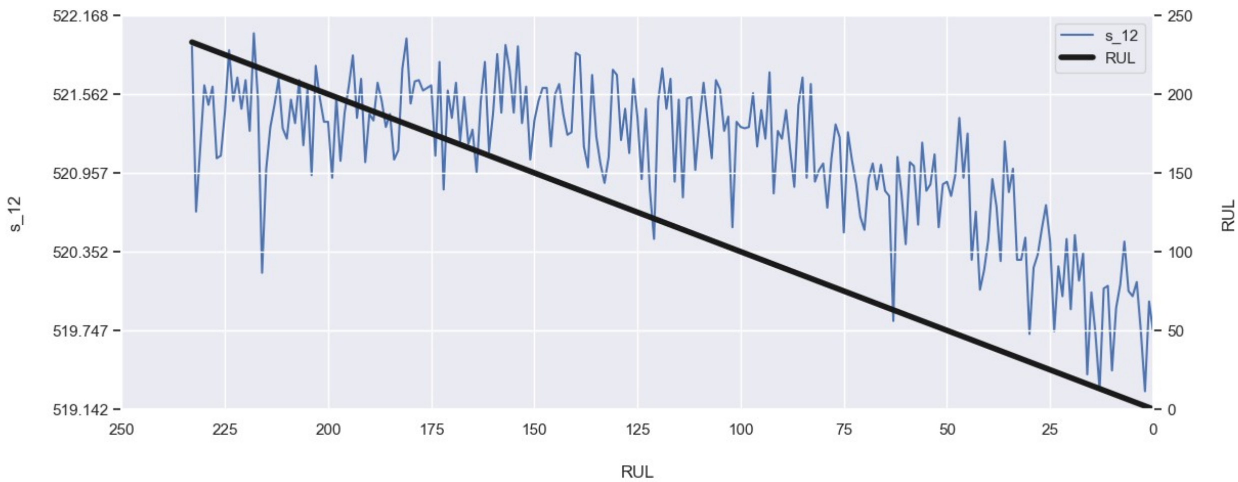
4. Abordagem de Referência: Regressão Linear

```
# Criar a funcao de avaliacao
def avaliar(y_verdadeiro, y_calculado, label='teste'):
    mse = mean_squared_error(y_verdadeiro, y_calculado)
    rmse = np.sqrt(mse)
    variancia = r2_score(y_verdadeiro, y_calculado)
    print('conjunto de {} -> RMSE:{}, R2:{}'.format(label, rmse,
variancia))

# Preparando os dados

# Removendo os sensores desnecessarios
drop_sensors = ['s_1','s_5','s_6','s_10','s_16','s_18','s_19']
drop_labels = index_names+setting_names+drop_sensors

# Separar os valores de entrada e saida do modelo
X_train = train.drop(drop_labels, axis=1)
y_train = X_train.pop('RUL')

# Como os valores reais para o RUL no conjunto de treino são
fornecidos apenas para o último ciclo de tempo de cada motor,
# o conjunto de teste é subdividido para representar o mesmo
X_test =
test.groupby('unidade').last().reset_index().drop(drop_labels, axis=1)

print(X_train.columns)  # Verificar as colunas restantes

Index(['s_2', 's_3', 's_4', 's_7', 's_8', 's_9', 's_11', 's_12',
's_13',
       's_14', 's_15', 's_17', 's_20', 's_21'],
      dtype='object')
```

```
# Criar e treinar o modelo
lm = LinearRegression()
lm.fit(X_train, y_train)

# Testar e avaliar o modelo treinado
y_hat_train = lm.predict(X_train)
avaliar(y_train, y_hat_train, 'treino')

y_hat_test = lm.predict(X_test)
avaliar(y_test, y_hat_test)
```

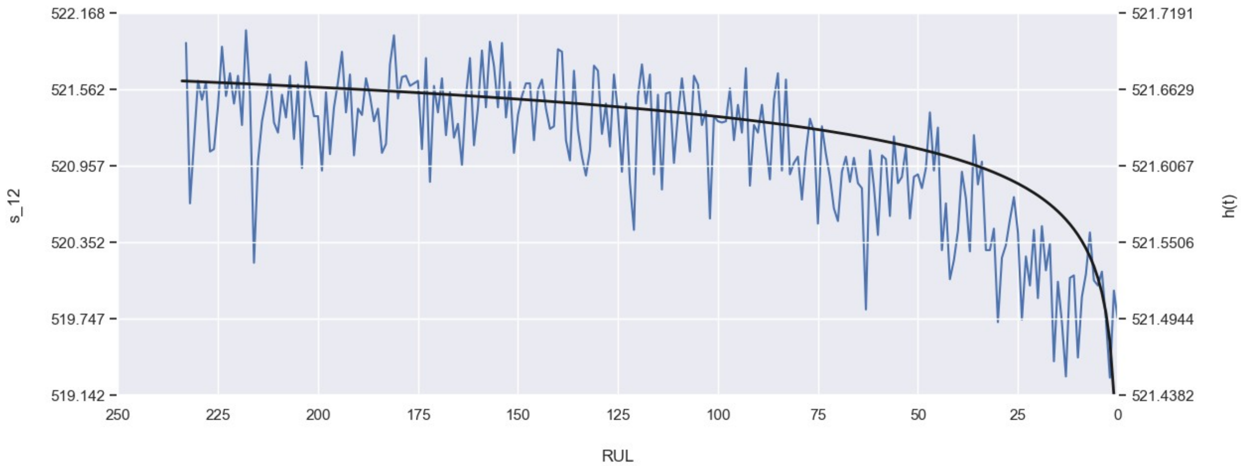
```
conjunto de treino -> RMSE:44.66819159545402, R2:0.5794486527796813
conjunto de teste -> RMSE:31.952633027743115, R2:0.40877368076569287
```

```
fig, ax1 = plt.subplots(1,1, figsize=(13,5))

sinal = ax1.plot('RUL', 's_12', 'b',
                 data=train.loc[train['unidade']==20])
plt.xlim(250, 0)  # Reverter o eixo x
plt.xticks(np.arange(0, 275, 25))
ax1.set_ylabel('s_12', labelpad=20)
ax1.set_xlabel('RUL', labelpad=20)

ax2 = ax1.twinx()
linha_rul = ax2.plot('RUL', 'RUL', 'k', linewidth=4,
                   data=train.loc[train['unidade']==20])
ax2.set_ylabel('RUL', labelpad=20)

# código para ter espaçamento igual de xticks para ambos os eixos,
para que as linhas de grade se alinhem
# fonte https://stackoverflow.com/questions/20243683/matplotlib-align-
twinx-tick-marks?rq=1
ax2.set_ylim(0, 250)  # defina os limites do eixo que você deseja
exibir
ax2.set_yticks(
    np.linspace(ax2.get_ybound()[0], ax2.get_ybound()[1], 6))  #
escolha um número inteiro para dividir seu eixo, no nosso caso 6
ax1.set_yticks(
    np.linspace(ax1.get_ybound()[0], ax1.get_ybound()[1], 6))  #
aplicar o mesmo espaçamento a outro eixo

# código para ter uma legenda unificada
# fonte https://stackoverflow.com/questions/5484922/secondary-axis-
with-twinx-how-to-add-to-legend
linhas = sinal+linha_rul
etiquetas = [linha.get_label() for linha in linhas]
ax1.legend(linhas, etiquetas, loc=0)


plt.show()
```




```
# Bloco que cria pontos de uma funcao exponencial com comportamento
semelhante a funcao de desgaste esperado

# Alguns parametros importante para criar os pontos da funcao
primeiro_valor_s12 = train.loc[train['unidade']==20]['s_12'].iloc[0]
ultimo_ciclo = train.loc[train['unidade']==20]['ciclo_tempo'].iloc[-1]

# Criando o vetor de tempo que vai do ciclo 1 ao último ciclo de tempo
daquela unidade
t = np.linspace(1, ultimo_ciclo, 234)

# Definindo os parametros a, b e d
a = 0.4
b = -0.1
d = -primeiro_valor_s12

# Calcular os valores de h(t) para cada valor de t
h = 1 - d - np.exp(a * t ** b)

# Grafico dessa funcao exponencial
fig, ax1 = plt.subplots(1,1, figsize=(13,5))

sinal = ax1.plot('RUL', 's_12', 'b',
                 data=train.loc[train['unidade']==20])

plt.xlim(250, 0)
plt.xticks(np.arange(0, 275, 25))
ax1.set_ylabel('s_12', labelpad=20)
ax1.set_xlabel('RUL', labelpad=20)

ax2 = ax1.twinx()
linha_h = ax2.plot(t, h, 'k', linewidth=2,
                   data=train.loc[train['unidade']==20])
ax2.set_ylabel('h(t)', labelpad=20)
```

```
ax2.set_ylim(min(h), max(h) + 0.05)
ax2.set_yticks(
    np.linspace(ax2.get_ybound()[0], ax2.get_ybound()[1], 6))
ax1.set_yticks(
    np.linspace(ax1.get_ybound()[0], ax1.get_ybound()[1], 6))

plt.show()
```

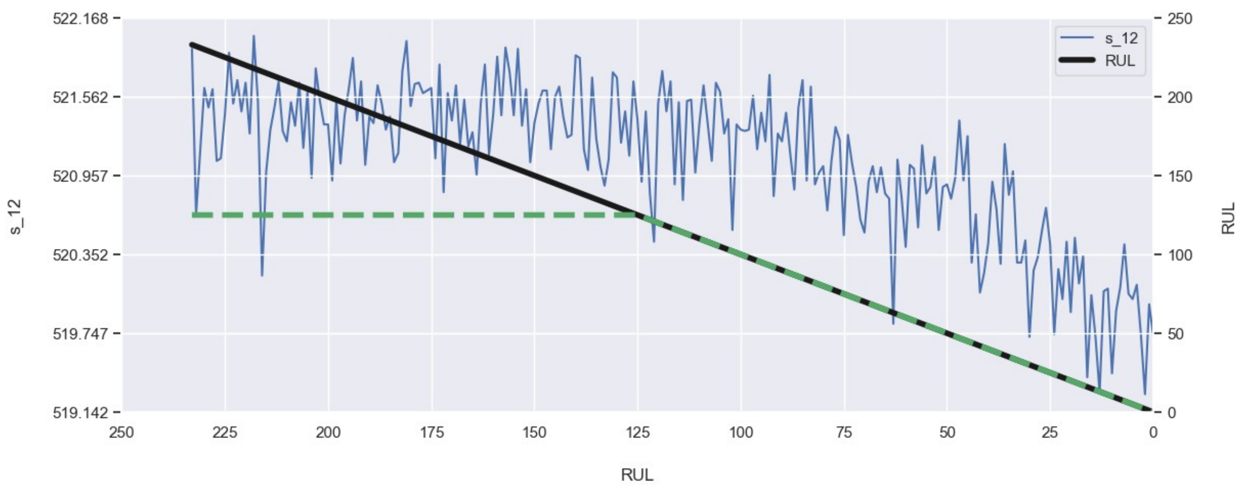


5. Reexaminando o RUL

```
fig, ax1 = plt.subplots(1,1, figsize=(13,5))

sinal = ax1.plot('RUL', 's_12', 'b',
                 data=train.loc[train['unidade']==20])
plt.xlim(250, 0)  # reverter o eixo x
plt.xticks(np.arange(0, 275, 25))
ax1.set_ylabel('s_12', labelpad=20)
ax1.set_xlabel('RUL', labelpad=20)

ax2 = ax1.twinx()
linha_rul = ax2.plot('RUL', 'RUL', 'k', linewidth=4,
                   data=train.loc[train['unidade']==20])
rul = train.loc[train['unidade']==20, 'RUL']
linha_rul2 = ax2.plot(rul, rul.where(rul <= 125, 125), '--g',
linewidth=4, label='rul_cortada')
ax2.set_ylabel('RUL', labelpad=20)

# código para ter espaçamento igual de xticks para ambos os eixos,
para que as linhas de grade se alinhem
# fonte https://stackoverflow.com/questions/20243683/matplotlib-align-
twinx-tick-marks?rq=1
ax2.set_ylim(0, 250)  # defina os limites do eixo que você deseja
exibir
```

```
ax2.set_yticks(
    np.linspace(ax2.get_ybound()[0], ax2.get_ybound()[1], 6))  #
escolha um número inteiro para dividir seu eixo, no nosso caso 6
ax1.set_yticks(
    np.linspace(ax1.get_ybound()[0], ax1.get_ybound()[1], 6))  #
aplicar o mesmo espaçamento a outro eixo

# código para ter uma legenda unificada
# fonte https://stackoverflow.com/questions/5484922/secondary-axis-
with-twinx-how-to-add-to-legend
linhas = sinal+linha_rul
etiquetas = [linha.get_label() for linha in linhas]
ax1.legend(linhas, etiquetas, loc=0)
```

```
<matplotlib.legend.Legend at 0x1fa0735f700>
```




```
# Limitar o RUL a 125
# O objetivo é ter uma funcao mais proxima ao comportamento de
decaimento exponencial
y_train_clipped = y_train.clip(upper=125)

# Criar e ajustar o modelo
lm = LinearRegression()
lm.fit(X_train, y_train_clipped)

# Testar e avaliar o modelo
y_hat_train = lm.predict(X_train)
avaliar(y_train_clipped, y_hat_train, 'treino')

y_hat_test = lm.predict(X_test)
avaliar(y_test, y_hat_test)
```

```
conjunto de treino -> RMSE:21.491018701515276, R2:0.7340432868050482
conjunto de teste -> RMSE:21.900213406888167, R2:0.7222608196546836
```

6. Regressão com Vetores de Suporte - Support Vector Regression (SVR)

```
svr = SVR(kernel='linear')
svr.fit(X_train, y_train_clipped)

# Testar e avaliar o modelo
y_hat_train = svr.predict(X_train)
avaliar(y_train_clipped, y_hat_train, 'treino')

y_hat_test = svr.predict(X_test)
avaliar(y_test, y_hat_test)

conjunto de treino -> RMSE:31.162854665387698, R2:0.44079451641915235
conjunto de teste -> RMSE:31.762230542307382, R2:0.4157988020504362
```

6.1. Normalização

```
# Normalizacao
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# SVR + RUL cortada + normalizacao
svr = SVR(kernel='linear')
svr.fit(X_train_scaled, y_train_clipped)

# Testar e avaliar o modelo
y_hat_train = svr.predict(X_train_scaled)
avaliar(y_train_clipped, y_hat_train, 'treino')

y_hat_test = svr.predict(X_test_scaled)
avaliar(y_test, y_hat_test)

conjunto de treino -> RMSE:21.578263975067905, R2:0.7318795396979628
conjunto de teste -> RMSE:21.580480163289657, R2:0.7303113540952145
```

6.2. Engenharia de Parâmetros

```
from sklearn.preprocessing import PolynomialFeatures
# Polinômio de 2º grau dos parâmetros [a, b] torna-se [1, a, b, a^2,
ab, b^2]
poly = PolynomialFeatures(2)
X_train_transformed = poly.fit_transform(X_train_scaled)
X_test_transformed = poly.fit_transform(X_test_scaled)
```

```
print(X_train_scaled.shape)
print(X_train_transformed.shape)
```

```
(20631, 14)
(20631, 120)
```

```
# SVR + RUL cortada + engenharia de parametros
svr_f = SVR(kernel='linear')
svr_f.fit(X_train_transformed, y_train_clipped)

# Testar e avaliar o modelo
y_hat_train = svr_f.predict(X_train_transformed)
avaliar(y_train_clipped, y_hat_train, 'treino')

y_hat_test = svr_f.predict(X_test_transformed)
avaliar(y_test, y_hat_test)
```

```
conjunto de treino -> RMSE:19.71678973113082, R2:0.7761436785704148
conjunto de teste -> RMSE:20.585402508370784, R2:0.7546086882115255
```

6.3. Seleção de Parâmetros

```
# Engenharia de parametros + selecao de parametros
from sklearn.feature_selection import SelectFromModel
select_features = SelectFromModel(svr_f, threshold='mean',
prefit=True)
select_features.get_support()
feature_names = poly.get_feature_names_out()

print('Parâmetros originais:\n', X_train.columns)
print('Melhores parâmetros:\n', np.array(feature_names)
[select_features.get_support()])
np.array(feature_names)[select_features.get_support()].shape
```

```
Parâmetros originais:
 Index(['s_2', 's_3', 's_4', 's_7', 's_8', 's_9', 's_11', 's_12',
's_13',
       's_14', 's_15', 's_17', 's_20', 's_21'],
      dtype='object')
Melhores parâmetros:
 ['x0' 'x1' 'x2' 'x3' 'x5' 'x6' 'x7' 'x9' 'x10' 'x11' 'x12' 'x13' 'x2
x5'
 'x2 x8' 'x2 x9' 'x3 x5' 'x3 x8' 'x3 x9' 'x4^2' 'x4 x6' 'x4 x7' 'x4
x8'
 'x5^2' 'x5 x6' 'x5 x7' 'x5 x9' 'x5 x12' 'x5 x13' 'x6^2' 'x6 x8' 'x6
x9'
 'x7 x8' 'x7 x9' 'x8^2' 'x9^2' 'x9 x12' 'x9 x13']

(37,)
```

```
# SVR + RUL cortada + engenharia de parametros + selecao de parametros
svr = SVR(kernel='linear')
svr.fit(X_train_transformed[:, select_features.get_support()],
y_train_clipped)

# Testar e avaliar o modelo
y_hat_train = svr.predict(X_train_transformed[:,
select_features.get_support()])
avaliar(y_train_clipped, y_hat_train, 'treino')

y_hat_test = svr.predict(X_test_transformed[:,
select_features.get_support()])
avaliar(y_test, y_hat_test)
```

```
conjunto de treino -> RMSE:19.74678910148113, R2:0.7754619593165268
conjunto de teste -> RMSE:20.556138196054615, R2:0.75530589134507
```

6.4. Modelo Final

```
epsilon = [0.4, 0.3, 0.2, 0.1, 0.05]

for e in epsilon:
    svr = SVR(kernel='linear', epsilon=e)
    svr.fit(X_train_transformed[:, select_features.get_support()],
y_train_clipped)

    # Testar e avaliar o modelo
    y_hat = svr.predict(X_train_transformed[:,
select_features.get_support()])
    mse = mean_squared_error(y_train_clipped, y_hat)
    rmse = np.sqrt(mse)
    variance = r2_score(y_train_clipped, y_hat)
    print("epsilon:", e, "RMSE:", rmse, "R2:", variance)
```

```
epsilon: 0.4 RMSE: 19.74772556660335 R2: 0.7754406619776464
epsilon: 0.3 RMSE: 19.74758076106985 R2: 0.7754439552496147
epsilon: 0.2 RMSE: 19.746600078171717 R2: 0.775466258012399
epsilon: 0.1 RMSE: 19.74678910148113 R2: 0.7754619593165268
epsilon: 0.05 RMSE: 19.746532456984 R2: 0.7754677958176169
```

```
svr = SVR(kernel='linear', epsilon=0.2)
svr.fit(X_train_transformed[:, select_features.get_support()],
y_train_clipped)

# Testar e avaliar o modelo
y_hat_train = svr.predict(X_train_transformed[:,
select_features.get_support()])
avaliar(y_train_clipped, y_hat_train, 'treino')

y_hat_test = svr.predict(X_test_transformed[:,
```

```
select_features.get_support()])
avaliar(y_test, y_hat_test)
```

```
conjunto de treino -> RMSE:19.746600078171717, R2:0.775466258012399
conjunto de teste -> RMSE:20.544124820773813, R2:0.7555918150093471
```

7. Análise de série temporal

7.1. Carregando a base de dados

```
# carregar dados
train = pd.read_csv((dir_path+'train_FD001.txt'), sep='\s+',
header=None, names=col_names)
test = pd.read_csv((dir_path+'test_FD001.txt'), sep='\s+',
header=None, names=col_names)
y_test = pd.read_csv((dir_path+'RUL_FD001.txt'), sep='\s+',
header=None, names=['RUL'])

# avaliar as primeiras linhas
train.head()
```

```
   unidade  ciclo_tempo  config_1  config_2  config_3     s_1     s_2
\
0        1            1   -0.0007   -0.0004     100.0  518.67  641.82

1        1            2    0.0019   -0.0003     100.0  518.67  642.15

2        1            3   -0.0043    0.0003     100.0  518.67  642.35

3        1            4    0.0007    0.0000     100.0  518.67  642.35

4        1            5   -0.0019   -0.0002     100.0  518.67  642.37


       s_3      s_4    s_5  ...     s_12     s_13     s_14    s_15
s_16  s_17  \
0  1589.70  1400.60  14.62  ...  521.66  2388.02  8138.62  8.4195
0.03   392
1  1591.82  1403.14  14.62  ...  522.28  2388.07  8131.49  8.4318
0.03   392
2  1587.99  1404.20  14.62  ...  522.42  2388.03  8133.23  8.4178
0.03   390
3  1582.79  1401.87  14.62  ...  522.86  2388.08  8133.83  8.3682
0.03   392
4  1582.85  1406.22  14.62  ...  522.19  2388.04  8133.80  8.4294
0.03   393

   s_18   s_19   s_20     s_21
```

```
0  2388  100.0  39.06  23.4190
1  2388  100.0  39.00  23.4236
2  2388  100.0  38.95  23.3442
3  2388  100.0  38.88  23.3739
4  2388  100.0  38.90  23.4044

[5 rows x 26 columns]

train = add_RUL(train)
train[index_names+['RUL']].head()

   unidade  ciclo_tempo  RUL
0        1            1  191
1        1            2  190
2        1            3  189
3        1            4  188
4        1            5  187

# eliminar colunas indesejadas com base na análise inicial realizada
anteriormente
drop_sensors = ['s_1','s_5','s_6','s_10','s_16','s_18','s_19']
drop_labels = setting_names+drop_sensors

X_train = train.drop(drop_labels, axis=1)
# y_treino = X_treino.pop('RUL')
# vamos separar o RUL após remover as linhas contendo NaN introduzidos
pelo metodo de variaveis atrasadas

X_test_interim = test.drop(drop_labels, axis=1)
X_test_interim.head()

   unidade  ciclo_tempo     s_2      s_3      s_4     s_7      s_8
s_9  \
0        1            1  643.02  1585.29  1398.21  553.90  2388.04
9050.17
1        1            2  641.71  1588.45  1395.42  554.85  2388.01
9054.42
2        1            3  642.46  1586.94  1401.34  554.11  2388.05
9056.96
3        1            4  642.44  1584.12  1406.42  554.07  2388.03
9045.29
4        1            5  642.51  1587.19  1401.92  554.16  2388.01
9044.55

    s_11    s_12     s_13     s_14    s_15  s_17   s_20     s_21
0  47.20  521.72  2388.03  8125.55  8.4052   392  38.86  23.3735
1  47.50  522.16  2388.06  8139.62  8.3803   393  39.02  23.3916
2  47.50  521.97  2388.03  8130.10  8.4441   393  39.08  23.4166
3  47.28  521.38  2388.05  8132.90  8.3917   391  39.00  23.3737
4  47.31  522.15  2388.03  8129.54  8.4031   390  38.99  23.4130
```

7.2. Adicionando variáveis defasadas

```
# Exemplo de como adicionar as variaveis atrasadas e os efeitos dessa
adicao no dataset
np.random.seed(42)
df_exemplo = pd.DataFrame({'t': np.random.rand(5)})
df_exemplo['t-1'] = df_exemplo['t'].shift(1)
df_exemplo['t-2'] = df_exemplo['t'].shift(2)
display(df_exemplo)
```

	t	t-1	t-2
0	0.374540	NaN	NaN
1	0.950714	0.374540	NaN
2	0.731994	0.950714	0.374540
3	0.598658	0.731994	0.950714
4	0.156019	0.598658	0.731994

```
# Criar as variaveis atrasadas
remaining_sensors = X_train.columns.difference(index_names+['RUL'])
lag1 = [col + '_lag_1' for col in remaining_sensors]

X_train[lag1] = X_train.groupby('unidade')[remaining_sensors].shift(1)
X_train.dropna(inplace=True)

X_test_interim[lag1] = X_test_interim.groupby('unidade')
[remaining_sensors].shift(1)
X_test_interim.dropna(inplace=True)


# Variavel alvo
y_train = X_train.pop('RUL')

# preparar os dados do teste, já que os valores reais para o RUL no
conjunto de teste são fornecidos apenas para o último ciclo de tempo
# de cada motor, o conjunto de teste é subdividido para representar o
mesmo
X_test = X_test_interim.groupby('unidade').last().reset_index()

X_train.head()
```

	unidade	ciclo_tempo	s_2	s_3	s_4	s_7	s_8	s_9 \
1	1	2	642.15	1591.82	1403.14	553.75	2388.04	9044.07
2	1	3	642.35	1587.99	1404.20	554.26	2388.08	9052.94
3	1	4	642.35	1582.79	1401.87	554.45	2388.11	9049.48
4	1	5	642.37	1582.85	1406.22	554.00	2388.06	9055.15
5	1	6	642.10	1584.47	1398.37	554.67	2388.02	

```
9049.68

    s_11    s_12  ...  s_15_lag_1  s_17_lag_1  s_2_lag_1  
s_20_lag_1  \
1  47.49  522.28  ...      8.4195       392.0     641.82       39.06   

2  47.27  522.42  ...      8.4318       392.0     642.15       39.00   

3  47.13  522.86  ...      8.4178       390.0     642.35       38.95   

4  47.28  522.19  ...      8.3682       392.0     642.35       38.88   

5  47.16  521.68  ...      8.4294       393.0     642.37       38.90   


   s_21_lag_1  s_3_lag_1  s_4_lag_1  s_7_lag_1  s_8_lag_1  s_9_lag_1  
1     23.4190    1589.70    1400.60     554.36    2388.06    9046.19  
2     23.4236    1591.82    1403.14     553.75    2388.04    9044.07  
3     23.3442    1587.99    1404.20     554.26    2388.08    9052.94  
4     23.3739    1582.79    1401.87     554.45    2388.11    9049.48  
5     23.4044    1582.85    1406.22     554.00    2388.06    9055.15  

[5 rows x 30 columns]
```

```
# Criar e treinar o modelo
lm = LinearRegression()
lm.fit(X_train, y_train)

# Testar o modelo e avaliar os resultados
y_hat_train = lm.predict(X_train)
avaliar(y_train, y_hat_train, 'treino')

y_hat_test = lm.predict(X_test)
avaliar(y_test, y_hat_test)
```

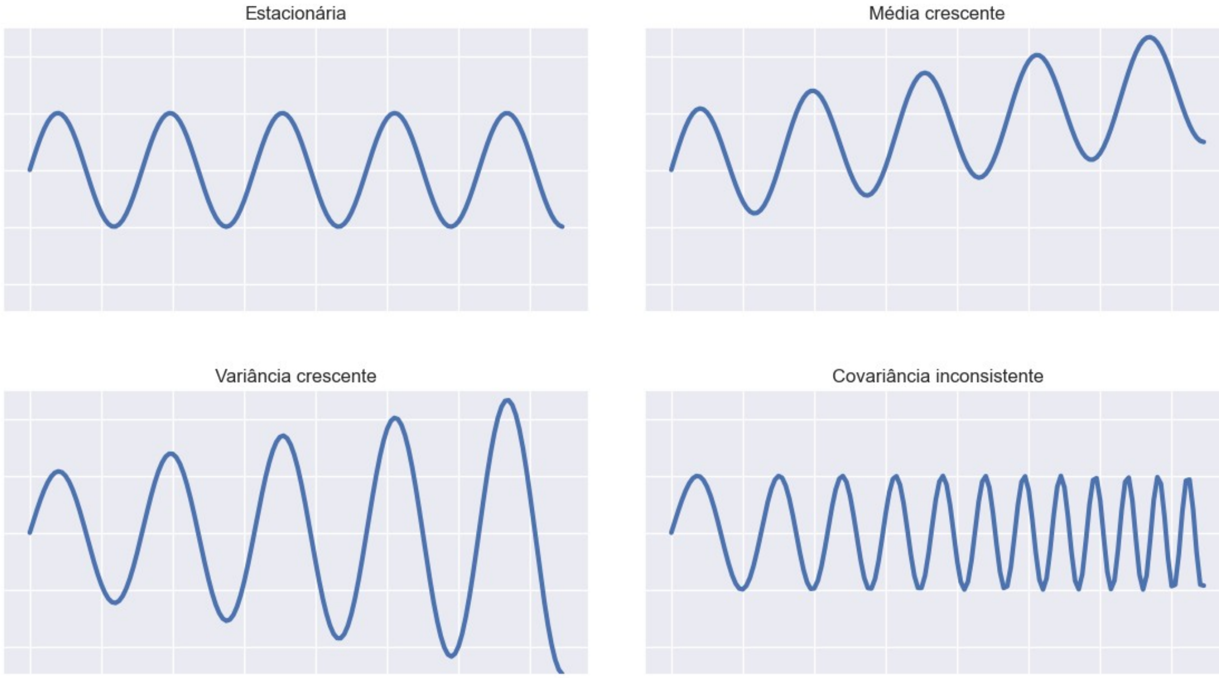
```
conjunto de treino -> RMSE:39.36769235113502, R2:0.6709929595362336
conjunto de teste -> RMSE:31.423109839703113, R2:0.42820706786543306
```

```
# Funcao para adicionar atrasos de 1 a n
def add_lagged_variables(df_input, nr_of_lags, columns):
    df = df_input.copy()
    for i in range(nr_of_lags):
        lagged_columns = [col + '_lag_{}'.format(i+1) for col in 
columns]
        df[lagged_columns] = df.groupby('unidade')[columns].shift(i+1)
    df.dropna(inplace=True)
    return df

# Funcao para adicionar atrasos específicos
def add_specific_lags(df_input, list_of_lags, columns):
    df = df_input.copy()
```

```
    for i in list_of_lags:
        lagged_columns = [col + '_lag_{}'.format(i) for col in
columns]
        df[lagged_columns] = df.groupby('unidade')[columns].shift(i)
    df.dropna(inplace=True)
    return df
```

7.3. Estacionaridade

```
# Graficos de exemplo de estacionaridade
t = np.arange(0,150)
fator = 0.2
estacionaria = np.sin(fator*t)
media_crescente = np.sin(fator*t) + t/100
variancia_crescente = np.sin(fator*t) * (1 + t/100)
covariancia_inconsistente = np.sin((fator + t/500) * t)

plotlist = [estacionaria, media_crescente, variancia_crescente,
covariancia_inconsistente]
plotnames = ['Estacionária', 'Média crescente', 'Variância crescente',
'Covariância inconsistente']

plt.subplots(2, 2, figsize=(12,7))  # inicia subplot
ylim = 2.5

for i in range(len(plotlist)):

    plt.subplot(2, 2, i+1)  # define qual subplot preencher, o
intervalo começa em 0, e aumenta de 1 em 1
    plt.plot(t, plotlist[i], linewidth=3)
    plt.ylim(-ylim, ylim)
    plt.tick_params(which='both', bottom=False, labelbottom=False,
left=False, labelleft=False)  # remover marcações e rótulos do eixo
    plt.title(str(plotnames[i]))

plt.tight_layout(pad=3)  # especifica o layout e o preenchimento de
espaços em branco entre os gráficos
plt.show()
```

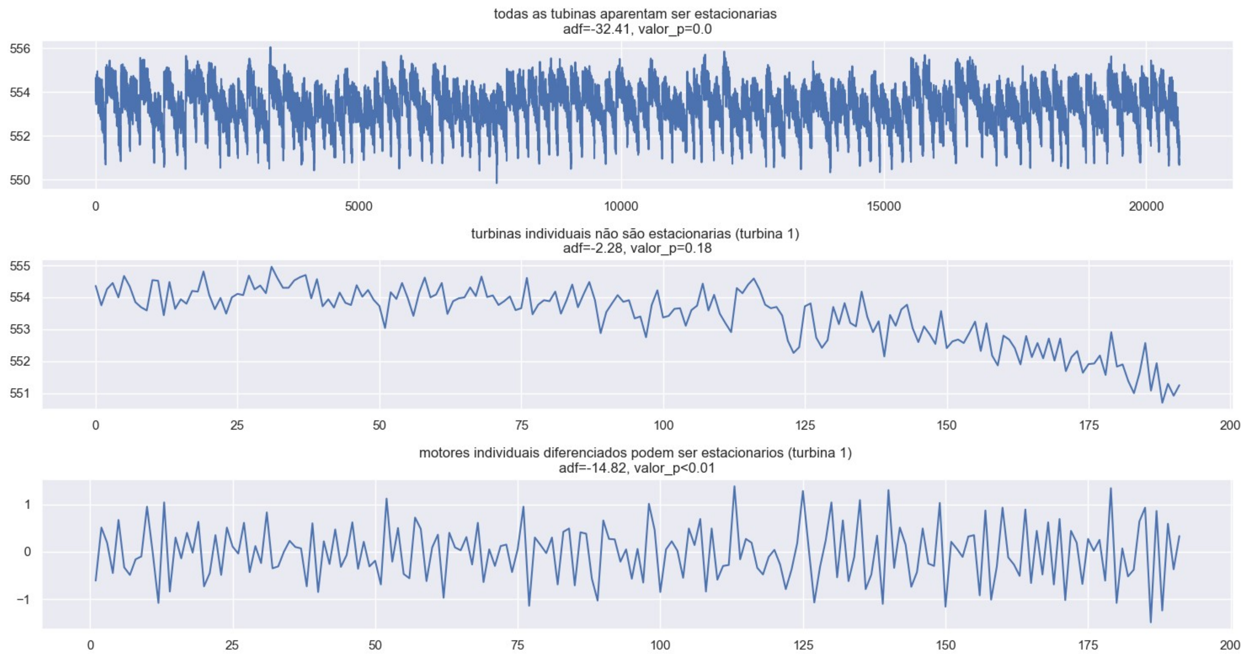



```
from statsmodels.tsa.stattools import adfuller

test_series = train.loc[train['unidade']==1, 's_7']
adf, valor_p, atraso_usado, n_obs, valores_criticos, melhor_ic =
adfuller(test_series, maxlag=1)
print('uma unidade, uma coluna de teste - resultados:')
print('adf: {} \nvalor_p: {}'.format(adf, valor_p))
print('significancia: {}\n'.format(valor_p < 0.05))

uma unidade, uma coluna de teste - resultados:
adf: -2.276666927948035
valor_p: 0.17960380425358008
significancia: False


test_series = test_series.diff(1).dropna()
adf, valor_p, atraso_usado, n_obs, valores_criticos, melhor_ic =
adfuller(test_series, maxlag=1)
print('uma unidade, uma coluna de teste após diferenciação')
print('adf: {} \nvalor_p: {}'.format(adf, valor_p))
print('significancia: {}\n'.format(valor_p < 0.05))

uma unidade, uma coluna de teste após diferenciação
adf: -14.824168539718976
valor_p: 1.943101859639205e-27
significancia: True


adf, valor_p, atraso_usado, n_obs, valores_criticos, melhor_ic =
adfuller(train[ 's_7'], maxlag=1)
```

```python
print('todas as unidades, uma coluna de teste - resultados:')
print('adf: {} \nvalor_p: {}'.format(adf, valor_p))
print('significancia: {}'.format(valor_p < 0.05))
print('OBS: Testando uma coluna com valores de 100 turbinas, todos os
motores juntos são estacionarios, mas motores individuais não!\n')
```

```
todas as unidades, uma coluna de teste - resultados:
adf: -32.413164189329294
valor_p: 0.0
significancia: True
OBS: Testando uma coluna com valores de 100 turbinas, todos os motores
juntos são estacionarios, mas motores individuais não!
```

```python
# todas as turbinas vs uma unica turbina
plt.subplots(3,1, figsize=(15,8))

plt.subplot(3,1,1)
plt.plot(train[ 's_7'])
plt.title('todas as tubinas aparentam ser estacionarias \n\
adf=-32.41, valor_p=0.0')


plt.subplot(3,1,2)
plt.plot(train.loc[train['unidade']==1, 's_7'])
plt.title('turbinas individuais não são estacionarias (turbina 1) \n\
adf=-2.28, valor_p=0.18')


plt.subplot(3,1,3)
plt.plot(train.loc[train['unidade']==1, 's_7'].diff(1).dropna())
plt.title('motores individuais diferenciados podem ser estacionarios
(turbina 1) \n\
adf=-14.82, valor_p<0.01')


plt.tight_layout()
plt.show()
```




```python
# Funcao para realizar a diferenciacao dos dados para garantir a estacionaridade
def find_max_diff(series):
    maxdiff = 0
    do = True
    adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(series, maxlag=1)
    if pvalue < 0.05:
        do = False

    while do:
        maxdiff += 1
        adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(series.diff(maxdiff).dropna(), maxlag=1)
        if pvalue < 0.05:  # Se significante, parar de diferenciar e testar estacionaridade
            do = False
    return maxdiff

# Funcao que torna os dados estacionarios
def make_stationary(df_input, columns):
    df = df_input.copy()
    for unit_nr in range(1, df['unidade'].max()+1):
        for col in columns:
            maxdiff = find_max_diff(df.loc[df['unidade']==unit_nr, col])
            if maxdiff > 0:
                df.loc[df['unidade']==unit_nr, col] = df.loc[df['unidade']==unit_nr, col].diff(maxdiff)
```

```
    df.dropna(inplace=True)
    return df
```

7.4. Aplicando o conceito de estacionaridade aos dados

```
# fazer com que todos os sensores restantes sejam estacionarios para
cada unidade
intermediate_df = train.drop(drop_labels, axis=1)
intermediate_df = make_stationary(intermediate_df, remaining_sensors)

intermediate_df.head()  # dados estacionarios
```

```
   unidade  ciclo_tempo     s_2      s_3   s_4   s_7   s_8      s_9
s_11  \
1        1            2  642.15  1591.82  2.54 -0.61 -0.02  9044.07
0.02
2        1            3  642.35  1587.99  1.06  0.51  0.04  9052.94 -
0.22
3        1            4  642.35  1582.79 -2.33  0.19  0.03  9049.48 -
0.14
4        1            5  642.37  1582.85  4.35 -0.45 -0.05  9055.15
0.15
5        1            6  642.10  1584.47 -7.85  0.67 -0.04  9049.68 -
0.12

   s_12  s_13  s_14    s_15   s_17   s_20     s_21  RUL
1  0.62  0.05 -7.13  8.4318  392.0  39.00  23.4236  190
2  0.14 -0.04  1.74  8.4178  390.0  38.95  23.3442  189
3  0.44  0.05  0.60  8.3682  392.0  38.88  23.3739  188
4 -0.67 -0.04 -0.03  8.4294  393.0  38.90  23.4044  187
5 -0.51 -0.01 -0.95  8.4108  391.0  38.98  23.3669  186
```

```
# Criar e treinar o modelo
lm = LinearRegression()
lm.fit(intermediate_df[remaining_sensors], intermediate_df['RUL'])

# Testar o modelo e avaliar os resultados
y_hat_train = lm.predict(intermediate_df[remaining_sensors])
avaliar(intermediate_df['RUL'], y_hat_train, 'treino')
```

```
conjunto de treino -> RMSE:51.633756137911796, R2:0.43408640113852304
```

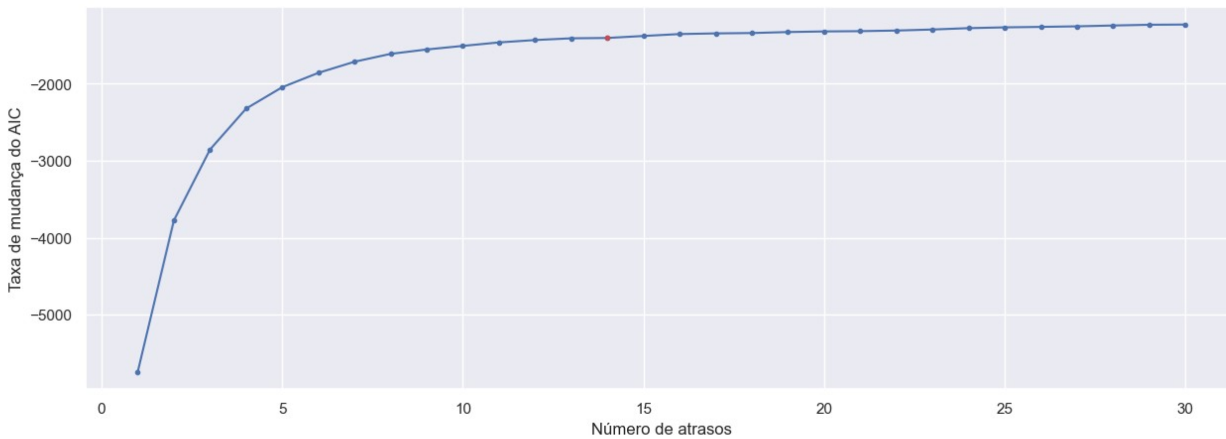
7.5. AIC: Procurando o número correto de defasagens

```
# Adicionar atrasos e avaliar os modelos para encontrar o numero ideal
de atrasos
import statsmodels.api as sm

metrics = pd.DataFrame(columns=['rmse', 'AIC', 'BIC'])
nr_of_lags = 30
```

```
for i in range(0, nr_of_lags+1):
    X_train = add_lagged_variables(intermediate_df, i,
remaining_sensors)
    X_train = X_train.drop(index_names, axis=1)
    y_train = X_train.pop('RUL')

    model = sm.OLS(y_train, sm.add_constant(X_train.values))
    result = model.fit()

    metrics = pd.concat([metrics, pd.DataFrame(
        data=[[np.sqrt(result.mse_resid), round(result.aic,2),
round(result.bic,2)]],columns=['rmse', 'AIC', 'BIC']
        )],
                        ignore_index=True)

display(metrics)
```

	rmse	AIC	BIC
0	51.652627	220271.65	220390.59
1	46.060503	214530.30	214760.12
2	43.058562	210753.82	211094.38
3	41.151503	207898.35	208349.51
4	39.833453	205574.27	206135.89
5	38.813294	203528.00	204199.94
6	37.986450	201667.95	202450.06
7	37.304649	199952.23	200844.37
8	36.723366	198338.72	199340.76
9	36.198014	196782.37	197894.15
10	35.717277	195272.37	196493.75
11	35.279048	193807.53	195138.36
12	34.869837	192373.83	193813.96
13	34.481184	190962.43	192511.71
14	34.096930	189555.58	191213.86
15	33.735926	188174.60	189941.74
16	33.396878	186818.44	188694.27
17	33.065376	185471.02	187455.40
18	32.737658	184128.15	186220.90
19	32.420900	182798.11	184999.09
20	32.110808	181476.10	183785.15
21	31.803284	180157.46	182574.43
22	31.502414	178846.97	181371.69
23	31.212569	177549.83	180182.14
24	30.937293	176270.33	179010.06
25	30.670196	175001.12	177848.10
26	30.407545	173737.89	176691.96
27	30.150377	172481.90	175542.91
28	29.901632	171236.72	174404.47
29	29.660232	170001.12	173275.46
30	29.420456	168768.36	172149.11

```
# Gráfico da taxa de mudança do AIC versus número de atrasos adicionados
plt.figure(figsize=(15,5))
plt.plot(metrics['AIC'].diff(), marker='.')  # traca a diferença para ver onde a curva se achata
plt.plot(14, metrics['AIC'].diff()[14], '.r')
plt.xlabel("Número de atrasos")
plt.ylabel("Taxa de mudança do AIC")
plt.show()
plt.close()
```



7.5.1. Resumo dos resultados AIC

```
result.summary()  # Cheque as notas ao final dos resultados

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results

==============================================================================
========
Dep. Variable:                    RUL   R-squared:
0.777
Model:                            OLS   Adj. R-squared:
0.771
Method:                 Least Squares   F-statistic:
136.9
Date:                Fri, 09 Feb 2024   Prob (F-statistic):
0.00
Time:                        15:10:16   Log-Likelihood:
-83949.
No. Observations:               17533   AIC:
1.688e+05
Df Residuals:                   17098   BIC:
1.721e+05
Df Model:                         434
```

```
Covariance Type:            nonrobust
==============================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
const       1.908e+04    165.898    114.981      0.000    1.88e+04
1.94e+04
x1            -0.5162      0.722     -0.715      0.474      -1.931
0.899
x2            -0.3837      0.055     -6.974      0.000      -0.492
-0.276
x3            -0.2225      0.047     -4.728      0.000      -0.315
-0.130
x4             1.2576      0.471      2.671      0.008       0.335
2.180
x5            14.4389      6.395      2.258      0.024       1.904
26.974
x6            -0.0607      0.046     -1.325      0.185      -0.150
0.029
x7            -7.1442      1.865     -3.830      0.000     -10.800
-3.488
x8             1.5610      0.624      2.501      0.012       0.338
2.784
x9            12.0334      6.284      1.915      0.056      -0.284
24.351
x10            0.0454      0.059      0.764      0.445      -0.071
0.162
x11          -32.1409     10.207     -3.149      0.002     -52.148
-12.134
x12           -0.4505      0.228     -1.975      0.048      -0.897
-0.003
x13            3.2219      2.087      1.544      0.123      -0.869
7.313
x14            5.7930      3.539      1.637      0.102      -1.144
12.730
x15          -11.3690      2.129     -5.341      0.000     -15.541
-7.197
x16            2.2548      0.675      3.342      0.001       0.932
3.577
x17           13.9514      6.514      2.142      0.032       1.183
26.720
x18            0.0655      0.063      1.035      0.301      -0.059
0.190
x19          -31.2511     10.278     -3.041      0.002     -51.397
-11.106
```

x20	-0.4099	0.228	-1.797	0.072	-0.857	0.037
x21	-0.5227	0.723	-0.723	0.470	-1.939	0.894
x22	3.5827	2.090	1.714	0.087	-0.515	7.680
x23	5.4823	3.544	1.547	0.122	-1.464	12.429
x24	-0.3639	0.055	-6.605	0.000	-0.472	-0.256
x25	-0.3235	0.051	-6.390	0.000	-0.423	-0.224
x26	1.6130	0.494	3.263	0.001	0.644	2.582
x27	16.1230	6.638	2.429	0.015	3.112	29.134
x28	-0.0789	0.048	-1.633	0.103	-0.174	0.016
x29	-13.5609	2.187	-6.200	0.000	-17.848	-9.273
x30	2.6131	0.679	3.850	0.000	1.283	3.943
x31	12.4625	6.550	1.903	0.057	-0.377	25.302
x32	0.0639	0.064	1.003	0.316	-0.061	0.189
x33	-27.0154	10.276	-2.629	0.009	-47.157	-6.874
x34	-0.3297	0.228	-1.443	0.149	-0.777	0.118
x35	-0.8074	0.724	-1.116	0.264	-2.226	0.611
x36	4.0887	2.093	1.953	0.051	-0.014	8.192
x37	5.5043	3.544	1.553	0.120	-1.443	12.451
x38	-0.3612	0.055	-6.553	0.000	-0.469	-0.253
x39	-0.3632	0.051	-7.130	0.000	-0.463	-0.263
x40	1.7238	0.494	3.486	0.000	0.755	2.693
x41	10.6883	6.669	1.603	0.109	-2.383	23.759
x42	-0.1016	0.048	-2.100	0.036	-0.196	-0.007
x43	-15.1007	2.199	-6.866	0.000	-19.411	-10.790
x44	2.8442	0.680	4.185	0.000	1.512	

```
4.176
x45               9.1309        6.556        1.393        0.164        -3.719
21.981
x46               0.0554        0.064        0.871        0.384        -0.069
0.180
x47             -24.7710       10.284       -2.409        0.016       -44.929
-4.613
x48              -0.2381        0.228       -1.042        0.297        -0.686
0.210
x49              -0.7158        0.724       -0.988        0.323        -2.135
0.704
x50               3.4893        2.096        1.665        0.096        -0.618
7.597
x51               5.1919        3.550        1.463        0.144        -1.766
12.150
x52              -0.3593        0.055       -6.515        0.000        -0.467
-0.251
x53              -0.3455        0.051       -6.773        0.000        -0.446
-0.246
x54               1.7473        0.495        3.530        0.000         0.777
2.717
x55               9.0302        6.688        1.350        0.177        -4.079
22.139
x56              -0.1013        0.048       -2.088        0.037        -0.196
-0.006
x57             -15.4788        2.204       -7.022        0.000       -19.800
-11.158
x58               2.9086        0.682        4.263        0.000         1.571
4.246
x59               7.3085        6.564        1.113        0.266        -5.557
20.174
x60               0.0421        0.064        0.660        0.509        -0.083
0.167
x61             -21.8150       10.299       -2.118        0.034       -42.002
-1.628
x62              -0.1948        0.229       -0.852        0.394        -0.643
0.253
x63              -0.8302        0.724       -1.147        0.252        -2.249
0.589
x64               3.2161        2.095        1.535        0.125        -0.891
7.323
x65               4.9624        3.553        1.397        0.163        -2.002
11.927
x66              -0.3468        0.055       -6.281        0.000        -0.455
-0.239
x67              -0.3194        0.051       -6.242        0.000        -0.420
-0.219
x68               1.7426        0.496        3.512        0.000         0.770
2.715
```

x69	7.0777	6.699	1.056	0.291	-6.053	20.209
x70	-0.1018	0.049	-2.094	0.036	-0.197	-0.007
x71	-14.9779	2.212	-6.772	0.000	-19.313	-10.643
x72	2.8179	0.685	4.114	0.000	1.475	4.161
x73	6.6674	6.578	1.014	0.311	-6.225	19.560
x74	0.0457	0.064	0.713	0.476	-0.080	0.172
x75	-19.7584	10.310	-1.916	0.055	-39.967	0.450
x76	-0.2005	0.229	-0.876	0.381	-0.649	0.248
x77	-0.6308	0.724	-0.871	0.384	-2.050	0.789
x78	2.0938	2.097	0.999	0.318	-2.016	6.203
x79	3.9833	3.556	1.120	0.263	-2.986	10.953
x80	-0.3651	0.055	-6.608	0.000	-0.473	-0.257
x81	-0.2959	0.051	-5.759	0.000	-0.397	-0.195
x82	1.6565	0.498	3.328	0.001	0.681	2.632
x83	5.5047	6.711	0.820	0.412	-7.650	18.659
x84	-0.1070	0.049	-2.199	0.028	-0.202	-0.012
x85	-13.8846	2.222	-6.250	0.000	-18.239	-9.530
x86	2.7048	0.688	3.932	0.000	1.356	4.053
x87	7.0066	6.596	1.062	0.288	-5.922	19.935
x88	0.0433	0.064	0.672	0.502	-0.083	0.170
x89	-18.8011	10.324	-1.821	0.069	-39.037	1.435
x90	-0.1365	0.229	-0.597	0.551	-0.585	0.312
x91	-0.4272	0.724	-0.590	0.555	-1.846	0.991
x92	2.0504	2.099	0.977	0.329	-2.063	6.164
x93	4.4078	3.566	1.236	0.216	-2.582	

```
11.397
x94         -0.3755      0.055     -6.796      0.000      -0.484
-0.267
x95         -0.2764      0.052     -5.354      0.000      -0.378
-0.175
x96          1.6884      0.500      3.379      0.001       0.709
2.668
x97          5.1293      6.724      0.763      0.446      -8.050
18.309
x98         -0.1128      0.049     -2.312      0.021      -0.208
-0.017
x99        -12.9024      2.234     -5.776      0.000     -17.280
-8.524
x100         2.2658      0.689      3.287      0.001       0.915
3.617
x101         4.6505      6.610      0.704      0.482      -8.307
17.608
x102         0.0396      0.065      0.613      0.540      -0.087
0.166
x103       -13.7061     10.350     -1.324      0.185     -33.993
6.581
x104        -0.0764      0.229     -0.333      0.739      -0.525
0.373
x105        -0.5312      0.724     -0.733      0.463      -1.951
0.888
x106         2.3639      2.099      1.126      0.260      -1.750
6.478
x107         4.5611      3.568      1.278      0.201      -2.432
11.554
x108        -0.3823      0.055     -6.917      0.000      -0.491
-0.274
x109        -0.2663      0.052     -5.139      0.000      -0.368
-0.165
x110         1.6381      0.501      3.268      0.001       0.656
2.621
x111         7.2430      6.734      1.076      0.282      -5.957
20.443
x112        -0.1066      0.049     -2.178      0.029      -0.202
-0.011
x113       -10.6341      2.247     -4.732      0.000     -15.039
-6.230
x114         1.9949      0.690      2.890      0.004       0.642
3.348
x115         2.6262      6.622      0.397      0.692     -10.353
15.605
x116         0.0296      0.065      0.457      0.648      -0.097
0.157
x117       -10.4773     10.368     -1.011      0.312     -30.800
9.845
```

x118	-0.0967	0.229	-0.423	0.673	-0.545	0.352
x119	-0.5067	0.724	-0.699	0.484	-1.927	0.913
x120	2.8134	2.099	1.341	0.180	-1.300	6.927
x121	3.9524	3.576	1.105	0.269	-3.056	10.961
x122	-0.3728	0.055	-6.741	0.000	-0.481	-0.264
x123	-0.2275	0.052	-4.377	0.000	-0.329	-0.126
x124	1.4263	0.503	2.834	0.005	0.440	2.413
x125	7.4667	6.750	1.106	0.269	-5.765	20.698
x126	-0.1018	0.049	-2.076	0.038	-0.198	-0.006
x127	-8.7999	2.258	-3.897	0.000	-13.226	-4.373
x128	1.8563	0.692	2.683	0.007	0.500	3.212
x129	2.0702	6.625	0.312	0.755	-10.915	15.055
x130	0.0217	0.065	0.334	0.738	-0.106	0.149
x131	-5.1450	10.379	-0.496	0.620	-25.489	15.199
x132	-0.0579	0.229	-0.253	0.800	-0.507	0.391
x133	-0.3834	0.724	-0.529	0.597	-1.803	1.036
x134	2.2275	2.099	1.061	0.289	-1.887	6.342
x135	3.2209	3.575	0.901	0.368	-3.787	10.229
x136	-0.3848	0.055	-6.966	0.000	-0.493	-0.276
x137	-0.1958	0.052	-3.762	0.000	-0.298	-0.094
x138	1.2698	0.504	2.518	0.012	0.281	2.258
x139	10.0694	6.760	1.490	0.136	-3.180	23.319
x140	-0.0927	0.049	-1.886	0.059	-0.189	0.004
x141	-6.2933	2.271	-2.771	0.006	-10.745	-1.841
x142	1.8086	0.693	2.608	0.009	0.450	

```
3.168
x143              1.8443       6.623       0.278       0.781     -11.137
14.825
x144             -0.0069       0.065      -0.106       0.916      -0.135
0.121
x145             -5.0708      10.390      -0.488       0.626     -25.436
15.295
x146             -0.0833       0.229      -0.364       0.716      -0.532
0.366
x147             -0.4933       0.724      -0.681       0.496      -1.912
0.926
x148              1.3388       2.101       0.637       0.524      -2.780
5.458
x149              2.1763       3.578       0.608       0.543      -4.836
9.189
x150             -0.3829       0.055      -6.928       0.000      -0.491
-0.275
x151             -0.1500       0.052      -2.882       0.004      -0.252
-0.048
x152              1.0327       0.505       2.043       0.041       0.042
2.023
x153              8.4858       6.770       1.253       0.210      -4.785
21.756
x154             -0.0634       0.049      -1.287       0.198      -0.160
0.033
x155             -4.3992       2.283      -1.927       0.054      -8.874
0.076
x156              1.3358       0.695       1.923       0.054      -0.025
2.697
x157              1.8226       6.621       0.275       0.783     -11.156
14.801
x158              0.0017       0.065       0.026       0.979      -0.126
0.130
x159             -7.6734      10.392      -0.738       0.460     -28.044
12.697
x160             -0.0072       0.229      -0.031       0.975      -0.457
0.443
x161             -0.3715       0.724      -0.513       0.608      -1.792
1.048
x162              1.4676       2.103       0.698       0.485      -2.654
5.589
x163              1.9654       3.577       0.549       0.583      -5.047
8.977
x164             -0.3749       0.055      -6.779       0.000      -0.483
-0.267
x165             -0.1227       0.052      -2.359       0.018      -0.225
-0.021
x166              0.9971       0.506       1.970       0.049       0.005
1.989
```

x167	5.8206	6.775	0.859	0.390	-7.458	19.099
x168	-0.0344	0.049	-0.698	0.485	-0.131	0.062
x169	-2.7328	2.292	-1.192	0.233	-7.225	1.759
x170	0.8297	0.695	1.194	0.233	-0.532	2.192
x171	4.1320	6.624	0.624	0.533	-8.851	17.115
x172	0.0273	0.065	0.418	0.676	-0.101	0.156
x173	-6.4106	10.405	-0.616	0.538	-26.805	13.983
x174	0.0515	0.230	0.224	0.823	-0.399	0.502
x175	-0.1279	0.725	-0.176	0.860	-1.549	1.293
x176	1.3598	2.100	0.647	0.517	-2.757	5.477
x177	0.8711	3.579	0.243	0.808	-6.143	7.886
x178	-0.3654	0.055	-6.597	0.000	-0.474	-0.257
x179	-0.0816	0.052	-1.568	0.117	-0.184	0.020
x180	0.7633	0.506	1.507	0.132	-0.229	1.756
x181	3.9174	6.781	0.578	0.563	-9.374	17.208
x182	-0.0357	0.049	-0.723	0.469	-0.132	0.061
x183	-1.2055	2.294	-0.525	0.599	-5.703	3.292
x184	0.4223	0.696	0.607	0.544	-0.941	1.786
x185	2.2019	6.635	0.332	0.740	-10.803	15.207
x186	0.0225	0.066	0.343	0.732	-0.106	0.151
x187	-1.1606	10.399	-0.112	0.911	-21.544	19.222
x188	-0.0040	0.230	-0.017	0.986	-0.455	0.447
x189	-0.2719	0.724	-0.375	0.707	-1.692	1.148
x190	1.1216	2.100	0.534	0.593	-2.996	5.239
x191	1.5336	3.578	0.429	0.668	-5.479	

```
8.547
x192          -0.3778      0.055     -6.817      0.000      -0.486
-0.269
x193          -0.0407      0.052     -0.783      0.433      -0.143
0.061
x194           0.4769      0.507      0.941      0.346      -0.516
1.470
x195           2.3115      6.783      0.341      0.733     -10.984
15.607
x196          -0.0239      0.049     -0.485      0.628      -0.121
0.073
x197          -0.2498      2.293     -0.109      0.913      -4.745
4.246
x198           0.0700      0.696      0.101      0.920      -1.295
1.435
x199           0.8303      6.640      0.125      0.900     -12.185
13.845
x200           0.0039      0.066      0.059      0.953      -0.125
0.133
x201           4.0567     10.398      0.390      0.696     -16.324
24.437
x202           0.0744      0.230      0.323      0.746      -0.377
0.525
x203           0.1844      0.724      0.255      0.799      -1.235
1.604
x204           0.2693      2.099      0.128      0.898      -3.845
4.384
x205           1.5386      3.574      0.430      0.667      -5.467
8.544
x206          -0.3764      0.055     -6.791      0.000      -0.485
-0.268
x207          -0.0177      0.052     -0.340      0.734      -0.120
0.084
x208           0.2945      0.506      0.582      0.561      -0.698
1.287
x209           0.5148      6.783      0.076      0.940     -12.780
13.810
x210          -0.0206      0.049     -0.418      0.676      -0.117
0.076
x211           0.2289      2.294      0.100      0.921      -4.268
4.726
x212           0.0253      0.697      0.036      0.971      -1.341
1.392
x213           0.7268      6.638      0.109      0.913     -12.284
13.737
x214           0.0038      0.066      0.058      0.954      -0.125
0.133
x215           5.9335     10.379      0.572      0.568     -14.410
26.277
```

x216	0.0366	0.230	0.159	0.874	-0.414	0.487
x217	0.1033	0.724	0.143	0.887	-1.316	1.522
x218	-0.2131	2.101	-0.101	0.919	-4.331	3.905
x219	0.2786	3.569	0.078	0.938	-6.717	7.275
x220	-0.3844	0.055	-6.936	0.000	-0.493	-0.276
x221	0.0115	0.052	0.221	0.825	-0.091	0.114
x222	0.0242	0.507	0.048	0.962	-0.969	1.017
x223	-2.7317	6.777	-0.403	0.687	-16.016	10.553
x224	-0.0045	0.049	-0.092	0.927	-0.101	0.092
x225	0.8917	2.294	0.389	0.697	-3.604	5.388
x226	-0.2261	0.698	-0.324	0.746	-1.593	1.141
x227	0.4126	6.641	0.062	0.950	-12.604	13.429
x228	-0.0111	0.066	-0.169	0.866	-0.140	0.118
x229	4.4351	10.378	0.427	0.669	-15.906	24.776
x230	0.0145	0.230	0.063	0.950	-0.436	0.465
x231	-0.1002	0.723	-0.139	0.890	-1.517	1.317
x232	-0.4879	2.104	-0.232	0.817	-4.612	3.636
x233	-0.2328	3.567	-0.065	0.948	-7.225	6.759
x234	-0.3880	0.055	-7.004	0.000	-0.497	-0.279
x235	0.0384	0.052	0.737	0.461	-0.064	0.140
x236	0.0080	0.506	0.016	0.987	-0.985	1.001
x237	-3.6343	6.779	-0.536	0.592	-16.922	9.653
x238	0.0131	0.049	0.265	0.791	-0.084	0.110
x239	2.4893	2.291	1.087	0.277	-2.001	6.979
x240	-0.6263	0.697	-0.898	0.369	-1.993	0.740

```
x241             -1.3535       6.639      -0.204       0.838     -14.366
11.659
x242             -0.0310       0.066      -0.472       0.637      -0.160
0.098
x243              2.5591      10.369       0.247       0.805     -17.765
22.884
x244              0.1001       0.230       0.435       0.664      -0.351
0.551
x245              0.0334       0.724       0.046       0.963      -1.385
1.452
x246             -0.5095       2.104      -0.242       0.809      -4.634
3.615
x247             -0.3139       3.569      -0.088       0.930      -7.310
6.682
x248             -0.3913       0.055      -7.062       0.000      -0.500
-0.283
x249              0.0721       0.052       1.387       0.165      -0.030
0.174
x250             -0.1190       0.506      -0.235       0.814      -1.112
0.874
x251             -3.3842       6.770      -0.500       0.617     -16.655
9.887
x252              0.0348       0.049       0.707       0.480      -0.062
0.131
x253              3.7801       2.290       1.651       0.099      -0.708
8.268
x254             -0.8911       0.695      -1.283       0.200      -2.253
0.471
x255             -2.1465       6.646      -0.323       0.747     -15.173
10.880
x256             -0.0447       0.065      -0.684       0.494      -0.173
0.083
x257              4.8066      10.359       0.464       0.643     -15.498
25.111
x258              0.1080       0.230       0.470       0.639      -0.343
0.559
x259              0.0630       0.723       0.087       0.931      -1.354
1.480
x260             -0.6131       2.104      -0.291       0.771      -4.737
3.511
x261              0.8345       3.568       0.234       0.815      -6.158
7.827
x262             -0.4051       0.055      -7.312       0.000      -0.514
-0.297
x263              0.0886       0.052       1.703       0.089      -0.013
0.191
x264             -0.3742       0.507      -0.738       0.460      -1.368
0.619
x265             -4.9969       6.770      -0.738       0.460     -18.267
```

```
8.273
x266              0.0446       0.049       0.904       0.366      -0.052
0.141
x267              5.2140       2.278       2.289       0.022       0.749
9.679
x268             -1.2435       0.693      -1.794       0.073      -2.602
0.115
x269             -4.6603       6.640      -0.702       0.483     -17.675
8.355
x270             -0.0576       0.065      -0.882       0.378      -0.186
0.070
x271              6.5251      10.347       0.631       0.528     -13.756
26.806
x272              0.1131       0.230       0.491       0.623      -0.338
0.564
x273              0.0399       0.722       0.055       0.956      -1.375
1.455
x274             -1.1274       2.106      -0.535       0.592      -5.254
3.000
x275              0.7470       3.565       0.210       0.834      -6.242
7.736
x276             -0.4016       0.055      -7.248       0.000      -0.510
-0.293
x277              0.1379       0.052       2.651       0.008       0.036
0.240
x278             -0.5930       0.506      -1.172       0.241      -1.585
0.399
x279             -7.2725       6.754      -1.077       0.282     -20.511
5.966
x280              0.0430       0.049       0.874       0.382      -0.053
0.139
x281              6.4691       2.265       2.856       0.004       2.029
10.909
x282             -1.4789       0.692      -2.136       0.033      -2.836
-0.122
x283             -6.1314       6.639      -0.924       0.356     -19.144
6.881
x284             -0.0766       0.065      -1.177       0.239      -0.204
0.051
x285              5.9424      10.332       0.575       0.565     -14.308
26.193
x286              0.1537       0.230       0.668       0.504      -0.297
0.605
x287              0.1351       0.722       0.187       0.852      -1.280
1.550
x288             -1.5838       2.102      -0.753       0.451      -5.704
2.536
x289             -1.1352       3.561      -0.319       0.750      -8.114
5.844
```

x290	-0.3957	0.055	-7.147	0.000	-0.504	-0.287
x291	0.1755	0.052	3.372	0.001	0.073	0.277
x292	-0.7041	0.505	-1.394	0.163	-1.694	0.286
x293	-5.8743	6.740	-0.872	0.383	-19.085	7.336
x294	0.0449	0.049	0.914	0.361	-0.051	0.141
x295	7.9755	2.251	3.542	0.000	3.562	12.389
x296	-1.5593	0.692	-2.253	0.024	-2.916	-0.202
x297	-8.0959	6.639	-1.219	0.223	-21.109	4.917
x298	-0.0674	0.065	-1.039	0.299	-0.195	0.060
x299	7.9464	10.326	0.770	0.442	-12.294	28.186
x300	0.1026	0.230	0.446	0.656	-0.348	0.554
x301	0.3929	0.722	0.544	0.586	-1.022	1.807
x302	-1.8884	2.100	-0.899	0.368	-6.004	2.227
x303	-2.5268	3.561	-0.710	0.478	-9.506	4.453
x304	-0.3952	0.055	-7.139	0.000	-0.504	-0.287
x305	0.2048	0.052	3.935	0.000	0.103	0.307
x306	-0.9682	0.504	-1.920	0.055	-1.957	0.020
x307	-4.9093	6.728	-0.730	0.466	-18.096	8.278
x308	0.0439	0.049	0.895	0.371	-0.052	0.140
x309	9.1863	2.245	4.093	0.000	4.787	13.586
x310	-1.9824	0.691	-2.869	0.004	-3.337	-0.628
x311	-9.3473	6.633	-1.409	0.159	-22.348	3.653
x312	-0.0586	0.065	-0.905	0.366	-0.186	0.068
x313	14.1523	10.318	1.372	0.170	-6.073	34.377
x314	0.1593	0.230	0.692	0.489	-0.292	

```
0.610
x315          0.4220     0.721     0.585     0.559     -0.992
1.836
x316         -1.8293     2.100    -0.871     0.384     -5.945
2.287
x317         -3.4409     3.554    -0.968     0.333    -10.408
3.526
x318         -0.3955     0.055    -7.139     0.000     -0.504
-0.287
x319          0.2266     0.052     4.354     0.000      0.125
0.329
x320         -1.2564     0.503    -2.499     0.012     -2.242
-0.271
x321         -6.1372     6.723    -0.913     0.361    -19.315
7.041
x322          0.0505     0.049     1.033     0.302     -0.045
0.146
x323         11.3733     2.231     5.097     0.000      7.000
15.747
x324         -2.0226     0.690    -2.932     0.003     -3.375
-0.670
x325         -8.6984     6.627    -1.313     0.189    -21.688
4.292
x326         -0.0574     0.065    -0.889     0.374     -0.184
0.069
x327         13.1084    10.310     1.271     0.204     -7.100
33.316
x328          0.2055     0.230     0.894     0.371     -0.245
0.656
x329          0.4966     0.722     0.688     0.491     -0.918
1.911
x330         -2.0833     2.099    -0.992     0.321     -6.198
2.032
x331         -3.8273     3.554    -1.077     0.282    -10.793
3.139
x332         -0.4216     0.055    -7.616     0.000     -0.530
-0.313
x333          0.2320     0.052     4.473     0.000      0.130
0.334
x334         -1.6187     0.501    -3.234     0.001     -2.600
-0.638
x335         -8.0149     6.707    -1.195     0.232    -21.162
5.132
x336          0.0594     0.049     1.217     0.223     -0.036
0.155
x337         12.7181     2.226     5.713     0.000      8.354
17.082
x338         -2.2714     0.689    -3.298     0.001     -3.621
-0.922
```

x339	-9.8137	6.604	-1.486	0.137	-22.759	3.131
x340	-0.0506	0.064	-0.787	0.431	-0.177	0.075
x341	15.4825	10.297	1.504	0.133	-4.701	35.666
x342	0.1411	0.230	0.614	0.539	-0.309	0.591
x343	0.5671	0.722	0.785	0.432	-0.849	1.983
x344	-2.7366	2.097	-1.305	0.192	-6.847	1.374
x345	-4.7097	3.548	-1.327	0.184	-11.664	2.245
x346	-0.4177	0.055	-7.546	0.000	-0.526	-0.309
x347	0.2513	0.052	4.859	0.000	0.150	0.353
x348	-1.7749	0.499	-3.554	0.000	-2.754	-0.796
x349	-6.6469	6.695	-0.993	0.321	-19.770	6.476
x350	0.0914	0.049	1.879	0.060	-0.004	0.187
x351	14.4313	2.225	6.487	0.000	10.070	18.792
x352	-2.6525	0.686	-3.864	0.000	-3.998	-1.307
x353	-8.9625	6.601	-1.358	0.175	-21.901	3.976
x354	-0.0364	0.064	-0.567	0.571	-0.162	0.089
x355	21.0831	10.286	2.050	0.040	0.922	41.244
x356	0.1930	0.230	0.839	0.402	-0.258	0.644
x357	0.6841	0.722	0.947	0.344	-0.732	2.100
x358	-2.8710	2.096	-1.370	0.171	-6.980	1.238
x359	-5.9620	3.540	-1.684	0.092	-12.901	0.977
x360	-0.4186	0.055	-7.561	0.000	-0.527	-0.310
x361	0.2650	0.052	5.137	0.000	0.164	0.366
x362	-1.9258	0.498	-3.864	0.000	-2.903	-0.949
x363	-8.0115	6.678	-1.200	0.230	-21.102	

```
5.079
x364           0.1151      0.049      2.373      0.018       0.020
0.210
x365          15.5927      2.226      7.003      0.000      11.229
19.957
x366          -2.9812      0.685     -4.354      0.000      -4.323
-1.639
x367          -9.8792      6.594     -1.498      0.134     -22.805
3.046
x368          -0.0205      0.064     -0.321      0.748      -0.146
0.105
x369          23.5514     10.282      2.291      0.022       3.398
43.705
x370           0.1495      0.230      0.650      0.516      -0.301
0.600
x371           0.8624      0.722      1.195      0.232      -0.553
2.277
x372          -3.0413      2.097     -1.450      0.147      -7.151
1.069
x373          -6.0344      3.540     -1.705      0.088     -12.972
0.904
x374          -0.4090      0.055     -7.388      0.000      -0.518
-0.301
x375           0.3097      0.051      6.015      0.000       0.209
0.411
x376          -2.0970      0.497     -4.219      0.000      -3.071
-1.123
x377         -10.9813      6.658     -1.649      0.099     -24.031
2.069
x378           0.1141      0.048      2.355      0.019       0.019
0.209
x379          15.4593      2.228      6.938      0.000      11.092
19.826
x380          -3.1955      0.684     -4.671      0.000      -4.536
-1.855
x381          -9.3293      6.584     -1.417      0.157     -22.235
3.577
x382          -0.0052      0.064     -0.082      0.935      -0.130
0.120
x383          25.1040     10.278      2.442      0.015       4.958
45.250
x384           0.1483      0.230      0.646      0.519      -0.302
0.599
x385           0.7582      0.722      1.050      0.294      -0.658
2.174
x386          -3.3316      2.097     -1.588      0.112      -7.443
0.780
x387          -5.7604      3.539     -1.628      0.104     -12.697
1.177
```

```
x388          -0.4262      0.055     -7.698      0.000      -0.535
-0.318
x389           0.3474      0.052      6.745      0.000       0.246
0.448
x390          -2.1278      0.496     -4.290      0.000      -3.100
-1.156
x391         -11.7073      6.658     -1.759      0.079     -24.757
1.342
x392           0.1105      0.048      2.279      0.023       0.015
0.205
x393          14.0420      2.219      6.328      0.000       9.693
18.391
x394          -2.9618      0.685     -4.323      0.000      -4.305
-1.619
x395          -6.7109      6.588     -1.019      0.308     -19.623
6.201
x396          -0.0003      0.064     -0.005      0.996      -0.126
0.125
x397          24.8123     10.272      2.415      0.016       4.677
44.947
x398           0.1521      0.230      0.661      0.508      -0.299
0.603
x399           0.7531      0.723      1.042      0.298      -0.664
2.170
x400          -4.0909      2.098     -1.950      0.051      -8.204
0.022
x401          -6.7870      3.533     -1.921      0.055     -13.712
0.138
x402          -0.4044      0.055     -7.303      0.000      -0.513
-0.296
x403           0.3512      0.052      6.809      0.000       0.250
0.452
x404          -2.1831      0.496     -4.402      0.000      -3.155
-1.211
x405         -11.7425      6.655     -1.765      0.078     -24.786
1.301
x406           0.1295      0.048      2.673      0.008       0.035
0.224
x407          11.9818      2.157      5.555      0.000       7.754
16.210
x408          -2.4515      0.680     -3.607      0.000      -3.784
-1.119
x409          -3.4016      6.592     -0.516      0.606     -16.323
9.520
x410           0.0102      0.064      0.159      0.873      -0.115
0.135
x411          26.8862     10.279      2.616      0.009       6.738
47.035
x412           0.1732      0.230      0.752      0.452      -0.278
```

```
0.624
x413              0.9290        0.723         1.285         0.199        -0.489
2.347
x414             -4.4944        2.099        -2.141         0.032        -8.609
-0.380
x415             -8.2558        3.532        -2.338         0.019       -15.178
-1.333
x416             -0.3820        0.055        -6.897         0.000        -0.491
-0.273
x417              0.3157        0.051         6.163         0.000         0.215
0.416
x418             -2.0250        0.496        -4.085         0.000        -2.997
-1.053
x419            -10.3500        6.647        -1.557         0.119       -23.378
2.678
x420              0.1406        0.048         2.902         0.004         0.046
0.236
x421              7.3578        1.887         3.899         0.000         3.659
11.057
x422             -1.7508        0.627        -2.793         0.005        -2.979
-0.522
x423             -1.3445        6.383        -0.211         0.833       -13.855
11.166
x424              0.0040        0.060         0.068         0.946        -0.113
0.121
x425             21.1604       10.197         2.075         0.038         1.173
41.148
x426              0.2148        0.230         0.933         0.351        -0.237
0.666
x427              0.8189        0.724         1.131         0.258        -0.600
2.238
x428             -3.5362        2.098        -1.686         0.092        -7.648
0.575
x429             -7.5083        3.525        -2.130         0.033       -14.419
-0.598
x430             -0.3696        0.055        -6.674         0.000        -0.478
-0.261
x431              0.2364        0.048         4.975         0.000         0.143
0.329
x432             -1.5751        0.471        -3.341         0.001        -2.499
-0.651
x433             -7.4253        6.423        -1.156         0.248       -20.015
5.165
x434              0.1129        0.046         2.453         0.014         0.023
0.203
======================================================================
========
Omnibus:                     2014.889   Durbin-Watson:
0.024
```

```
Prob(Omnibus):                    0.000   Jarque-Bera (JB):
3114.102
Skew:                             0.836   Prob(JB):
0.00
Kurtosis:                         4.211   Cond. No.
3.27e+07
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 3.27e+07. This might indicate that
there are
strong multicollinearity or other numerical problems.
"""
```

7.6. Multicolinearidade

```
from statsmodels.stats.outliers_influence import
variance_inflation_factor as vif
X_train = add_lagged_variables(intermediate_df, 14, remaining_sensors)
X_train = X_train.drop(index_names, axis=1)

# observação, isso leva alguns minutos para calcular
vifs = {X_train.columns[i]:round(vif(X_train.values, i), 2) for i in
range(len(X_train.columns))}
display(vifs)

{'s_2': 4081069.37,
 's_3': 148827.31,
 's_4': 34450.38,
 's_7': 676636.48,
 's_8': 2591641803.36,
 's_9': 1580259.79,
 's_11': 28779.21,
 's_12': 686940.38,
 's_13': 2480791096.44,
 's_14': 1724157.4,
 's_15': 116497.33,
 's_17': 152527.42,
 's_20': 113760.29,
 's_21': 119734.69,
 'RUL': 5.11,
 's_11_lag_1': 36657.17,
 's_12_lag_1': 785930.67,
 's_13_lag_1': 2613853860.37,
 's_14_lag_1': 1926076.65,
 's_15_lag_1': 116696.91,
```

```
 's_17_lag_1': 152525.25,
 's_2_lag_1': 4074895.76,
 's_20_lag_1': 113444.5,
 's_21_lag_1': 119475.96,
 's_3_lag_1': 148979.81,
 's_4_lag_1': 38889.32,
 's_7_lag_1': 733701.46,
 's_8_lag_1': 2748462855.3,
 's_9_lag_1': 1716566.66,
 's_11_lag_2': 37805.01,
 's_12_lag_2': 780833.03,
 's_13_lag_2': 2620758747.64,
 's_14_lag_2': 1917277.53,
 's_15_lag_2': 116346.14,
 's_17_lag_2': 152686.28,
 's_2_lag_2': 4072573.42,
 's_20_lag_2': 113685.05,
 's_21_lag_2': 119653.41,
 's_3_lag_2': 149009.07,
 's_4_lag_2': 38847.79,
 's_7_lag_2': 730031.66,
 's_8_lag_2': 2753903979.86,
 's_9_lag_2': 1701499.91,
 's_11_lag_3': 37105.7,
 's_12_lag_3': 773526.33,
 's_13_lag_3': 2618785248.58,
 's_14_lag_3': 1896597.22,
 's_15_lag_3': 116127.43,
 's_17_lag_3': 152610.2,
 's_2_lag_3': 4068060.65,
 's_20_lag_3': 114361.87,
 's_21_lag_3': 119613.2,
 's_3_lag_3': 148800.49,
 's_4_lag_3': 38497.52,
 's_7_lag_3': 728089.62,
 's_8_lag_3': 2749195207.39,
 's_9_lag_3': 1683956.51,
 's_11_lag_4': 36059.48,
 's_12_lag_4': 771185.41,
 's_13_lag_4': 2624330020.99,
 's_14_lag_4': 1886173.65,
 's_15_lag_4': 115823.95,
 's_17_lag_4': 153272.33,
 's_2_lag_4': 4062599.0,
 's_20_lag_4': 114088.67,
 's_21_lag_4': 119312.75,
 's_3_lag_4': 148722.78,
 's_4_lag_4': 38584.48,
 's_7_lag_4': 729044.21,
```

's_8_lag_4': 2743933140.15,
's_9_lag_4': 1683268.62,
's_11_lag_5': 35244.87,
's_12_lag_5': 772466.32,
's_13_lag_5': 2641501136.77,
's_14_lag_5': 1889391.77,
's_15_lag_5': 115853.68,
's_17_lag_5': 153607.6,
's_2_lag_5': 4063127.86,
's_20_lag_5': 113685.06,
's_21_lag_5': 119507.78,
's_3_lag_5': 148932.1,
's_4_lag_5': 38679.79,
's_7_lag_5': 727778.41,
's_8_lag_5': 2735546298.33,
's_9_lag_5': 1681368.16,
's_11_lag_6': 34798.69,
's_12_lag_6': 774980.99,
's_13_lag_6': 2650576379.39,
's_14_lag_6': 1891928.65,
's_15_lag_6': 115892.67,
's_17_lag_6': 153120.03,
's_2_lag_6': 4056163.35,
's_20_lag_6': 113991.08,
's_21_lag_6': 119621.05,
's_3_lag_6': 149196.52,
's_4_lag_6': 38921.14,
's_7_lag_6': 728141.57,
's_8_lag_6': 2737289603.74,
's_9_lag_6': 1680383.52,
's_11_lag_7': 34504.61,
's_12_lag_7': 774738.12,
's_13_lag_7': 2664214961.26,
's_14_lag_7': 1890207.16,
's_15_lag_7': 116125.94,
's_17_lag_7': 153305.33,
's_2_lag_7': 4053766.67,
's_20_lag_7': 114005.35,
's_21_lag_7': 119658.72,
's_3_lag_7': 149094.91,
's_4_lag_7': 38889.24,
's_7_lag_7': 724862.22,
's_8_lag_7': 2726961215.99,
's_9_lag_7': 1681104.36,
's_11_lag_8': 34722.55,
's_12_lag_8': 773448.24,
's_13_lag_8': 2657202566.2,
's_14_lag_8': 1892490.36,
's_15_lag_8': 116075.99,

```
 's_17_lag_8': 153127.73,
 's_2_lag_8': 4057249.05,
 's_20_lag_8': 113882.63,
 's_21_lag_8': 119457.58,
 's_3_lag_8': 149090.83,
 's_4_lag_8': 38831.86,
 's_7_lag_8': 726609.31,
 's_8_lag_8': 2732378792.92,
 's_9_lag_8': 1676326.04,
 's_11_lag_9': 35139.75,
 's_12_lag_9': 769744.07,
 's_13_lag_9': 2649938498.41,
 's_14_lag_9': 1894940.8,
 's_15_lag_9': 116156.64,
 's_17_lag_9': 153503.58,
 's_2_lag_9': 4061020.86,
 's_20_lag_9': 113625.1,
 's_21_lag_9': 119239.28,
 's_3_lag_9': 148797.63,
 's_4_lag_9': 38674.95,
 's_7_lag_9': 726300.5,
 's_8_lag_9': 2724589688.57,
 's_9_lag_9': 1678515.58,
 's_11_lag_10': 35884.42,
 's_12_lag_10': 770020.38,
 's_13_lag_10': 2630826238.38,
 's_14_lag_10': 1895242.94,
 's_15_lag_10': 116047.01,
 's_17_lag_10': 153038.83,
 's_2_lag_10': 4060044.03,
 's_20_lag_10': 114102.31,
 's_21_lag_10': 118944.38,
 's_3_lag_10': 148624.82,
 's_4_lag_10': 38519.52,
 's_7_lag_10': 725865.13,
 's_8_lag_10': 2726371041.82,
 's_9_lag_10': 1681292.35,
 's_11_lag_11': 36892.92,
 's_12_lag_11': 773025.62,
 's_13_lag_11': 2608847308.54,
 's_14_lag_11': 1910711.51,
 's_15_lag_11': 116061.38,
 's_17_lag_11': 152730.91,
 's_2_lag_11': 4062510.43,
 's_20_lag_11': 114335.75,
 's_21_lag_11': 119145.99,
 's_3_lag_11': 148282.2,
 's_4_lag_11': 38674.45,
 's_7_lag_11': 725708.69,
```

```
 's_8_lag_11': 2729653667.05,
 's_9_lag_11': 1680967.62,
 's_11_lag_12': 37626.5,
 's_12_lag_12': 782627.0,
 's_13_lag_12': 2615374839.28,
 's_14_lag_12': 1932547.11,
 's_15_lag_12': 116172.02,
 's_17_lag_12': 153241.62,
 's_2_lag_12': 4062099.93,
 's_20_lag_12': 113786.28,
 's_21_lag_12': 119052.67,
 's_3_lag_12': 148384.8,
 's_4_lag_12': 39051.6,
 's_7_lag_12': 728254.64,
 's_8_lag_12': 2742823506.32,
 's_9_lag_12': 1694380.25,
 's_11_lag_13': 36642.27,
 's_12_lag_13': 786005.95,
 's_13_lag_13': 2642486873.59,
 's_14_lag_13': 1947122.52,
 's_15_lag_13': 116420.99,
 's_17_lag_13': 152905.08,
 's_2_lag_13': 4055096.01,
 's_20_lag_13': 113656.05,
 's_21_lag_13': 119027.09,
 's_3_lag_13': 148386.05,
 's_4_lag_13': 39132.88,
 's_7_lag_13': 732498.08,
 's_8_lag_13': 2763723655.45,
 's_9_lag_13': 1709258.03,
 's_11_lag_14': 28840.05,
 's_12_lag_14': 687369.03,
 's_13_lag_14': 2525867132.5,
 's_14_lag_14': 1747444.47,
 's_15_lag_14': 115928.84,
 's_17_lag_14': 153167.49,
 's_2_lag_14': 4062462.88,
 's_20_lag_14': 114126.98,
 's_21_lag_14': 119271.89,
 's_3_lag_14': 147985.56,
 's_4_lag_14': 34630.32,
 's_7_lag_14': 676496.63,
 's_8_lag_14': 2629757810.34,
 's_9_lag_14': 1575418.83}

from sklearn.preprocessing import StandardScaler

intermediate_df = train.drop(drop_labels, axis=1)
scaler = StandardScaler()
scaler.fit(intermediate_df[remaining_sensors])
```

```
intermediate_df[remaining_sensors] =
scaler.transform(intermediate_df[remaining_sensors])

intermediate_df = make_stationary(intermediate_df, remaining_sensors)

X_train = add_lagged_variables(intermediate_df, 14, remaining_sensors)
X_train = X_train.drop(index_names, axis=1)

vifs = {X_train.columns[i]:round(vif(X_train.values, i), 2) for i in
range(len(X_train.columns))}
display(vifs)
```

```
{'s_2': 2.58,
 's_3': 2.31,
 's_4': 2.09,
 's_7': 2.21,
 's_8': 2.39,
 's_9': 4.15,
 's_11': 1.9,
 's_12': 1.97,
 's_13': 2.37,
 's_14': 4.75,
 's_15': 2.55,
 's_17': 2.47,
 's_20': 2.64,
 's_21': 2.72,
 'RUL': 2.34,
 's_11_lag_1': 2.39,
 's_12_lag_1': 2.22,
 's_13_lag_1': 2.47,
 's_14_lag_1': 5.27,
 's_15_lag_1': 2.51,
 's_17_lag_1': 2.42,
 's_2_lag_1': 2.54,
 's_20_lag_1': 2.59,
 's_21_lag_1': 2.67,
 's_3_lag_1': 2.27,
 's_4_lag_1': 2.32,
 's_7_lag_1': 2.36,
 's_8_lag_1': 2.51,
 's_9_lag_1': 4.48,
 's_11_lag_2': 2.42,
 's_12_lag_2': 2.18,
 's_13_lag_2': 2.45,
 's_14_lag_2': 5.22,
 's_15_lag_2': 2.47,
 's_17_lag_2': 2.39,
 's_2_lag_2': 2.49,
 's_20_lag_2': 2.55,
 's_21_lag_2': 2.63,
```

```
 's_3_lag_2': 2.24,
 's_4_lag_2': 2.29,
 's_7_lag_2': 2.31,
 's_8_lag_2': 2.48,
 's_9_lag_2': 4.42,
 's_11_lag_3': 2.35,
 's_12_lag_3': 2.13,
 's_13_lag_3': 2.42,
 's_14_lag_3': 5.14,
 's_15_lag_3': 2.43,
 's_17_lag_3': 2.35,
 's_2_lag_3': 2.45,
 's_20_lag_3': 2.53,
 's_21_lag_3': 2.59,
 's_3_lag_3': 2.2,
 's_4_lag_3': 2.24,
 's_7_lag_3': 2.28,
 's_8_lag_3': 2.46,
 's_9_lag_3': 4.36,
 's_11_lag_4': 2.28,
 's_12_lag_4': 2.11,
 's_13_lag_4': 2.41,
 's_14_lag_4': 5.09,
 's_15_lag_4': 2.39,
 's_17_lag_4': 2.33,
 's_2_lag_4': 2.41,
 's_20_lag_4': 2.48,
 's_21_lag_4': 2.55,
 's_3_lag_4': 2.17,
 's_4_lag_4': 2.21,
 's_7_lag_4': 2.26,
 's_8_lag_4': 2.43,
 's_9_lag_4': 4.35,
 's_11_lag_5': 2.23,
 's_12_lag_5': 2.1,
 's_13_lag_5': 2.41,
 's_14_lag_5': 5.08,
 's_15_lag_5': 2.36,
 's_17_lag_5': 2.29,
 's_2_lag_5': 2.38,
 's_20_lag_5': 2.44,
 's_21_lag_5': 2.51,
 's_3_lag_5': 2.14,
 's_4_lag_5': 2.2,
 's_7_lag_5': 2.24,
 's_8_lag_5': 2.42,
 's_9_lag_5': 4.33,
 's_11_lag_6': 2.2,
 's_12_lag_6': 2.09,
```

```
 'S_13_lag_6': 2.4,
 's_14_lag_6': 5.07,
 's_15_lag_6': 2.33,
 's_17_lag_6': 2.26,
 's_2_lag_6': 2.35,
 's_20_lag_6': 2.41,
 's_21_lag_6': 2.47,
 's_3_lag_6': 2.11,
 's_4_lag_6': 2.19,
 's_7_lag_6': 2.22,
 's_8_lag_6': 2.4,
 's_9_lag_6': 4.33,
 's_11_lag_7': 2.19,
 's_12_lag_7': 2.08,
 's_13_lag_7': 2.4,
 's_14_lag_7': 5.05,
 's_15_lag_7': 2.3,
 's_17_lag_7': 2.23,
 's_2_lag_7': 2.32,
 's_20_lag_7': 2.38,
 's_21_lag_7': 2.44,
 's_3_lag_7': 2.09,
 's_4_lag_7': 2.17,
 's_7_lag_7': 2.18,
 's_8_lag_7': 2.38,
 's_9_lag_7': 4.33,
 's_11_lag_8': 2.21,
 's_12_lag_8': 2.07,
 's_13_lag_8': 2.38,
 's_14_lag_8': 5.04,
 's_15_lag_8': 2.27,
 's_17_lag_8': 2.2,
 's_2_lag_8': 2.28,
 's_20_lag_8': 2.34,
 's_21_lag_8': 2.41,
 's_3_lag_8': 2.06,
 's_4_lag_8': 2.15,
 's_7_lag_8': 2.17,
 's_8_lag_8': 2.37,
 's_9_lag_8': 4.31,
 's_11_lag_9': 2.24,
 's_12_lag_9': 2.05,
 's_13_lag_9': 2.35,
 's_14_lag_9': 5.03,
 's_15_lag_9': 2.25,
 's_17_lag_9': 2.17,
 's_2_lag_9': 2.26,
 's_20_lag_9': 2.31,
 's_21_lag_9': 2.37,
```

```
 's_3_lag_9': 2.03,
 's_4_lag_9': 2.13,
 's_7_lag_9': 2.14,
 's_8_lag_9': 2.35,
 's_9_lag_9': 4.31,
 's_11_lag_10': 2.29,
 's_12_lag_10': 2.04,
 's_13_lag_10': 2.33,
 's_14_lag_10': 5.02,
 's_15_lag_10': 2.21,
 's_17_lag_10': 2.14,
 's_2_lag_10': 2.23,
 's_20_lag_10': 2.29,
 's_21_lag_10': 2.34,
 's_3_lag_10': 2.01,
 's_4_lag_10': 2.11,
 's_7_lag_10': 2.12,
 's_8_lag_10': 2.34,
 's_9_lag_10': 4.31,
 's_11_lag_11': 2.34,
 's_12_lag_11': 2.03,
 's_13_lag_11': 2.3,
 's_14_lag_11': 5.05,
 's_15_lag_11': 2.19,
 's_17_lag_11': 2.11,
 's_2_lag_11': 2.2,
 's_20_lag_11': 2.26,
 's_21_lag_11': 2.31,
 's_3_lag_11': 1.98,
 's_4_lag_11': 2.1,
 's_7_lag_11': 2.1,
 's_8_lag_11': 2.33,
 's_9_lag_11': 4.3,
 's_11_lag_12': 2.36,
 's_12_lag_12': 2.04,
 's_13_lag_12': 2.29,
 's_14_lag_12': 5.1,
 's_15_lag_12': 2.16,
 's_17_lag_12': 2.09,
 's_2_lag_12': 2.18,
 's_20_lag_12': 2.23,
 's_21_lag_12': 2.28,
 's_3_lag_12': 1.96,
 's_4_lag_12': 2.09,
 's_7_lag_12': 2.08,
 's_8_lag_12': 2.32,
 's_9_lag_12': 4.33,
 's_11_lag_13': 2.25,
 's_12_lag_13': 2.02,
```

```
 's_13_lag_13': 2.29,
 's_14_lag_13': 5.11,
 's_15_lag_13': 2.14,
 's_17_lag_13': 2.07,
 's_2_lag_13': 2.15,
 's_20_lag_13': 2.2,
 's_21_lag_13': 2.25,
 's_3_lag_13': 1.94,
 's_4_lag_13': 2.06,
 's_7_lag_13': 2.07,
 's_8_lag_13': 2.32,
 's_9_lag_13': 4.35,
 's_11_lag_14': 1.73,
 's_12_lag_14': 1.73,
 's_13_lag_14': 2.15,
 's_14_lag_14': 4.54,
 's_15_lag_14': 2.09,
 's_17_lag_14': 2.05,
 's_2_lag_14': 2.14,
 's_20_lag_14': 2.18,
 's_21_lag_14': 2.22,
 's_3_lag_14': 1.92,
 's_4_lag_14': 1.78,
 's_7_lag_14': 1.87,
 's_8_lag_14': 2.18,
 's_9_lag_14': 3.98}
```

7.7. Testando e avaliando o modelo

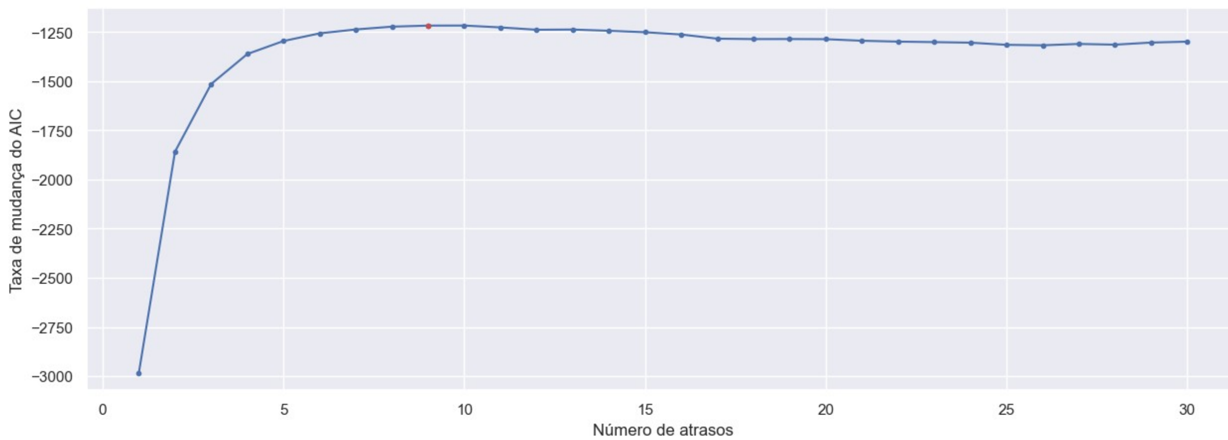
```
# primeiro vamos reexaminar a quantidade ideal de atrasos como fizemos
antes
# execute etapas de processamento de dados para garantir que estamos
trabalhando com os dados processados corretamente
intermediate_df = train.drop(drop_labels, axis=1)
scaler = StandardScaler()
scaler.fit(intermediate_df[remaining_sensors])
intermediate_df[remaining_sensors] =
scaler.transform(intermediate_df[remaining_sensors])

intermediate_df = make_stationary(intermediate_df, remaining_sensors)

# calcular as metricas do modelo
metrics = pd.DataFrame(columns=['rmse', 'AIC', 'BIC'])
nr_of_lags = 30
for i in range(0, nr_of_lags+1):
    X_train = add_lagged_variables(intermediate_df, i,
remaining_sensors)
    X_train = X_train.drop(index_names, axis=1)
    y_train = X_train.pop('RUL')
```

```python
    model = sm.OLS(y_train, sm.add_constant(X_train.values))
    result = model.fit()

    metrics = pd.concat([metrics, pd.DataFrame(
        data=[[np.sqrt(result.mse_resid), round(result.aic,2),
round(result.bic,2)]],columns=['rmse', 'AIC', 'BIC']
        )],
                        ignore_index=True)

display(metrics)
```

	rmse	AIC	BIC
0	45.311459	214892.78	215011.72
1	43.199826	211909.99	212139.82
2	42.321539	210051.72	210392.28
3	41.806392	208537.26	208988.42
4	41.449199	207175.30	207736.92
5	41.159315	205879.37	206551.31
6	40.908577	204622.43	205404.54
7	40.677353	203385.46	204277.60
8	40.459553	202162.57	203164.60
9	40.246294	200945.11	202056.89
10	40.032297	199727.93	200949.31
11	39.807564	198501.28	199832.11
12	39.569598	197262.72	198702.85
13	39.331775	196025.28	197574.57
14	39.086446	194781.51	196439.79
15	38.832622	193530.39	195297.52
16	38.565998	192267.69	194143.52
17	38.276907	190983.82	192968.19
18	37.984763	189697.81	191790.56
19	37.692030	188412.06	190613.04
20	37.397693	187125.56	189434.62
21	37.094592	185831.20	188248.16
22	36.786163	184532.32	187057.03
23	36.474461	183230.92	185863.23
24	36.159212	181926.72	184666.44
25	35.832347	180611.53	183458.52
26	35.502479	179293.95	182248.03
27	35.178855	177983.30	181044.31
28	34.850913	176668.91	179836.67
29	34.532843	175365.21	178639.55
30	34.218579	174066.08	177446.83

```python
plt.figure(figsize=(15,5))
plt.plot(metrics['AIC'].diff(), marker='.')  # traca a diferença para
ver onde a curva se achata
plt.plot(9, metrics['AIC'].diff()[9], '.r')
plt.xlabel("Número de atrasos")
```

```
plt.ylabel("Taxa de mudança do AIC")
plt.show()
plt.close()
```




```
# Treinar e avaliar o modelo com 0 a 9 atrasos inseridos
lags = 9

# Preparo dos dados
X_train_interim = train.drop(drop_labels, axis=1)
X_train_interim[remaining_sensors] =
scaler.transform(X_train_interim[remaining_sensors])
X_train_interim = make_stationary(X_train_interim, remaining_sensors)
X_train_interim = add_lagged_variables(X_train_interim, lags,
remaining_sensors)
X_train_interim = sm.add_constant(X_train_interim)
X_train = X_train_interim.drop(index_names, axis=1)
y_train = X_train.pop("RUL")

X_test_interim = test.drop(drop_labels, axis=1)
X_test_interim[remaining_sensors] =
scaler.transform(X_test_interim[remaining_sensors])
X_test_interim = make_stationary(X_test_interim, remaining_sensors)
X_test_interim = add_lagged_variables(X_test_interim, lags,
remaining_sensors)
X_test_interim =
X_test_interim.groupby('unidade').last().reset_index()
X_test_interim = sm.add_constant(X_test_interim)
X_test = X_test_interim.drop(index_names, axis=1)

# Ajuste do modelo
model = sm.OLS(y_train.clip(upper=125), X_train)  # apply clipped RUL
from last post
model_fitted = model.fit()

# Teste
```

```
y_hat_train = model_fitted.predict(X_train)
y_hat = model_fitted.predict(X_test)

# Avaliar
avaliar(y_train.clip(upper=125), y_hat_train, 'treino')
avaliar(y_test, y_hat)
```

```
conjunto de treino -> RMSE:20.80233928213837, R2:0.7523208224684113
conjunto de teste -> RMSE:21.148958570124005, R2:0.7409888687595046
```

```
# Treinar e avaliar modelo com atrasos específicos
specific_lags = [1,2,3,4,5,10,20]

# Preparo dos dados
X_train_interim = train.drop(drop_labels, axis=1)
X_train_interim[remaining_sensors] =
scaler.transform(X_train_interim[remaining_sensors])
X_train_interim = make_stationary(X_train_interim, remaining_sensors)
X_train_interim = add_specific_lags(X_train_interim, specific_lags,
remaining_sensors)
X_train_interim = sm.add_constant(X_train_interim)
X_train = X_train_interim.drop(index_names, axis=1)
y_train = X_train.pop("RUL")

X_test_interim = test.drop(drop_labels, axis=1)
X_test_interim[remaining_sensors] =
scaler.transform(X_test_interim[remaining_sensors])
X_test_interim = make_stationary(X_test_interim, remaining_sensors)
X_test_interim = add_specific_lags(X_test_interim, specific_lags,
remaining_sensors)
X_test_interim =
X_test_interim.groupby('unidade').last().reset_index()
X_test_interim = sm.add_constant(X_test_interim)
X_test = X_test_interim.drop(index_names, axis=1)

# Ajuste do modelo
model = sm.OLS(y_train.clip(upper=125), X_train)
model_fitted = model.fit()

# Testar o modelo
y_hat_train = model_fitted.predict(X_train)
y_hat = model_fitted.predict(X_test)

# Avaliar
avaliar(y_train.clip(upper=125), y_hat_train, 'treino')
avaliar(y_test, y_hat)
```

```
conjunto de treino -> RMSE:20.742950740267165, R2:0.7542608298904978
conjunto de teste -> RMSE:20.85223486440749, R2:0.7482058292992063
```

8. Análise de sobrevivência

8.1. Carregando a base de dados

```
# Definir o caminho para os dados
dir_path = './CMAPSSData/'

# Definir o nome das colunas
index_names = ['unidade', 'ciclo_tempo']
setting_names = ['config_1', 'config_2', 'config_3']
sensor_names = ['s_{}'.format(i) for i in range(1,22)]
col_names = index_names + setting_names + sensor_names


# Ker os dados
train = pd.read_csv((dir_path+'train_FD001.txt'), sep='\s+',
header=None, names=col_names)
test = pd.read_csv((dir_path+'test_FD001.txt'), sep='\s+',
header=None, names=col_names)
y_test = pd.read_csv((dir_path+'RUL_FD001.txt'), sep='\s+',
header=None, names=['RUL'])

train.head()
```

```
   unidade  ciclo_tempo  config_1  config_2  config_3     s_1     s_2
\
0        1            1   -0.0007   -0.0004     100.0  518.67  641.82

1        1            2    0.0019   -0.0003     100.0  518.67  642.15

2        1            3   -0.0043    0.0003     100.0  518.67  642.35

3        1            4    0.0007    0.0000     100.0  518.67  642.35

4        1            5   -0.0019   -0.0002     100.0  518.67  642.37


       s_3      s_4    s_5  ...     s_12     s_13     s_14    s_15
s_16  s_17  \
0  1589.70  1400.60  14.62  ...   521.66  2388.02  8138.62  8.4195
0.03   392
1  1591.82  1403.14  14.62  ...   522.28  2388.07  8131.49  8.4318
0.03   392
2  1587.99  1404.20  14.62  ...   522.42  2388.03  8133.23  8.4178
0.03   390
3  1582.79  1401.87  14.62  ...   522.86  2388.08  8133.83  8.3682
0.03   392
4  1582.85  1406.22  14.62  ...   522.19  2388.04  8133.80  8.4294
0.03   393
```

```
   s_18   s_19   s_20     s_21
0  2388  100.0  39.06  23.4190
1  2388  100.0  39.00  23.4236
2  2388  100.0  38.95  23.3442
3  2388  100.0  38.88  23.3739
4  2388  100.0  38.90  23.4044

[5 rows x 26 columns]
```

```
train = add_RUL(train)  # Adicionar o RUL
display(train[index_names+['RUL']].head())
```

```
   unidade  ciclo_tempo  RUL
0        1            1  191
1        1            2  190
2        1            3  189
3        1            4  188
4        1            5  187
```

```
# Corte do RUL, conforme visto anteriormente
train['RUL'].clip(upper=125, inplace=True)

# Remocao dos parametros nao significativos
drop_sensors = ['s_1','s_5','s_6','s_10','s_16','s_18','s_19']
drop_labels = setting_names + drop_sensors
train.drop(labels=drop_labels, axis=1, inplace=True)

remaining_sensors = ['s_2', 's_3', 's_4', 's_7', 's_8', 's_9',
       's_11', 's_12', 's_13', 's_14', 's_15', 's_17', 's_20', 's_21']

print(train.columns)
```

```
Index(['unidade', 'ciclo_tempo', 's_2', 's_3', 's_4', 's_7', 's_8',
's_9',
       's_11', 's_12', 's_13', 's_14', 's_15', 's_17', 's_20', 's_21',
'RUL'],
      dtype='object')
```

8.2. Preparação dos dados

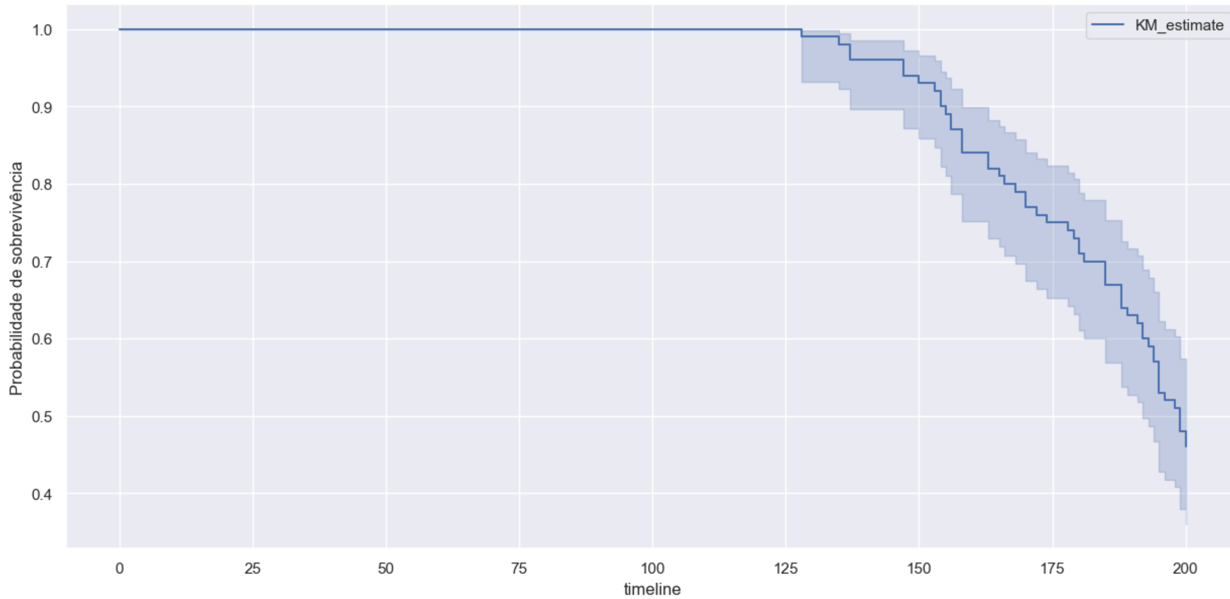
```
train['falha'] = 0
idx_ultimo_registro = train.reset_index().groupby(by='unidade')
['index'].last()
for i in idx_ultimo_registro:
    train.at[i, 'falha'] = 1

train['start'] = train['ciclo_tempo'] - 1
train.tail() # varificar os dados
```

```
       unidade  ciclo_tempo     s_2      s_3      s_4     s_7      s_8
\
```

```
20626       100          196  643.49  1597.98  1428.63  551.43  2388.19

20627       100          197  643.54  1604.50  1433.58  550.86  2388.23

20628       100          198  643.42  1602.46  1428.18  550.94  2388.24

20629       100          199  643.23  1605.26  1426.53  550.68  2388.25

20630       100          200  643.85  1600.38  1432.14  550.79  2388.26


           s_9   s_11    s_12     s_13     s_14    s_15  s_17   s_20
s_21  \
20626  9065.52  48.07  519.49  2388.26  8137.60  8.4956   397  38.49
22.9735
20627  9065.11  48.04  519.68  2388.22  8136.50  8.5139   395  38.30
23.1594
20628  9065.90  48.09  520.01  2388.24  8141.05  8.5646   398  38.44
22.9333
20629  9073.72  48.39  519.67  2388.23  8139.29  8.5389   395  38.29
23.0640
20630  9061.48  48.20  519.30  2388.26  8137.33  8.5036   396  38.37
23.0522

       RUL  falha  start
20626    4      0    195
20627    3      0    196
20628    2      0    197
20629    1      0    198
20630    0      1    199

cut_off = 200
train_censored = train[train['ciclo_tempo'] <=
cut_off].copy().reset_index(drop=True)
```

8.3. Curva de KaplanMeier

```
# Criando a curva
data = train_censored[index_names+['falha']].groupby('unidade').last()

plt.figure(figsize=(15,7))
survival = KaplanMeierFitter()
survival.fit(data['ciclo_tempo'], data['falha'])
survival.plot()
plt.ylabel("Probabilidade de sobrevivência")
plt.show()
plt.close()
```



8.4. Modelos de riscos proporcionais de Cox (Cox Proportional Hazards models) - CoxTimeVaryingFitter

```
train_cols = index_names + remaining_sensors + ['start', 'falha']
predict_cols = ['ciclo_tempo'] + remaining_sensors + ['start',
'falha']  # valor que indica falha sera 0

ctv = CoxTimeVaryingFitter()
ctv.fit(train_censored[train_cols], id_col="unidade",
event_col='falha',
        start_col='start', stop_col='ciclo_tempo', show_progress=True)
```

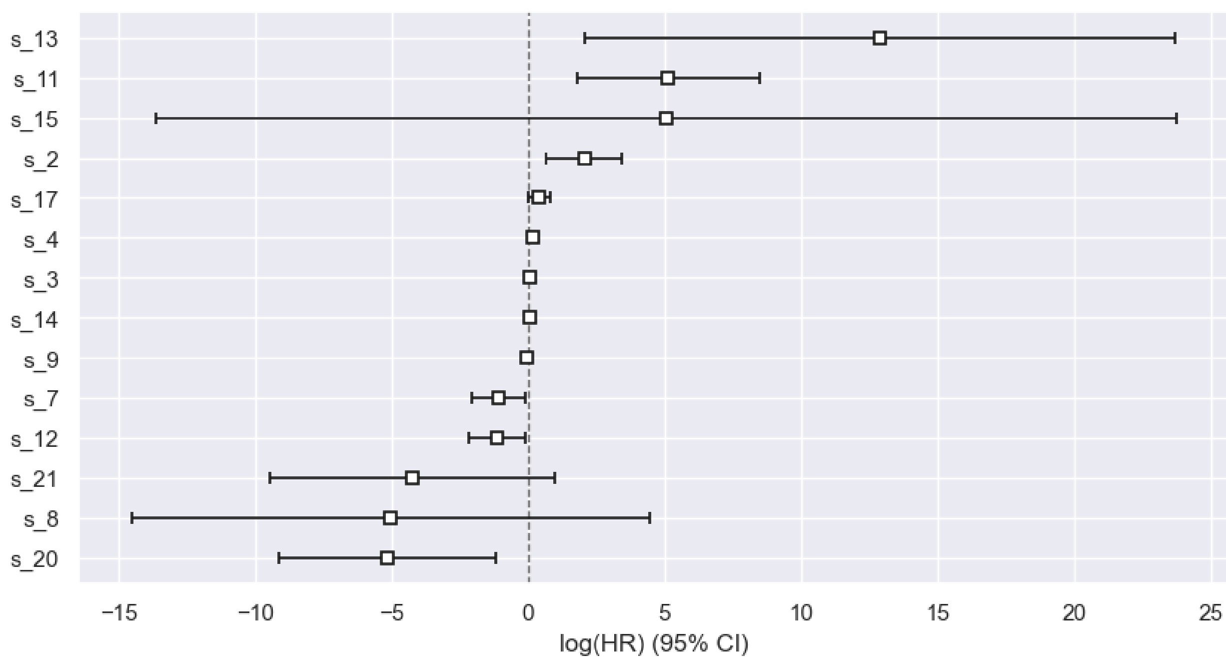
```
Iteration 1: norm_delta = 9.03e-01, step_size = 0.9500, log_lik = -
230.78680, newton_decrement = 1.07e+02, seconds_since_start = 0.0
Iteration 2: norm_delta = 7.17e-01, step_size = 0.9500, log_lik = -
109.79194, newton_decrement = 2.32e+01, seconds_since_start = 0.0
Iteration 3: norm_delta = 6.79e-01, step_size = 0.9500, log_lik = -
81.15203, newton_decrement = 1.03e+01, seconds_since_start = 0.0
Iteration 4: norm_delta = 6.29e-01, step_size = 1.0000, log_lik = -
68.53253, newton_decrement = 3.52e+00, seconds_since_start = 0.0
Iteration 5: norm_delta = 3.05e-01, step_size = 1.0000, log_lik = -
64.35996, newton_decrement = 4.53e-01, seconds_since_start = 0.0
Iteration 6: norm_delta = 4.80e-02, step_size = 1.0000, log_lik = -
63.86912, newton_decrement = 8.90e-03, seconds_since_start = 0.1
Iteration 7: norm_delta = 9.95e-04, step_size = 1.0000, log_lik = -
63.86010, newton_decrement = 3.78e-06, seconds_since_start = 0.1
Iteration 8: norm_delta = 4.27e-07, step_size = 1.0000, log_lik = -
63.86010, newton_decrement = 7.09e-13, seconds_since_start = 0.1
Convergence completed after 8 iterations.
```

```
<lifelines.CoxTimeVaryingFitter: fitted with 18627 periods, 100
subjects, 54 events>
```

```
ctv.print_summary()

plt.figure(figsize=(10,5))
ctv.plot()
plt.show()
plt.close()
```

```
<lifelines.CoxTimeVaryingFitter: fitted with 18627 periods, 100
subjects, 54 events>
         event col = 'falha'
number of subjects = 100
 number of periods = 18627
  number of events = 54
partial log-likelihood = -63.86
  time fit was run = 2024-02-09 18:12:42 UTC

---
            coef  exp(coef)   se(coef)   coef lower 95%   coef upper
95%  exp(coef) lower 95%  exp(coef) upper 95%
covariate

s_2         2.05       7.75       0.72             0.64
3.45                 1.90                 31.64
s_3         0.07       1.07       0.04            -0.01
0.15                 0.99                  1.16
s_4         0.16       1.18       0.05             0.07
0.26                 1.07                  1.29
s_7        -1.09       0.34       0.50            -2.07             -
0.11                 0.13                  0.90
s_8        -5.03       0.01       4.84           -14.51
4.45                 0.00                 85.79
s_9        -0.02       0.98       0.04            -0.10
0.05                 0.91                  1.05
s_11        5.12     167.43       1.70             1.78
8.46                 5.95               4714.95
s_12       -1.14       0.32       0.53            -2.17             -
0.10                 0.11                  0.90
s_13       12.90   3.99e+05       5.51             2.09
23.70                 8.09               1.97e+10
s_14        0.04       1.04       0.04            -0.04
0.12                 0.96                  1.13
s_15        5.07     159.96       9.53           -13.61
23.76                 0.00               2.09e+10
s_17        0.41       1.51       0.20             0.01
0.81                 1.01                  2.25
s_20       -5.13       0.01       2.02            -9.09             -
1.17                 0.00                  0.31
```

```
s_21          -4.23        0.01        2.67                -9.46
1.01                 0.00                 2.75

            cmp to      z       p   -log2(p)
covariate
s_2           0.00   2.85 <0.005       7.84
s_3           0.00   1.73   0.08       3.57
s_4           0.00   3.49 <0.005      11.03
s_7           0.00  -2.17   0.03       5.06
s_8           0.00  -1.04   0.30       1.74
s_9           0.00  -0.58   0.56       0.82
s_11          0.00   3.01 <0.005       8.56
s_12          0.00  -2.15   0.03       4.99
s_13          0.00   2.34   0.02       5.69
s_14          0.00   0.95   0.34       1.54
s_15          0.00   0.53   0.59       0.75
s_17          0.00   2.03   0.04       4.56
s_20          0.00  -2.54   0.01       6.49
s_21          0.00  -1.58   0.11       3.14
---
Partial AIC = 155.72
log-likelihood ratio test = 333.85 on 14 df
-log2(p) of ll-ratio test = 205.97
```



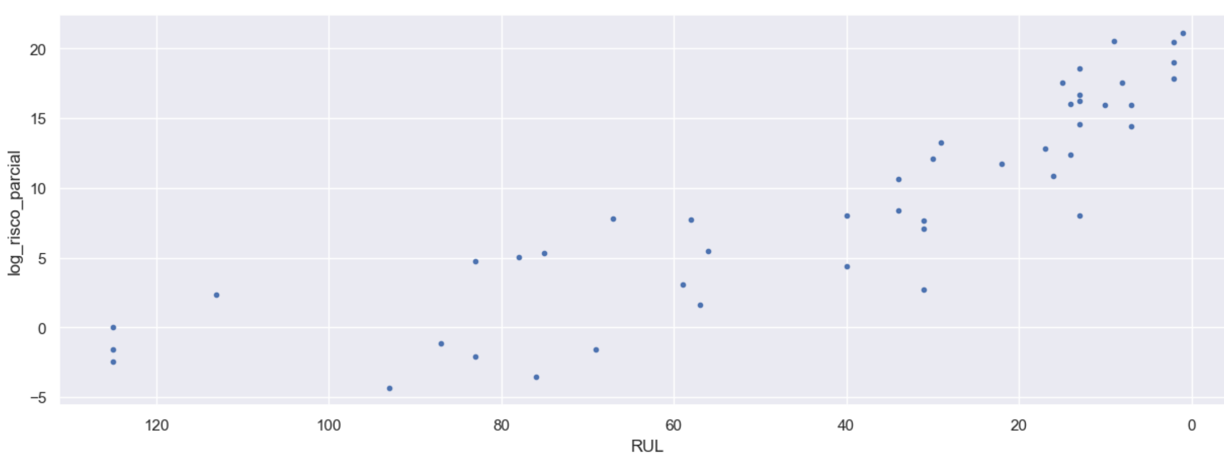
8.5. Testar e avaliar o modelo

```
df = train_censored.groupby("unidade").last()
df = df[df['falha'] == 0]  # obter motores do conjunto de dados que
ainda estão funcionando para que possamos prever seu RUL
```

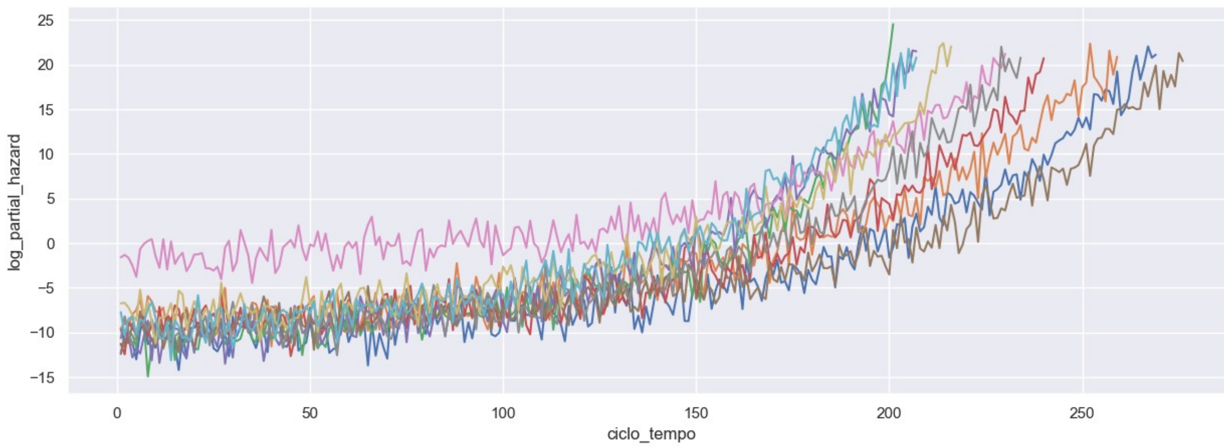
```
df_to_predict = df[df['falha'] == 0].copy()

predictions =
ctv.predict_log_partial_hazard(df_to_predict[predict_cols])
predictions = predictions.to_frame()
predictions.rename(columns={0: "previsoes"}, inplace=True)

df_unidade = df.reset_index()["unidade"].to_frame()
resultado = pd.concat([df_unidade,predictions],
axis=1).set_index("unidade")["previsoes"]
resultado.index.name = None
predictions = resultado.to_frame()
predictions['RUL'] = df_to_predict['RUL']

predictions.head(10)
```

	previsoes	RUL
2	-1.128757	87
5	-1.607369	69
7	3.073379	59
9	21.145492	1
10	11.751951	22
11	4.391817	40
15	14.439742	7
16	20.540832	9
17	-3.512471	76
20	10.624723	34

```
plt.figure(figsize=(15,5))
plt.plot(predictions['RUL'], predictions['previsoes'], '.b')
xlim = plt.gca().get_xlim()
plt.xlim(xlim[1], xlim[0])
plt.xlabel('RUL')
plt.ylabel('log_risco_parcial')

plt.show()
```



```
X = train.loc[train['unidade'].isin(df_to_predict.index)]
X_unico = X['unidade'].unique()

plt.figure(figsize=(15,5))

for i in range(1, len(X_unico), 2):    # para plotar apenas algumas
linhas, estamos selecionando apenas alguns valores
    if i in X_unico:
        X_sub = X.loc[X['unidade'] == i]
        predictions = ctv.predict_partial_hazard(X_sub).values
        plt.plot(X_sub['ciclo_tempo'].values, np.log(predictions))

plt.xlabel('ciclo_tempo')
plt.ylabel('log_partial_hazard')
plt.show()
```

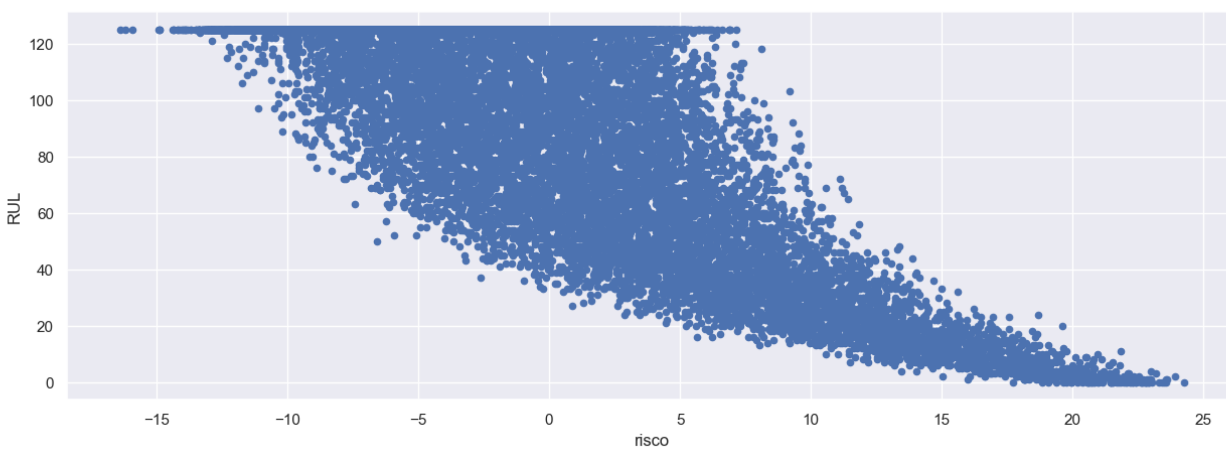


8.6. Regressão do risco parcial para RUL

```
df_hazard = train_censored.copy()
df_hazard['risco'] = ctv.predict_log_partial_hazard(df_hazard)
df_hazard.head()
```

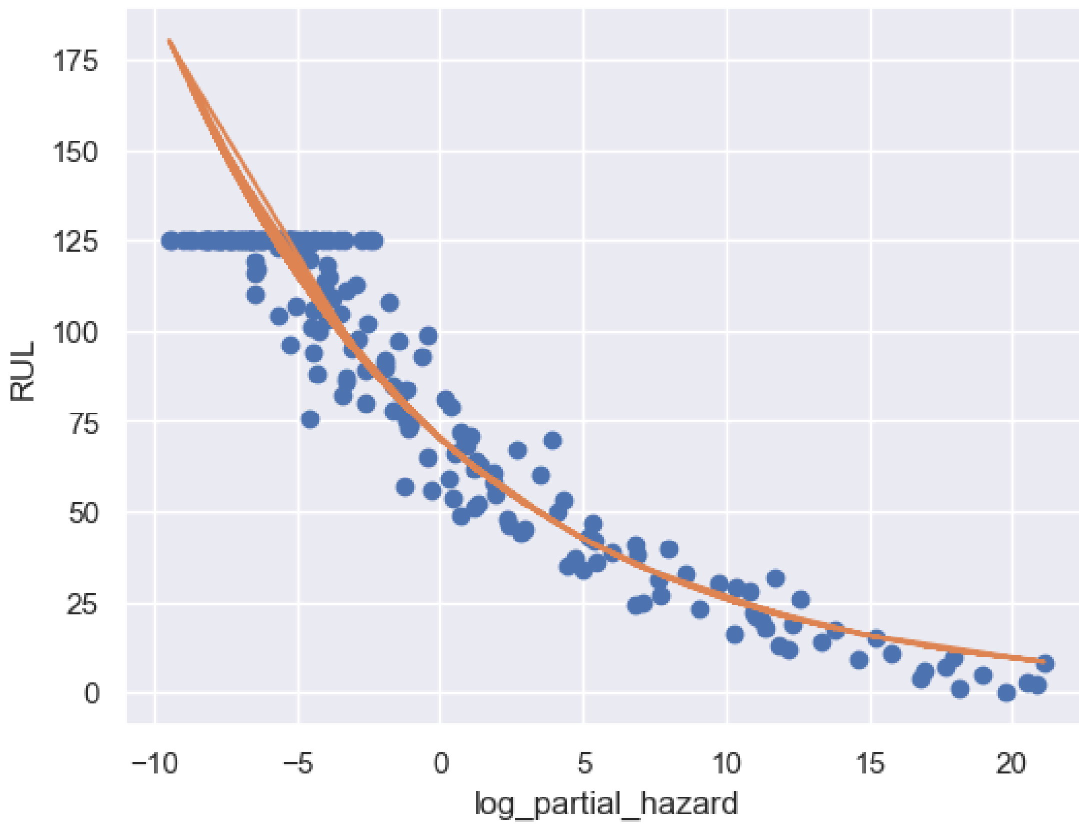
```
   unidade  ciclo_tempo     s_2      s_3      s_4     s_7      s_8
s_9  \
0        1            1  641.82  1589.70  1400.60  554.36  2388.06
9046.19
1        1            2  642.15  1591.82  1403.14  553.75  2388.04
9044.07
2        1            3  642.35  1587.99  1404.20  554.26  2388.08
9052.94
3        1            4  642.35  1582.79  1401.87  554.45  2388.11
9049.48
4        1            5  642.37  1582.85  1406.22  554.00  2388.06
9055.15
```

```
    s_11    s_12     s_13     s_14    s_15  s_17   s_20     s_21  RUL
falha  \
0  47.47  521.66  2388.02  8138.62  8.4195   392  39.06  23.4190  125
0
1  47.49  522.28  2388.07  8131.49  8.4318   392  39.00  23.4236  125
0
2  47.27  522.42  2388.03  8133.23  8.4178   390  38.95  23.3442  125
0
3  47.13  522.86  2388.08  8133.83  8.3682   392  38.88  23.3739  125
0
4  47.28  522.19  2388.04  8133.80  8.4294   393  38.90  23.4044  125
0

   start     risco
0      0 -6.856341
1      1 -4.700971
2      2 -7.370774
3      3 -8.140738
4      4 -5.260691

df_hazard.plot('risco', 'RUL', 'scatter', figsize=(15,5))
plt.xlabel('risco')
plt.ylabel('RUL')
plt.show()
```




```
# https://stackoverflow.com/questions/52930401/how-to-get-a-robust-
nonlinear-regression-fit-using-scipy-optimize-least-squares
from scipy.optimize import curve_fit

def exponential_model(z, a, b):
    return a * np.exp(-b * z)

# checar por unidade especifica
y_hat = exponential_model(df_hazard.loc[df_hazard['unidade']==1,
'risco'], 70, 0.1)
```

```
plt.plot(df_hazard.loc[df_hazard['unidade']==1, 'risco'],
df_hazard.loc[df_hazard['unidade']==1, 'RUL'], 'o',
        df_hazard.loc[df_hazard['unidade']==1, 'risco'], y_hat)
plt.xlabel("log_partial_hazard")
plt.ylabel("RUL")
plt.show()
plt.close()
```




```
popt, pcov = curve_fit(exponential_model, df_hazard['risco'],
df_hazard['RUL'])
print(popt)

[8.85954699e+01 4.35302167e-02]

# Preparar grupo de teste
test = test.drop(labels=drop_labels, axis=1)
test['falha'] = 0
test['start'] = test['ciclo_tempo'] - 1

# Testar e avaliar o modelo
y_hat = exponential_model(df_hazard['risco'], *popt)
avaliar(df_hazard['RUL'], y_hat, 'treino')
```

```
y_pred =
ctv.predict_log_partial_hazard(test.groupby('unidade').last())
y_hat = exponential_model(y_pred, *popt)
avaliar(y_test, y_hat)
```

```
conjunto de treino -> RMSE:26.30285842224388, R2:0.5487597218187092
conjunto de teste -> RMSE:27.13524416898879, R2:0.5736091039470262
```

8.7. Aplicando a todo o conjunto de dados

```
ctv2 = CoxTimeVaryingFitter()
ctv2.fit(train[train_cols], id_col="unidade", event_col='falha',
        start_col='start', stop_col='ciclo_tempo', show_progress=True)

train['risco'] = ctv2.predict_log_partial_hazard(train)
popt2, pcov2 = curve_fit(exponential_model, train['risco'],
train['RUL'])

y_hat = exponential_model(train['risco'], *popt2)
avaliar(train['RUL'], y_hat, 'treino')

y_pred =
ctv2.predict_log_partial_hazard(test.groupby('unidade').last())
y_hat = exponential_model(y_pred, *popt2)
avaliar(y_test, y_hat)
```

```
Iteration 1: norm_delta = 1.00e+00, step_size = 0.9500, log_lik = -
363.73938, newton_decrement = 1.69e+02, seconds_since_start = 0.0
Iteration 2: norm_delta = 1.23e+00, step_size = 0.9500, log_lik = -
196.95744, newton_decrement = 6.53e+01, seconds_since_start = 0.0
Iteration 3: norm_delta = 7.49e-01, step_size = 0.9500, log_lik = -
141.32406, newton_decrement = 2.53e+01, seconds_since_start = 0.1
Iteration 4: norm_delta = 7.27e-01, step_size = 0.9310, log_lik = -
129.43964, newton_decrement = 1.68e+01, seconds_since_start = 0.1
Iteration 5: norm_delta = 2.30e-01, step_size = 1.0000, log_lik = -
115.92018, newton_decrement = 1.13e+00, seconds_since_start = 0.1
Iteration 6: norm_delta = 4.04e-02, step_size = 1.0000, log_lik = -
114.78357, newton_decrement = 1.24e-02, seconds_since_start = 0.1
Iteration 7: norm_delta = 5.62e-04, step_size = 1.0000, log_lik = -
114.77107, newton_decrement = 2.21e-06, seconds_since_start = 0.1
Iteration 8: norm_delta = 1.18e-07, step_size = 1.0000, log_lik = -
114.77106, newton_decrement = 9.32e-14, seconds_since_start = 0.1
Convergence completed after 8 iterations.
conjunto de treino -> RMSE:26.22636478059728, R2:0.603928906030835
conjunto de teste -> RMSE:26.58098880820965, R2:0.5908498441213126
```