1. Análise Exploratória de Dados

```
# Importando bibliotecas necessárias
import os
seed = 42
os.environ['PYTHONHASHSEED']=str(seed)

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

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GroupShuffleSplit

import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Masking,
TimeDistributed

# Definindo a seed para garantir a reprodutibilidade dos resultados
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)

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

# Definindo caminho dos dados
dir_path = './CMAPSSData/'
train_file = 'train_FD004.txt'
test_file = 'test_FD004.txt'

# 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+1) for i in range(0,21)]
col_names = index_names + setting_names + sensor_names

# Lendo os dados
train = pd.read_csv((dir_path+train_file), sep='\s+', header=None,
                    names=col_names)
test = pd.read_csv((dir_path+test_file), sep='\s+', header=None,
                    names=col_names)
```

```
y_test = pd.read_csv((dir_path+'RUL_FD004.txt'), sep='\s+',
header=None,
                 names=['RemainingUsefulLife'])

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

```
(61249, 26)
```

```
   unidade  ciclo_tempo  config_1  config_2  config_3     s_1     s_2
\
0        1            1   42.0049    0.8400     100.0  445.00  549.68

1        1            2   20.0020    0.7002     100.0  491.19  606.07

2        1            3   42.0038    0.8409     100.0  445.00  548.95

3        1            4   42.0000    0.8400     100.0  445.00  548.70

4        1            5   25.0063    0.6207      60.0  462.54  536.10


       s_3      s_4   s_5  ...    s_12     s_13     s_14     s_15
s_16  s_17  \
0  1343.43  1112.93  3.91  ...  129.78  2387.99  8074.83   9.3335
0.02   330
1  1477.61  1237.50  9.35  ...  312.59  2387.73  8046.13   9.1913
0.02   361
2  1343.12  1117.05  3.91  ...  129.62  2387.97  8066.62   9.4007
0.02   329
3  1341.24  1118.03  3.91  ...  129.80  2388.02  8076.05   9.3369
0.02   328
4  1255.23  1033.59  7.05  ...  164.11  2028.08  7865.80  10.8366
0.02   305

   s_18    s_19   s_20     s_21
0  2212  100.00  10.62   6.3670
1  2324  100.00  24.37  14.6552
2  2212  100.00  10.48   6.4213
3  2212  100.00  10.54   6.4176
4  1915   84.93  14.03   8.6754

[5 rows x 26 columns]
```

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

    # Mesclar o valor do ciclo maximo no dataframe de origem
```

```
    result_frame = df.merge(max_cycle.to_frame(name='ciclo_max'),
left_on='unidade', right_index=True)

    # Calcular o RUL para cada linha
    remaining_useful_life = result_frame["ciclo_max"] -
result_frame["ciclo_tempo"]
    result_frame["RUL"] = remaining_useful_life

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

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

	unidade	ciclo_tempo	RUL
0	1	1	320
1	1	2	319
2	1	3	318
3	1	4	317
4	1	5	316

2. Modelo de referência

```
# Criando 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))

# Usando uma regressao linear simples como modelo de referencia

# Separando dados de treino
X_train = train[setting_names + sensor_names].copy()
y_train = train['RUL'].copy()
y_train_clipped = y_train.clip(upper=125)

# Usando apenas a ultima linha para cada unidade para obter o valor
real do RUL
X_test = test.drop('ciclo_tempo',
axis=1).groupby('unidade').last().copy()

# Criando e ajustando o modelo
lm = LinearRegression()
lm.fit(X_train, y_train)
```

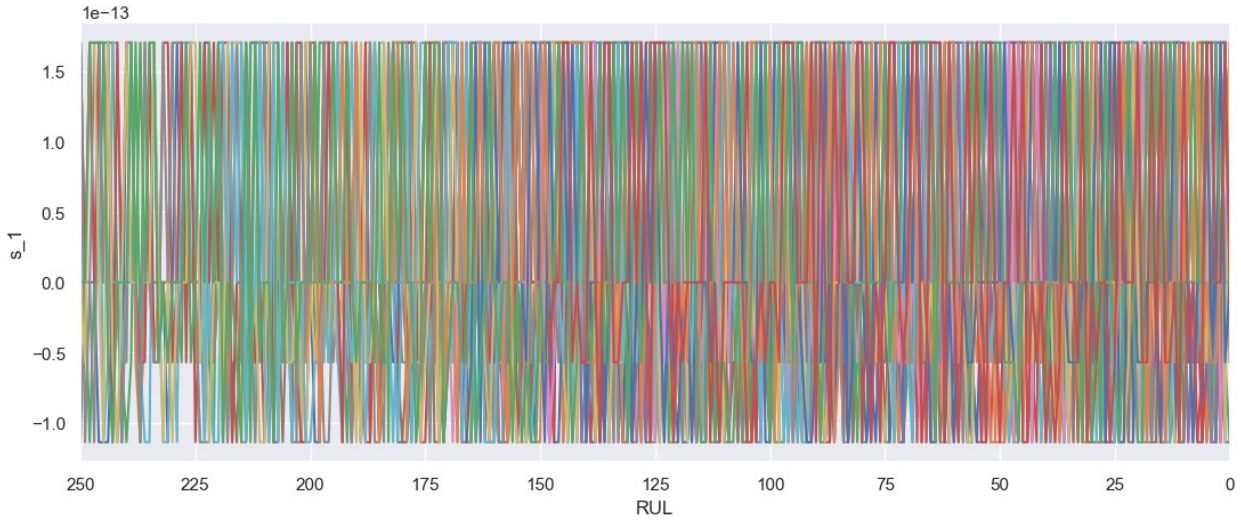
```
# Testando e avaliando o modelo treinado
y_hat_train = lm.predict(X_train)
avaliar(y_train, y_hat_train, 'treino')

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

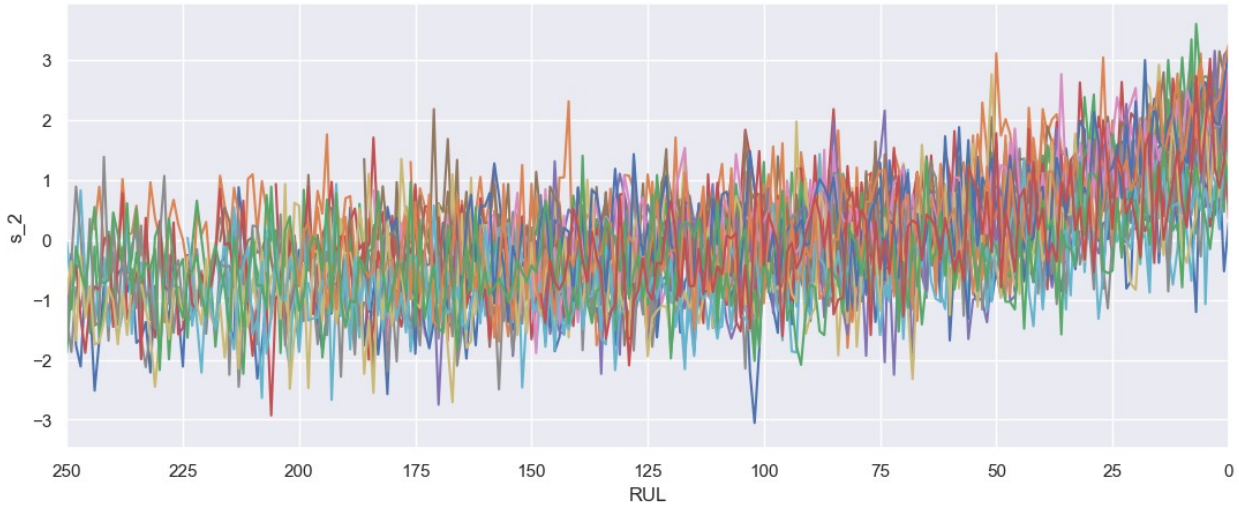
```
conjunto de treino -> RMSE:60.28599555343543, R2:0.5491334954684846
conjunto de teste -> RMSE:47.75764832870212, R2:0.23279090539900815
```

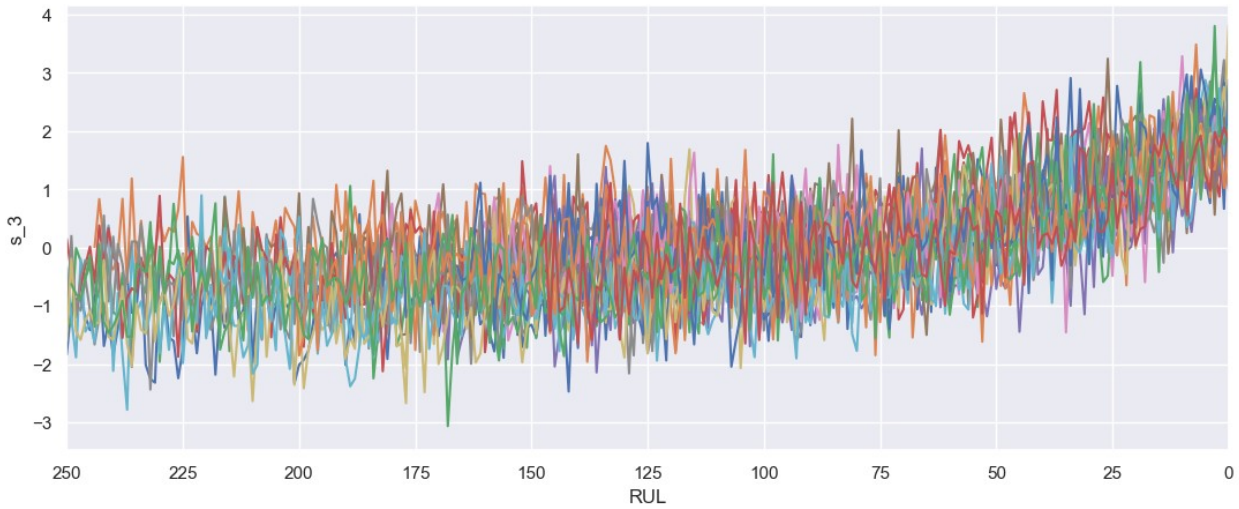
3. Análise gráfica do comportamento dos sensores

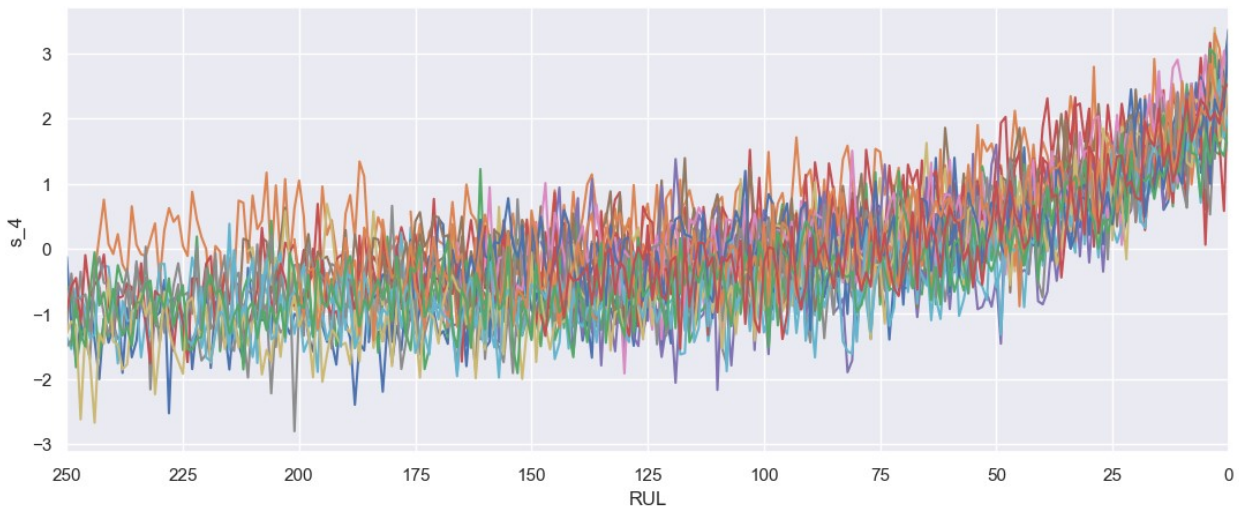
```
# Implementando a normalizacao por condicao de operacao

# Adicionando a condicao de operacao
def add_op_cond(df):
    df_op_cond = df.copy()

    df_op_cond['config_1'] = df_op_cond['config_1'].round()
    df_op_cond['config_2'] = df_op_cond['config_2'].round(decimals=2)

    df_op_cond['op_cond'] = df_op_cond['config_1'].astype(str) + '_' +
\
                        df_op_cond['config_2'].astype(str) + '_' + \
                        df_op_cond['config_3'].astype(str)

    return df_op_cond

X_train_condition = add_op_cond(train)
X_test_condition = add_op_cond(X_test)



# Normalizando baseado na condicao de operacao
def condition_scaler(df_train, df_test, sensor_names):
    scaler = StandardScaler()
    for condition in df_train['op_cond'].unique():
        scaler.fit(df_train.loc[df_train['op_cond']==condition,
sensor_names])
        df_train.loc[df_train['op_cond']==condition, sensor_names] =
scaler.transform(df_train.loc[df_train['op_cond']==condition,
sensor_names])
        df_test.loc[df_test['op_cond']==condition, sensor_names] =
scaler.transform(df_test.loc[df_test['op_cond']==condition,
sensor_names])
    return df_train, df_test
```

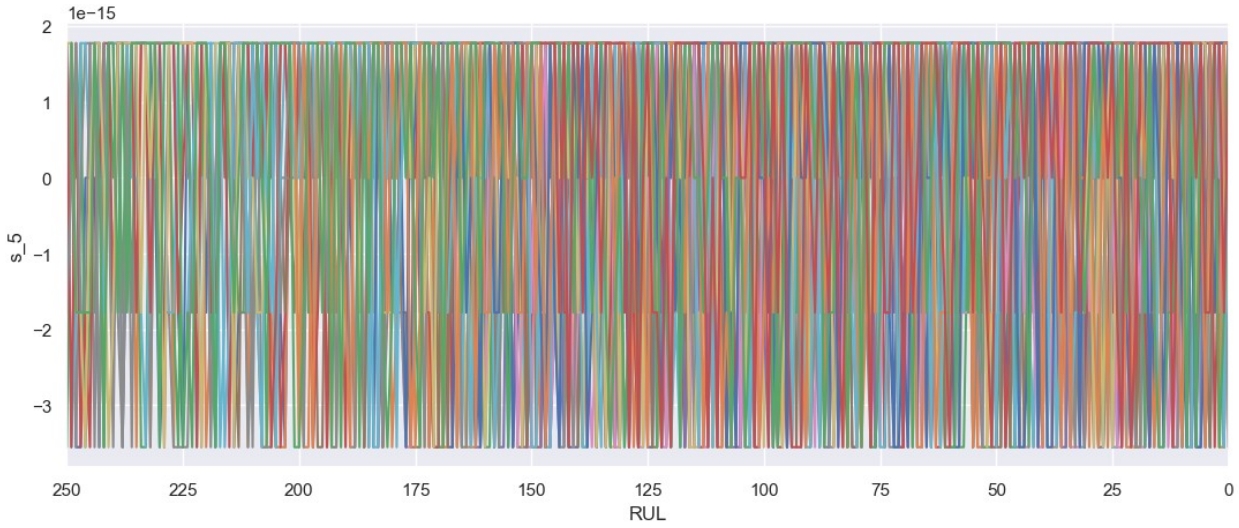
```
X_train_condition_scaled, X_test_condition_scaled =
condition_scaler(X_train_condition, X_test_condition, sensor_names)

def plot_sinal(df, signal_name, unit_nr=None):
    plt.figure(figsize=(13,5))

    if unit_nr:
        plt.plot('RUL', signal_name,
                 data=df[df['unidade']==unit_nr])
    else:
        for i in train['unidade'].unique():
            if (i % 10 == 0):
                plt.plot('RUL', signal_name,
                         data=df[df['unidade']==i])
    plt.xlim(250, 0)  # Inverte o eixo x para ir de 250 ate 0
    plt.xticks(np.arange(0, 275, 25))
    plt.ylabel(signal_name)
    plt.xlabel('RUL')
    plt.show()

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

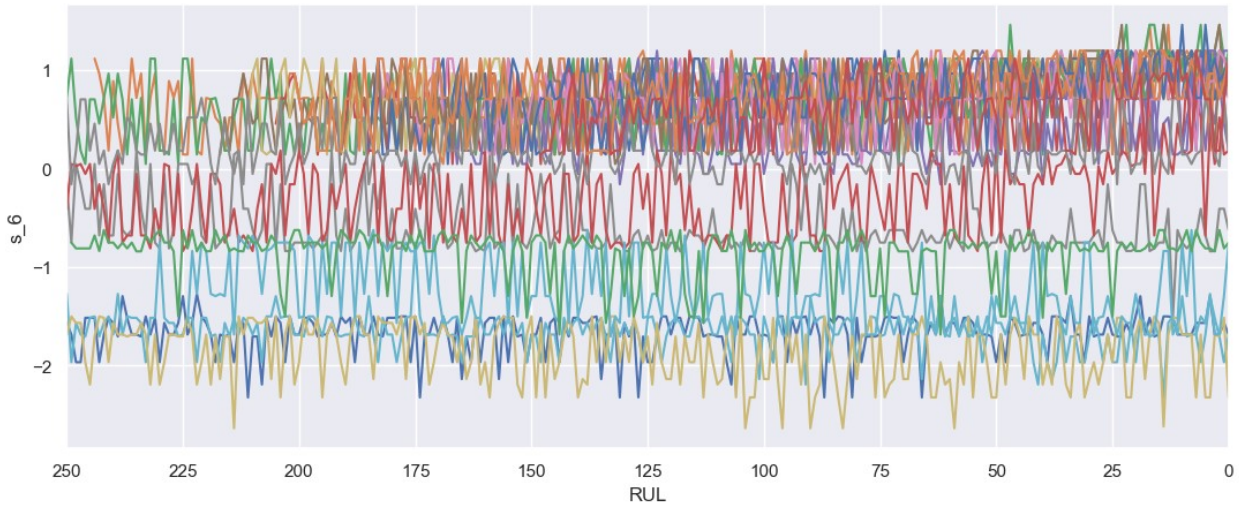


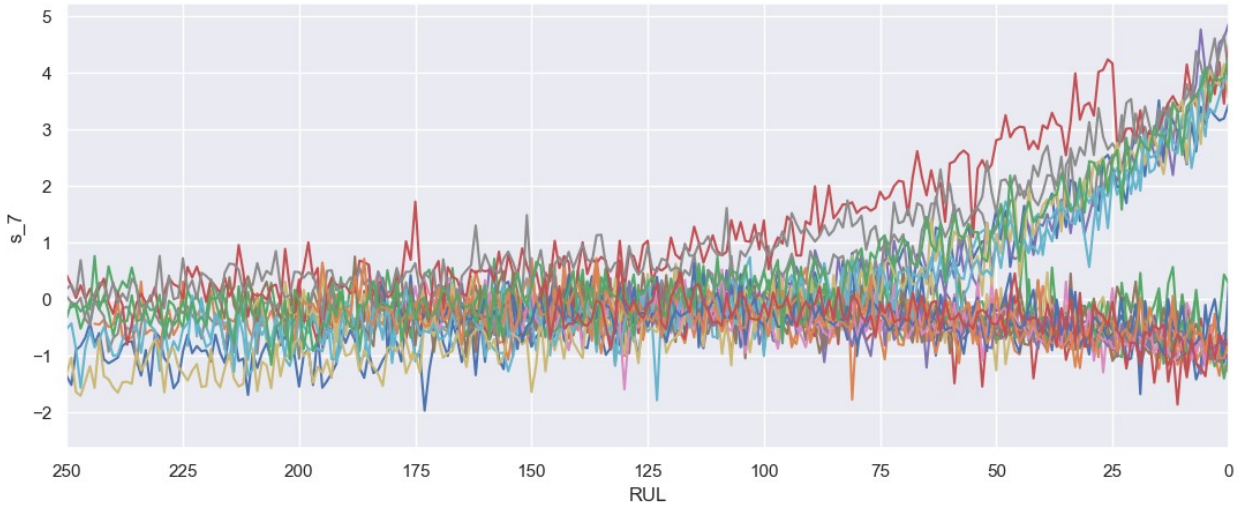

s_2
3
2
1
0
−1
−2
−3
250
225
200
175
150
125
100
75
50
25
0
RUL

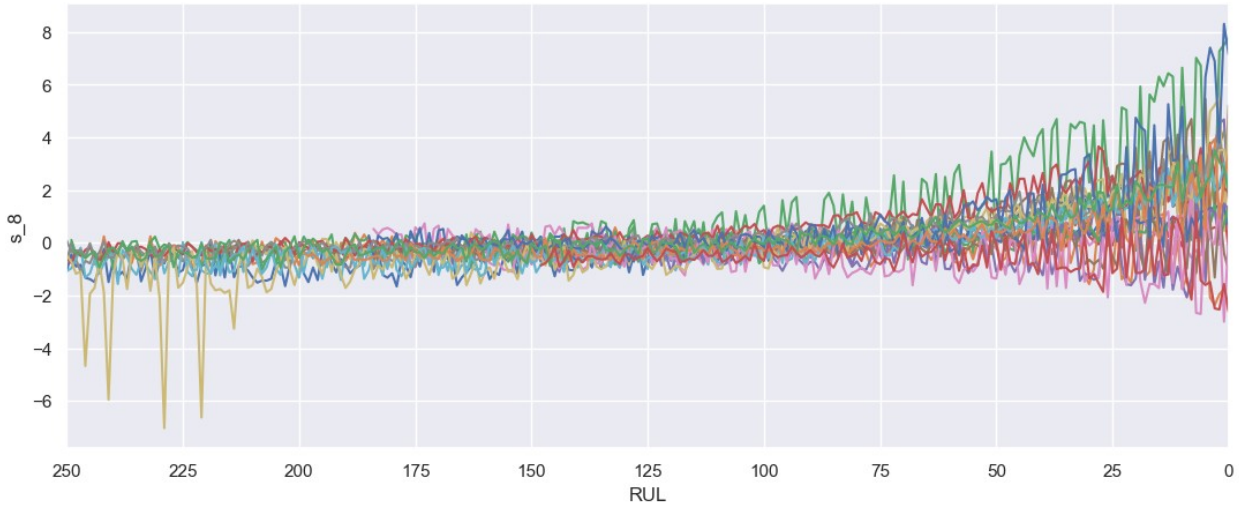

s_3
4
3
2
1
0
−1
−2
−3
250
225
200
175
150
125
100
75
50
25
0
RUL

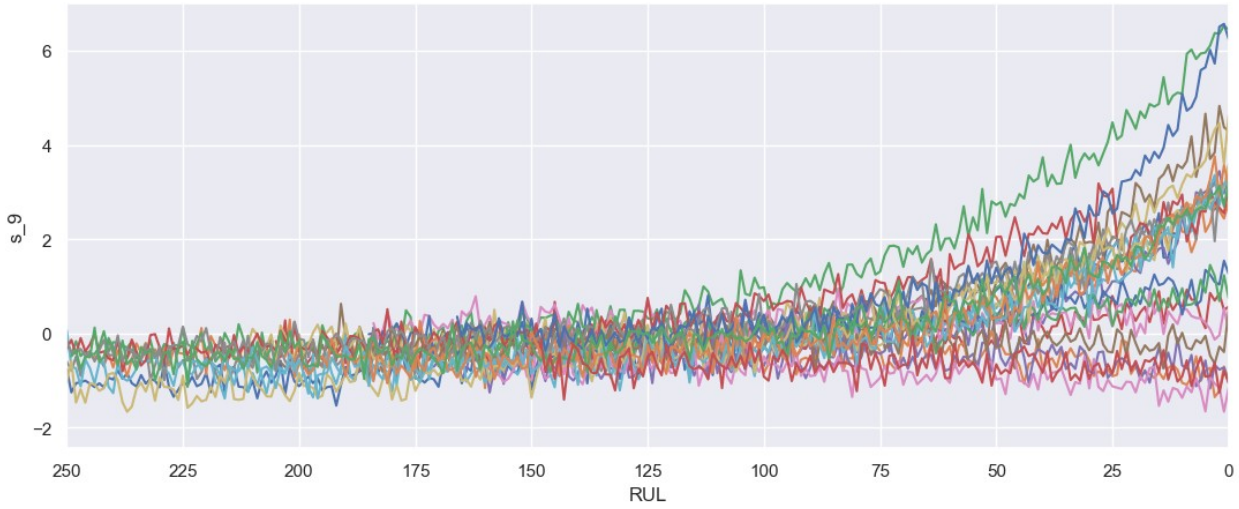

s_4
3
2
1
0
−1
−2
−3
250
225
200
175
150
125
100
75
50
25
0
RUL

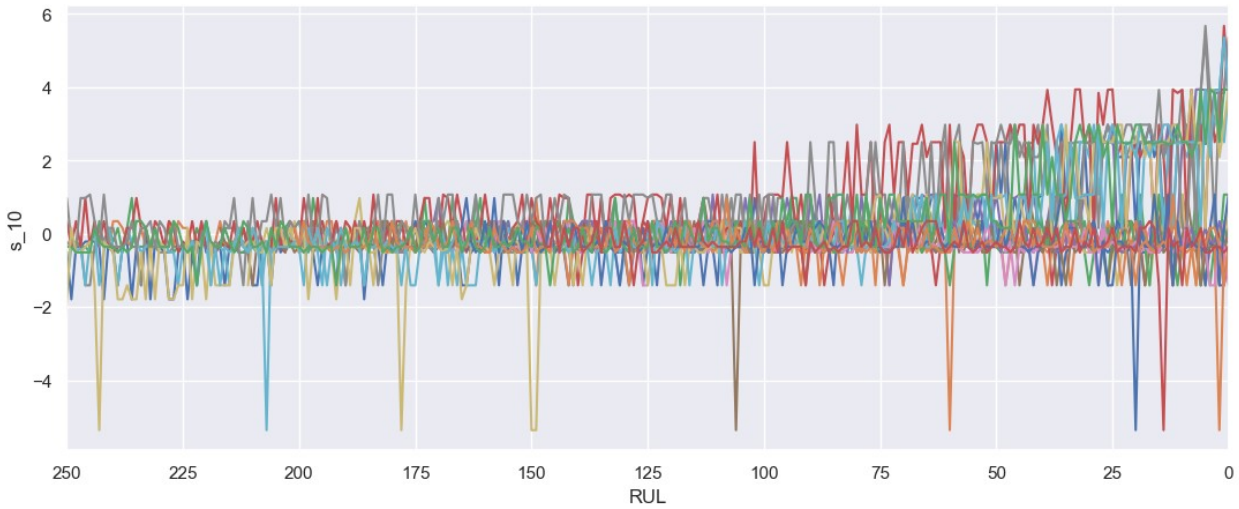

1e−15
2
1
0
−1
−2
−3
s_5
250
225
200
175
150
125
100
75
50
25
0
RUL

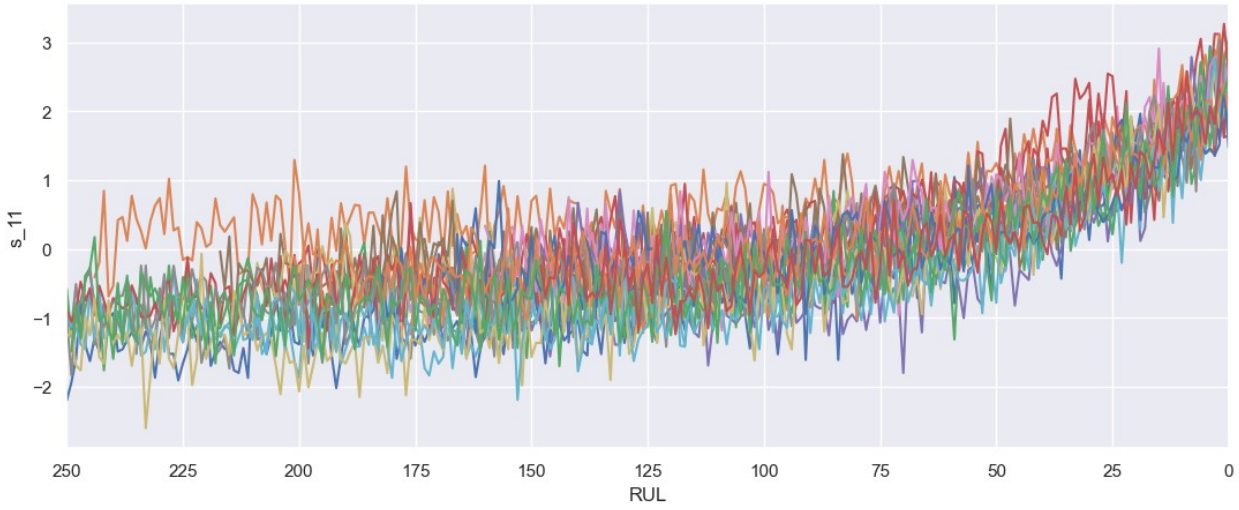

1
0
−1
−2
s_6
250
225
200
175
150
125
100
75
50
25
0
RUL

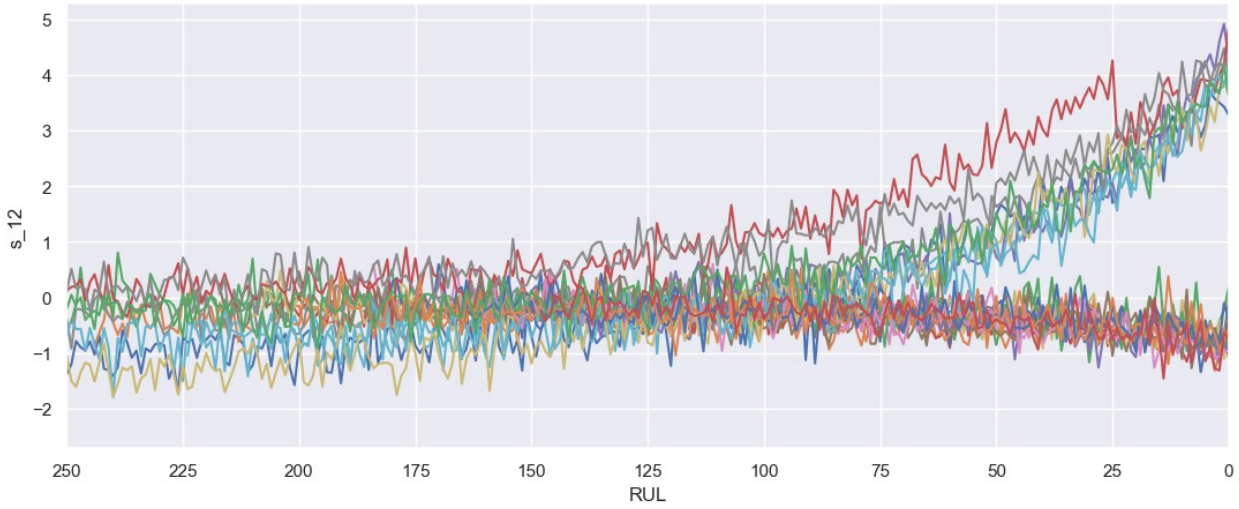

5
4
3
2
1
0
−1
−2
s_7
250
225
200
175
150
125
100
75
50
25
0
RUL

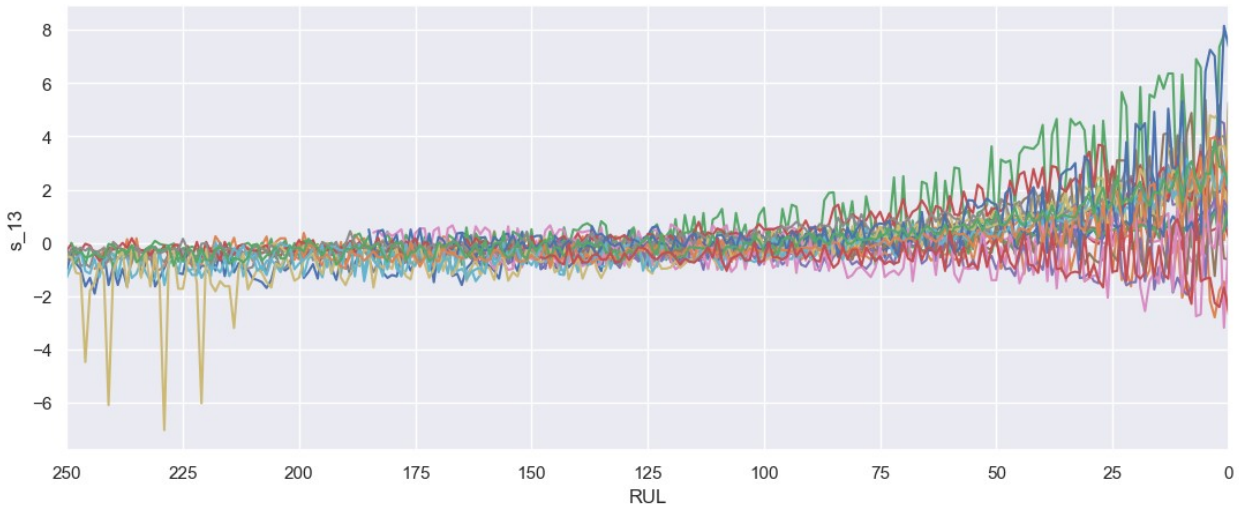

s_8
8
6
4
2
0
−2
−4
−6
250
225
200
175
150
125
100
75
50
25
0
RUL

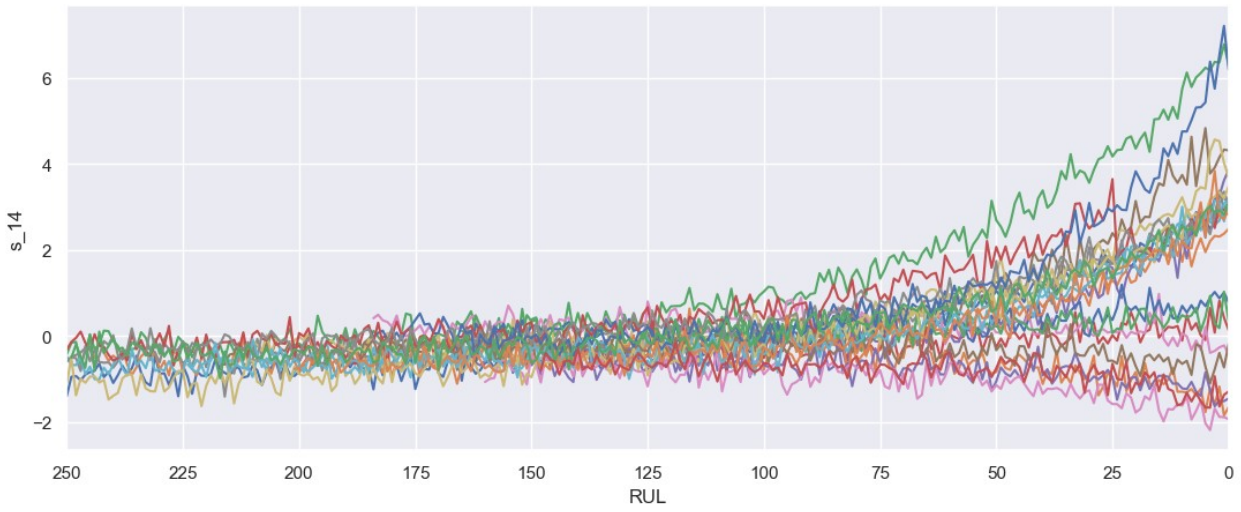

s_9
6
4
2
0
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

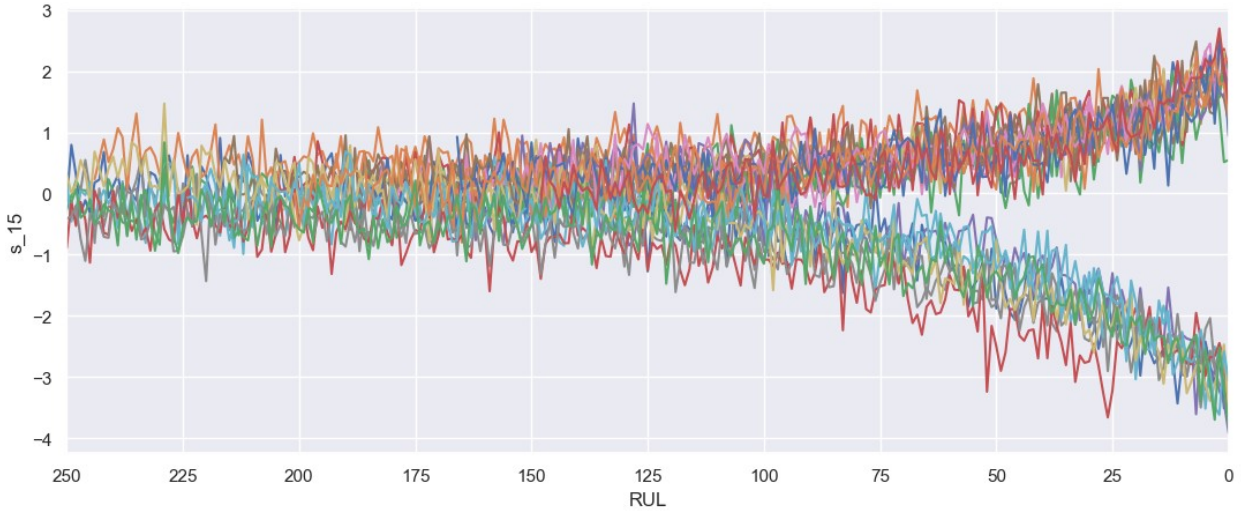

s_10
6
4
2
0
−2
−4
250
225
200
175
150
125
100
75
50
25
0
RUL

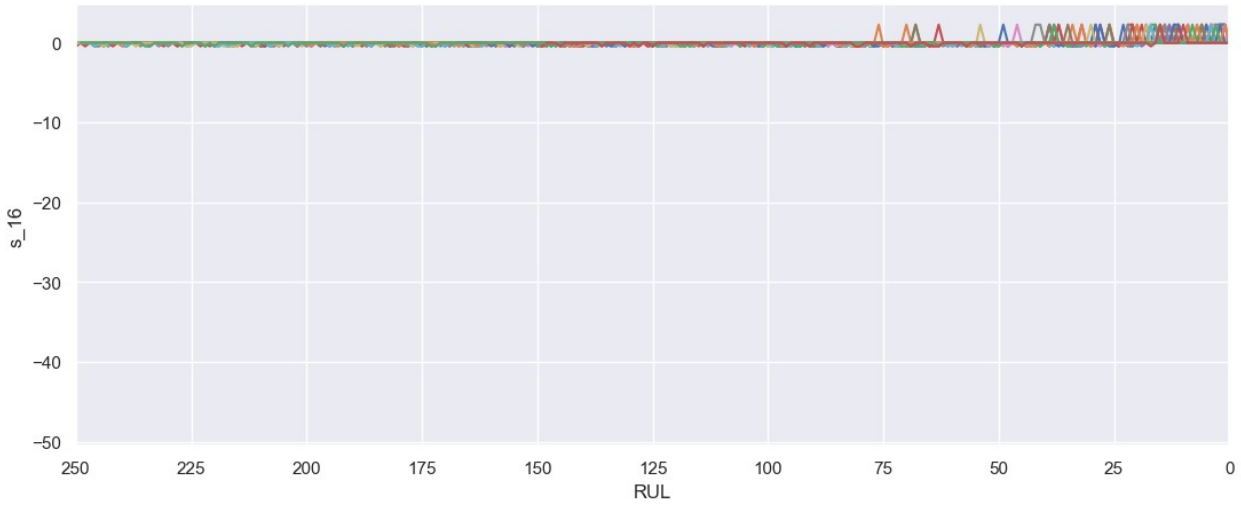

s_11
3
2
1
0
−1
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

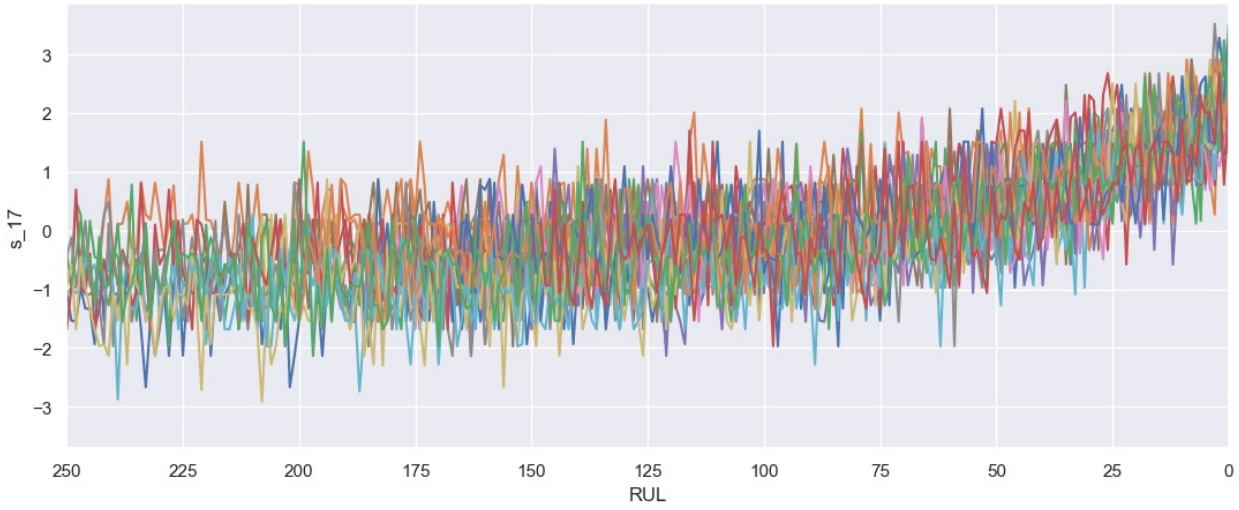

s_12
5
4
3
2
1
0
−1
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

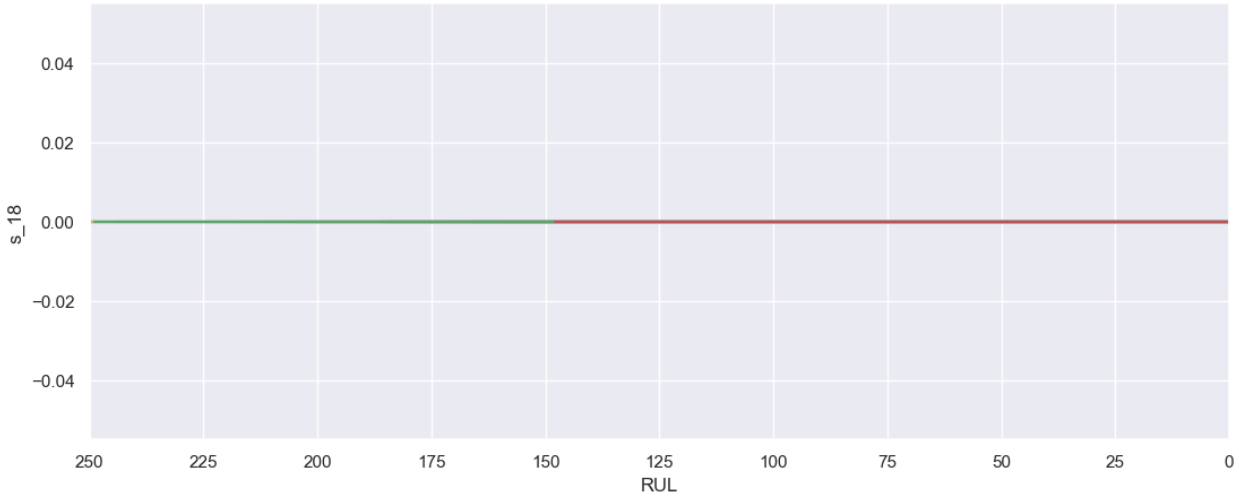

s_13
8
6
4
2
0
−2
−4
−6
250
225
200
175
150
125
100
75
50
25
0
RUL

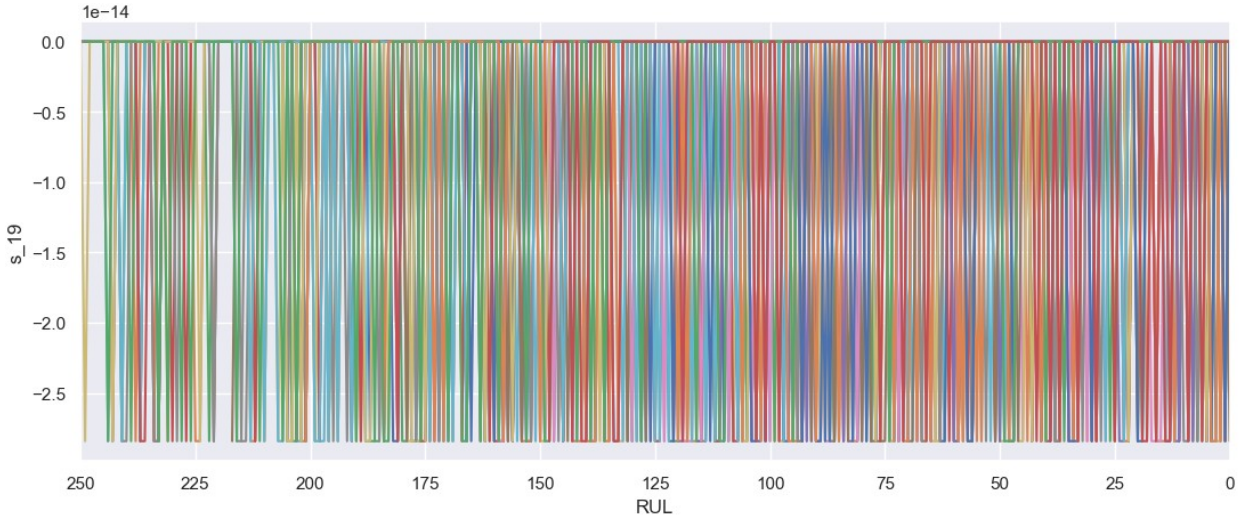

s_14
6
4
2
0
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

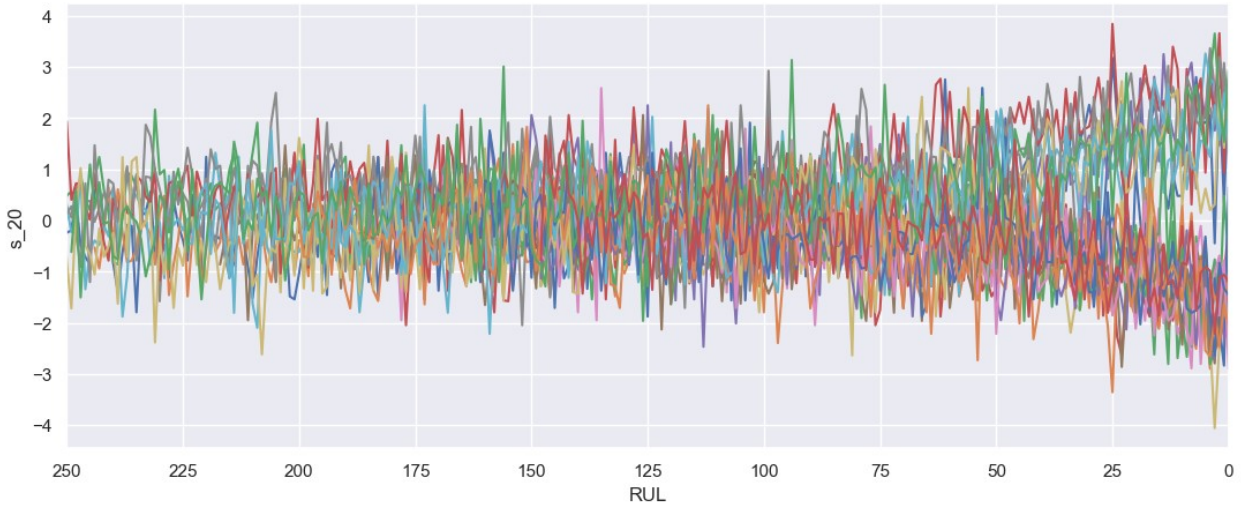

s_15
3
2
1
0
−1
−2
−3
−4
250
225
200
175
150
125
100
75
50
25
0
RUL

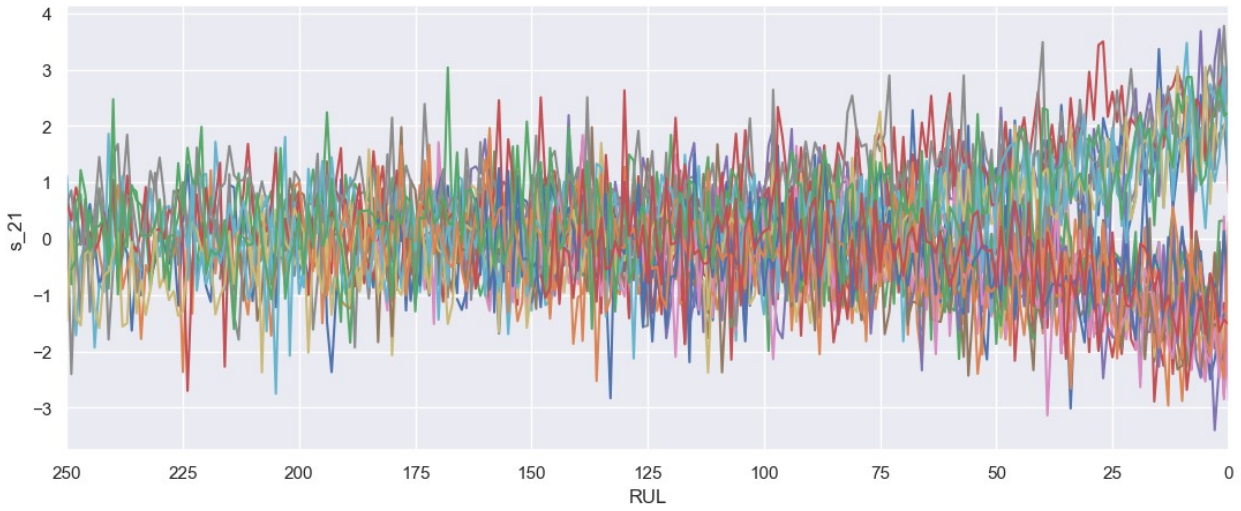

s_16
0
−10
−20
−30
−40
−50
250
225
200
175
150
125
100
75
50
25
0
RUL


s_17
3
2
1
0
−1
−2
−3
250
225
200
175
150
125
100
75
50
25
0
RUL


s_18
0.04
0.02
0.00
−0.02
−0.04
250
225
200
175
150
125
100
75
50
25
0
RUL


1e−14
s_19
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
250
225
200
175
150
125
100
75
50
25
0
RUL






```
# Sensores uteis
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']

# Dados desnecessarios
drop_sensors = [element for element in sensor_names if element not in
remaining_sensors]
drop_sensors
```

```
['s_1', 's_5', 's_6', 's_10', 's_16', 's_18', 's_19']
```

4. Preparação dos dados

```
# Suavizacao
def exponential_smoothing(df, sensors, n_samples, alpha=0.4):
    df = df.copy()
```

```
    df[sensors] = df.groupby('unidade')
[sensors].ewm(alpha=0.4).mean().reset_index()[sensors]

    def create_mask(data, samples):
        result = np.ones_like(data)
        result[0:samples] = 0
        return result

    mask = df.groupby('unidade')['unidade'].transform(create_mask,
samples=n_samples).astype(bool)
    df = df[mask]

    return df

# Criando grupos de teste e validacao
gss = GroupShuffleSplit(n_splits=1, train_size=0.80, random_state=42)


def train_val_group_split(X, y, gss, groups, print_groups=True):

    for idx_train, idx_val in gss.split(X, y, groups=groups):
        if print_groups:
            print('unidades_grupo_treino', train.iloc[idx_train]
['unidade'].unique(), '\n')
            print('unidades_grupo_validacao', train.iloc[idx_val]
['unidade'].unique(), '\n')

        X_train_split = X.iloc[idx_train].copy()
        y_train_split = y.iloc[idx_train].copy()
        X_val_split = X.iloc[idx_val].copy()
        y_val_split = y.iloc[idx_val].copy()
    return X_train_split, y_train_split, X_val_split, y_val_split

split_result = train_val_group_split(X_train, y_train_clipped, gss,
train['unidade'])
X_train_split, y_train_clipped_split, X_val_split, y_val_clipped_split
= split_result
```

```
unidades_grupo_treino [  1   2   3   4   5   6   8   9  12  13  14  15
18  21  22  23  24  27
  28  29  30  32  33  35  36  37  38  40  41  42  43  44  45  47  48
49
  50  51  52  53  54  55  57  58  59  60  62  63  64  65  66  67  71
72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
90
  91  92  93  94  95  96  99 100 101 102 103 104 106 107 108 110 111
114
 116 117 118 120 122 123 124 125 126 127 128 129 130 131 132 133 134
135
```

```
 136 137 139 140 142 143 144 145 146 147 148 149 150 151 152 153 154
155
 156 157 158 161 162 164 165 166 167 168 169 170 171 172 175 177 178
179
 180 182 183 186 187 188 189 190 192 193 198 199 203 204 205 206 207
208
 209 210 211 212 213 214 215 217 218 219 220 221 222 223 225 226 227
228
 229 230 231 232 233 234 235 236 237 238 239 241 242 243 244 245 246
248
 249]

unidades_grupo_validacao [  7  10  11  16  17  19  20  25  26  31  34
39  46  56  61  68  69  70
  97  98 105 109 112 113 115 119 121 138 141 159 160 163 173 174 176
181
 184 185 191 194 195 196 197 200 201 202 216 224 240 247]
```

5. Sequencias de dados

```
# Funcao que cria sequencias de comprimento determinado para as
colunas especificadas de uma determinada unidade do dataframe
# Em resumo, a funcao "corta" os dados de uma unidade em arrays do
comprimento determinado
# Cada sequencia serviria como input de uma rede neural
def gen_train_data(df, sequence_length, columns):
    data = df[columns].values
    num_elements = data.shape[0]

    for start, stop in zip(range(0, num_elements-(sequence_length-1)),
range(sequence_length, num_elements+1)):
        yield data[start:stop, :]

# Funcao para aplicar a gen_train_data para varias unidades do
dataframe
def gen_data_wrapper(df, sequence_length, columns,
unit_nrs=np.array([])):
    if unit_nrs.size <= 0:
        unit_nrs = df['unidade'].unique()

    data_gen = (list(gen_train_data(df[df['unidade']==unit_nr],
sequence_length, columns))
                for unit_nr in unit_nrs)
    data_array = np.concatenate(list(data_gen)).astype(np.float32)
    return data_array
```

```
# Funcao similar a gen_train_data, mas para pegar o valor do RUL para
o ultimo elemento de cada sequencia para cada unidade
# Cada valor do RUL tem a funcao de label para uma rede neural
def gen_labels(df, sequence_length, label):
    data_matrix = df[label].values
    num_elements = data_matrix.shape[0]

    return data_matrix[sequence_length-1:num_elements, :]

# Similar a funcao gen_data_wrapper, mas para as labels
def gen_label_wrapper(df, sequence_length, label,
unit_nrs=np.array([])):
    if unit_nrs.size <= 0:
        unit_nrs = df['unidade'].unique()

    label_gen = [gen_labels(df[df['unidade']==unit_nr],
sequence_length, label)
                 for unit_nr in unit_nrs]
    label_array = np.concatenate(label_gen).astype(np.float32)
    return label_array

# Similar a funcao gen_train_data, mas para os dados de teste
# Os dados de teste possuem a particularidade que, para algumas
unidades, podem existir menos dados que o comprimento da sequencia
# Para resolver esse problema, para esses casos, as linhas sao
preenchidas com um valor generico que depois sera filtrado na rede
neural
def gen_test_data(df, sequence_length, columns, mask_value):
    if df.shape[0] < sequence_length:
        data_matrix = np.full(shape=(sequence_length, len(columns)),
fill_value=mask_value) # Cria uma matriz so com os valores genericos
        idx = data_matrix.shape[0] - df.shape[0]
        data_matrix[idx:,:] = df[columns].values  # E preenche com os
dados disponiveis
    else:
        data_matrix = df[columns].values

    # Gera especificamente a ultima sequencia possivel
    stop = num_elements = data_matrix.shape[0]
    start = stop - sequence_length
    for i in list(range(1)):
        yield data_matrix[start:stop, :]
```

5.1. Exemplo das funções que geram as sequencias

```
# Dados para exemplo das funcoes criadas
d = {'unidade': [1]*5 + [2]*5,
     'X': [10., 10.2, 10.4, 10.6, 10.8, 20., 20.2, 20.4, 20.6, 20.8],
     'y': [1.4, 1.4, 1.3, 1.2, 1.1, 2.4, 2.4, 2.3, 2.2, 2.1]}
```

```
exemplo = pd.DataFrame(data=d)
exemplo
```

```
   unidade     X    y
0        1  10.0  1.4
1        1  10.2  1.4
2        1  10.4  1.3
3        1  10.6  1.2
4        1  10.8  1.1
5        2  20.0  2.4
6        2  20.2  2.4
7        2  20.4  2.3
8        2  20.6  2.2
9        2  20.8  2.1
```

```
# Seguindo com o exemplo
ex_train_data = gen_train_data(exemplo[exemplo['unidade']==1],  # 
Aplicando a funcao na unidade 1
                               sequence_length=4,               # Para 
gerar sequencias de comprimento 4
                               columns=['X']                    # Para 
a coluna X
                               )
list(ex_train_data)
```

```
[array([[10. ],
        [10.2],
        [10.4],
        [10.6]]),
 array([[10.2],
        [10.4],
        [10.6],
        [10.8]])]
```

```
# Seguindo com o exemplo
ex_data_array = gen_data_wrapper(exemplo,           # Aplicando a 
funcao para todas as unidades no dataframe de exemplo
                                 sequence_length=4, # Para gerar 
sequencias de comprimento 4
                                 columns=['X']      # Para a coluna X
                                 )
ex_data_array
```

```
array([[[10. ],
        [10.2],
        [10.4],
        [10.6]],

       [[10.2],
        [10.4],
```

```
        [10.6],
        [10.8]],

       [[20. ],
        [20.2],
        [20.4],
        [20.6]],

       [[20.2],
        [20.4],
        [20.6],
        [20.8]]], dtype=float32)
```

```
# Seguindo com o exemplo
ex_label = gen_labels(exemplo[exemplo['unidade']==1],   # Aplicando a
funcao na unidade 1
                      sequence_length=4,                # Informando
que cada sequencia tem comprimento 4
                      label=['y']                       # Para obter
as labels na coluna y
                      )
ex_label
```

```
array([[1.2],
       [1.1]])
```

```
# Seguindo com o exemplo
label_array = gen_label_wrapper(exemplo,            # Aplicando a
funcao para todas as unidades no dataframe de exemplo
                                sequence_length=4,  # Informando que
cada sequencia tem comprimento 4
                                label=['y']         # Para obter as
labels na coluna y
                                )
label_array
```

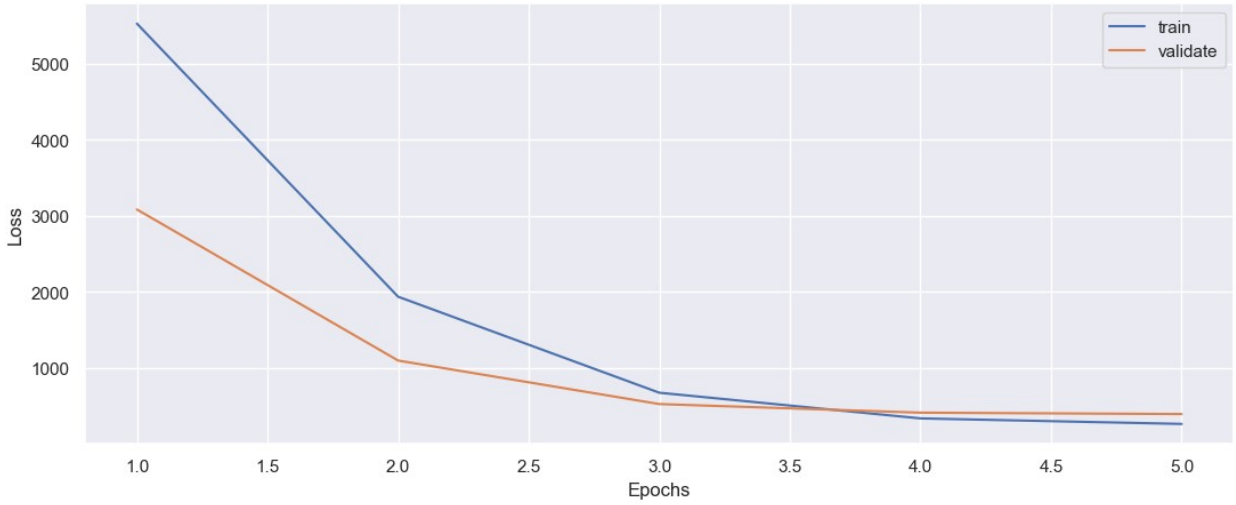
```
array([[1.2],
       [1.1],
       [2.2],
       [2.1]], dtype=float32)
```

6. Primeiro LSTM

```
# Combinando todas as etapas de tratamento dos dados
sequence_length = 20
train['RUL'].clip(upper=125, inplace=True)

X_train_interim = add_op_cond(train.drop(drop_sensors, axis=1))
X_test_interim = add_op_cond(test.drop(drop_sensors, axis=1))
```

```
X_train_interim, X_test_interim = condition_scaler(X_train_interim,
X_test_interim, remaining_sensors)

X_train_interim = exponential_smoothing(X_train_interim,
remaining_sensors, 0, 0.4)
X_test_interim = exponential_smoothing(X_test_interim,
remaining_sensors, 0, 0.4)

# Separando em grupos de treino e validacao
gss = GroupShuffleSplit(n_splits=1, train_size=0.80, random_state=42)
for train_unit, val_unit in
gss.split(X_train_interim['unidade'].unique(),
groups=X_train_interim['unidade'].unique()):
    train_unit = X_train_interim['unidade'].unique()[train_unit]
    val_unit = X_train_interim['unidade'].unique()[val_unit]

    train_split_array = gen_data_wrapper(X_train_interim,
sequence_length, remaining_sensors, train_unit)
    train_split_label = gen_label_wrapper(X_train_interim,
sequence_length, ['RUL'], train_unit)

    val_split_array = gen_data_wrapper(X_train_interim,
sequence_length, remaining_sensors, val_unit)
    val_split_label = gen_label_wrapper(X_train_interim,
sequence_length, ['RUL'], val_unit)

# Criando as sequencias de treino e teste
train_array = gen_data_wrapper(X_train_interim, sequence_length,
remaining_sensors)
label_array = gen_label_wrapper(X_train_interim, sequence_length,
['RUL'])

test_gen =
(list(gen_test_data(X_test_interim[X_test_interim['unidade']==unit_nr]
, sequence_length, remaining_sensors, -99.))
           for unit_nr in X_test_interim['unidade'].unique())
test_array = np.concatenate(list(test_gen)).astype(np.float32)

# Arquitetura da rede neural
model = Sequential()
model.add(Masking(mask_value=-99., input_shape=(sequence_length,
train_array.shape[2])))
model.add(LSTM(32, activation='tanh'))
model.add(Dense(1))

# Compilando o modelo e salvando os pesos para reprodutibilidade
futura consistente
model.compile(loss='mean_squared_error', optimizer='adam')
model.save_weights('simple_lstm_weights.h5')
```

```
model.compile(loss='mean_squared_error', optimizer='adam')  #
Recompilando para reiniciar o optimizer
model.load_weights('simple_lstm_weights.h5')                #
Carregando os pesos salvos

# Treinando o modelo
history = model.fit(train_split_array, train_split_label,
                    validation_data=(val_split_array,
val_split_label),
                    epochs=5,
                   batch_size=32)

Epoch 1/5
1423/1423 [==============================] - 17s 8ms/step - loss:
5526.1470 - val_loss: 3083.2695
Epoch 2/5
1423/1423 [==============================] - 10s 7ms/step - loss:
1936.5970 - val_loss: 1097.5002
Epoch 3/5
1423/1423 [==============================] - 10s 7ms/step - loss:
674.6016 - val_loss: 525.6874
Epoch 4/5
1423/1423 [==============================] - 10s 7ms/step - loss:
337.1608 - val_loss: 412.3534
Epoch 5/5
1423/1423 [==============================] - 10s 7ms/step - loss:
264.3943 - val_loss: 394.3087

# Funcao para fazer o grafico com o historico de treinamento
def plot_loss(fit_history):
    plt.figure(figsize=(13,5))
    plt.plot(range(1, len(fit_history.history['loss'])+1),
fit_history.history['loss'], label='train')
    plt.plot(range(1, len(fit_history.history['val_loss'])+1),
fit_history.history['val_loss'], label='validate')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

plot_loss(history)
```




```
# Avaliando o modelo
y_hat_train = model.predict(train_array)
avaliar(label_array, y_hat_train, 'treino')

y_hat_test = model.predict(test_array)
avaliar(y_test, y_hat_test)
```

```
1767/1767 [==============================] - 6s 3ms/step
conjunto de treino -> RMSE:16.584558486938477, R2:0.8381487437445712
8/8 [==============================] - 0s 6ms/step
conjunto de teste -> RMSE:29.166704034821986, R2:0.7138441786050604
```

7. Ajuste de hiperparâmetros

```
# Definindo os limites de cada parametro para o ajuste de
hiperparametros
alpha_list = [0.01, 0.05] + list(np.arange(10,60+1,10)/100)
sequence_list = list(np.arange(10,40+1,5))
epoch_list = list(np.arange(5,20+1,5))
nodes_list = [[32], [64], [128], [256], [32, 64], [64, 128], [128,
256]]
dropouts = list(np.arange(1,5)/10)  # O dropout = 0 geraria alguns
resultados melhores no treino, mas pior generalização devido ao
overfitting
activation_functions = ['tanh', 'sigmoid']  # Testes reveleram que o
uso da relu gerava uma performance significativamente pior
batch_size_list = [32, 64, 128, 256]
sensor_list = [['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'],
               ['s_2', 's_3', 's_4', 's_7', 's_11', 's_12', 's_15',
's_17', 's_20', 's_21']]
```

```
tuning_options = np.prod([len(alpha_list),
                          len(sequence_list),
                          len(epoch_list),
                          len(nodes_list),
                          len(dropouts),
                          len(activation_functions),
                          len(batch_size_list),
                          len(sensor_list)])
tuning_options
```

```
100352
```

```
train['RUL'].clip(upper=125, inplace=True)

# Funcao para realizar o preparo dos dados
def prep_data(train, test, drop_sensors, remaining_sensors, alpha):
    X_train_interim = add_op_cond(train.drop(drop_sensors, axis=1))
    X_test_interim = add_op_cond(test.drop(drop_sensors, axis=1))

    X_train_interim, X_test_interim =
condition_scaler(X_train_interim, X_test_interim, remaining_sensors)

    X_train_interim = exponential_smoothing(X_train_interim,
remaining_sensors, 0, alpha)
    X_test_interim = exponential_smoothing(X_test_interim,
remaining_sensors, 0, alpha)

    return X_train_interim, X_test_interim

# Funcao que cria os modelos baseado nos hiperparametros possiveis
def create_model(input_shape, nodes_per_layer, dropout, activation,
weights_file):
    model = Sequential()
    model.add(Masking(mask_value=-99., input_shape=input_shape))
    if len(nodes_per_layer) <= 1:
        model.add(LSTM(nodes_per_layer[0], activation=activation))
        model.add(Dropout(dropout))
    else:
        model.add(LSTM(nodes_per_layer[0], activation=activation,
return_sequences=True))
        model.add(Dropout(dropout))
        model.add(LSTM(nodes_per_layer[1], activation=activation))
        model.add(Dropout(dropout))
    model.add(Dense(1))

    model.compile(loss='mean_squared_error', optimizer='adam')
    model.save_weights(weights_file)
    return model
```

```
# Bloco que aplica um conjunto diferente de hiperparametros a cada
teste e registra os resultados

import time
import datetime

ITERATIONS = 100

results = pd.DataFrame(columns=['MSE', 'std_MSE', 'alpha',
                                'epochs', 'nodes', 'dropout',
                                'activation', 'batch_size',
                                'sequence_length', 'sensor_length'])

weights_file = 'lstm_hyper_parameter_weights.h5'

time_start = time.time()
print("Início: ", time.ctime(time_start))
time_history = [time_start]
prediction_history = []

for i in range(ITERATIONS):

    mse = []

    # Parametros de iniciacao
    alpha = random.sample(alpha_list, 1)[0]
    sequence_length = random.sample(sequence_list, 1)[0]
    epochs = random.sample(epoch_list, 1)[0]
    nodes_per_layer = random.sample(nodes_list, 1)[0]
    dropout = random.sample(dropouts, 1)[0]
    activation = random.sample(activation_functions, 1)[0]
    batch_size = random.sample(batch_size_list, 1)[0]
    remaining_sensors = random.sample(sensor_list, 1)[0]
    drop_sensors = [element for element in sensor_names if element not
in remaining_sensors]

    # Criando o modelo
    input_shape = (sequence_length, len(remaining_sensors))
    model = create_model(input_shape, nodes_per_layer, dropout,
activation, weights_file)

    # Separando em grupos de treino e validacao
    X_train_interim, X_test_interim = prep_data(train, test,
drop_sensors, remaining_sensors, alpha)
    gss = GroupShuffleSplit(n_splits=3, train_size=0.80,
random_state=42)
    for train_unit, val_unit in
gss.split(X_train_interim['unidade'].unique(),
groups=X_train_interim['unidade'].unique()):
        train_unit = X_train_interim['unidade'].unique()[train_unit]
```

```
        train_split_array = gen_data_wrapper(X_train_interim,
sequence_length, remaining_sensors, train_unit)
        train_split_label = gen_label_wrapper(X_train_interim,
sequence_length, ['RUL'], train_unit)

        val_unit = X_train_interim['unidade'].unique()[val_unit]
        val_split_array = gen_data_wrapper(X_train_interim,
sequence_length, remaining_sensors, val_unit)
        val_split_label = gen_label_wrapper(X_train_interim,
sequence_length, ['RUL'], val_unit)

        # Treinando e avaliando o modelo
        model.compile(loss='mean_squared_error', optimizer='adam')
        model.load_weights(weights_file)  # Reiniciando o optimizer e
os pesos do modelo antes de cada treinamento

        history = model.fit(train_split_array, train_split_label,
                            validation_data=(val_split_array,
val_split_label),
                            epochs=epochs,
                            batch_size=batch_size,
                            verbose=0)
        mse.append(history.history['val_loss'][-1])

    # Registrando os resultados
    d = {'MSE':np.mean(mse), 'std_MSE':np.std(mse), 'alpha':alpha,
         'epochs':epochs, 'nodes':str(nodes_per_layer),
'dropout':dropout,
         'activation':activation, 'batch_size':batch_size,
'sequence_length':sequence_length,
         'sensor_length':len(remaining_sensors)}
    results = pd.concat([results, pd.DataFrame(d, index=[0])],
ignore_index=True)


    # Informando previsao ate conclusao
    time_now = time.time()
    time_history.append(time_now)

    delta_t = time_now-time_history[i]
    prediction_history.append(delta_t)
    previsao = np.mean(prediction_history)

    print("Iteração ", i+1, " de ", ITERATIONS, " concluída em ",
datetime.timedelta(seconds=delta_t))
    print("Tempo restante previsto: ",
datetime.timedelta(seconds=((previsao)*(ITERATIONS-i+1))))
    print("Previsão para conclusão: ", time.ctime(time.time()+
```

```
(previsao)*(ITERATIONS-i+1)))
    print()

Início:  Sat Feb 10 14:07:55 2024

Iteração  1  de  100  concluída em  0:01:02.887340
Tempo restante previsto:  1:45:51.621347
Previsão para conclusão:  Sat Feb 10 15:54:49 2024

WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_3 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  2  de  100  concluída em  0:30:11.302216
Tempo restante previsto:  1 day, 2:01:49.477794
Previsão para conclusão:  Sun Feb 11 16:40:59 2024

Iteração  3  de  100  concluída em  0:08:19.463167
Tempo restante previsto:  21:45:30.539846
Previsão para conclusão:  Sun Feb 11 12:32:59 2024

Iteração  4  de  100  concluída em  0:02:50.842293
Tempo restante previsto:  17:19:00.127877
Previsão para conclusão:  Sun Feb 11 08:09:19 2024

Iteração  5  de  100  concluída em  0:06:53.442030
Tempo restante previsto:  15:56:23.978680
Previsão para conclusão:  Sun Feb 11 06:53:37 2024

Iteração  6  de  100  concluída em  0:08:06.559417
Tempo restante previsto:  15:18:31.943394
Previsão para conclusão:  Sun Feb 11 06:23:51 2024

Iteração  7  de  100  concluída em  0:02:21.206586
Tempo restante previsto:  13:31:03.112797
Previsão para conclusão:  Sun Feb 11 04:38:44 2024

Iteração  8  de  100  concluída em  0:27:24.971728
Tempo restante previsto:  17:04:20.428622
Previsão para conclusão:  Sun Feb 11 08:39:26 2024

Iteração  9  de  100  concluída em  0:07:41.178826
Tempo restante previsto:  16:20:15.820553
Previsão para conclusão:  Sun Feb 11 08:03:03 2024

WARNING:tensorflow:Layer lstm_16 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_17 will not use cuDNN kernels since it
```

```
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  10  de  100  concluída em  2:19:32.149724
Tempo restante previsto:  1 day, 11:56:28.830603
Previsão para conclusão:  Mon Feb 12 05:58:48 2024

Iteração  11  de  100  concluída em  0:05:37.986573
Tempo restante previsto:  1 day, 9:05:43.734622
Previsão para conclusão:  Mon Feb 12 03:13:41 2024

WARNING:tensorflow:Layer lstm_20 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_21 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  12  de  100  concluída em  0:58:14.101900
Tempo restante previsto:  1 day, 13:17:00.688494
Previsão para conclusão:  Mon Feb 12 08:23:12 2024

WARNING:tensorflow:Layer lstm_22 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_23 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  13  de  100  concluída em  1:02:59.849983
Tempo restante previsto:  1 day, 17:13:16.832201
Previsão para conclusão:  Mon Feb 12 13:22:28 2024

Iteração  14  de  100  concluída em  0:09:06.247936
Tempo restante previsto:  1 day, 14:48:02.335373
Previsão para conclusão:  Mon Feb 12 11:06:19 2024

Iteração  15  de  100  concluída em  0:13:53.835736
Tempo restante previsto:  1 day, 13:08:44.947638
Previsão para conclusão:  Mon Feb 12 09:40:56 2024

WARNING:tensorflow:Layer lstm_27 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_28 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  16  de  100  concluída em  0:17:16.356028
Tempo restante previsto:  1 day, 11:58:16.550472
Previsão para conclusão:  Mon Feb 12 08:47:44 2024

Iteração  17  de  100  concluída em  0:07:40.374953
Tempo restante previsto:  1 day, 10:06:03.782181
```

```
Previsão para conclusão:  Mon Feb 12 07:03:11 2024

Iteração  18  de  100  concluída em  0:03:46.376809
Tempo restante previsto:  1 day, 8:07:15.955143
Previsão para conclusão:  Mon Feb 12 05:08:10 2024

Iteração  19  de  100  concluída em  0:25:51.513421
Tempo restante previsto:  1 day, 7:57:03.351225
Previsão para conclusão:  Mon Feb 12 05:23:49 2024

Iteração  20  de  100  concluída em  0:02:17.949492
Tempo restante previsto:  1 day, 6:08:41.244250
Previsão para conclusão:  Mon Feb 12 03:37:45 2024

Iteração  21  de  100  concluída em  0:03:55.188718
Tempo restante previsto:  1 day, 4:36:40.313096
Previsão para conclusão:  Mon Feb 12 02:09:39 2024

Iteração  22  de  100  concluída em  0:01:37.080232
Tempo restante previsto:  1 day, 3:04:17.691305
Previsão para conclusão:  Mon Feb 12 00:38:53 2024

WARNING:tensorflow:Layer lstm_38 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  23  de  100  concluída em  2:28:59.103508
Tempo restante previsto:  1 day, 10:05:59.022640
Previsão para conclusão:  Mon Feb 12 10:09:34 2024

Iteração  24  de  100  concluída em  0:11:14.584049
Tempo restante previsto:  1 day, 8:52:27.296165
Previsão para conclusão:  Mon Feb 12 09:07:17 2024

WARNING:tensorflow:Layer lstm_41 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  25  de  100  concluída em  0:50:25.928035
Tempo restante previsto:  1 day, 9:44:36.680559
Previsão para conclusão:  Mon Feb 12 10:49:52 2024

WARNING:tensorflow:Layer lstm_42 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_43 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  26  de  100  concluída em  0:25:05.103371
Tempo restante previsto:  1 day, 9:14:47.091904
Previsão para conclusão:  Mon Feb 12 10:45:08 2024
```

```
WARNING:tensorflow:Layer lstm_44 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  27  de  100  concluída em  0:48:23.272864
Tempo restante previsto:  1 day, 9:50:02.380379
Previsão para conclusão:  Mon Feb 12 12:08:46 2024

Iteração  28  de  100  concluída em  0:11:56.657827
Tempo restante previsto:  1 day, 8:43:00.289017
Previsão para conclusão:  Mon Feb 12 11:13:41 2024

WARNING:tensorflow:Layer lstm_47 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  29  de  100  concluída em  0:10:43.725727
Tempo restante previsto:  1 day, 7:36:42.570890
Previsão para conclusão:  Mon Feb 12 10:18:07 2024

Iteração  30  de  100  concluída em  0:03:33.130481
Tempo restante previsto:  1 day, 6:16:53.690331
Previsão para conclusão:  Mon Feb 12 09:01:51 2024

WARNING:tensorflow:Layer lstm_50 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_51 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  31  de  100  concluída em  1:06:39.634709
Tempo restante previsto:  1 day, 7:26:32.335590
Previsão para conclusão:  Mon Feb 12 11:18:09 2024

Iteração  32  de  100  concluída em  0:23:20.896900
Tempo restante previsto:  1 day, 6:52:55.099395
Previsão para conclusão:  Mon Feb 12 11:07:53 2024

Iteração  33  de  100  concluída em  0:19:24.428393
Tempo restante previsto:  1 day, 6:11:40.782945
Previsão para conclusão:  Mon Feb 12 10:46:03 2024

WARNING:tensorflow:Layer lstm_56 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  34  de  100  concluída em  0:49:58.476045
Tempo restante previsto:  1 day, 6:32:51.614037
Previsão para conclusão:  Mon Feb 12 11:57:12 2024

Iteração  35  de  100  concluída em  0:09:33.139591
Tempo restante previsto:  1 day, 5:32:35.697794
Previsão para conclusão:  Mon Feb 12 11:06:29 2024
```

```
WARNING:tensorflow:Layer lstm_59 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_60 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  36  de  100  concluída em  2:33:27.456759
Tempo restante previsto:  1 day, 8:58:58.406174
Previsão para conclusão:  Mon Feb 12 17:06:20 2024

Iteração  37  de  100  concluída em  0:06:17.398634
Tempo restante previsto:  1 day, 7:47:21.814327
Previsão para conclusão:  Mon Feb 12 16:01:00 2024

WARNING:tensorflow:Layer lstm_62 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_63 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  38  de  100  concluída em  0:07:27.873144
Tempo restante previsto:  1 day, 6:41:10.189719
Previsão para conclusão:  Mon Feb 12 15:02:17 2024

WARNING:tensorflow:Layer lstm_64 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_65 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  39  de  100  concluída em  0:05:59.861897
Tempo restante previsto:  1 day, 5:35:37.098300
Previsão para conclusão:  Mon Feb 12 14:02:43 2024

Iteração  40  de  100  concluída em  0:03:14.146441
Tempo restante previsto:  1 day, 4:28:45.809401
Previsão para conclusão:  Mon Feb 12 12:59:06 2024

WARNING:tensorflow:Layer lstm_68 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_69 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  41  de  100  concluída em  0:09:19.125416
Tempo restante previsto:  1 day, 3:34:03.740071
Previsão para conclusão:  Mon Feb 12 12:13:43 2024

WARNING:tensorflow:Layer lstm_70 will not use cuDNN kernels since it
```

```
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_71 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  42  de  100  concluída em  0:27:28.576052
Tempo restante previsto:  1 day, 3:07:27.692789
Previsão para conclusão:  Mon Feb 12 12:14:36 2024

WARNING:tensorflow:Layer lstm_72 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  43  de  100  concluída em  0:52:35.673964
Tempo restante previsto:  1 day, 3:15:17.080839
Previsão para conclusão:  Mon Feb 12 13:15:01 2024

Iteração  44  de  100  concluída em  0:05:07.603481
Tempo restante previsto:  1 day, 2:17:47.418615
Previsão para conclusão:  Mon Feb 12 12:22:39 2024

WARNING:tensorflow:Layer lstm_75 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_76 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  45  de  100  concluída em  0:27:51.895487
Tempo restante previsto:  1 day, 1:51:25.506655
Previsão para conclusão:  Mon Feb 12 12:24:09 2024

WARNING:tensorflow:Layer lstm_77 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  46  de  100  concluída em  2:25:44.561800
Tempo restante previsto:  1 day, 3:48:29.884834
Previsão para conclusão:  Mon Feb 12 16:46:58 2024

WARNING:tensorflow:Layer lstm_78 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_79 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  47  de  100  concluída em  0:20:42.781937
Tempo restante previsto:  1 day, 3:08:04.565728
Previsão para conclusão:  Mon Feb 12 16:27:15 2024

WARNING:tensorflow:Layer lstm_80 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
```

```
WARNING:tensorflow:Layer lstm_81 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  48  de  100  concluída em  0:48:31.502330
Tempo restante previsto:  1 day, 2:59:45.829445
Previsão para conclusão:  Mon Feb 12 17:07:28 2024

Iteração  49  de  100  concluída em  0:07:27.451334
Tempo restante previsto:  1 day, 2:05:23.414900
Previsão para conclusão:  Mon Feb 12 16:20:33 2024

WARNING:tensorflow:Layer lstm_84 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  50  de  100  concluída em  1:35:31.011949
Tempo restante previsto:  1 day, 2:44:28.501923
Previsão para conclusão:  Mon Feb 12 18:35:09 2024

WARNING:tensorflow:Layer lstm_85 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  51  de  100  concluída em  0:45:09.802223
Tempo restante previsto:  1 day, 2:27:55.669456
Previsão para conclusão:  Mon Feb 12 19:03:46 2024

WARNING:tensorflow:Layer lstm_86 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  52  de  100  concluída em  0:10:58.204774
Tempo restante previsto:  1 day, 1:37:24.109837
Previsão para conclusão:  Mon Feb 12 18:24:13 2024

Iteração  53  de  100  concluída em  0:18:23.134718
Tempo restante previsto:  1 day, 0:55:13.461103
Previsão para conclusão:  Mon Feb 12 18:00:25 2024

WARNING:tensorflow:Layer lstm_89 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  54  de  100  concluída em  0:08:36.065659
Tempo restante previsto:  1 day, 0:05:13.844096
Previsão para conclusão:  Mon Feb 12 17:19:02 2024

WARNING:tensorflow:Layer lstm_90 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  55  de  100  concluída em  2:16:55.008261
Tempo restante previsto:  1 day, 1:06:23.634452
Previsão para conclusão:  Mon Feb 12 20:37:07 2024
```

```
Iteração  56  de  100  concluída em  0:10:19.351766
Tempo restante previsto:  1 day, 0:16:29.678451
Previsão para conclusão:  Mon Feb 12 19:57:32 2024

Iteração  57  de  100  concluída em  0:05:10.034295
Tempo restante previsto:  23:23:54.843892
Previsão para conclusão:  Mon Feb 12 19:10:07 2024

Iteração  58  de  100  concluída em  0:03:07.216420
Tempo restante previsto:  22:31:24.933714
Previsão para conclusão:  Mon Feb 12 18:20:44 2024

Iteração  59  de  100  concluída em  0:05:50.534857
Tempo restante previsto:  21:42:34.482524
Previsão para conclusão:  Mon Feb 12 17:37:45 2024

WARNING:tensorflow:Layer lstm_97 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  60  de  100  concluída em  0:38:10.143173
Tempo restante previsto:  21:17:47.754366
Previsão para conclusão:  Mon Feb 12 17:51:08 2024

WARNING:tensorflow:Layer lstm_98 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  61  de  100  concluída em  0:14:42.807989
Tempo restante previsto:  20:36:48.770921
Previsão para conclusão:  Mon Feb 12 17:24:52 2024

WARNING:tensorflow:Layer lstm_99 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_100 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  62  de  100  concluída em  0:19:46.521426
Tempo restante previsto:  19:59:56.575997
Previsão para conclusão:  Mon Feb 12 17:07:46 2024

Iteração  63  de  100  concluída em  0:02:58.441085
Tempo restante previsto:  19:13:12.892402
Previsão para conclusão:  Mon Feb 12 16:24:01 2024

Iteração  64  de  100  concluída em  0:15:51.278726
Tempo restante previsto:  18:35:30.119985
Previsão para conclusão:  Mon Feb 12 16:02:09 2024

WARNING:tensorflow:Layer lstm_104 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
```

```
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_105 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  65  de  100  concluída em  3:56:40.409982
Tempo restante previsto:  20:04:09.514396
Previsão para conclusão:  Mon Feb 12 21:27:29 2024

WARNING:tensorflow:Layer lstm_106 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  66  de  100  concluída em  0:28:28.877418
Tempo restante previsto:  19:29:23.836367
Previsão para conclusão:  Mon Feb 12 21:21:12 2024

WARNING:tensorflow:Layer lstm_107 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  67  de  100  concluída em  0:42:39.698275
Tempo restante previsto:  19:02:13.864774
Previsão para conclusão:  Mon Feb 12 21:36:42 2024

WARNING:tensorflow:Layer lstm_108 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  68  de  100  concluída em  0:31:55.140533
Tempo restante previsto:  18:29:14.269407
Previsão para conclusão:  Mon Feb 12 21:35:38 2024

Iteração  69  de  100  concluída em  0:00:43.790971
Tempo restante previsto:  17:41:21.549028
Previsão para conclusão:  Mon Feb 12 20:48:29 2024

Iteração  70  de  100  concluída em  0:19:04.113030
Tempo restante previsto:  17:03:12.659573
Previsão para conclusão:  Mon Feb 12 20:29:24 2024

Iteração  71  de  100  concluída em  0:05:44.445084
Tempo restante previsto:  16:19:46.866548
Previsão para conclusão:  Mon Feb 12 19:51:43 2024

WARNING:tensorflow:Layer lstm_112 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_113 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  72  de  100  concluída em  0:20:48.892190
Tempo restante previsto:  15:43:40.741704
Previsão para conclusão:  Mon Feb 12 19:36:25 2024
```

```
Iteração  73  de  100  concluída em  0:05:25.824455
Tempo restante previsto:  15:01:53.048381
Previsão para conclusão:  Mon Feb 12 19:00:04 2024

WARNING:tensorflow:Layer lstm_116 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  74  de  100  concluída em  0:45:47.565639
Tempo restante previsto:  14:36:20.658988
Previsão para conclusão:  Mon Feb 12 19:20:19 2024

WARNING:tensorflow:Layer lstm_117 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  75  de  100  concluída em  0:23:20.247874
Tempo restante previsto:  14:02:10.830501
Previsão para conclusão:  Mon Feb 12 19:09:29 2024

Iteração  76  de  100  concluída em  0:02:00.343299
Tempo restante previsto:  13:21:00.234149
Previsão para conclusão:  Mon Feb 12 18:30:19 2024

WARNING:tensorflow:Layer lstm_119 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  77  de  100  concluída em  1:41:13.372354
Tempo restante previsto:  13:13:03.484971
Previsão para conclusão:  Mon Feb 12 20:03:35 2024

Iteração  78  de  100  concluída em  0:08:27.149723
Tempo restante previsto:  12:34:10.548749
Previsão para conclusão:  Mon Feb 12 19:33:10 2024

WARNING:tensorflow:Layer lstm_121 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  79  de  100  concluída em  0:14:05.200028
Tempo restante previsto:  11:57:42.254679
Previsão para conclusão:  Mon Feb 12 19:10:47 2024

WARNING:tensorflow:Layer lstm_122 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_123 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  80  de  100  concluída em  0:07:49.016770
Tempo restante previsto:  11:20:04.087564
Previsão para conclusão:  Mon Feb 12 18:40:57 2024
```

```
WARNING:tensorflow:Layer lstm_124 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  81  de  100  concluída em  0:25:17.687562
Tempo restante previsto:  10:47:41.974613
Previsão para conclusão:  Mon Feb 12 18:33:53 2024

Iteração  82  de  100  concluída em  0:13:52.807709
Tempo restante previsto:  10:12:43.169635
Previsão para conclusão:  Mon Feb 12 18:12:47 2024

WARNING:tensorflow:Layer lstm_126 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_127 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  83  de  100  concluída em  0:12:59.363765
Tempo restante previsto:  9:38:02.636459
Previsão para conclusão:  Mon Feb 12 17:51:06 2024

WARNING:tensorflow:Layer lstm_128 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  84  de  100  concluída em  0:51:00.818672
Tempo restante previsto:  9:12:01.966702
Previsão para conclusão:  Mon Feb 12 18:16:06 2024

WARNING:tensorflow:Layer lstm_129 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  85  de  100  concluída em  0:04:30.277352
Tempo restante previsto:  8:36:07.891059
Previsão para conclusão:  Mon Feb 12 17:44:42 2024

Iteração  86  de  100  concluída em  0:03:28.033327
Tempo restante previsto:  8:00:46.044394
Previsão para conclusão:  Mon Feb 12 17:12:48 2024

Iteração  87  de  100  concluída em  0:10:16.452718
Tempo restante previsto:  7:27:18.610576
Previsão para conclusão:  Mon Feb 12 16:49:37 2024

Iteração  88  de  100  concluída em  0:11:59.015153
Tempo restante previsto:  6:54:39.106715
Previsão para conclusão:  Mon Feb 12 16:28:57 2024

WARNING:tensorflow:Layer lstm_136 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
```

```
Iteração 89 de 100 concluída em 1:48:18.136765
Tempo restante previsto: 6:36:31.620363
Previsão para conclusão: Mon Feb 12 17:59:08 2024

Iteração 90 de 100 concluída em 0:07:02.963465
Tempo restante previsto: 6:02:53.874229
Previsão para conclusão: Mon Feb 12 17:32:33 2024

Iteração 91 de 100 concluída em 0:04:29.145643
Tempo restante previsto: 5:29:32.584901
Previsão para conclusão: Mon Feb 12 17:03:41 2024

Iteração 92 de 100 concluída em 0:02:28.107342
Tempo restante previsto: 4:56:35.794533
Previsão para conclusão: Mon Feb 12 16:33:12 2024

Iteração 93 de 100 concluída em 0:11:34.945210
Tempo restante previsto: 4:25:11.250476
Previsão para conclusão: Mon Feb 12 16:13:22 2024

Iteração 94 de 100 concluída em 0:05:18.268326
Tempo restante previsto: 3:53:39.959425
Previsão para conclusão: Mon Feb 12 15:47:09 2024

WARNING:tensorflow:Layer lstm_145 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração 95 de 100 concluída em 0:06:53.837771
Tempo restante previsto: 3:22:48.826601
Previsão para conclusão: Mon Feb 12 15:23:12 2024

WARNING:tensorflow:Layer lstm_146 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração 96 de 100 concluída em 0:11:28.651394
Tempo restante previsto: 2:52:44.813275
Previsão para conclusão: Mon Feb 12 15:04:37 2024

Iteração 97 de 100 concluída em 0:11:35.145137
Tempo restante previsto: 2:23:04.131832
Previsão para conclusão: Mon Feb 12 14:46:31 2024

Iteração 98 de 100 concluída em 0:02:21.410615
Tempo restante previsto: 1:53:23.002782
Previsão para conclusão: Mon Feb 12 14:19:11 2024

Iteração 99 de 100 concluída em 0:09:52.909294
Tempo restante previsto: 1:24:28.681135
Previsão para conclusão: Mon Feb 12 14:00:10 2024
```

```
WARNING:tensorflow:Layer lstm_152 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Iteração  100  de  100  concluída em  0:08:28.533190
Tempo restante previsto:  0:55:55.500213
Previsão para conclusão:  Mon Feb 12 13:40:05 2024
```

```
# Resultados
sorted_results = results.sort_values('MSE')
sorted_results
```

```
            MSE     std_MSE  alpha epochs       nodes  dropout
activation  \
83   203.720261   44.383698   0.20     10       [128]      0.1
sigmoid
45   207.975240   43.847645   0.05     20        [64]      0.2
sigmoid
35   212.387583   45.864200   0.40     20  [128, 256]      0.2
sigmoid
76   216.469981   38.163559   0.30     20       [128]      0.4
sigmoid
50   220.752625   34.394522   0.01     20       [256]      0.1
sigmoid
..          ...         ...    ...    ...         ...      ...
...
79  1714.035604    4.427255   0.40      5  [128, 256]      0.2
sigmoid
94  1846.295410   58.176646   0.20     10        [64]      0.4
sigmoid
40  2333.661540   61.908300   0.40      5   [64, 128]      0.3
sigmoid
99  3572.084635  115.423514   0.40     10        [32]      0.3
sigmoid
68  4669.416992  111.000534   0.10      5        [32]      0.3
tanh

   batch_size sequence_length sensor_length
83         32              35            14
45         32              40            14
35         64              40            14
76         32              35            14
50        128              40            10
..        ...             ...           ...
79        256              20            10
94        256              25            10
40        256              35            10
99        256              35            14
68        256              20            14
```

```
[100 rows x 10 columns]
```

8. Modelo final

```
# Definindo quais sensores geraram melhor resultado
sensor_length = int(sorted_results['sensor_length'].iloc[0])
if sensor_length == 10:
       remaining_sensors = ['s_2', 's_3', 's_4', 's_7', 's_11',
's_12', 's_15', 's_17', 's_20', 's_21']
else:
       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']


drop_sensors = [element for element in sensor_names if element not in
remaining_sensors]

# arametros
alpha = float(sorted_results['alpha'].iloc[0])
epochs = int(sorted_results['epochs'].iloc[0])
specific_lags = [1,2,3,4,5,10,20]
nodes_per_layer = [eval(i) for i in
sorted_results['nodes'].iloc[0].replace("
","").replace("[","").replace("]","").split(",")]
dropout = float(sorted_results['dropout'].iloc[0])
activation = str(sorted_results['activation'].iloc[0])
batch_size = int(sorted_results['batch_size'].iloc[0])
sequence_length = int(sorted_results['sequence_length'].iloc[0])

print("Parametros do modelo final:")
print()
print("alpha = ", alpha)
print("epochs = ", epochs)
print("nodes = ", nodes_per_layer)
print("dropout = ", dropout)
print("activation = ", activation)
print("batch_size = ", batch_size)
print("sequence_length = ", sequence_length)
print("sensor_length = ", sensor_length)
print("remaining_sensors = ", remaining_sensors)
```

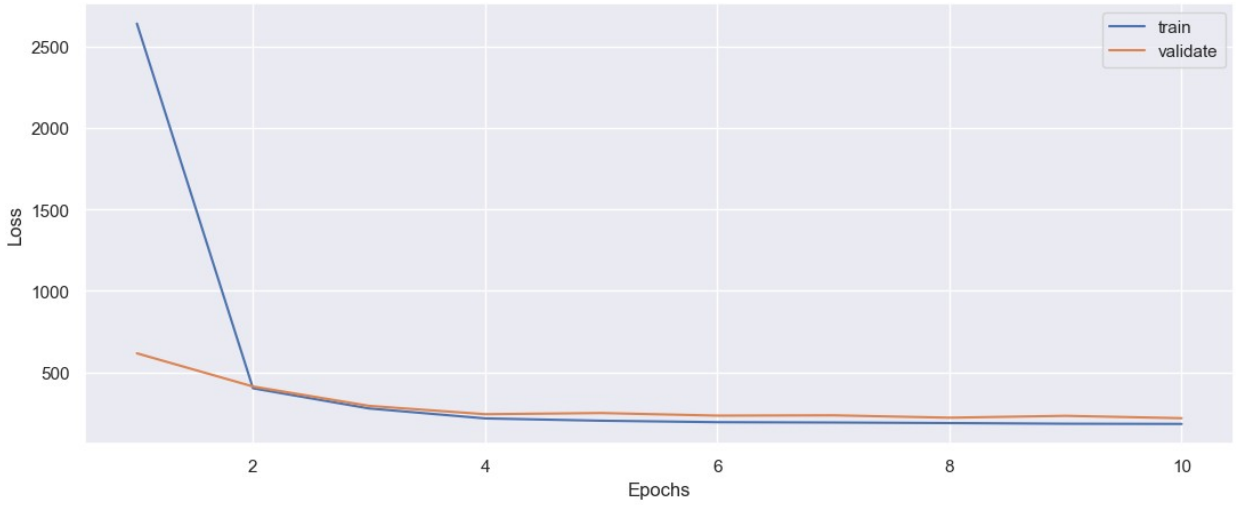
```
Parametros do modelo final:

alpha =  0.2
epochs =  10
nodes =  [128]
dropout =  0.1
```

```
activation =  sigmoid
batch_size =  32
sequence_length =  35
sensor_length =  14
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']

# Carregando pesos
weights_file = 'fd004_model_weights.m5'

# Preparando os dados
X_train_interim, X_test_interim = prep_data(train, test, drop_sensors,
remaining_sensors, alpha)

train_array = gen_data_wrapper(X_train_interim, sequence_length,
remaining_sensors)
label_array = gen_label_wrapper(X_train_interim, sequence_length,
['RUL'])

test_gen =
(list(gen_test_data(X_test_interim[X_test_interim['unidade']==unit_nr]
, sequence_length, remaining_sensors, -99.))
           for unit_nr in X_test_interim['unidade'].unique())
test_array = np.concatenate(list(test_gen)).astype(np.float32)

# Separando em grupos de treino e validacao
gss = GroupShuffleSplit(n_splits=1, train_size=0.80, random_state=42)
for train_unit, val_unit in
gss.split(X_train_interim['unidade'].unique(),
groups=X_train_interim['unidade'].unique()):
    train_unit = X_train_interim['unidade'].unique()[train_unit]
    val_unit = X_train_interim['unidade'].unique()[val_unit]

    train_split_array = gen_data_wrapper(X_train_interim,
sequence_length, remaining_sensors, train_unit)
    train_split_label = gen_label_wrapper(X_train_interim,
sequence_length, ['RUL'], train_unit)

    val_split_array = gen_data_wrapper(X_train_interim,
sequence_length, remaining_sensors, val_unit)
    val_split_label = gen_label_wrapper(X_train_interim,
sequence_length, ['RUL'], val_unit)

# Criando o modelo final
input_shape = (sequence_length, len(remaining_sensors))
final_model = create_model(input_shape, nodes_per_layer, dropout,
activation, weights_file)

final_model.compile(loss='mean_squared_error', optimizer='adam')
final_model.load_weights(weights_file)
```

```
# Treinando o modelo final
history = final_model.fit(train_array, label_array,
                validation_data=(val_split_array, val_split_label),
                epochs=epochs,
                batch_size=batch_size)
```

```
WARNING:tensorflow:Layer lstm_153 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
Epoch 1/10
1650/1650 [==============================] - 176s 106ms/step - loss:
2642.4353 - val_loss: 615.9604
Epoch 2/10
1650/1650 [==============================] - 174s 106ms/step - loss:
400.9756 - val_loss: 411.9127
Epoch 3/10
1650/1650 [==============================] - 170s 103ms/step - loss:
277.0396 - val_loss: 292.9207
Epoch 4/10
1650/1650 [==============================] - 171s 104ms/step - loss:
215.7764 - val_loss: 241.6957
Epoch 5/10
1650/1650 [==============================] - 169s 102ms/step - loss:
201.6434 - val_loss: 248.9064
Epoch 6/10
1650/1650 [==============================] - 173s 105ms/step - loss:
192.6869 - val_loss: 232.9602
Epoch 7/10
1650/1650 [==============================] - 174s 105ms/step - loss:
190.6402 - val_loss: 234.7803
Epoch 8/10
1650/1650 [==============================] - 174s 105ms/step - loss:
187.2545 - val_loss: 220.1553
Epoch 9/10
1650/1650 [==============================] - 172s 104ms/step - loss:
183.1554 - val_loss: 231.7045
Epoch 10/10
1650/1650 [==============================] - 173s 105ms/step - loss:
181.9132 - val_loss: 217.0807
```

```
plot_loss(history)
```




```
# Avaliando o modelo final
y_hat_train = final_model.predict(train_array)
avaliar(label_array, y_hat_train, 'treino')

y_hat_test = final_model.predict(test_array)
avaliar(y_test, y_hat_test)
```

```
1650/1650 [==============================] - 22s 13ms/step
conjunto de treino -> RMSE:13.021495819091797, R2:0.9020147651893394
8/8 [==============================] - 0s 16ms/step
conjunto de teste -> RMSE:26.502435564053055, R2:0.7637349673954448
```

```
The Kernel crashed while executing code in the the current cell or a
previous cell. Please review the code in the cell(s) to identify a
possible cause of the failure. Click <a
href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.
View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further
details.
```