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

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

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

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

# Definindo caminho dos dados
dir_path = './CMAPSSData/'
train_file = 'train_FD002.txt'
test_file = 'test_FD002.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_FD002.txt'), sep='\s+',
header=None,
                    names=['RemainingUsefulLife'])
```

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

```
(53759, 26)
```

```
   unidade  ciclo_tempo  config_1  config_2  config_3     s_1     s_2
\
0        1            1   34.9983    0.8400     100.0  449.44  555.32

1        1            2   41.9982    0.8408     100.0  445.00  549.90

2        1            3   24.9988    0.6218      60.0  462.54  537.31

3        1            4   42.0077    0.8416     100.0  445.00  549.51

4        1            5   25.0005    0.6203      60.0  462.54  537.07


       s_3      s_4   s_5  ...    s_12     s_13     s_14     s_15
s_16  s_17  \
0  1358.61  1137.23  5.48  ...  183.06  2387.72  8048.56   9.3461
0.02   334
1  1353.22  1125.78  3.91  ...  130.42  2387.66  8072.30   9.3774
0.02   330
2  1256.76  1047.45  7.05  ...  164.22  2028.03  7864.87  10.8941
0.02   309
3  1354.03  1126.38  3.91  ...  130.72  2387.61  8068.66   9.3528
0.02   329
4  1257.71  1047.93  7.05  ...  164.31  2028.00  7861.23  10.8963
0.02   309

   s_18    s_19   s_20    s_21
0  2223  100.00  14.73  8.8071
1  2212  100.00  10.41  6.2665
2  1915   84.93  14.08  8.6723
3  2212  100.00  10.59  6.4701
4  1915   84.93  14.13  8.5286

[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  148
1        1            2  147
2        1            3  146
3        1            4  145
4        1            5  144
```

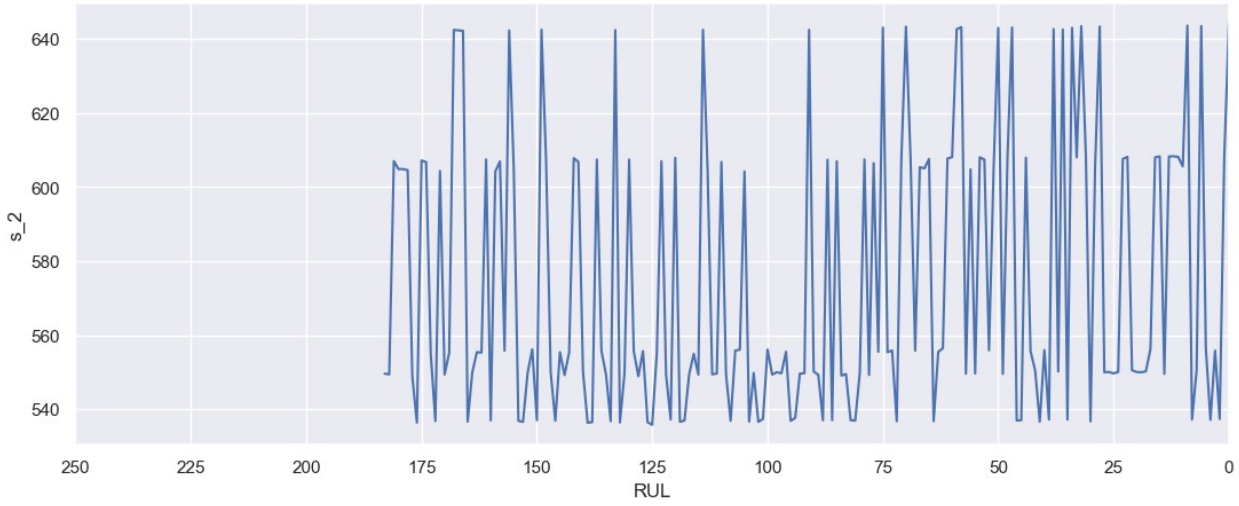
```
# Verificando as 6 condicoes de operacao
settings_df = train[setting_names].copy()
settings_df['config_1'] = settings_df['config_1'].round()
settings_df['config_2'] = settings_df['config_2'].round(decimals=2)
settings_df.groupby(by=setting_names).size()
```

```
config_1  config_2  config_3
0.0       0.00      100.0        8044
10.0      0.25      100.0        8096
20.0      0.70      100.0        8122
25.0      0.62      60.0         8002
35.0      0.84      100.0        8037
42.0      0.84      100.0       13458
dtype: int64
```

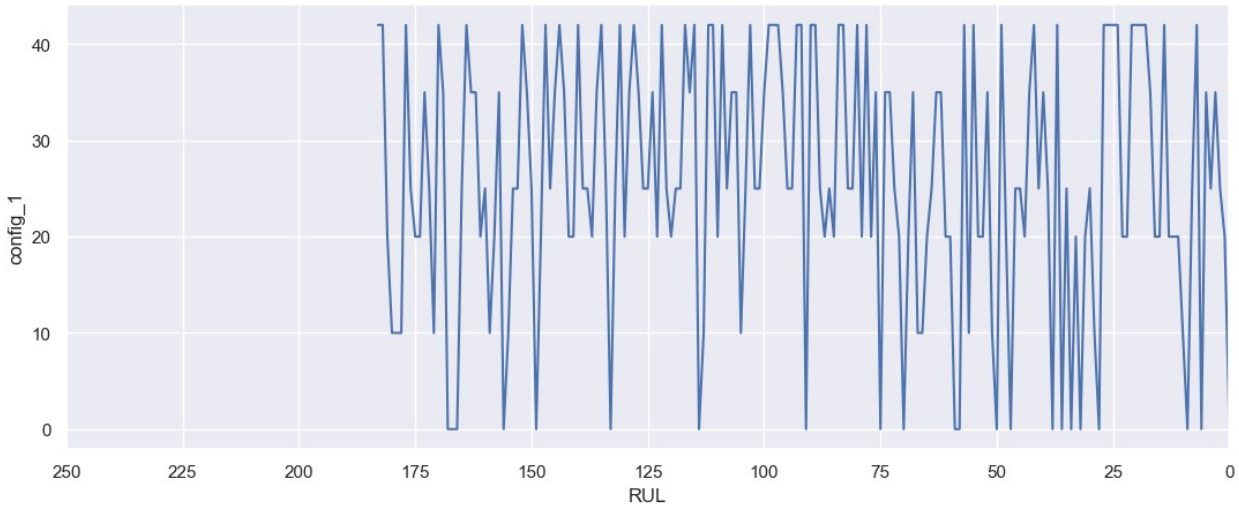
2. Gráficos

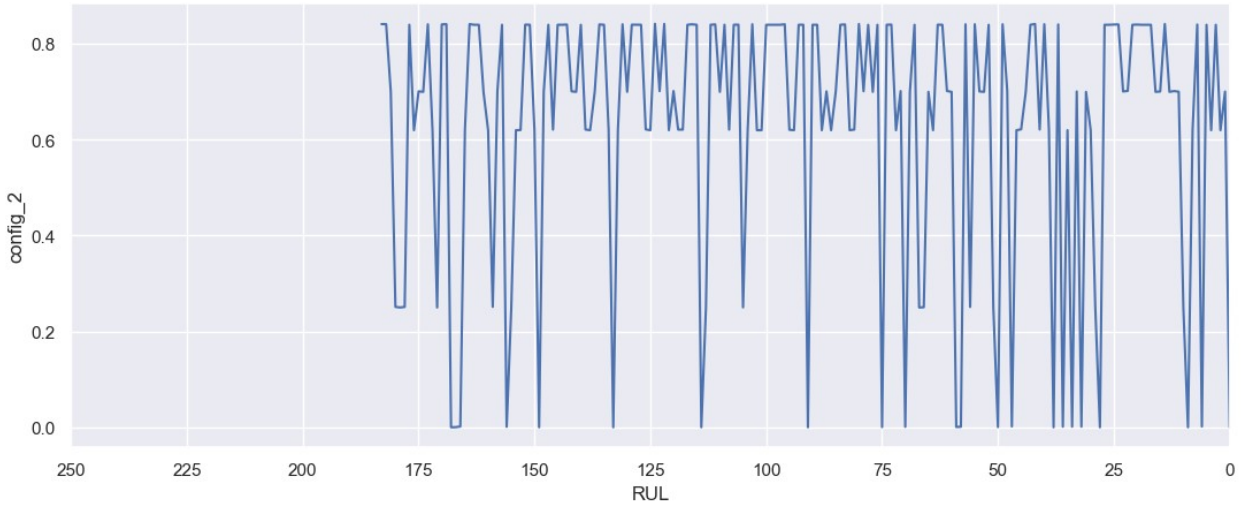
```
def plot_sinal(df, signal_name, unit_nr=None):
    plt.figure(figsize=(13,5))

    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
```

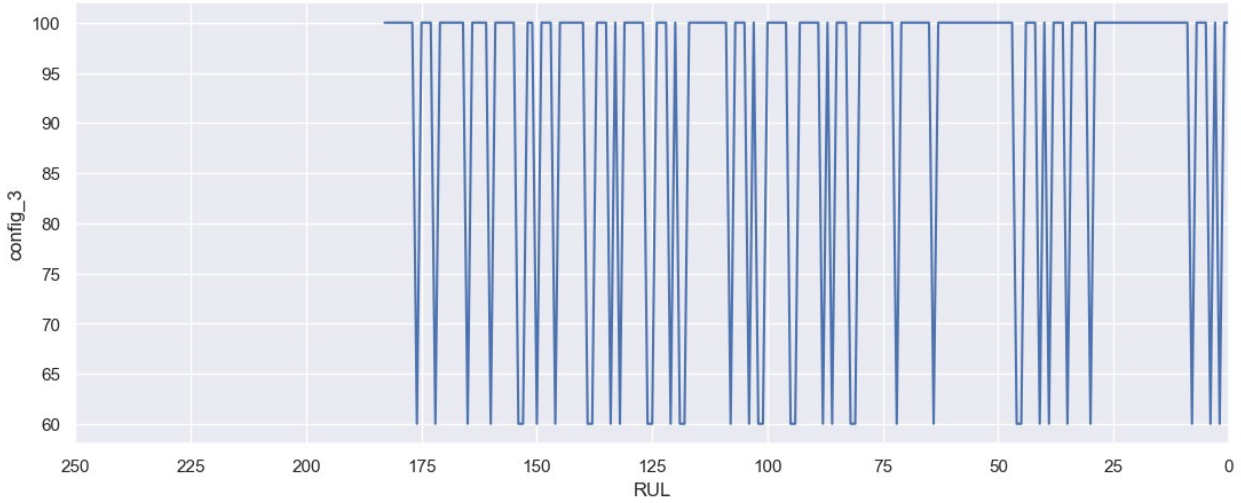
```
    plt.xticks(np.arange(0, 275, 25))
    plt.ylabel(signal_name)
    plt.xlabel('RUL')
    plt.show()

# Protando apenas o s_2 como exemplo
plot_sinal(train, 's_2', unit_nr=10) # s_2 vs RUL para unidade 10
```




```
# Plotando as 3 configuracoes para a unidade 10
for setting in setting_names:
    plot_sinal(train, setting, unit_nr=10)
```







3. 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)
```

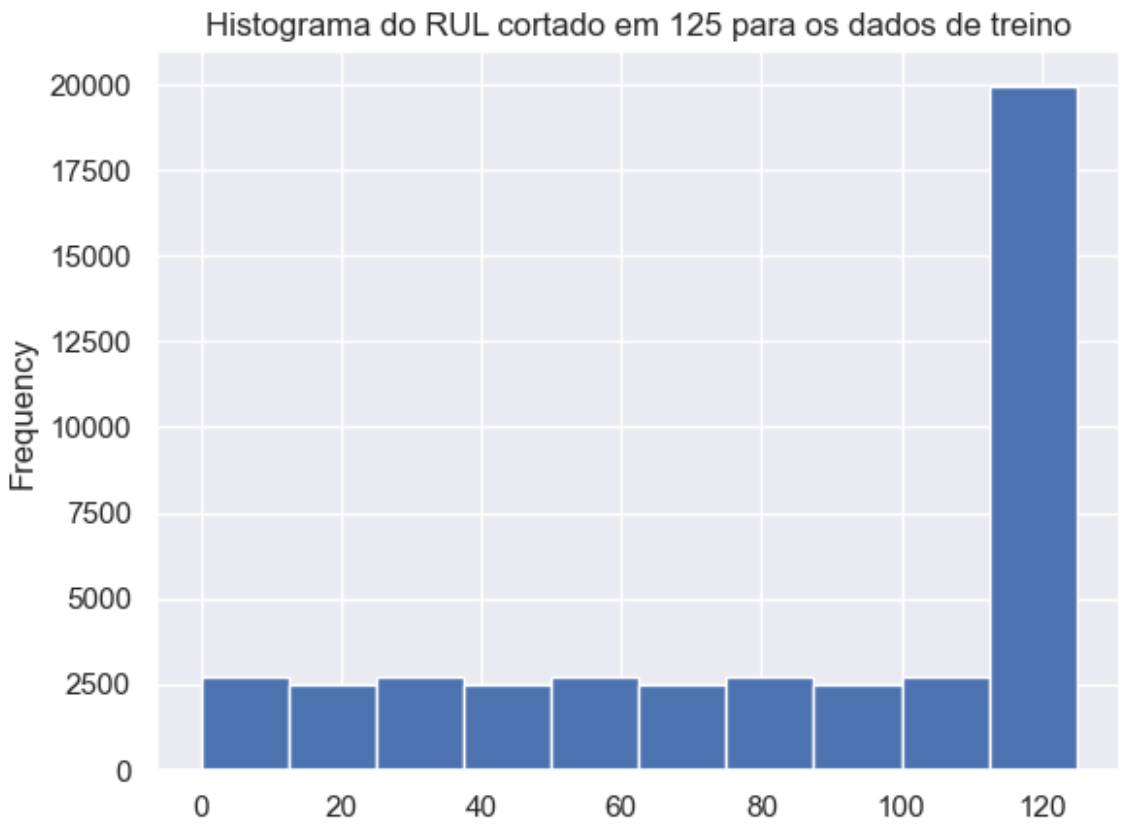
```
# Usando apenas a ultima linha para cada unidade para obter o valor
real do RUL
X_test = test.drop('ciclo_tempo',
axis=1).groupby('unidade').last().copy()

# 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)
```

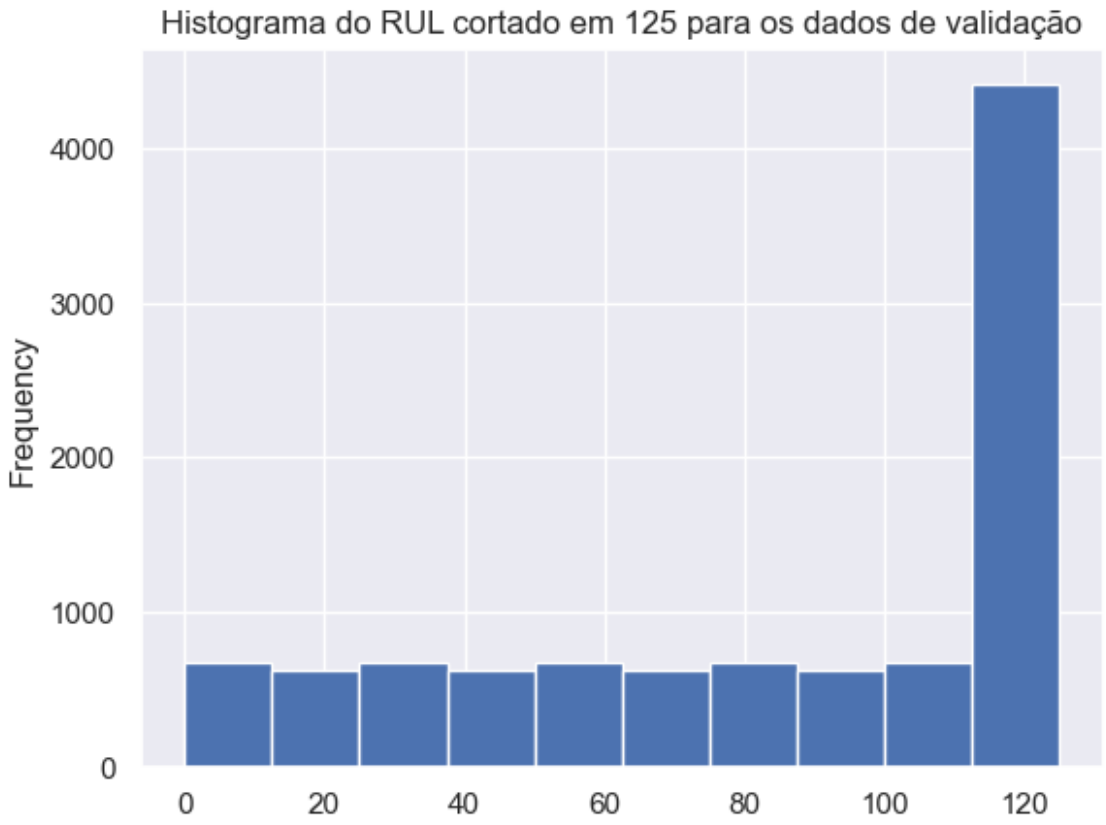
```
conjunto de treino -> RMSE:45.0297373781965, R2:0.5763193277684149
conjunto de teste -> RMSE:33.942725282228444, R2:0.6016440598062158
```

4. Conjunto de Validação

```
# Para obter uma ideia do overfitting antes de executar o modelo com
os dados de teste, eh usado um conjunto de validacao

from sklearn.model_selection import GroupShuffleSplit

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
17  18  21  22  23  24
  27  28  29  30  32  33  35  36  37  38  39  40  41  42  43  44  45
48
  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65
66
  67  68  70  71  72  73  74  75  77  79  80  81  82  83  84  85  86
87
  88  89  90  92  94  95  96  99 100 101 103 104 106 107 108 109 110
111
 112 113 116 117 118 119 121 122 123 124 125 126 127 128 129 130 131
132
 133 134 135 136 137 138 139 141 142 144 146 147 148 149 150 152 153
154
 156 157 158 160 161 162 163 164 165 166 167 169 170 171 172 173 175
176
 177 179 181 183 184 185 187 188 189 190 192 193 194 195 196 198 199
200
 201 203 204 208 209 210 211 215 216 217 218 219 220 222 223 225 226
227
 228 230 231 232 233 234 235 236 239 240 241 242 244 245 246 247 248
249
 250 251 252 253 254 255 256 257 258 260]

unidades_grupo_validacao [  7  10  11  16  19  20  25  26  31  34  46
47  69  76  78  91  93  97
  98 102 105 114 115 120 140 143 145 151 155 159 168 174 178 180 182
186
 191 197 202 205 206 207 212 213 214 221 224 229 237 238 243 259]


# A distribuicao dos dados de treino deve ser semelhante a
distribuicao dos dados de validacao

plt.figure()
y_train_clipped_split.plot(kind='hist')
plt.title("Histograma do RUL cortado em 125 para os dados de treino")
plt.show()
plt.close()

plt.figure()
y_val_clipped_split.plot(kind='hist')
plt.title("Histograma do RUL cortado em 125 para os dados de
validação")
plt.show()
plt.close()
```


Histograma do RUL cortado em 125 para os dados de treino
Frequency
20000
17500
15000
12500
10000
7500
5000
2500
0
0
20
40
60
80
100
120


Histograma do RUL cortado em 125 para os dados de validação
Frequency
4000
3000
2000
1000
0
0
20
40
60
80
100
120

5. MLP - 1ª Tentativa

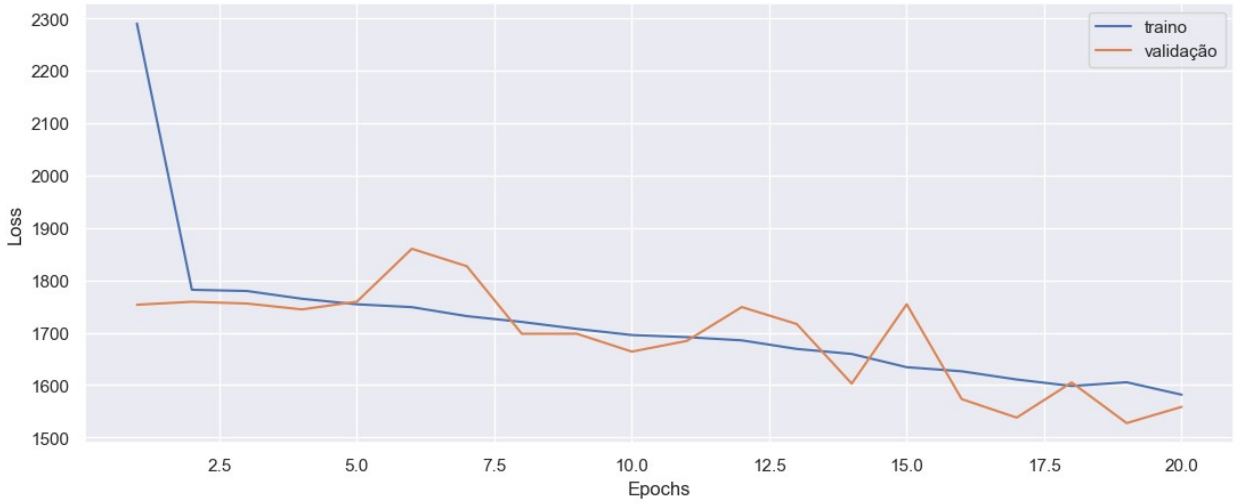
```
train_cols = setting_names+sensor_names # Dados de entrada
input_dim = len(train_cols) # Dimensao do input do modelo

# Criando a arquitetura da rede neural
model = Sequential()
model.add(Dense(16, input_dim=input_dim, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.save_weights('initial_weights_simple_mlp.h5')

epochs = 20

model.compile(loss='mean_squared_error', optimizer='adam')
model.load_weights('initial_weights_simple_mlp.h5') # Salvando os
pesos para pode repetir os testes de forma consistente

# Ajustando o modelo
history = model.fit(X_train_split[train_cols], y_train_clipped_split,
                    validation_data=(X_val_split[train_cols],
y_val_clipped_split),
                    epochs=epochs)
```

```
Epoch 1/20
1359/1359 [==============================] - 3s 2ms/step - loss:
2290.4268 - val_loss: 1753.4263
Epoch 2/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1782.0126 - val_loss: 1759.1102
Epoch 3/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1779.7086 - val_loss: 1755.7511
Epoch 4/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1764.8439 - val_loss: 1744.5570
Epoch 5/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1754.3258 - val_loss: 1759.2827
Epoch 6/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1748.8732 - val_loss: 1860.3269
Epoch 7/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1731.6879 - val_loss: 1827.0001
Epoch 8/20
1359/1359 [==============================] - 2s 1ms/step - loss:
```

```
1720.6494 - val_loss: 1698.0288
Epoch 9/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1707.3751 - val_loss: 1698.0837
Epoch 10/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1695.6295 - val_loss: 1663.9392
Epoch 11/20
1359/1359 [==============================] - 2s 2ms/step - loss:
1691.5673 - val_loss: 1684.3693
Epoch 12/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1685.4316 - val_loss: 1749.0297
Epoch 13/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1669.0699 - val_loss: 1716.6658
Epoch 14/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1659.4850 - val_loss: 1602.9062
Epoch 15/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1634.1747 - val_loss: 1754.4368
Epoch 16/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1626.3687 - val_loss: 1573.2368
Epoch 17/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1610.6757 - val_loss: 1537.8927
Epoch 18/20
1359/1359 [==============================] - 2s 2ms/step - loss:
1598.2765 - val_loss: 1605.0728
Epoch 19/20
1359/1359 [==============================] - 2s 2ms/step - loss:
1605.4086 - val_loss: 1527.4420
Epoch 20/20
1359/1359 [==============================] - 2s 2ms/step - loss:
1581.6495 - val_loss: 1558.4598
```

```
# Plot do 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='traino')
    plt.plot(range(1, len(fit_history.history['val_loss'])+1),
fit_history.history['val_loss'], label='validação')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```

```
plot_loss(history)
```




```
# Testando o modelo
y_hat_train = model.predict(X_train[train_cols])
avaliar(y_train_clipped, y_hat_train, 'treino')

y_hat_test = model.predict(X_test[train_cols])
avaliar(y_test, y_hat_test)
```

```
1680/1680 [==============================] - 2s 901us/step
conjunto de treino -> RMSE:39.91568124192894, R2:0.08221734762785737
9/9 [==============================] - 0s 876us/step
conjunto de teste -> RMSE:51.45479651588735, R2:0.08456058059890736
```

6. Normalização dos dados

```
# Normalizando os dados
from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
scaler.fit(X_train[sensor_names])
X_train_scaled = X_train.copy()
X_train_scaled[sensor_names] =
pd.DataFrame(scaler.transform(X_train[sensor_names]),
columns=sensor_names)


X_test_scaled = X_test.copy()
X_test_scaled[sensor_names] =
pd.DataFrame(scaler.transform(X_test[sensor_names]),
columns=sensor_names, index=X_test.index)
```

```
# Recriando os grupos de treino e validacao
split_result = train_val_group_split(X_train_scaled, y_train_clipped,
gss, train['unidade'], print_groups=True)
X_train_split_scaled, y_train_clipped_split_scaled,
X_val_split_scaled, y_val_clipped_split_scaled = split_result
```

```
unidades_grupo_treino [  1   2   3   4   5   6   8   9  12  13  14  15
17  18  21  22  23  24
  27  28  29  30  32  33  35  36  37  38  39  40  41  42  43  44  45
48
  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65
66
  67  68  70  71  72  73  74  75  77  79  80  81  82  83  84  85  86
87
  88  89  90  92  94  95  96  99 100 101 103 104 106 107 108 109 110
111
 112 113 116 117 118 119 121 122 123 124 125 126 127 128 129 130 131
132
 133 134 135 136 137 138 139 141 142 144 146 147 148 149 150 152 153
154
 156 157 158 160 161 162 163 164 165 166 167 169 170 171 172 173 175
176
 177 179 181 183 184 185 187 188 189 190 192 193 194 195 196 198 199
200
 201 203 204 208 209 210 211 215 216 217 218 219 220 222 223 225 226
227
 228 230 231 232 233 234 235 236 239 240 241 242 244 245 246 247 248
249
 250 251 252 253 254 255 256 257 258 260]

unidades_grupo_validacao [  7  10  11  16  19  20  25  26  31  34  46
47  69  76  78  91  93  97
  98 102 105 114 115 120 140 143 145 151 155 159 168 174 178 180 182
186
 191 197 202 205 206 207 212 213 214 221 224 229 237 238 243 259]
```

7. Perceptron de Múltiplas Camadas (MLP) - Multilayer Perceptron (MLP)

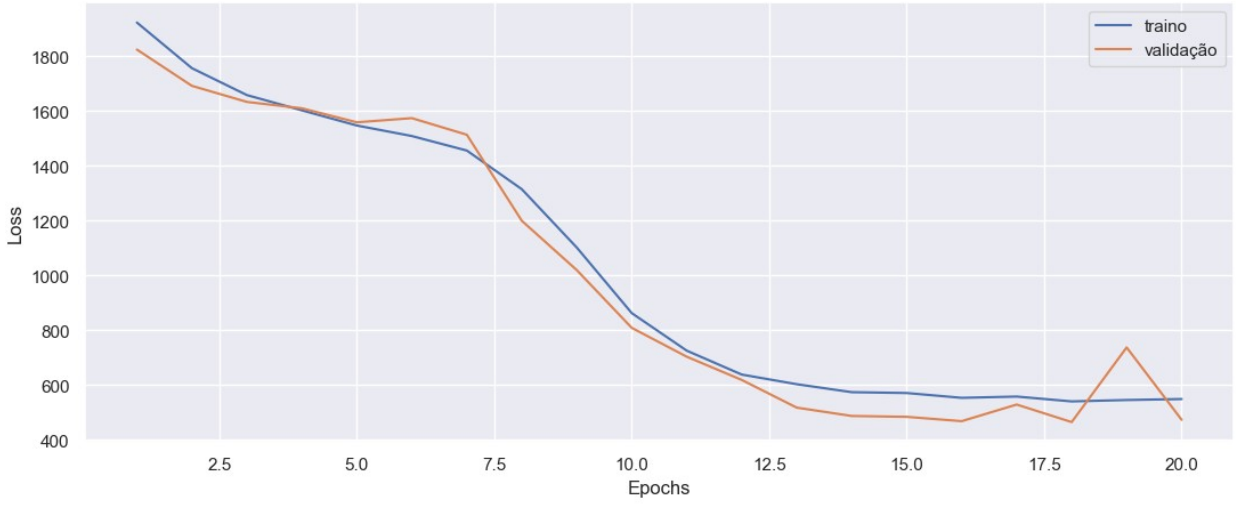
```
train_cols = setting_names+sensor_names # Dados de entrada
input_dim = len(train_cols) # Dimensao do input do modelo

# Criando a arquitetura da rede neural
model = Sequential()
model.add(Dense(16, input_dim=input_dim, activation='relu'))
model.add(Dense(32, activation='relu'))
```

```
model.add(Dense(64, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

epochs = 20

# Ajustando o modelo
history = model.fit(X_train_split_scaled[train_cols],
y_train_clipped_split_scaled,
                    validation_data=(X_val_split_scaled[train_cols],
y_val_clipped_split_scaled),
                    epochs=epochs)
```

```
Epoch 1/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1923.0657 - val_loss: 1824.4838
Epoch 2/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1756.5615 - val_loss: 1692.0946
Epoch 3/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1658.3896 - val_loss: 1633.4777
Epoch 4/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1602.7003 - val_loss: 1610.1454
Epoch 5/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1547.3088 - val_loss: 1559.0049
Epoch 6/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1508.9701 - val_loss: 1574.2631
Epoch 7/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1456.0228 - val_loss: 1513.7056
Epoch 8/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1315.0337 - val_loss: 1199.7887
Epoch 9/20
1359/1359 [==============================] - 2s 1ms/step - loss:
1102.0618 - val_loss: 1019.7873
Epoch 10/20
1359/1359 [==============================] - 2s 1ms/step - loss:
862.8146 - val_loss: 809.0702
Epoch 11/20
1359/1359 [==============================] - 2s 1ms/step - loss:
725.8895 - val_loss: 703.8035
Epoch 12/20
1359/1359 [==============================] - 2s 1ms/step - loss:
638.6942 - val_loss: 619.1010
Epoch 13/20
```

```
1359/1359 [==============================] - 2s 1ms/step - loss:
603.4861 - val_loss: 518.0450
Epoch 14/20
1359/1359 [==============================] - 2s 1ms/step - loss:
574.4549 - val_loss: 487.6472
Epoch 15/20
1359/1359 [==============================] - 2s 1ms/step - loss:
571.3380 - val_loss: 484.4385
Epoch 16/20
1359/1359 [==============================] - 2s 1ms/step - loss:
553.7935 - val_loss: 468.6904
Epoch 17/20
1359/1359 [==============================] - 2s 1ms/step - loss:
558.4355 - val_loss: 529.3086
Epoch 18/20
1359/1359 [==============================] - 2s 1ms/step - loss:
540.7634 - val_loss: 465.2841
Epoch 19/20
1359/1359 [==============================] - 2s 1ms/step - loss:
545.8635 - val_loss: 737.6691
Epoch 20/20
1359/1359 [==============================] - 2s 1ms/step - loss:
549.4098 - val_loss: 473.6833
```

```
# Plot do historico de treinamento
plot_loss(history)
```




```
#Avaliando o modelo
y_hat_train = model.predict(X_train_scaled[train_cols])
avaliar(y_train_clipped, y_hat_train, 'treino')

y_hat_test = model.predict(X_test_scaled[train_cols])
avaliar(y_test, y_hat_test)
```

```
1680/1680 [==============================] - 1s 830us/step
conjunto de treino -> RMSE:22.893595090021503, R2:0.6980877410075903
9/9 [==============================] - 0s 1ms/step
conjunto de teste -> RMSE:34.74786999576291, R2:0.5825213645033326
```

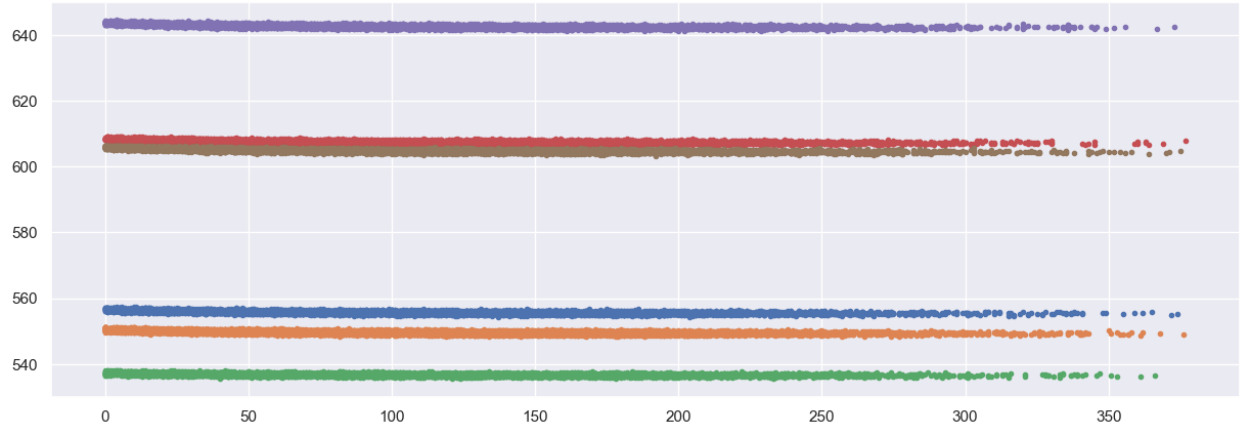
8. Normalização baseada na condição de operação

```
# Listando as condicoes de operacao

# Arredondando os valores para que sejam interpretados como 6
condicoes unicas
X_train_condition = X_train.copy()
X_train_condition['config_1'] = X_train_condition['config_1'].round()
X_train_condition['config_2'] =
X_train_condition['config_2'].round(decimals=2)


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

op_cond = list(X_train_condition['op_cond'].unique())
op_cond
```

```
['35.0_0.84_100.0',
 '42.0_0.84_100.0',
 '25.0_0.62_60.0',
 '20.0_0.7_100.0',
 '0.0_0.0_100.0',
 '10.0_0.25_100.0']
```

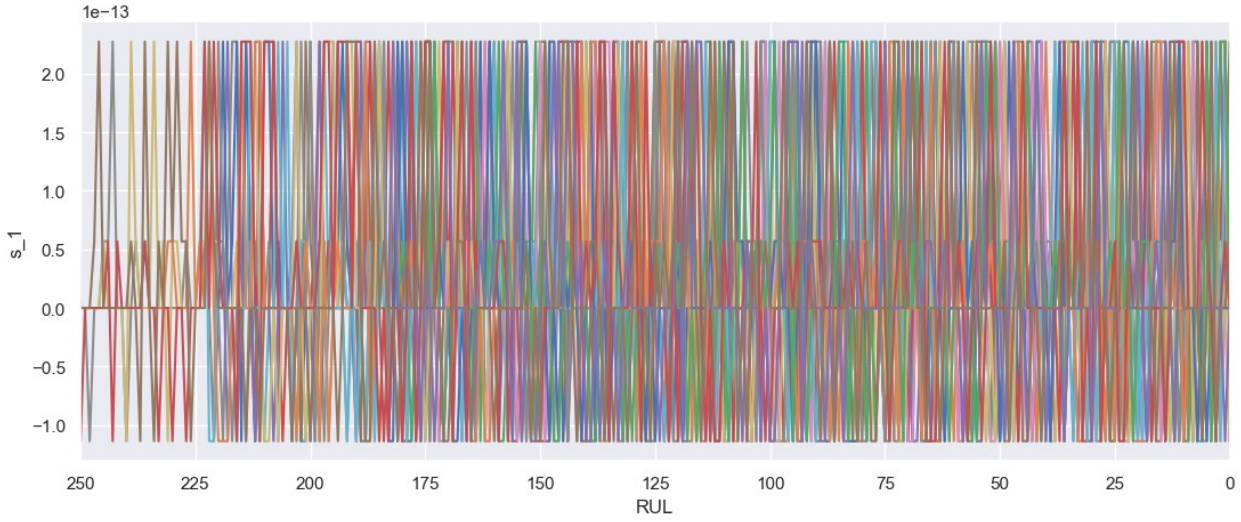
```
# Adicionando a condição de operação ao dataframe
def add_op_cond(df):
    df_op_cond = df.copy()

    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
```

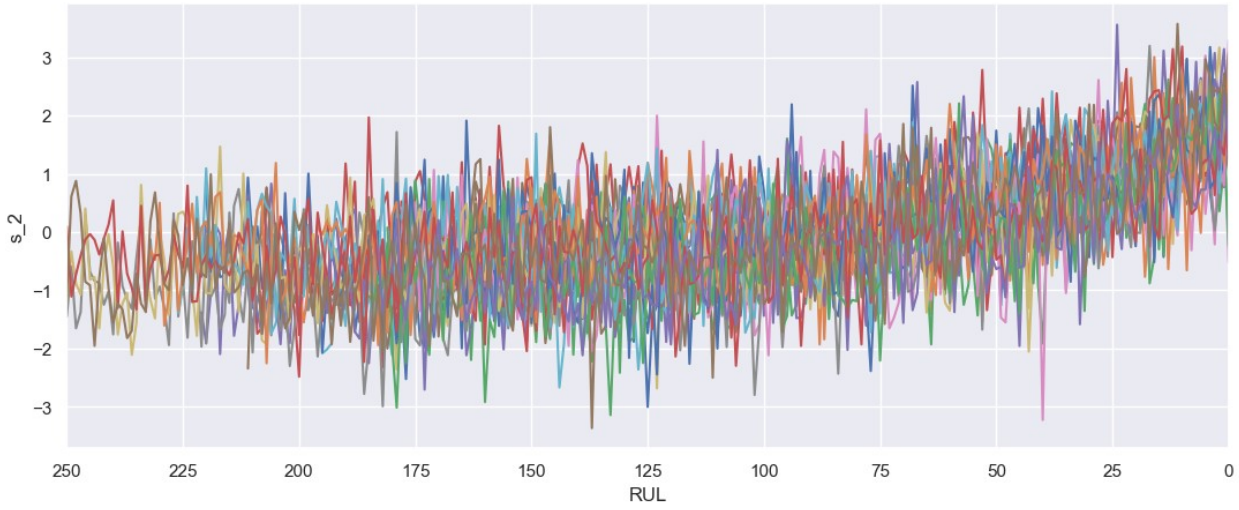
```
X_train_condition = add_op_cond(train)
X_test_condition = add_op_cond(X_test)

# Grafico de todos os motores para o s_2
# O objetivo eh identificar se as condicoes de operacao sao claramente
distinguiveis
plt.figure(figsize=(15,5))

for condition in op_cond:
    data =
X_train_condition.loc[X_train_condition['op_cond']==condition]
    plt.plot(data['RUL'], data['s_2'], '.')
```

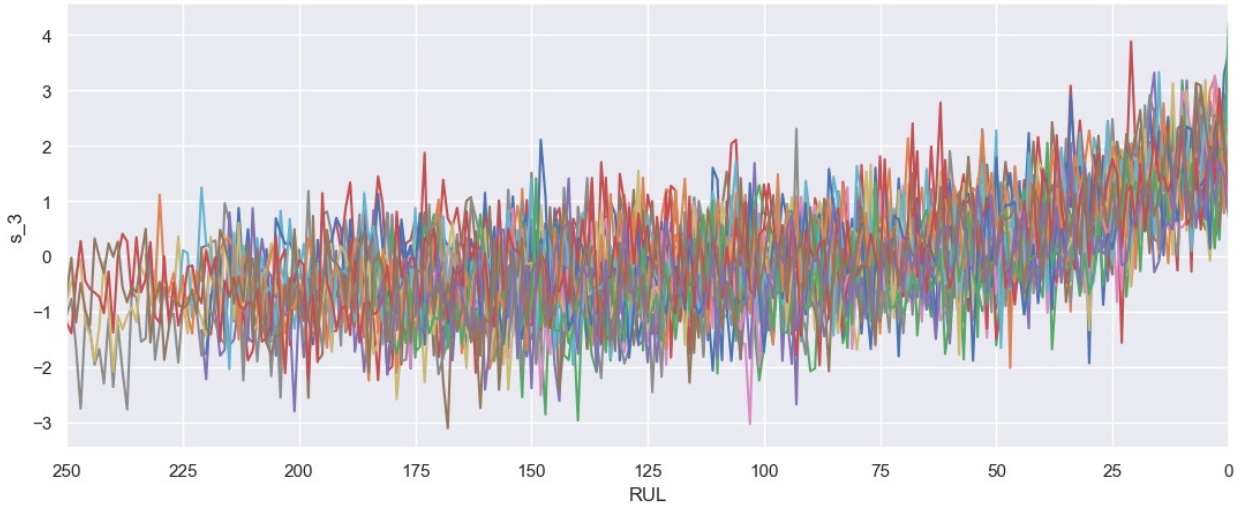


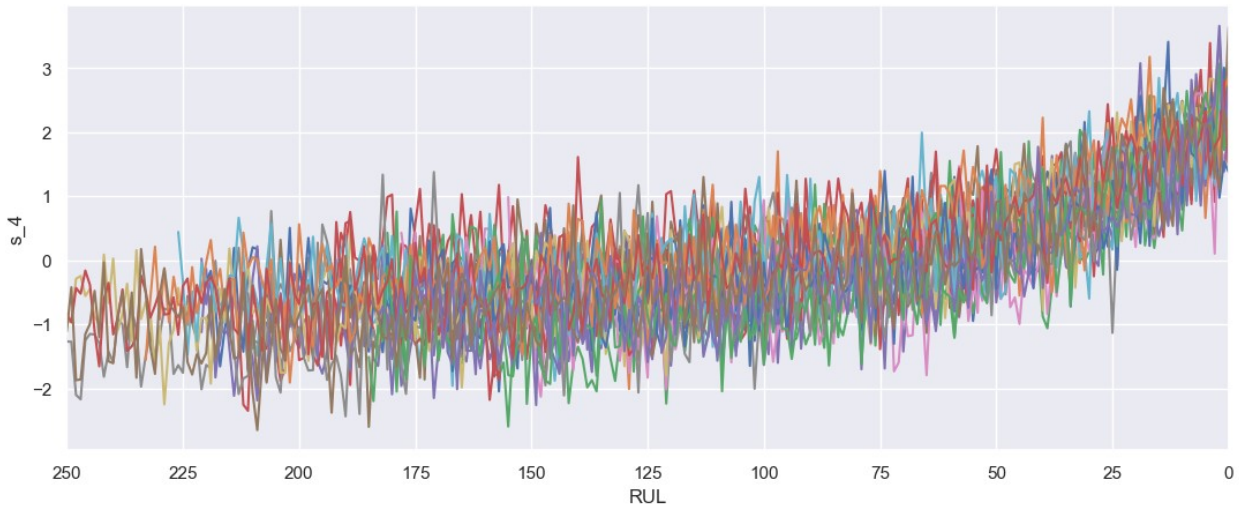

```
# Normalizando baseado na condicao de operacao

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)

# Plotando o grafico de cada sensor apos a normalizacao por op_cond
for sensor in sensor_names:
    plot_sinal(X_train_condition_scaled, sensor)
```

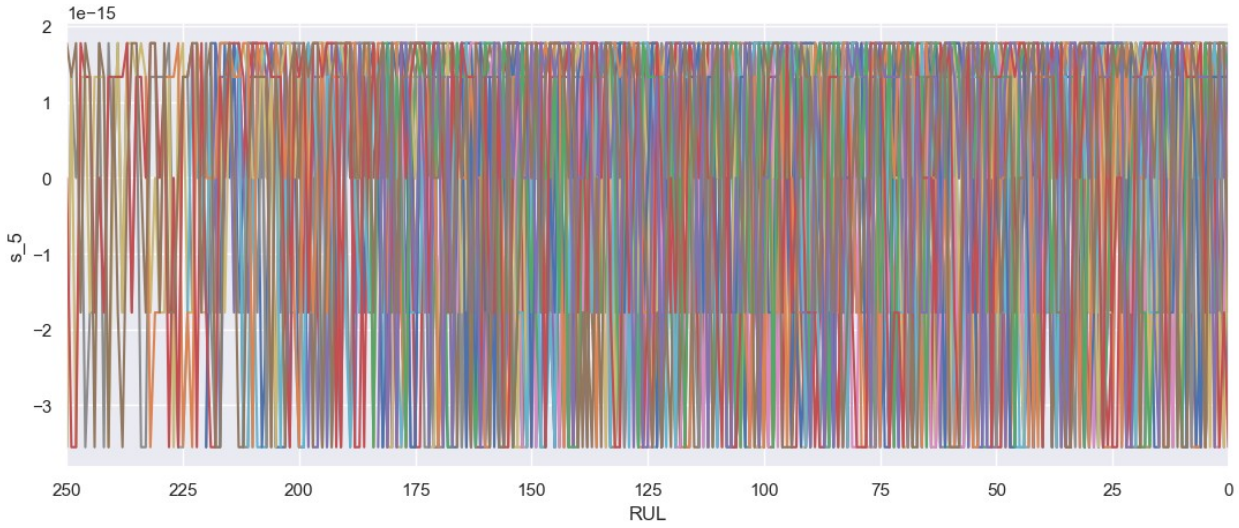
```
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\1051512065.py:7:
FutureWarning: Setting an item of incompatible dtype is deprecated and
will raise in a future error of pandas. Value '[-0.1674041  -
1.59188977 -0.1674041  ...  0.54483874  1.96932442
  2.68156725]' has dtype incompatible with int64, please explicitly
cast to a compatible dtype first.
  df_train.loc[df_train['op_cond']==condition, sensor_names] =
scaler.transform(df_train.loc[df_train['op_cond']==condition,
sensor_names])
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\1051512065.py:8:
FutureWarning: Setting an item of incompatible dtype is deprecated and
will raise in a future error of pandas. Value '[-1.59188977 -
0.87964693  0.54483874 -0.1674041  -1.59188977 -0.1674041
  1.96932442 -0.87964693 -0.87964693  0.54483874 -0.1674041
0.54483874
 -0.1674041  -0.1674041  -0.87964693 -0.87964693 -1.59188977
0.54483874
 -0.1674041   1.25708158  0.54483874 -0.1674041   0.54483874
1.25708158
  0.54483874  1.25708158 -0.87964693 -0.1674041  -0.87964693 -
0.1674041
  0.54483874 -1.59188977  0.54483874  0.54483874 -0.87964693 -
0.87964693]' has dtype incompatible with int64, please explicitly cast
to a compatible dtype first.
  df_test.loc[df_test['op_cond']==condition, sensor_names] =
scaler.transform(df_test.loc[df_test['op_cond']==condition,
sensor_names])
```

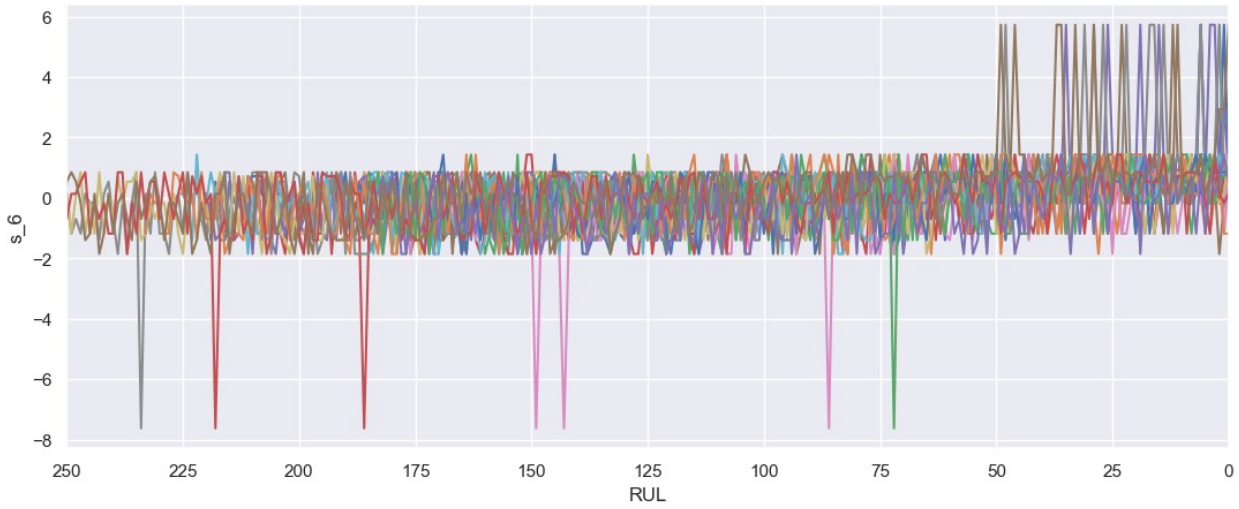


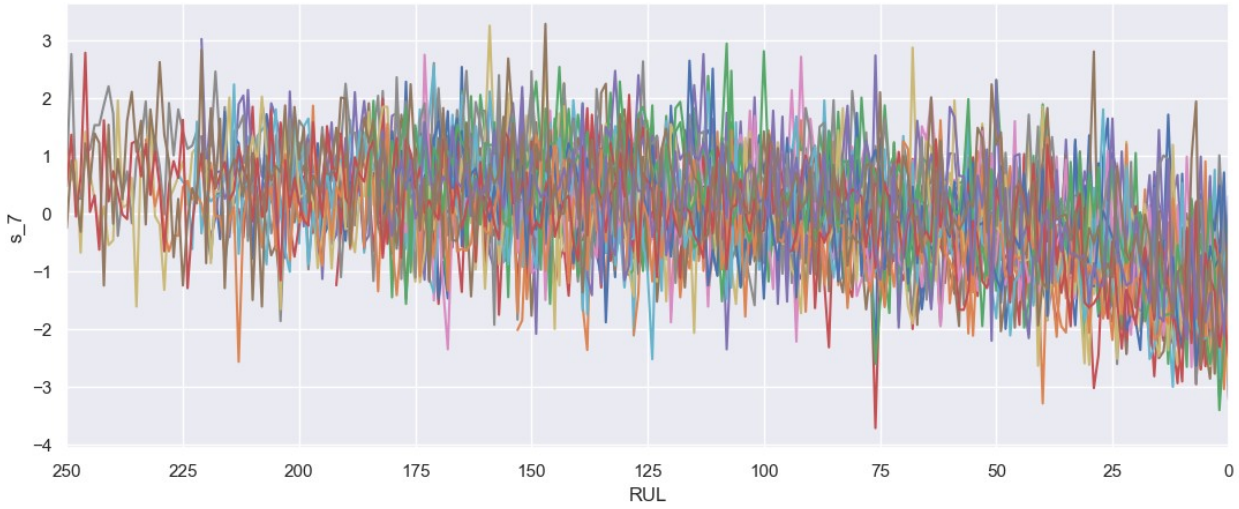

s_2
3
2
1
0
−1
−2
−3
250
225
200
175
150
125
100
75
50
25
0
RUL

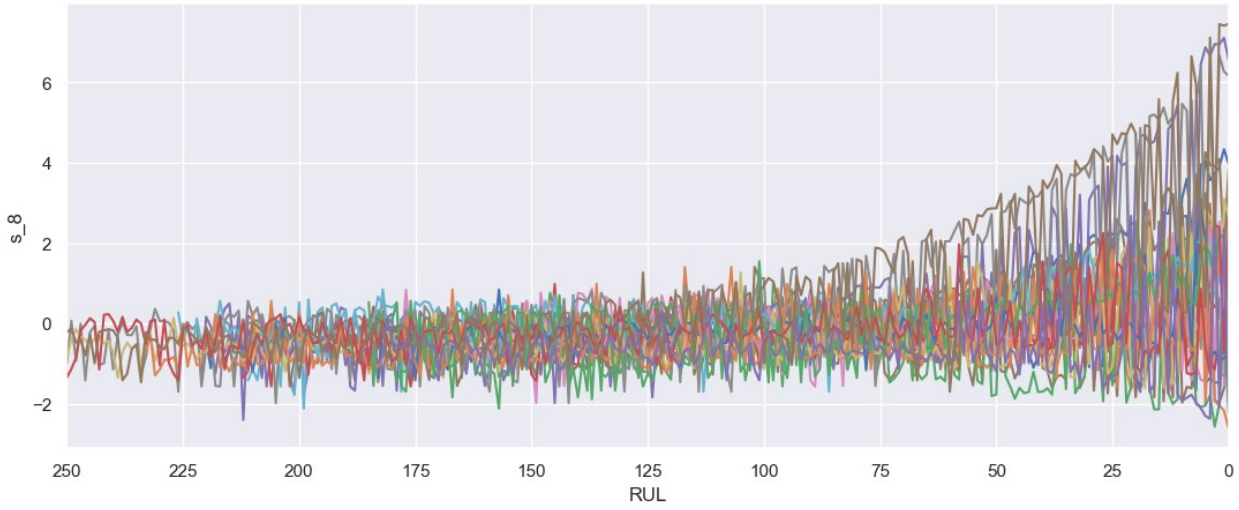

s_3
4
3
2
1
0
−1
−2
−3
250
225
200
175
150
125
100
75
50
25
0
RUL

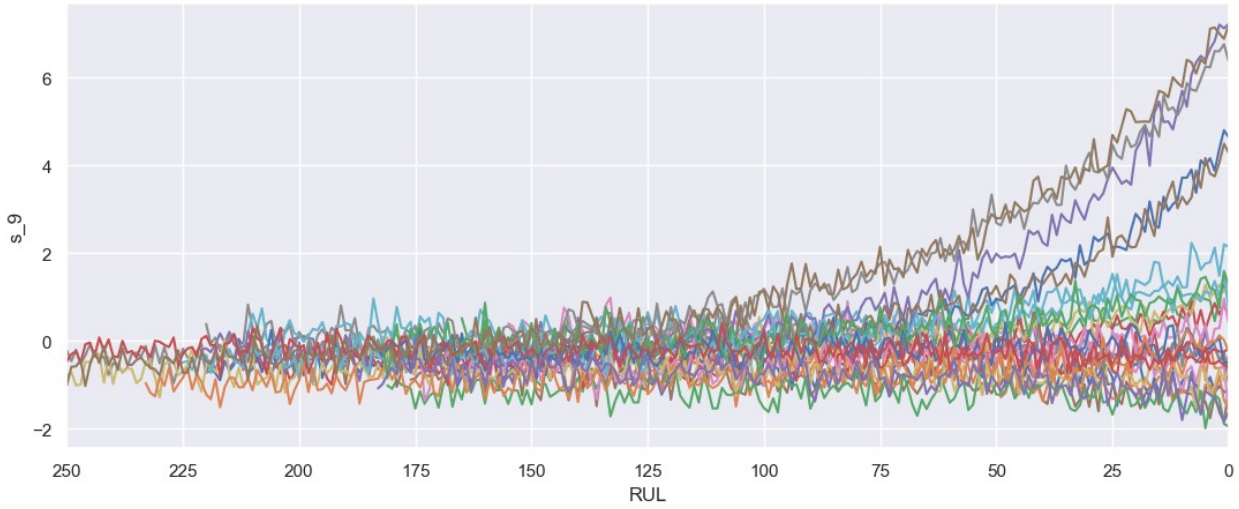

s_4
3
2
1
0
−1
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

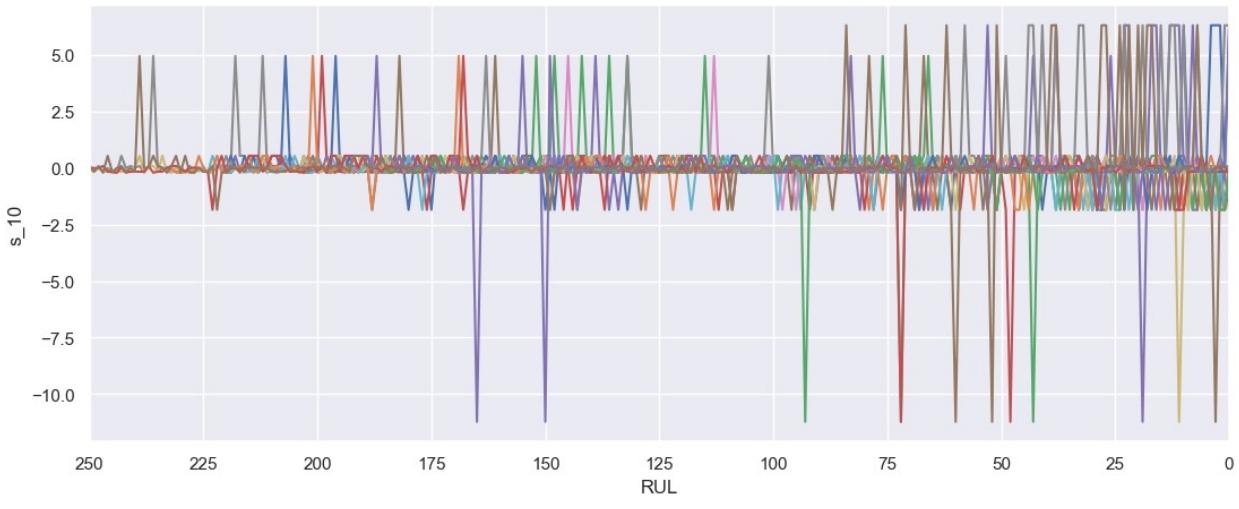

1e−15
2
1
0
−1
−2
−3
s_5
250
225
200
175
150
125
100
75
50
25
0
RUL

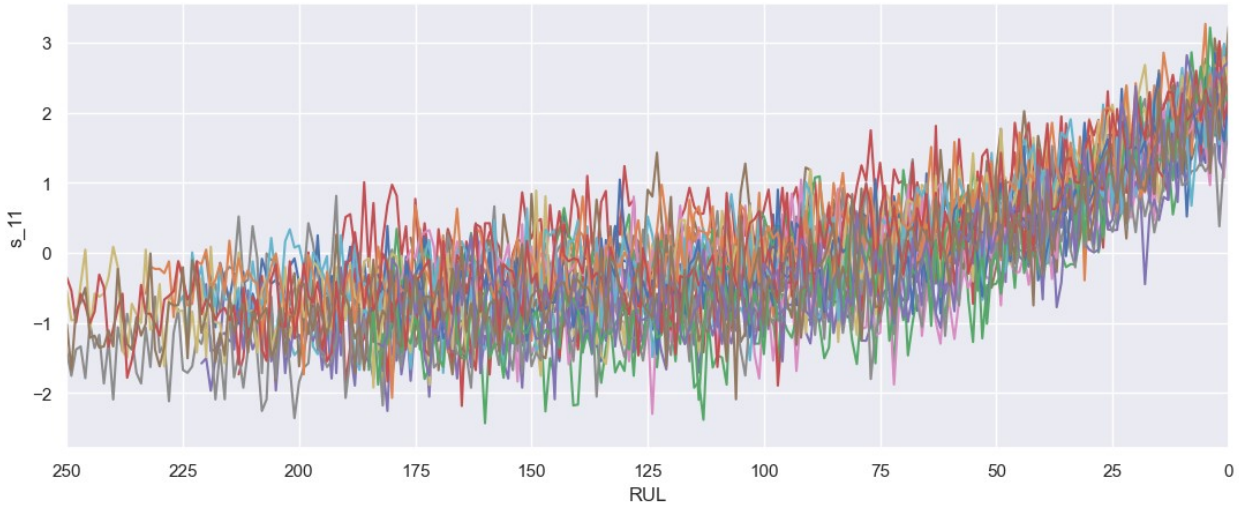

6
4
2
0
−2
−4
−6
−8
s_6
250
225
200
175
150
125
100
75
50
25
0
RUL

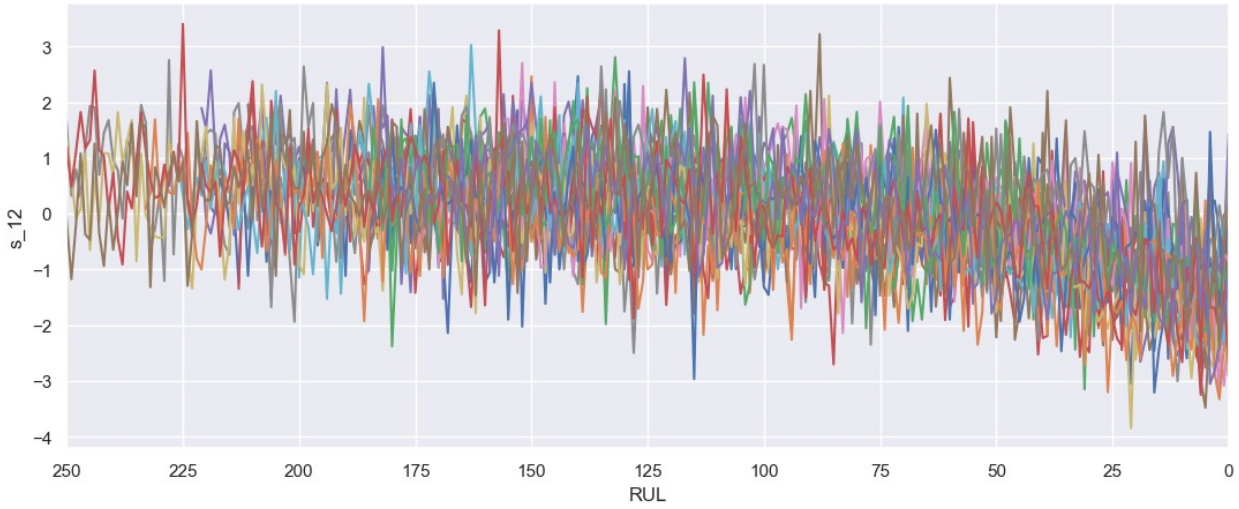

3
2
1
0
−1
−2
−3
−4
s_7
250
225
200
175
150
125
100
75
50
25
0
RUL

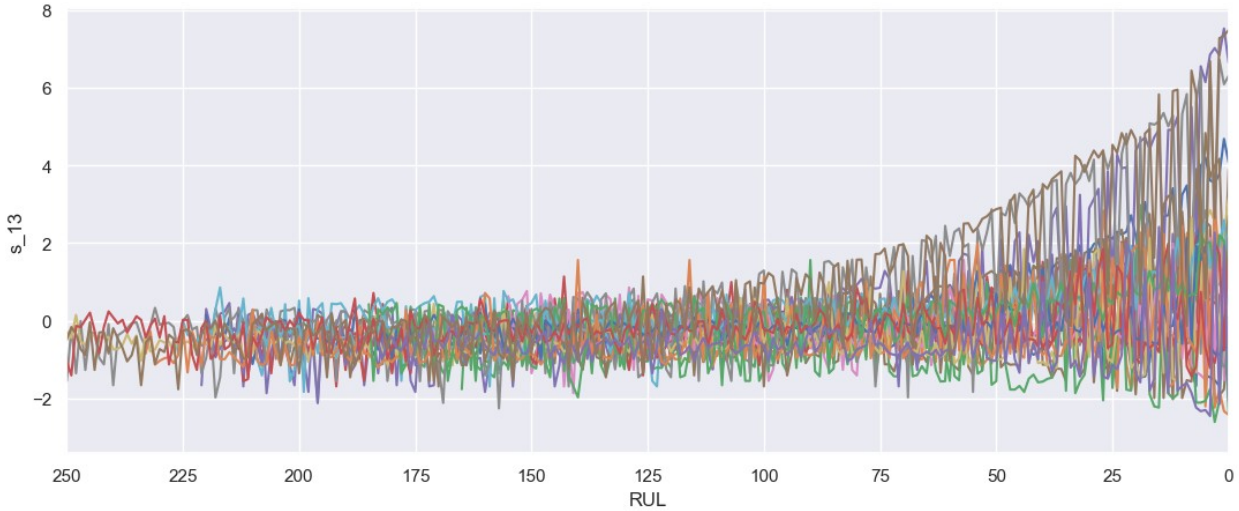

s_8
6
4
2
0
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

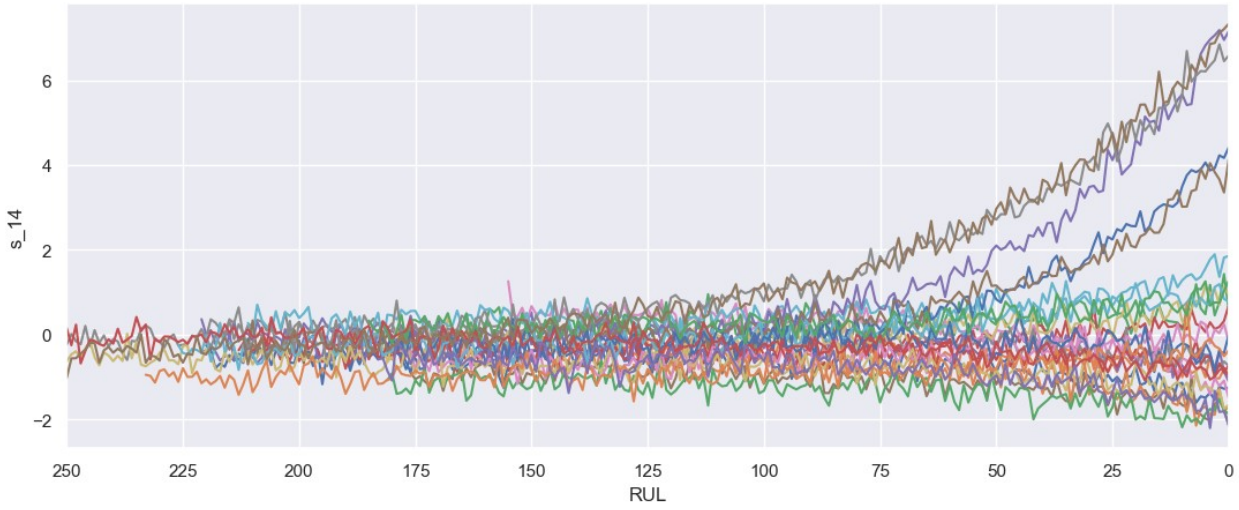

s_9
6
4
2
0
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

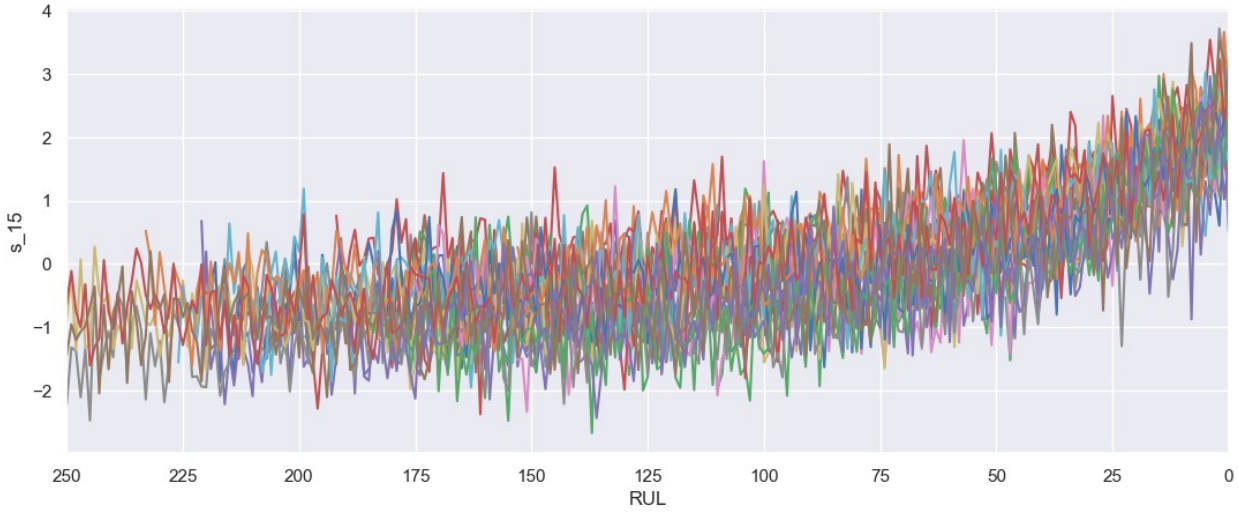

s_10
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
250
225
200
175
150
125
100
75
50
25
0
RUL

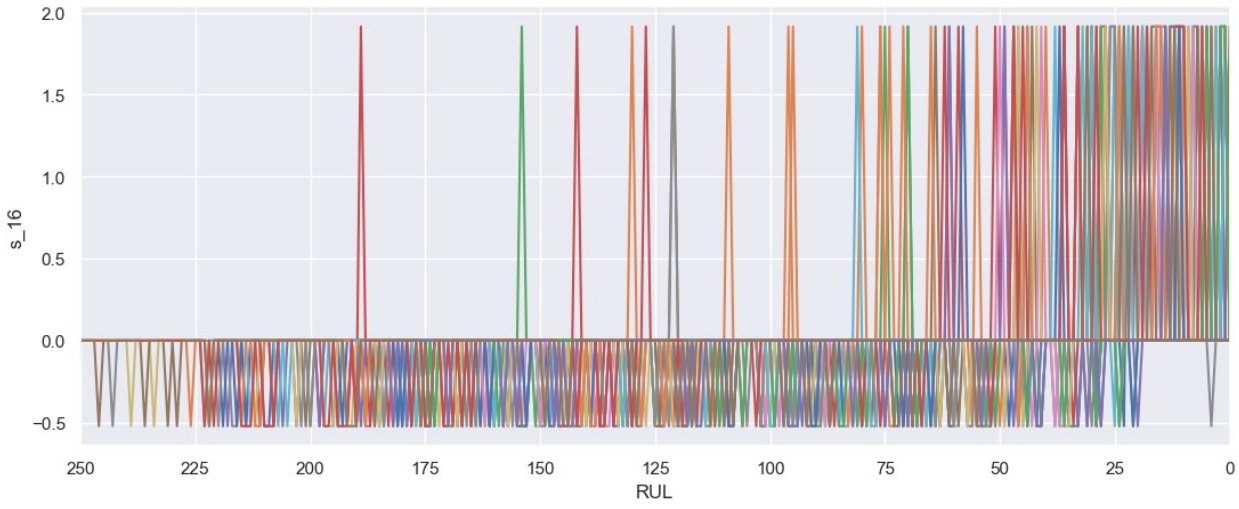

s_11
3
2
1
0
−1
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

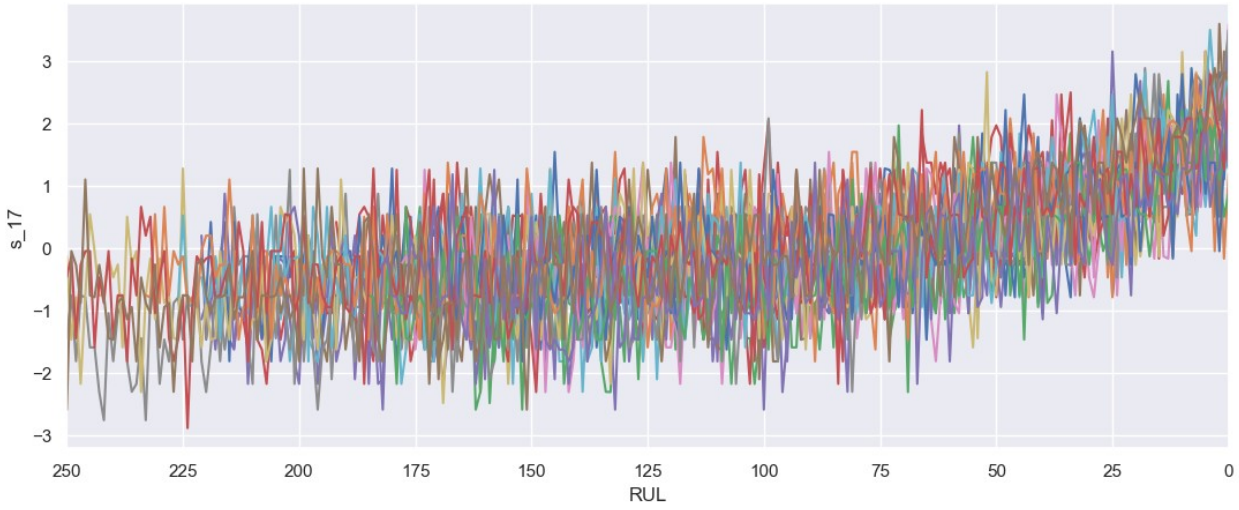

s_12
3
2
1
0
−1
−2
−3
−4
250
225
200
175
150
125
100
75
50
25
0
RUL

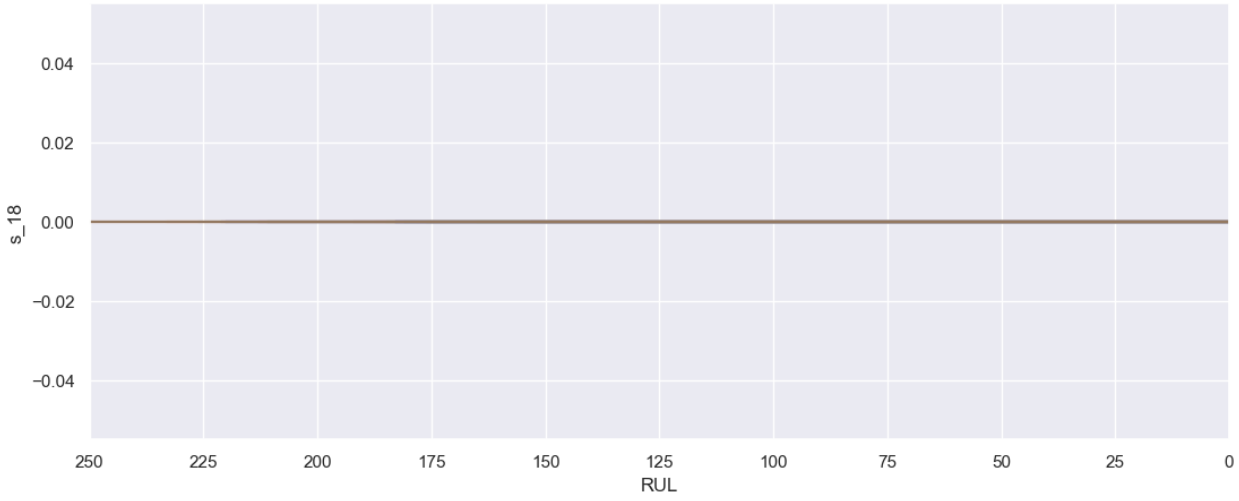

s_13
8
6
4
2
0
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

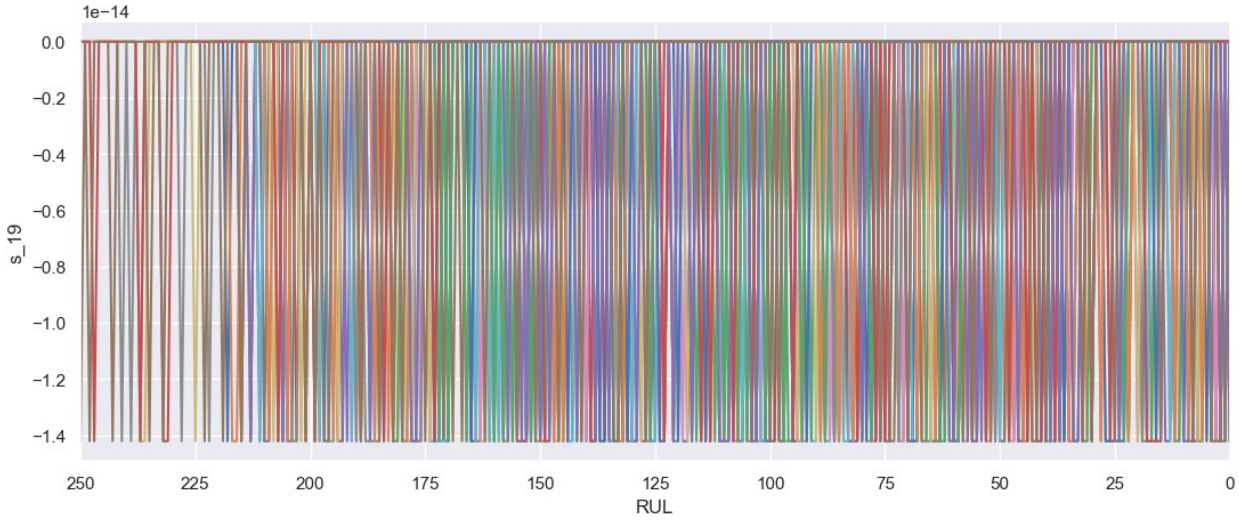

s_14
6
4
2
0
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

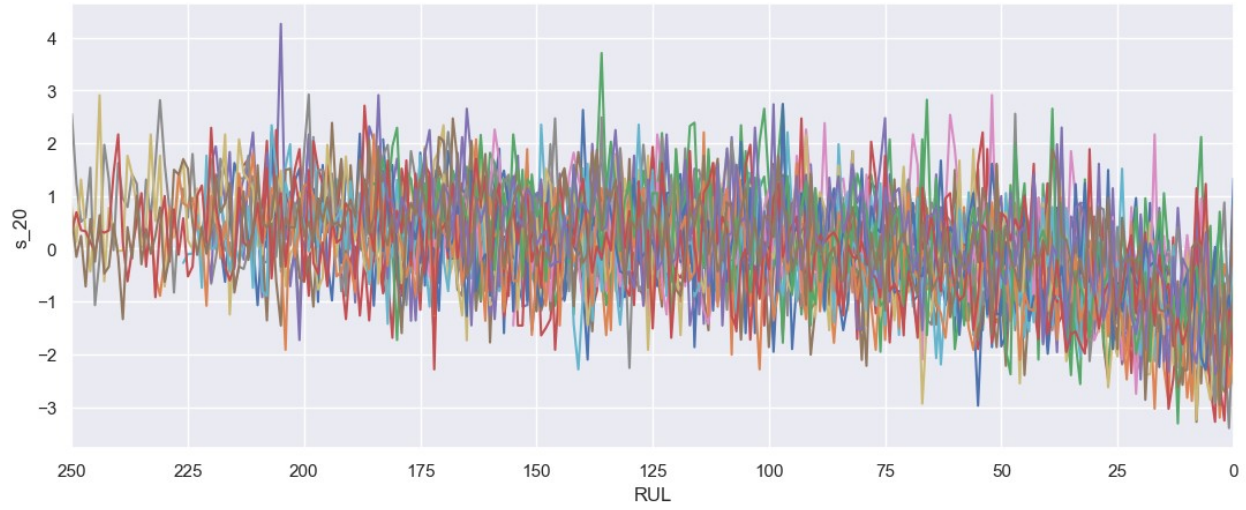

s_15
4
3
2
1
0
−1
−2
250
225
200
175
150
125
100
75
50
25
0
RUL

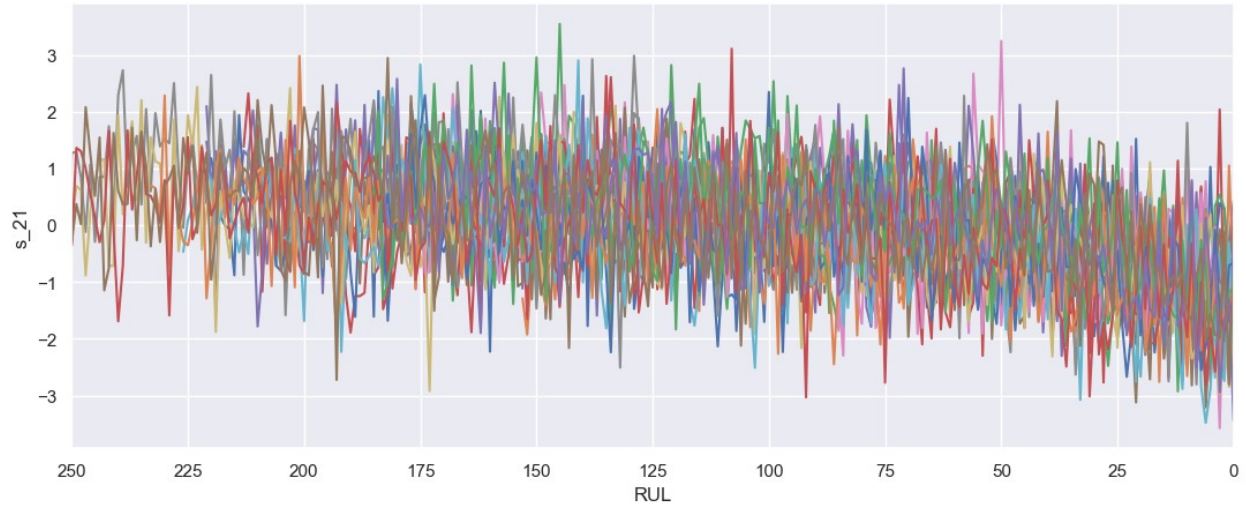

s_16
2.0
1.5
1.0
0.5
0.0
−0.5
250
225
200
175
150
125
100
75
50
25
0
RUL


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
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
250
225
200
175
150
125
100
75
50
25
0
RUL





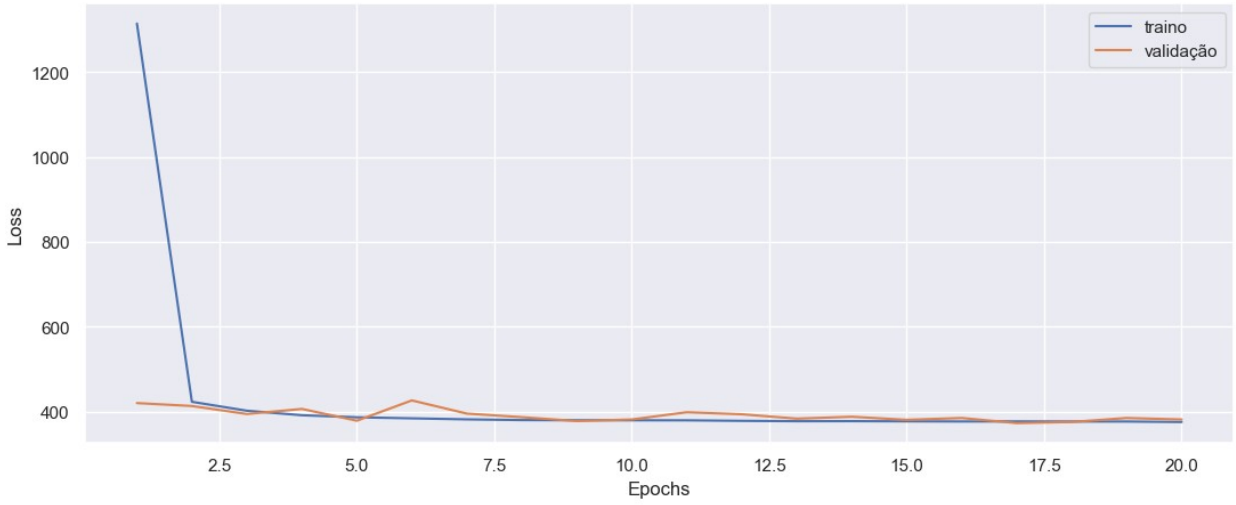

```
# Os sensores uteis sao
remaining_sensors = ['s_2', 's_3', 's_4', 's_7', 's_8', 's_9',
       's_11', 's_12', 's_13', 's_14', 's_15', 's_17', 's_20', 's_21']

# 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']
```

```
# Ajustando um modelo com as altercoes feitas

split_result = train_val_group_split(X_train_condition_scaled,
y_train_clipped, gss, train['unidade'], print_groups=False)
X_train_split_condition_scaled, y_train_clipped_split_condition_scaled
= split_result[:2]
X_val_split_condition_scaled, y_val_clipped_split_condition_scaled =
```

```
split_result[2:]

input_dim =
len(X_train_split_condition_scaled[remaining_sensors].columns)

model = Sequential()
model.add(Dense(16, input_dim=input_dim, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit(X_train_split_condition_scaled[remaining_sensors],
y_train_clipped_split_condition_scaled,

validation_data=(X_val_split_condition_scaled[remaining_sensors],
y_val_clipped_split_condition_scaled),
                    epochs=epochs)
plot_loss(history)

Epoch 1/20

1359/1359 [==============================] - 2s 1ms/step - loss:
1314.5349 - val_loss: 419.9603
Epoch 2/20
1359/1359 [==============================] - 2s 1ms/step - loss:
422.9648 - val_loss: 413.0465
Epoch 3/20
1359/1359 [==============================] - 2s 1ms/step - loss:
401.9226 - val_loss: 394.2321
Epoch 4/20
1359/1359 [==============================] - 2s 1ms/step - loss:
391.3197 - val_loss: 406.1784
Epoch 5/20
1359/1359 [==============================] - 2s 1ms/step - loss:
386.4547 - val_loss: 378.3869
Epoch 6/20
1359/1359 [==============================] - 2s 1ms/step - loss:
384.0154 - val_loss: 426.2339
Epoch 7/20
1359/1359 [==============================] - 2s 1ms/step - loss:
381.5667 - val_loss: 395.1648
Epoch 8/20
1359/1359 [==============================] - 2s 1ms/step - loss:
379.9193 - val_loss: 386.7646
Epoch 9/20
1359/1359 [==============================] - 2s 1ms/step - loss:
379.5007 - val_loss: 377.6779
Epoch 10/20
1359/1359 [==============================] - 2s 1ms/step - loss:
```

```
379.4348 - val_loss: 381.3127
Epoch 11/20
1359/1359 [==============================] - 2s 1ms/step - loss:
379.3094 - val_loss: 398.3730
Epoch 12/20
1359/1359 [==============================] - 2s 1ms/step - loss:
378.0742 - val_loss: 393.5484
Epoch 13/20
1359/1359 [==============================] - 2s 1ms/step - loss:
377.2549 - val_loss: 383.2940
Epoch 14/20
1359/1359 [==============================] - 2s 1ms/step - loss:
377.3785 - val_loss: 387.7382
Epoch 15/20
1359/1359 [==============================] - 2s 1ms/step - loss:
376.9606 - val_loss: 380.5042
Epoch 16/20
1359/1359 [==============================] - 2s 1ms/step - loss:
376.5861 - val_loss: 384.8141
Epoch 17/20
1359/1359 [==============================] - 2s 1ms/step - loss:
376.7711 - val_loss: 372.7856
Epoch 18/20
1359/1359 [==============================] - 2s 1ms/step - loss:
376.4984 - val_loss: 375.4015
Epoch 19/20
1359/1359 [==============================] - 2s 1ms/step - loss:
376.4906 - val_loss: 384.7971
Epoch 20/20
1359/1359 [==============================] - 2s 1ms/step - loss:
375.3661 - val_loss: 381.2710
```



```
# Testando o modelo
y_hat_train =
model.predict(X_train_condition_scaled[remaining_sensors])
avaliar(y_train_clipped, y_hat_train, 'treino')

y_hat_test = model.predict(X_test_condition_scaled[remaining_sensors])
avaliar(y_test, y_hat_test)
```

```
1680/1680 [==============================] - 1s 829us/step
conjunto de treino -> RMSE:19.324283600118097, R2:0.7848905162604213
9/9 [==============================] - 0s 876us/step
conjunto de teste -> RMSE:29.417860818760126, R2:0.7007734969530757
```

9. Variáveis defasadas

```
# De forma semelhante ao que foi feito anteriormente, pode-se
adicionar variaveis defasadas para tentar melhorar os resultados do
modelo

# Exemplo
np.random.seed(42)
toy_df = pd.DataFrame({'value': np.random.rand(5)})
toy_df['value_lag_1'] = toy_df['value'].shift(1)
toy_df['value_lag_2'] = toy_df['value'].shift(2)
display(toy_df)
```

```
      value  value_lag_1  value_lag_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
```

```
# Funcao para adicionar as variaveis defasadas de forma especifica
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

# Variaveis defasadas que serao inseridas
specific_lags = [1,2,3,4,5,10,20]  # Mesmo conjunto que funcionou bem
com o FD001

# Preparando os dados
X_train_condition = add_op_cond(train.drop(drop_sensors, axis=1))
```

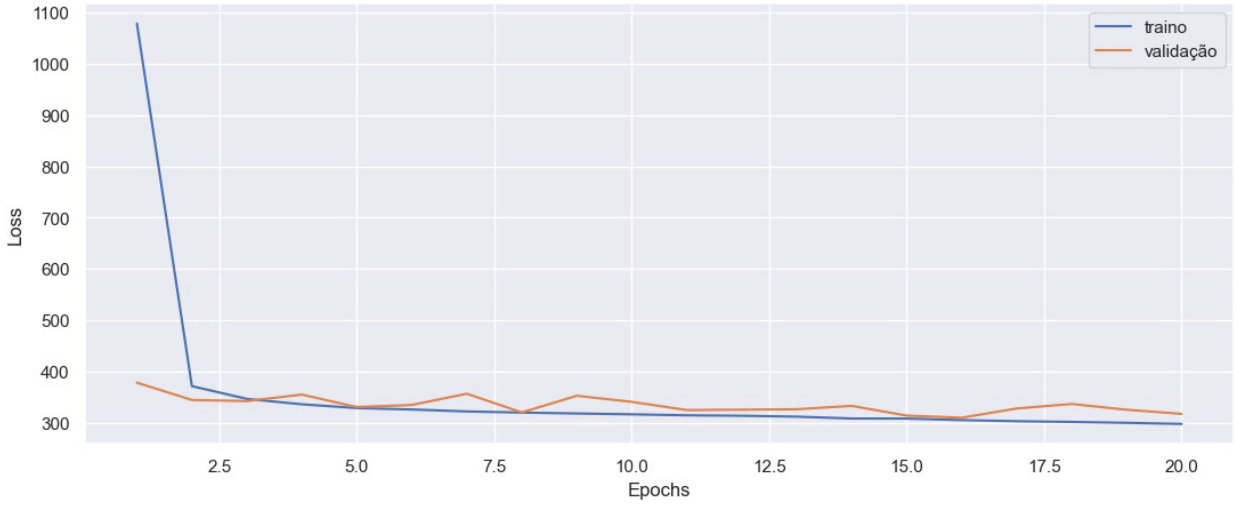
```
X_test_condition = add_op_cond(test.drop(drop_sensors, axis=1))

X_train_condition_scaled, X_test_condition_scaled =
condition_scaler(X_train_condition, X_test_condition,
remaining_sensors)

X_train_condition_scaled_lagged =
add_specific_lags(X_train_condition_scaled, specific_lags,
remaining_sensors)
X_test_condition_scaled_lagged =
add_specific_lags(X_test_condition_scaled, specific_lags,
remaining_sensors)

X_train_condition_scaled_lagged.drop(index_names+setting_names+
['op_cond', 'RUL'], axis=1, inplace=True)
X_test_condition_scaled_lagged =
X_test_condition_scaled_lagged.drop(['ciclo_tempo', 'op_cond']
+setting_names, axis=1).groupby('unidade').last().copy()

idx = X_train_condition_scaled_lagged.index
y_train_clipped_lagged = y_train_clipped.iloc[idx]

split_result = train_val_group_split(X_train_condition_scaled_lagged,
y_train_clipped_lagged, gss, train.iloc[idx]['unidade'],
print_groups=False)
X_train_split_condition_scaled_lagged, y_train_clipped_split_lagged =
split_result[:2]
X_val_split_condition_scaled_lagged, y_val_clipped_split_lagged =
split_result[2:]

C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\1051512065.py:7:
FutureWarning: Setting an item of incompatible dtype is deprecated and
will raise in a future error of pandas. Value '[-0.1674041  -
1.59188977 -0.1674041  ...  0.54483874  1.96932442
  2.68156725]' has dtype incompatible with int64, please explicitly
cast to a compatible dtype first.
  df_train.loc[df_train['op_cond']==condition, sensor_names] =
scaler.transform(df_train.loc[df_train['op_cond']==condition,
sensor_names])
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\1051512065.py:8:
FutureWarning: Setting an item of incompatible dtype is deprecated and
will raise in a future error of pandas. Value '[-0.87964693
1.25708158 -0.87964693 ... -0.1674041  -1.59188977
 -0.1674041 ]' has dtype incompatible with int64, please explicitly
cast to a compatible dtype first.
  df_test.loc[df_test['op_cond']==condition, sensor_names] =
scaler.transform(df_test.loc[df_test['op_cond']==condition,
sensor_names])
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\3611489979.py:6:
PerformanceWarning: DataFrame is highly fragmented.  This is usually
```

```
the result of calling `frame.insert` many times, which has poor
performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe
= frame.copy()`
  df[lagged_columns] = df.groupby('unidade')[columns].shift(i)
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\3611489979.py:6:
PerformanceWarning: DataFrame is highly fragmented.  This is usually
the result of calling `frame.insert` many times, which has poor
performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe
= frame.copy()`
  df[lagged_columns] = df.groupby('unidade')[columns].shift(i)
```

```
# Criando novamente o modelo com as alteracoes feitas

input_dim = len(X_train_split_condition_scaled_lagged.columns)

model = Sequential()
model.add(Dense(16, input_dim=input_dim, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.save_weights('specific_lagged_weights.h5')

model.compile(loss='mean_squared_error', optimizer='adam')
model.load_weights('specific_lagged_weights.h5')
epochs=20

history = model.fit(X_train_split_condition_scaled_lagged,
y_train_clipped_split_lagged,

validation_data=(X_val_split_condition_scaled_lagged,
y_val_clipped_split_lagged),
                    epochs=epochs)

plot_loss(history)
```

```
Epoch 1/20
1229/1229 [==============================] - 2s 2ms/step - loss:
1078.7202 - val_loss: 377.8046
Epoch 2/20
1229/1229 [==============================] - 2s 1ms/step - loss:
370.9806 - val_loss: 344.0408
Epoch 3/20
1229/1229 [==============================] - 2s 1ms/step - loss:
346.0805 - val_loss: 341.8675
Epoch 4/20
1229/1229 [==============================] - 2s 1ms/step - loss:
```

```
335.5905 - val_loss: 354.4585
Epoch 5/20
1229/1229 [==============================] - 2s 1ms/step - loss:
328.2046 - val_loss: 330.1830
Epoch 6/20
1229/1229 [==============================] - 2s 1ms/step - loss:
325.4110 - val_loss: 334.2445
Epoch 7/20
1229/1229 [==============================] - 2s 1ms/step - loss:
321.6201 - val_loss: 356.1645
Epoch 8/20
1229/1229 [==============================] - 2s 1ms/step - loss:
319.3730 - val_loss: 319.7221
Epoch 9/20
1229/1229 [==============================] - 2s 1ms/step - loss:
317.6744 - val_loss: 352.1667
Epoch 10/20
1229/1229 [==============================] - 2s 1ms/step - loss:
315.9741 - val_loss: 340.3294
Epoch 11/20
1229/1229 [==============================] - 2s 1ms/step - loss:
314.0656 - val_loss: 324.3356
Epoch 12/20
1229/1229 [==============================] - 2s 1ms/step - loss:
313.1969 - val_loss: 325.0709
Epoch 13/20
1229/1229 [==============================] - 2s 1ms/step - loss:
311.4585 - val_loss: 325.9850
Epoch 14/20
1229/1229 [==============================] - 2s 1ms/step - loss:
307.8198 - val_loss: 332.4072
Epoch 15/20
1229/1229 [==============================] - 2s 1ms/step - loss:
307.7088 - val_loss: 313.4813
Epoch 16/20
1229/1229 [==============================] - 2s 1ms/step - loss:
304.7116 - val_loss: 309.3174
Epoch 17/20
1229/1229 [==============================] - 2s 1ms/step - loss:
302.5725 - val_loss: 327.3765
Epoch 18/20
1229/1229 [==============================] - 2s 1ms/step - loss:
301.1960 - val_loss: 336.1879
Epoch 19/20
1229/1229 [==============================] - 2s 1ms/step - loss:
299.3936 - val_loss: 324.8922
Epoch 20/20
1229/1229 [==============================] - 2s 1ms/step - loss:
297.2462 - val_loss: 316.8986
```




```
# Avaliando o novo modelo
y_hat_train = model.predict(X_train_condition_scaled_lagged)
avaliar(y_train_clipped_lagged, y_hat_train, 'treino')

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

```
1518/1518 [==============================] - 1s 868us/step
conjunto de treino -> RMSE:17.142227063381085, R2:0.8321461522998501
9/9 [==============================] - 0s 1ms/step
conjunto de teste -> RMSE:28.56709980915771, R2:0.7178304252548284
```

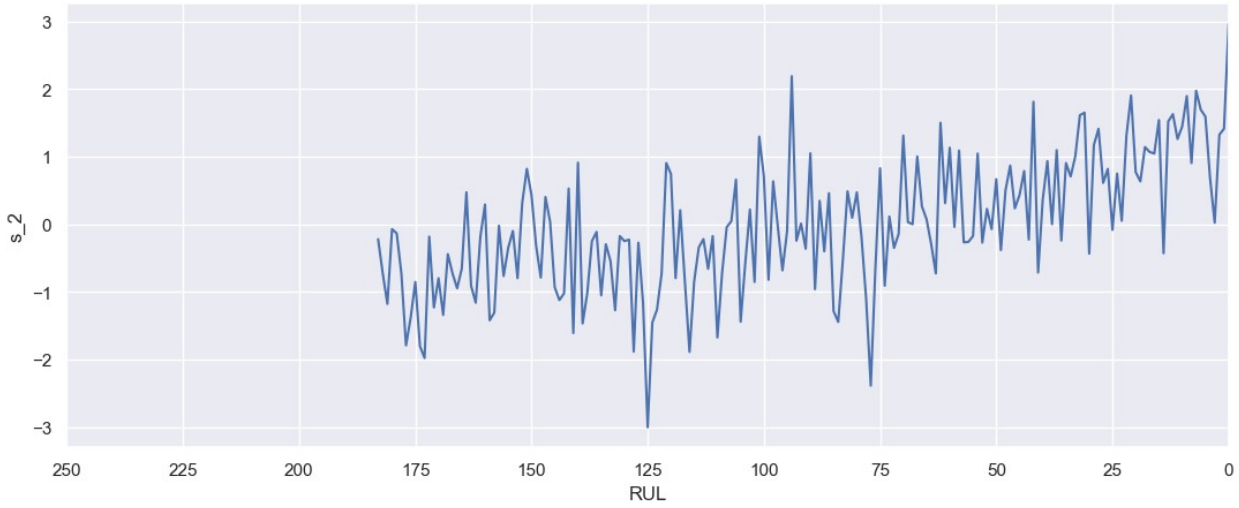
10. Estacionaridade e Suavização dos dados

```
# De forma semelhante ao feito com o conjunto FD001, será realizados
testes para avaliar a estacionaridade dos dados
# e aplicar processos que garantem a estacionaridade e suavizacao dos
dados

from statsmodels.tsa.stattools import adfuller

# 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

X_train_condition = add_op_cond(train.drop(drop_sensors, axis=1))
X_test_condition = add_op_cond(test.drop(drop_sensors, axis=1))

X_train_condition_scaled, _ = condition_scaler(X_train_condition,
X_test_condition, remaining_sensors)


adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(

X_train_condition_scaled.loc[X_train_condition_scaled['unidade'] ==
10,'s_2'],
                                                maxlag=1)
print('Sinal original é considerado estacionário? ', pvalue < 0.05)
plot_sinal(X_train_condition_scaled, 's_2', 10) # s_2 vs RUL apos
normalizacao baseada na op_cond
```

```
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\1051512065.py:7:
FutureWarning: Setting an item of incompatible dtype is deprecated and
will raise in a future error of pandas. Value '[-0.1674041  -
1.59188977 -0.1674041  ...  0.54483874  1.96932442
  2.68156725]' has dtype incompatible with int64, please explicitly
cast to a compatible dtype first.
  df_train.loc[df_train['op_cond']==condition, sensor_names] =
scaler.transform(df_train.loc[df_train['op_cond']==condition,
sensor_names])
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\1051512065.py:8:
FutureWarning: Setting an item of incompatible dtype is deprecated and
will raise in a future error of pandas. Value '[-0.87964693
1.25708158 -0.87964693 ... -0.1674041  -1.59188977
```

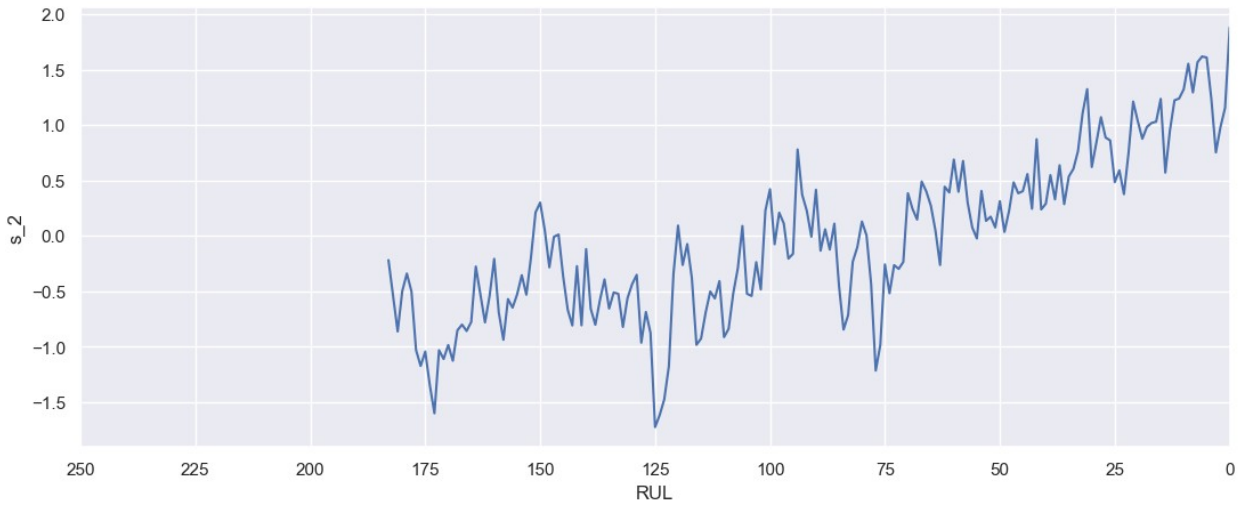
```
 -0.1674041 ]' has dtype incompatible with int64, please explicitly
cast to a compatible dtype first.
  df_test.loc[df_test['op_cond']==condition, sensor_names] =
scaler.transform(df_test.loc[df_test['op_cond']==condition,
sensor_names])

Sinal original é considerado estacionário?  True
```



11. Introduzindo suavização aos dados

```
# Valores menores para alpha geram um efeito de filtro maior
X_train_condition_scaled_smoothed = X_train_condition_scaled.copy()

# Agrupa por unidade e separa apenas o sinal dos sensores desejados
para aplicar a suavização com a função do Pandas.
# Depois desfaz o agrupamento e seleciona apenas os valores dos
sensores restantes para substituir o valor original.
X_train_condition_scaled_smoothed[remaining_sensors] =
X_train_condition_scaled_smoothed.groupby('unidade')
[remaining_sensors].ewm(alpha=0.4).mean().reset_index()
[remaining_sensors]

# Testando a estacionaridade apos normalizacao baseada na op_cond e
suavizacao
adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(

X_train_condition_scaled_smoothed.loc[X_train_condition_scaled_smoothe
d['unidade'] == 10,'s_2'],
                                maxlag=1)
print('Sinal suavizado é considerado estacionário? ', pvalue < 0.05)
plot_sinal(X_train_condition_scaled_smoothed, 's_2', 10)
```

```
Sinal suavizado é considerado estacionário?  False
```



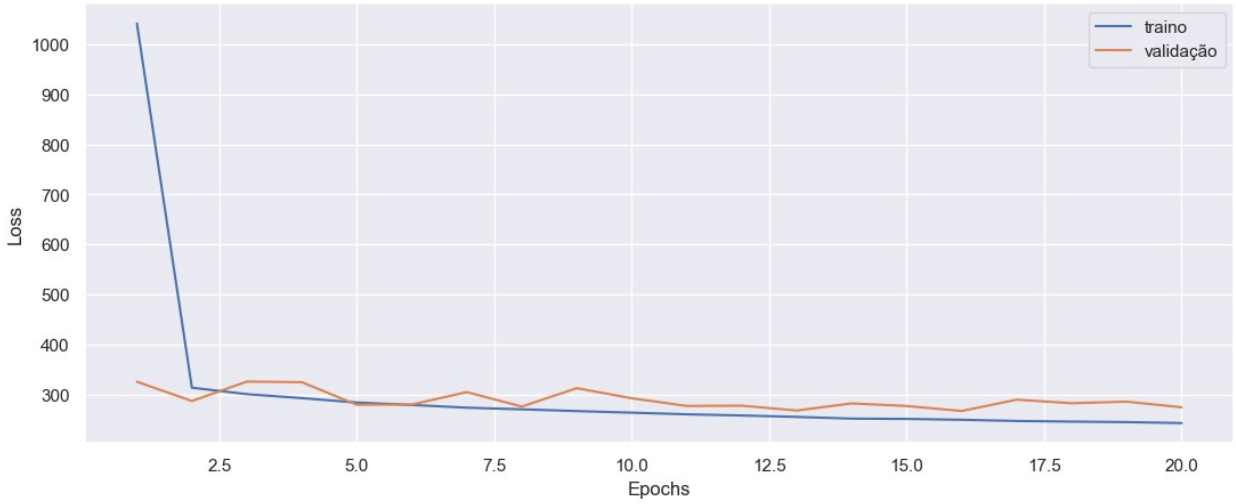

```
# Adicionando a suavizacao ao tratamento dos dados
def exponential_smoothing(df, sensors, n_samples, alpha=0.4):
    df = df.copy()
    df[sensors] = df.groupby('unidade')
[sensors].ewm(alpha=0.4).mean().reset_index()[sensors]

    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

# Preparando os dados
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)

X_train_interim = add_specific_lags(X_train_interim, specific_lags,
remaining_sensors)
```

```
X_test_interim = add_specific_lags(X_test_interim, specific_lags,
remaining_sensors)

X_train_smooth_lagged =
X_train_interim.drop(index_names+setting_names+['op_cond', 'RUL'],
axis=1)
X_test_smooth_lagged = X_test_interim.drop(['ciclo_tempo', 'op_cond']
+setting_names,

axis=1).groupby('unidade').last().copy()

idx = X_train_smooth_lagged.index
y_train_clipped_lagged = y_train_clipped.iloc[idx]

# Separando em dados de treino e validacao
split_result = train_val_group_split(X_train_smooth_lagged,
y_train_clipped_lagged, gss, train.iloc[idx]['unidade'],
print_groups=False)
X_train_split_smooth_lagged, y_train_clipped_split_lagged =
split_result[:2]
X_val_split_smooth_lagged, y_val_clipped_split_lagged =
split_result[2:]
```

```
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\1051512065.py:7:
FutureWarning: Setting an item of incompatible dtype is deprecated and
will raise in a future error of pandas. Value '[-0.1674041  -
1.59188977 -0.1674041  ...  0.54483874  1.96932442
  2.68156725]' has dtype incompatible with int64, please explicitly
cast to a compatible dtype first.
  df_train.loc[df_train['op_cond']==condition, sensor_names] =
scaler.transform(df_train.loc[df_train['op_cond']==condition,
sensor_names])
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\1051512065.py:8:
FutureWarning: Setting an item of incompatible dtype is deprecated and
will raise in a future error of pandas. Value '[-0.87964693
1.25708158 -0.87964693 ... -0.1674041  -1.59188977
 -0.1674041 ]' has dtype incompatible with int64, please explicitly
cast to a compatible dtype first.
  df_test.loc[df_test['op_cond']==condition, sensor_names] =
scaler.transform(df_test.loc[df_test['op_cond']==condition,
sensor_names])
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\3611489979.py:6:
PerformanceWarning: DataFrame is highly fragmented.  This is usually
the result of calling `frame.insert` many times, which has poor
performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe
= frame.copy()`
  df[lagged_columns] = df.groupby('unidade')[columns].shift(i)
C:\Users\lucas\AppData\Local\Temp\ipykernel_53108\3611489979.py:6:
PerformanceWarning: DataFrame is highly fragmented.  This is usually
```

```
the result of calling `frame.insert` many times, which has poor
performance.  Consider joining all columns at once using
pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe
= frame.copy()`
  df[lagged_columns] = df.groupby('unidade')[columns].shift(i)

# Criando um novo modelo com as alteracoes feitas
model.compile(loss='mean_squared_error', optimizer='adam')
model.load_weights('specific_lagged_weights.h5')
epochs=20

history = model.fit(X_train_split_smooth_lagged,
y_train_clipped_split_lagged,
                    validation_data=(X_val_split_smooth_lagged,
y_val_clipped_split_lagged),
                    epochs=epochs)

plot_loss(history)

Epoch 1/20
1229/1229 [==============================] - 2s 2ms/step - loss:
1041.1525 - val_loss: 325.7604
Epoch 2/20
1229/1229 [==============================] - 2s 1ms/step - loss:
313.8558 - val_loss: 287.2442
Epoch 3/20
1229/1229 [==============================] - 2s 1ms/step - loss:
300.9580 - val_loss: 326.0548
Epoch 4/20
1229/1229 [==============================] - 2s 1ms/step - loss:
292.9764 - val_loss: 324.8587
Epoch 5/20
1229/1229 [==============================] - 2s 1ms/step - loss:
284.2288 - val_loss: 279.7070
Epoch 6/20
1229/1229 [==============================] - 2s 1ms/step - loss:
279.4026 - val_loss: 280.2363
Epoch 7/20
1229/1229 [==============================] - 2s 1ms/step - loss:
273.8846 - val_loss: 305.0175
Epoch 8/20
1229/1229 [==============================] - 2s 1ms/step - loss:
270.6717 - val_loss: 276.1407
Epoch 9/20
1229/1229 [==============================] - 2s 1ms/step - loss:
267.1513 - val_loss: 312.9281
Epoch 10/20
1229/1229 [==============================] - 2s 1ms/step - loss:
263.9443 - val_loss: 292.7228
Epoch 11/20
```

```
1229/1229 [==============================] - 2s 1ms/step - loss:
260.6991 - val_loss: 277.4293
Epoch 12/20
1229/1229 [==============================] - 2s 1ms/step - loss:
258.3707 - val_loss: 277.8871
Epoch 13/20
1229/1229 [==============================] - 2s 1ms/step - loss:
255.5267 - val_loss: 268.2882
Epoch 14/20
1229/1229 [==============================] - 2s 1ms/step - loss:
252.0996 - val_loss: 282.4602
Epoch 15/20
1229/1229 [==============================] - 2s 1ms/step - loss:
251.4923 - val_loss: 277.2478
Epoch 16/20
1229/1229 [==============================] - 2s 1ms/step - loss:
249.7743 - val_loss: 267.2833
Epoch 17/20
1229/1229 [==============================] - 2s 1ms/step - loss:
247.3762 - val_loss: 289.9220
Epoch 18/20
1229/1229 [==============================] - 2s 1ms/step - loss:
246.1172 - val_loss: 282.8774
Epoch 19/20
1229/1229 [==============================] - 2s 1ms/step - loss:
245.1279 - val_loss: 285.9709
Epoch 20/20
1229/1229 [==============================] - 2s 1ms/step - loss:
243.1477 - val_loss: 274.7104
```




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

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

```
1518/1518 [==============================] - 1s 851us/step
conjunto de treino -> RMSE:15.668579517421401, R2:0.8597651138908445
9/9 [==============================] - 0s 1ms/step
conjunto de teste -> RMSE:27.205850900454198, R2:0.7440810138524186
```

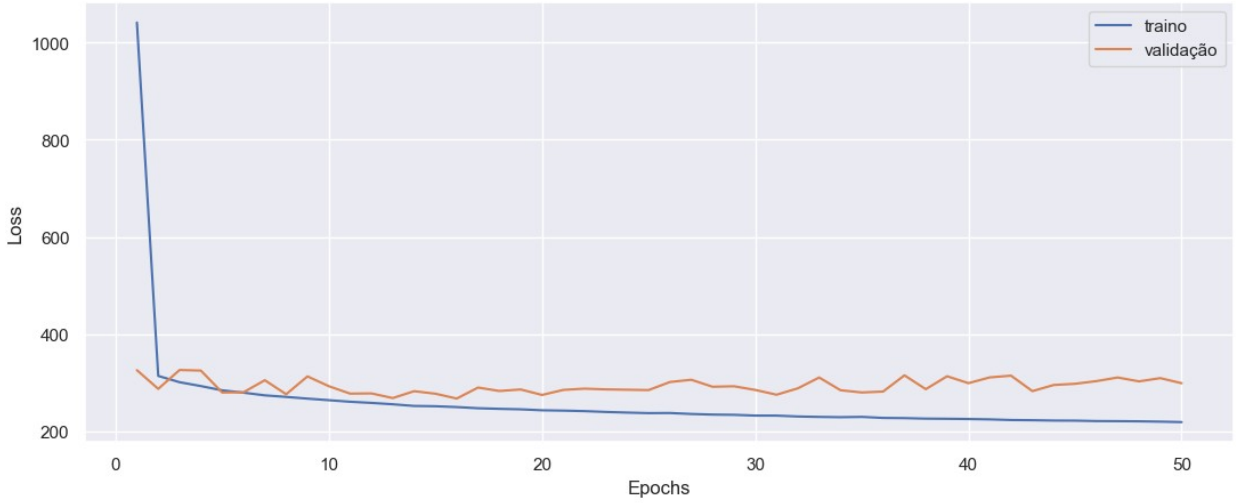
```
# Usando o mesmo modelo mas testando o efeito de aumentar o numero de
epochs
model.compile(loss='mean_squared_error', optimizer='adam')
model.load_weights('specific_lagged_weights.h5')
epochs=50

history = model.fit(X_train_split_smooth_lagged,
y_train_clipped_split_lagged,
                    validation_data=(X_val_split_smooth_lagged,
y_val_clipped_split_lagged),
                    epochs=epochs)
```

```
Epoch 1/50
1229/1229 [==============================] - 2s 2ms/step - loss:
1041.1525 - val_loss: 325.7604
Epoch 2/50
1229/1229 [==============================] - 2s 1ms/step - loss:
313.8558 - val_loss: 287.2442
Epoch 3/50
1229/1229 [==============================] - 2s 1ms/step - loss:
300.9580 - val_loss: 326.0548
Epoch 4/50
1229/1229 [==============================] - 2s 1ms/step - loss:
292.9764 - val_loss: 324.8587
Epoch 5/50
1229/1229 [==============================] - 2s 1ms/step - loss:
284.2288 - val_loss: 279.7070
Epoch 6/50
1229/1229 [==============================] - 2s 1ms/step - loss:
279.4026 - val_loss: 280.2363
Epoch 7/50
1229/1229 [==============================] - 2s 1ms/step - loss:
273.8846 - val_loss: 305.0175
Epoch 8/50
1229/1229 [==============================] - 2s 1ms/step - loss:
270.6717 - val_loss: 276.1407
Epoch 9/50
1229/1229 [==============================] - 2s 1ms/step - loss:
267.1513 - val_loss: 312.9281
Epoch 10/50
1229/1229 [==============================] - 2s 1ms/step - loss:
```

```
263.9443 - val_loss: 292.7228
Epoch 11/50
1229/1229 [==============================] - 2s 1ms/step - loss:
260.6991 - val_loss: 277.4293
Epoch 12/50
1229/1229 [==============================] - 2s 1ms/step - loss:
258.3707 - val_loss: 277.8871
Epoch 13/50
1229/1229 [==============================] - 2s 1ms/step - loss:
255.5267 - val_loss: 268.2882
Epoch 14/50
1229/1229 [==============================] - 2s 1ms/step - loss:
252.0996 - val_loss: 282.4602
Epoch 15/50
1229/1229 [==============================] - 2s 1ms/step - loss:
251.4923 - val_loss: 277.2478
Epoch 16/50
1229/1229 [==============================] - 2s 1ms/step - loss:
249.7743 - val_loss: 267.2833
Epoch 17/50
1229/1229 [==============================] - 2s 1ms/step - loss:
247.3762 - val_loss: 289.9220
Epoch 18/50
1229/1229 [==============================] - 2s 2ms/step - loss:
246.1172 - val_loss: 282.8774
Epoch 19/50
1229/1229 [==============================] - 2s 1ms/step - loss:
245.1279 - val_loss: 285.9709
Epoch 20/50
1229/1229 [==============================] - 2s 1ms/step - loss:
243.1477 - val_loss: 274.7104
Epoch 21/50
1229/1229 [==============================] - 2s 1ms/step - loss:
242.4549 - val_loss: 285.1052
Epoch 22/50
1229/1229 [==============================] - 2s 1ms/step - loss:
241.4262 - val_loss: 287.6934
Epoch 23/50
1229/1229 [==============================] - 2s 1ms/step - loss:
239.7136 - val_loss: 286.1598
Epoch 24/50
1229/1229 [==============================] - 2s 1ms/step - loss:
238.4043 - val_loss: 285.4720
Epoch 25/50
1229/1229 [==============================] - 2s 1ms/step - loss:
237.3276 - val_loss: 284.7283
Epoch 26/50
1229/1229 [==============================] - 2s 1ms/step - loss:
237.5481 - val_loss: 301.3198
```

```
Epoch 27/50
1229/1229 [==============================] - 2s 1ms/step - loss:
235.5423 - val_loss: 305.9800
Epoch 28/50
1229/1229 [==============================] - 2s 1ms/step - loss:
234.2957 - val_loss: 291.5582
Epoch 29/50
1229/1229 [==============================] - 2s 1ms/step - loss:
233.8119 - val_loss: 292.6014
Epoch 30/50
1229/1229 [==============================] - 2s 1ms/step - loss:
232.2418 - val_loss: 285.0339
Epoch 31/50
1229/1229 [==============================] - 2s 1ms/step - loss:
232.1657 - val_loss: 275.1656
Epoch 32/50
1229/1229 [==============================] - 2s 2ms/step - loss:
230.3734 - val_loss: 288.3420
Epoch 33/50
1229/1229 [==============================] - 2s 1ms/step - loss:
229.5123 - val_loss: 310.6277
Epoch 34/50
1229/1229 [==============================] - 2s 1ms/step - loss:
228.9370 - val_loss: 284.4604
Epoch 35/50
1229/1229 [==============================] - 2s 1ms/step - loss:
229.5443 - val_loss: 279.7957
Epoch 36/50
1229/1229 [==============================] - 2s 1ms/step - loss:
227.4201 - val_loss: 281.7409
Epoch 37/50
1229/1229 [==============================] - 2s 1ms/step - loss:
227.0335 - val_loss: 315.0797
Epoch 38/50
1229/1229 [==============================] - 2s 1ms/step - loss:
225.9923 - val_loss: 286.6010
Epoch 39/50
1229/1229 [==============================] - 2s 1ms/step - loss:
225.6736 - val_loss: 313.1949
Epoch 40/50
1229/1229 [==============================] - 2s 1ms/step - loss:
225.1939 - val_loss: 298.9798
Epoch 41/50
1229/1229 [==============================] - 2s 1ms/step - loss:
224.4666 - val_loss: 310.8296
Epoch 42/50
1229/1229 [==============================] - 2s 1ms/step - loss:
223.0988 - val_loss: 314.3835
Epoch 43/50
```

```
1229/1229 [==============================] - 2s 1ms/step - loss:
222.7210 - val_loss: 282.5793
Epoch 44/50
1229/1229 [==============================] - 2s 1ms/step - loss:
222.0095 - val_loss: 295.2156
Epoch 45/50
1229/1229 [==============================] - 2s 1ms/step - loss:
221.8514 - val_loss: 297.6723
Epoch 46/50
1229/1229 [==============================] - 2s 1ms/step - loss:
220.9864 - val_loss: 303.1740
Epoch 47/50
1229/1229 [==============================] - 2s 1ms/step - loss:
220.8002 - val_loss: 310.6229
Epoch 48/50
1229/1229 [==============================] - 2s 1ms/step - loss:
220.4447 - val_loss: 302.6555
Epoch 49/50
1229/1229 [==============================] - 2s 1ms/step - loss:
219.7761 - val_loss: 309.3281
Epoch 50/50
1229/1229 [==============================] - 2s 1ms/step - loss:
218.8459 - val_loss: 298.8487

plot_loss(history)
```




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

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

```
1518/1518 [==============================] - 2s 1ms/step
conjunto de treino -> RMSE:19.479221319109726, R2:0.7832595363037521
9/9 [==============================] - 0s 1ms/step
conjunto de teste -> RMSE:29.95703194009458, R2:0.6897045236400179
```

12. Ajuste de hiperparametros

```
# 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)
epoch_list = list(np.arange(10,30+1,5))
nodes_list = [[16, 32, 64], [32, 64, 128], [64, 128, 256], [128, 256,
512]]
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, 512]

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

```
6400
```

```
# Preparando os dados
def prep_data(df_train, train_label, df_test, remaining_sensors, lags,
alpha, n=0):
    X_train_interim = add_op_cond(df_train)
    X_test_interim = add_op_cond(df_test)

    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, n, alpha)
    X_test_interim = exponential_smoothing(X_test_interim,
remaining_sensors, n, alpha)

    X_train_interim = add_specific_lags(X_train_interim, lags,
remaining_sensors)
    X_test_interim = add_specific_lags(X_test_interim, lags,
remaining_sensors)
```

```
    X_train_interim.drop(index_names+setting_names+['op_cond', 'RUL'],
axis=1, inplace=True)
    X_test_interim = X_test_interim.drop(
        ['ciclo_tempo', 'op_cond']+setting_names,
axis=1).groupby('unidade').last().copy()

    idx = X_train_interim.index
    train_label = train_label.iloc[idx]
    return X_train_interim, train_label, X_test_interim, idx

# Definindo a funcao que vai gerar de forma automatica os modelos a
serem testados com os hiperparametros
def create_model(input_dim, nodes_per_layer, dropout, activation,
weights_file):
    model = Sequential()
    model.add(Dense(nodes_per_layer[0], input_dim=input_dim,
activation=activation))
    model.add(Dropout(dropout))
    model.add(Dense(nodes_per_layer[1], activation=activation))
    model.add(Dropout(dropout))
    model.add(Dense(nodes_per_layer[2], 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'])
weights_file = 'mlp_hyper_parameter_weights.h5'
specific_lags = [1,2,3,4,5,10,20]

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]
    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]

    # Criando o dataset
    df_train, train_label, _, idx =
prep_data(df_train=train.drop(drop_sensors, axis=1),

train_label=y_train_clipped,

df_test=test.drop(drop_sensors, axis=1),

remaining_sensors=remaining_sensors,
                                          lags=specific_lags,
                                          alpha=alpha)

    # Criando o modelo
    input_dim = len(df_train.columns)
    model = create_model(input_dim, nodes_per_layer, dropout,
activation, weights_file)


    # Separando em grupos de treino e validacao
    gss_search = GroupShuffleSplit(n_splits=3, train_size=0.80,
random_state=42)
    for idx_train, idx_val in gss_search.split(df_train, train_label,
groups=train.iloc[idx]['unidade']):
        X_train_split = df_train.iloc[idx_train].copy()
        y_train_split = train_label.iloc[idx_train].copy()
        X_val_split = df_train.iloc[idx_val].copy()
        y_val_split = train_label.iloc[idx_val].copy()

        # Treinando e avaliando o modelo
        model.compile(loss='mean_squared_error', optimizer='adam')
        model.load_weights(weights_file)
        history = model.fit(X_train_split, y_train_split,
                            validation_data=(X_val_split,
y_val_split),
                            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}
    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:  Thu Feb  8 15:27:29 2024

Iteração  1  de  100  concluída em  0:00:33.789344
Tempo restante previsto:  0:56:52.723703
Previsão para conclusão:  Thu Feb  8 16:24:56 2024

Iteração  2  de  100  concluída em  0:00:46.597985
Tempo restante previsto:  1:06:59.366407
Previsão para conclusão:  Thu Feb  8 16:35:49 2024

Iteração  3  de  100  concluída em  0:01:56.911473
Tempo restante previsto:  1:48:30.860447
Previsão para conclusão:  Thu Feb  8 17:19:17 2024

Iteração  4  de  100  concluída em  0:01:19.756272
Tempo restante previsto:  1:53:07.849295
Previsão para conclusão:  Thu Feb  8 17:25:14 2024

Iteração  5  de  100  concluída em  0:01:20.365503
Tempo restante previsto:  1:55:33.959176
Previsão para conclusão:  Thu Feb  8 17:29:00 2024

Iteração  6  de  100  concluída em  0:00:25.858095
```

```
Tempo restante previsto:  1:42:12.458744
Previsão para conclusão:  Thu Feb  8 17:16:05 2024

Iteração  7  de  100  concluída em  0:02:33.596657
Tempo restante previsto:  2:01:26.165167
Previsão para conclusão:  Thu Feb  8 17:37:52 2024

Iteração  8  de  100  concluída em  0:00:24.959073
Tempo restante previsto:  1:50:01.554210
Previsão para conclusão:  Thu Feb  8 17:26:52 2024

Iteração  9  de  100  concluída em  0:00:33.886527
Tempo restante previsto:  1:42:35.782917
Previsão para conclusão:  Thu Feb  8 17:20:01 2024

Iteração  10  de  100  concluída em  0:01:23.533006
Tempo restante previsto:  1:44:09.136192
Previsão para conclusão:  Thu Feb  8 17:22:57 2024

Iteração  11  de  100  concluída em  0:02:45.496483
Tempo restante previsto:  1:56:28.389815
Previsão para conclusão:  Thu Feb  8 17:38:02 2024

Iteração  12  de  100  concluída em  0:00:11.788085
Tempo restante previsto:  1:47:04.038765
Previsão para conclusão:  Thu Feb  8 17:28:50 2024

Iteração  13  de  100  concluída em  0:00:35.283087
Tempo restante previsto:  1:41:45.547800
Previsão para conclusão:  Thu Feb  8 17:24:06 2024

Iteração  14  de  100  concluída em  0:00:18.991301
Tempo restante previsto:  1:35:25.109592
Previsão para conclusão:  Thu Feb  8 17:18:05 2024

Iteração  15  de  100  concluída em  0:00:27.898413
Tempo restante previsto:  1:30:44.525557
Previsão para conclusão:  Thu Feb  8 17:13:52 2024

Iteração  16  de  100  concluída em  0:02:36.472780
Tempo restante previsto:  1:38:06.614443
Previsão para conclusão:  Thu Feb  8 17:23:51 2024

Iteração  17  de  100  concluída em  0:01:21.585960
Tempo restante previsto:  1:38:03.850213
Previsão para conclusão:  Thu Feb  8 17:25:10 2024

Iteração  18  de  100  concluída em  0:00:25.275261
Tempo restante previsto:  1:33:29.544752
Previsão para conclusão:  Thu Feb  8 17:21:01 2024
```

```
Iteração  19  de  100  concluída em  0:00:32.661570
Tempo restante previsto:  1:29:53.719503
Previsão para conclusão:  Thu Feb  8 17:17:57 2024

Iteração  20  de  100  concluída em  0:01:47.191263
Tempo restante previsto:  1:31:41.782362
Previsão para conclusão:  Thu Feb  8 17:21:33 2024

Iteração  21  de  100  concluída em  0:00:11.148197
Tempo restante previsto:  1:26:58.893002
Previsão para conclusão:  Thu Feb  8 17:17:01 2024

Iteração  22  de  100  concluída em  0:00:46.267331
Tempo restante previsto:  1:24:48.413328
Previsão para conclusão:  Thu Feb  8 17:15:37 2024

Iteração  23  de  100  concluída em  0:00:43.307151
Tempo restante previsto:  1:22:35.088889
Previsão para conclusão:  Thu Feb  8 17:14:07 2024

Iteração  24  de  100  concluída em  0:00:57.257769
Tempo restante previsto:  1:21:14.605401
Previsão para conclusão:  Thu Feb  8 17:13:43 2024

Iteração  25  de  100  concluída em  0:01:43.219708
Tempo restante previsto:  1:22:17.542743
Previsão para conclusão:  Thu Feb  8 17:16:30 2024

Iteração  26  de  100  concluída em  0:00:33.101424
Tempo restante previsto:  1:19:42.737634
Previsão para conclusão:  Thu Feb  8 17:14:28 2024

Iteração  27  de  100  concluída em  0:00:17.978981
Tempo restante previsto:  1:16:34.940829
Previsão para conclusão:  Thu Feb  8 17:11:38 2024

Iteração  28  de  100  concluída em  0:01:50.100786
Tempo restante previsto:  1:17:42.738638
Previsão para conclusão:  Thu Feb  8 17:14:36 2024

Iteração  29  de  100  concluída em  0:00:18.750842
Tempo restante previsto:  1:14:48.317719
Previsão para conclusão:  Thu Feb  8 17:12:00 2024

Iteração  30  de  100  concluída em  0:00:26.730306
Tempo restante previsto:  1:12:23.425519
Previsão para conclusão:  Thu Feb  8 17:10:02 2024

Iteração  31  de  100  concluída em  0:00:17.321781
```

```
Tempo restante previsto:  1:09:44.608110
Previsão para conclusão:  Thu Feb  8 17:07:41 2024

Iteração  32  de  100  concluída em  0:00:48.077127
Tempo restante previsto:  1:08:21.911496
Previsão para conclusão:  Thu Feb  8 17:07:06 2024

Iteração  33  de  100  concluída em  0:00:41.973149
Tempo restante previsto:  1:06:48.550170
Previsão para conclusão:  Thu Feb  8 17:06:15 2024

Iteração  34  de  100  concluída em  0:03:51.850077
Tempo restante previsto:  1:11:37.965535
Previsão para conclusão:  Thu Feb  8 17:14:56 2024

Iteração  35  de  100  concluída em  0:03:15.268705
Tempo restante previsto:  1:14:47.567104
Previsão para conclusão:  Thu Feb  8 17:21:21 2024

Iteração  36  de  100  concluída em  0:00:29.299791
Tempo restante previsto:  1:12:31.510650
Previsão para conclusão:  Thu Feb  8 17:19:34 2024

Iteração  37  de  100  concluída em  0:04:29.877551
Tempo restante previsto:  1:17:23.861432
Previsão para conclusão:  Thu Feb  8 17:28:56 2024

Iteração  38  de  100  concluída em  0:00:47.487609
Tempo restante previsto:  1:15:32.069766
Previsão para conclusão:  Thu Feb  8 17:27:52 2024

Iteração  39  de  100  concluída em  0:01:31.720084
Tempo restante previsto:  1:14:55.028204
Previsão para conclusão:  Thu Feb  8 17:28:47 2024

Iteração  40  de  100  concluída em  0:01:00.130831
Tempo restante previsto:  1:13:26.289375
Previsão para conclusão:  Thu Feb  8 17:28:18 2024

Iteração  41  de  100  concluída em  0:00:16.677973
Tempo restante previsto:  1:10:54.296683
Previsão para conclusão:  Thu Feb  8 17:26:03 2024

Iteração  42  de  100  concluída em  0:01:33.130504
Tempo restante previsto:  1:10:17.965451
Previsão para conclusão:  Thu Feb  8 17:27:00 2024

Iteração  43  de  100  concluída em  0:00:26.819218
Tempo restante previsto:  1:08:08.007139
Previsão para conclusão:  Thu Feb  8 17:25:16 2024
```

```
Iteração  44  de  100  concluída em  0:01:22.084222
Tempo restante previsto:  1:07:15.586291
Previsão para conclusão:  Thu Feb  8 17:25:46 2024

Iteração  45  de  100  concluída em  0:00:53.495668
Tempo restante previsto:  1:05:45.634903
Previsão para conclusão:  Thu Feb  8 17:25:10 2024

Iteração  46  de  100  concluída em  0:00:11.257115
Tempo restante previsto:  1:03:25.847700
Previsão para conclusão:  Thu Feb  8 17:23:01 2024

Iteração  47  de  100  concluída em  0:00:34.809804
Tempo restante previsto:  1:01:39.091518
Previsão para conclusão:  Thu Feb  8 17:21:49 2024

Iteração  48  de  100  concluída em  0:00:33.842561
Tempo restante previsto:  0:59:54.244954
Previsão para conclusão:  Thu Feb  8 17:20:38 2024

Iteração  49  de  100  concluída em  0:00:28.193678
Tempo restante previsto:  0:58:06.186496
Previsão para conclusão:  Thu Feb  8 17:19:18 2024

Iteração  50  de  100  concluída em  0:00:25.383071
Tempo restante previsto:  0:56:18.399599
Previsão para conclusão:  Thu Feb  8 17:17:56 2024

Iteração  51  de  100  concluída em  0:00:24.369086
Tempo restante previsto:  0:54:32.830239
Previsão para conclusão:  Thu Feb  8 17:16:35 2024

Iteração  52  de  100  concluída em  0:02:36.661966
Tempo restante previsto:  0:54:57.588658
Previsão para conclusão:  Thu Feb  8 17:19:36 2024

Iteração  53  de  100  concluída em  0:00:44.194987
Tempo restante previsto:  0:53:31.522120
Previsão para conclusão:  Thu Feb  8 17:18:54 2024

Iteração  54  de  100  concluída em  0:00:26.851789
Tempo restante previsto:  0:51:51.590205
Previsão para conclusão:  Thu Feb  8 17:17:41 2024

Iteração  55  de  100  concluída em  0:02:36.196578
Tempo restante previsto:  0:52:04.846750
Previsão para conclusão:  Thu Feb  8 17:20:31 2024

Iteração  56  de  100  concluída em  0:00:43.429060
```

```
Tempo restante previsto:  0:50:39.420937
Previsão para conclusão:  Thu Feb  8 17:19:49 2024

Iteração  57  de  100  concluída em  0:00:58.909680
Tempo restante previsto:  0:49:27.690236
Previsão para conclusão:  Thu Feb  8 17:19:36 2024

Iteração  58  de  100  concluída em  0:01:07.520505
Tempo restante previsto:  0:48:22.933989
Previsão para conclusão:  Thu Feb  8 17:19:39 2024

Iteração  59  de  100  concluída em  0:01:34.601228
Tempo restante previsto:  0:47:37.820837
Previsão para conclusão:  Thu Feb  8 17:20:28 2024

Iteração  60  de  100  concluída em  0:00:18.480594
Tempo restante previsto:  0:45:57.773638
Previsão para conclusão:  Thu Feb  8 17:19:06 2024

Iteração  61  de  100  concluída em  0:02:11.393107
Tempo restante previsto:  0:45:36.292771
Previsão para conclusão:  Thu Feb  8 17:20:56 2024

Iteração  62  de  100  concluída em  0:01:12.629227
Tempo restante previsto:  0:44:33.354167
Previsão para conclusão:  Thu Feb  8 17:21:06 2024

Iteração  63  de  100  concluída em  0:00:24.811836
Tempo restante previsto:  0:43:00.506683
Previsão para conclusão:  Thu Feb  8 17:19:58 2024

Iteração  64  de  100  concluída em  0:00:55.904170
Tempo restante previsto:  0:41:48.246386
Previsão para conclusão:  Thu Feb  8 17:19:42 2024

Iteração  65  de  100  concluída em  0:01:06.729653
Tempo restante previsto:  0:40:42.651552
Previsão para conclusão:  Thu Feb  8 17:19:43 2024

Iteração  66  de  100  concluída em  0:01:17.021357
Tempo restante previsto:  0:39:42.635986
Previsão para conclusão:  Thu Feb  8 17:20:00 2024

Iteração  67  de  100  concluída em  0:00:12.141387
Tempo restante previsto:  0:38:08.220264
Previsão para conclusão:  Thu Feb  8 17:18:38 2024

Iteração  68  de  100  concluída em  0:01:12.793731
Tempo restante previsto:  0:37:06.550547
Previsão para conclusão:  Thu Feb  8 17:18:49 2024
```

```
Iteração  69  de  100  concluída em  0:00:41.395251
Tempo restante previsto:  0:35:49.541730
Previsão para conclusão:  Thu Feb  8 17:18:13 2024

Iteração  70  de  100  concluída em  0:00:32.497833
Tempo restante previsto:  0:34:29.483052
Previsão para conclusão:  Thu Feb  8 17:17:25 2024

Iteração  71  de  100  concluída em  0:01:02.409904
Tempo restante previsto:  0:33:23.824317
Previsão para conclusão:  Thu Feb  8 17:17:22 2024

Iteração  72  de  100  concluída em  0:01:17.619559
Tempo restante previsto:  0:32:24.593184
Previsão para conclusão:  Thu Feb  8 17:17:41 2024

Iteração  73  de  100  concluída em  0:00:38.761703
Tempo restante previsto:  0:31:09.421575
Previsão para conclusão:  Thu Feb  8 17:17:04 2024

Iteração  74  de  100  concluída em  0:00:32.022824
Tempo restante previsto:  0:29:52.684171
Previsão para conclusão:  Thu Feb  8 17:16:19 2024

Iteração  75  de  100  concluída em  0:01:01.814538
Tempo restante previsto:  0:28:47.864174
Previsão para conclusão:  Thu Feb  8 17:16:16 2024

Iteração  76  de  100  concluída em  0:00:25.753116
Tempo restante previsto:  0:27:30.786465
Previsão para conclusão:  Thu Feb  8 17:15:25 2024

Iteração  77  de  100  concluída em  0:00:40.227504
Tempo restante previsto:  0:26:19.741339
Previsão para conclusão:  Thu Feb  8 17:14:54 2024

Iteração  78  de  100  concluída em  0:00:40.006383
Tempo restante previsto:  0:25:09.418371
Previsão para conclusão:  Thu Feb  8 17:14:24 2024

Iteração  79  de  100  concluída em  0:01:04.612955
Tempo restante previsto:  0:24:07.026851
Previsão para conclusão:  Thu Feb  8 17:14:26 2024

Iteração  80  de  100  concluída em  0:00:18.327286
Tempo restante previsto:  0:22:51.851236
Previsão para conclusão:  Thu Feb  8 17:13:29 2024

Iteração  81  de  100  concluída em  0:00:27.634666
```

```
Tempo restante previsto:  0:21:40.492308
Previsão para conclusão:  Thu Feb  8 17:12:46 2024

Iteração  82  de  100  concluída em  0:01:01.932489
Tempo restante previsto:  0:20:38.565147
Previsão para conclusão:  Thu Feb  8 17:12:46 2024

Iteração  83  de  100  concluída em  0:00:20.028007
Tempo restante previsto:  0:19:27.045266
Previsão para conclusão:  Thu Feb  8 17:11:54 2024

Iteração  84  de  100  concluída em  0:00:54.652529
Tempo restante previsto:  0:18:24.170923
Previsão para conclusão:  Thu Feb  8 17:11:46 2024

Iteração  85  de  100  concluída em  0:02:04.725112
Tempo restante previsto:  0:17:35.504550
Previsão para conclusão:  Thu Feb  8 17:13:02 2024

Iteração  86  de  100  concluída em  0:01:36.258952
Tempo restante previsto:  0:16:39.773340
Previsão para conclusão:  Thu Feb  8 17:13:43 2024

Iteração  87  de  100  concluída em  0:01:02.867985
Tempo restante previsto:  0:15:37.353395
Previsão para conclusão:  Thu Feb  8 17:13:43 2024

Iteração  88  de  100  concluída em  0:00:42.225927
Tempo restante previsto:  0:14:31.639303
Previsão para conclusão:  Thu Feb  8 17:13:20 2024

Iteração  89  de  100  concluída em  0:00:18.684047
Tempo restante previsto:  0:13:23.014333
Previsão para conclusão:  Thu Feb  8 17:12:30 2024

Iteração  90  de  100  concluída em  0:00:30.432599
Tempo restante previsto:  0:12:17.065635
Previsão para conclusão:  Thu Feb  8 17:11:54 2024

Iteração  91  de  100  concluída em  0:00:34.232650
Tempo restante previsto:  0:11:12.356858
Previsão para conclusão:  Thu Feb  8 17:11:24 2024

Iteração  92  de  100  concluída em  0:00:35.462374
Tempo restante previsto:  0:10:08.444270
Previsão para conclusão:  Thu Feb  8 17:10:55 2024

Iteração  93  de  100  concluída em  0:00:11.591370
Tempo restante previsto:  0:09:02.833418
Previsão para conclusão:  Thu Feb  8 17:10:01 2024
```

```
Iteração  94  de  100  concluída em  0:00:19.466745
Tempo restante previsto:  0:07:59.042162
Previsão para conclusão:  Thu Feb  8 17:09:17 2024

Iteração  95  de  100  concluída em  0:01:09.304313
Tempo restante previsto:  0:06:59.856295
Previsão para conclusão:  Thu Feb  8 17:09:27 2024

Iteração  96  de  100  concluída em  0:00:42.862517
Tempo restante previsto:  0:05:58.807014
Previsão para conclusão:  Thu Feb  8 17:09:09 2024

Iteração  97  de  100  concluída em  0:00:53.760073
Tempo restante previsto:  0:04:58.694449
Previsão para conclusão:  Thu Feb  8 17:09:02 2024

Iteração  98  de  100  concluída em  0:02:53.740899
Tempo restante previsto:  0:04:03.608702
Previsão para conclusão:  Thu Feb  8 17:11:01 2024

Iteração  99  de  100  concluída em  0:00:23.735442
Tempo restante previsto:  0:03:01.580262
Previsão para conclusão:  Thu Feb  8 17:10:23 2024

Iteração  100  de  100  concluída em  0:00:23.505545
Tempo restante previsto:  0:02:00.313084
Previsão para conclusão:  Thu Feb  8 17:09:45 2024
```

```
# Ordenando os resultados
sorted_results = results.sort_values('MSE')
sorted_results
```

```
            MSE     std_MSE  alpha epochs            nodes  dropout  \
68   253.056396   20.763111   0.05     15   [64, 128, 256]      0.1
97   255.200928   22.859067   0.01     30  [128, 256, 512]      0.4
32   256.940684   24.401943   0.30     25   [64, 128, 256]      0.1
84   258.184591   23.739028   0.10     15   [64, 128, 256]      0.1
94   258.454219   26.557427   0.10     30  [128, 256, 512]      0.4
..          ...         ...    ...    ...              ...      ...
28  1979.575887   86.500409   0.01     20     [16, 32, 64]      0.1
45  2044.514323  104.486543   0.40     10     [16, 32, 64]      0.3
20  2129.662842  106.493531   0.20     10     [16, 32, 64]      0.3
93  2354.812663  103.608973   0.01     20     [16, 32, 64]      0.2
92  3346.966878  134.908655   0.60     10     [16, 32, 64]      0.4

   activation batch_size
68       tanh        128
97       tanh         64
32       tanh        256
```

```
84       tanh          32
94       tanh         256
..        ...         ...
28    sigmoid         512
45       tanh         512
20       tanh         512
93    sigmoid         512
92    sigmoid         512

[100 rows x 8 columns]
```

13. Modelo Final

```
# Melhores parametros
alpha = float(sorted_results['alpha'].iloc[0])
epochs = int(sorted_results['epochs'].iloc[0])
specific_lags = [1,2,3,4,5,10,20]
nodes = [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])

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

```
Parametros do modelo final:

alpha =  0.05
epochs =  15
nodes =  [64, 128, 256]
dropout =  0.1
activation =  tanh
batch_size =  128
```

```
# Preparando os dados
df_train, train_label, df_test, _ =
prep_data(df_train=train.drop(drop_sensors, axis=1),
                                          train_label=y_train_clipped,

df_test=test.drop(drop_sensors, axis=1),
```

```
remaining_sensors=remaining_sensors,
                                        lags=specific_lags,
                                        alpha=alpha)

# Criar e carregar o modelo
input_dim = len(df_train.columns)
weights_file = 'fd002_model_weights.h5'
final_model = create_model(input_dim,
                           nodes_per_layer=nodes,
                           dropout=dropout,
                           activation=activation,
                           weights_file=weights_file)

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

# Ajustar o modelo
final_model.fit(df_train, train_label,
                epochs=epochs,
                batch_size=batch_size)
```

```
Epoch 1/15
380/380 [==============================] - 2s 3ms/step - loss:
2481.9802
Epoch 2/15
380/380 [==============================] - 1s 3ms/step - loss:
369.0403
Epoch 3/15
380/380 [==============================] - 1s 3ms/step - loss:
304.6987
Epoch 4/15
380/380 [==============================] - 1s 3ms/step - loss:
295.6360
Epoch 5/15
380/380 [==============================] - 1s 3ms/step - loss:
291.1004
Epoch 6/15
380/380 [==============================] - 1s 3ms/step - loss:
288.1526
Epoch 7/15
380/380 [==============================] - 1s 3ms/step - loss:
283.2098
Epoch 8/15
380/380 [==============================] - 1s 3ms/step - loss:
280.4359
Epoch 9/15
380/380 [==============================] - 1s 3ms/step - loss:
275.2274
Epoch 10/15
380/380 [==============================] - 1s 3ms/step - loss:
```

```
273.6574
Epoch 11/15
380/380 [==============================] - 1s 3ms/step - loss:
269.0629
Epoch 12/15
380/380 [==============================] - 1s 3ms/step - loss:
267.6482
Epoch 13/15
380/380 [==============================] - 1s 3ms/step - loss:
264.9823
Epoch 14/15
380/380 [==============================] - 1s 3ms/step - loss:
264.2480
Epoch 15/15
380/380 [==============================] - 1s 3ms/step - loss:
261.1174

<keras.callbacks.History at 0x1dd039376a0>

# Avaliando o modelo
y_hat_train = final_model.predict(df_train)
avaliar(train_label, y_hat_train, 'treino')

y_hat_test = final_model.predict(df_test)
avaliar(y_test, y_hat_test)

1518/1518 [==============================] - 2s 1ms/step
conjunto de treino -> RMSE:15.653994313889608, R2:0.8600260695681474
9/9 [==============================] - 0s 1ms/step
conjunto de teste -> RMSE:27.60127026814015, R2:0.7365877192521701
```