O3_reference_and_sensor_data_with_temp

February 14, 2024

0.0.1 Sensor Constants

[ ]:
```
sensor_name = 'alpha_o3_conc'
sensor_1_name = 'alpha_o3_1_conc'
sensor_2_name = 'alpha_o3_2_conc'
```

0.1 Upload Data from File

0.1.1 Sensor 1

[ ]:
```
import pandas as pd

directory_path = 'input/'
file_name = sensor_1_name + '_and_temp_valid_1HR.csv'
df_1 = pd.read_csv(directory_path + file_name)
df_1.head()
```

[ ]:

	DateTime	measuring 1	temperature	Hour	measuring 1 no Temp	Count 1	Tag
0	2022-12-14 14:30:00	42.267696	31.52000	14	56.369783	3	VALID
1	2022-12-14 15:30:00	50.822340	30.56750	15	69.328964	4	VALID
2	2022-12-14 16:30:00	67.516902	28.82875	16	94.063829	4	VALID
3	2022-12-14 17:30:00	68.069052	27.91125	17	98.858670	4	VALID
4	2022-12-14 18:30:00	84.294900	27.16250	18	118.546877	4	VALID

[ ]:
```
import locale
locale.setlocale(locale.LC_TIME, 'pt_BR')
```

[ ]:
```
'pt_BR'
```

0.1.2 Create Sensor Dataframe as Pandas Series

[ ]:
```
# Remove the first column with the indexes and save data into web dataframe
dataframe = df_1.drop(df_1.columns[0], axis='columns')
dataframe['DateTime'] = (pd.to_datetime(df_1['DateTime'],␣
 ↪infer_datetime_format=True))

# Resample data with 15 mins period and create sensor dataframe
sensor_1_dataframe = dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_1_dataframe.index = sensor_1_dataframe['DateTime']
sensor_1_dataframe = sensor_1_dataframe.drop(columns=['DateTime', 'Hour'])
sensor_1_dataframe = sensor_1_dataframe.rename(columns={'temperature':␣
 ↪'temperature 1'})
sensor_1_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_6358/1180347280.py:3:
UserWarning: The argument 'infer_datetime_format' is deprecated and will be
removed in a future version. A strict version of it is now the default, see
https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-parsing.html. You
can safely remove this argument.
  dataframe['DateTime'] = (pd.to_datetime(df_1['DateTime'],
infer_datetime_format=True))
```

[ ]:

DateTime	measuring 1	temperature 1	measuring 1 no Temp	Count 1	\
2022-12-14 14:30:00	42.267696	31.52000	56.369783	3	
2022-12-14 15:30:00	50.822340	30.56750	69.328964	4	
2022-12-14 16:30:00	67.516902	28.82875	94.063829	4	
2022-12-14 17:30:00	68.069052	27.91125	98.858670	4	
2022-12-14 18:30:00	84.294900	27.16250	118.546877	4	
…	…	…	…	…	
2023-04-19 16:30:00	50.962218	28.84750	77.422442	4	
2023-04-19 17:30:00	56.130342	27.83625	87.266774	4	
2023-04-19 18:30:00	40.233330	25.86125	80.502529	4	
2023-04-19 19:30:00	51.222342	23.64875	101.722552	4	
2023-04-20 20:30:00	38.675040	22.74000	93.377479	3	

DateTime	Tag
2022-12-14 14:30:00	VALID
2022-12-14 15:30:00	VALID
2022-12-14 16:30:00	VALID
2022-12-14 17:30:00	VALID
2022-12-14 18:30:00	VALID
…	…
2023-04-19 16:30:00	VALID
2023-04-19 17:30:00	VALID

```
2023-04-19 18:30:00  VALID
2023-04-19 19:30:00  VALID
2023-04-20 20:30:00  VALID

[1021 rows x 5 columns]
```

0.1.3 Sensor 2

[ ]:
```
import pandas as pd

directory_path = 'input/'
file_name = sensor_2_name + '_and_temp_valid_1HR.csv'
df_2 = pd.read_csv(directory_path + file_name)
df_2.head()
```

[ ]:
```
              DateTime  measuring 2  temperature  Hour  measuring 2 no Temp  \
0  2022-11-28 11:30:00    53.759778     30.10750    11             11.239952
1  2022-11-28 12:30:00    53.445666     29.88250    12             11.476654
2  2022-11-28 13:30:00    54.100884     30.24125    13             11.253630
3  2022-11-28 14:30:00    53.921742     30.13250    14             11.340715
4  2022-11-28 15:30:00    53.494746     29.89875    15             11.485953

   Count 2    Tag
0        4  VALID
1        4  VALID
2        4  VALID
3        4  VALID
4        4  VALID
```

0.1.4 Create Sensor Dataframe as Pandas Series

[ ]:
```
# Remove the first column with the indexes and save data into web dataframe
dataframe = df_2.drop(df_2.columns[0], axis='columns')
dataframe['DateTime'] = (pd.to_datetime(df_2['DateTime'],␣
 ↪infer_datetime_format=True))

# Resample data with 15 mins period and create sensor dataframe
sensor_2_dataframe = dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_2_dataframe.index = sensor_2_dataframe['DateTime']
sensor_2_dataframe = sensor_2_dataframe.drop(columns=['DateTime', 'Hour'])
sensor_2_dataframe = sensor_2_dataframe.rename(columns={'temperature':␣
 ↪'temperature 2'})
sensor_2_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_6358/3647602612.py:3:
UserWarning: The argument 'infer_datetime_format' is deprecated and will be
removed in a future version. A strict version of it is now the default, see
```

```
https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-parsing.html. You
can safely remove this argument.
  dataframe['DateTime'] = (pd.to_datetime(df_2['DateTime'],
infer_datetime_format=True))
```

[ ]:

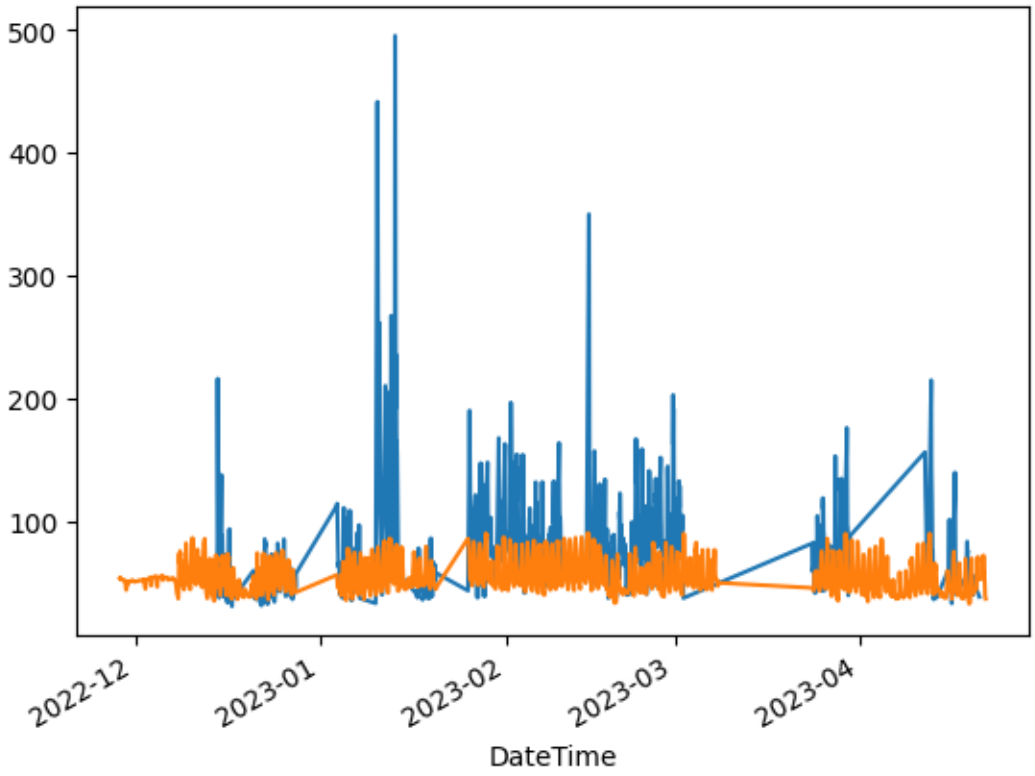
DateTime	measuring 2	temperature 2	measuring 2 no Temp	Count 2	Tag
2022-11-28 11:30:00	53.759778	30.10750	11.239952	4	VALID
2022-11-28 12:30:00	53.445666	29.88250	11.476654	4	VALID
2022-11-28 13:30:00	54.100884	30.24125	11.253630	4	VALID
2022-11-28 14:30:00	53.921742	30.13250	11.340715	4	VALID
2022-11-28 15:30:00	53.494746	29.89875	11.485953	4	VALID
…	…	…	…	…	…
2023-04-21 17:30:00	50.964672	30.53625	7.395241	4	VALID
2023-04-21 18:30:00	42.360948	27.13750	7.111857	4	VALID
2023-04-21 19:30:00	39.303264	25.32750	8.485160	4	VALID
2023-04-21 20:30:00	37.688532	24.45875	8.997179	4	VALID
2023-04-21 21:30:00	37.048038	23.90625	9.709237	4	VALID

[2603 rows x 5 columns]

0.1.5 Plot raw data

[ ]:
```
sensor_1_dataframe['measuring 1'].plot()
sensor_2_dataframe['measuring 2'].plot()
```

[ ]:
```
<Axes: xlabel='DateTime'>
```



0.2 Compare with original data

[ ]:
```
input_data_directory = 'input/'
reference_data_path = input_data_directory + 'ref_air_quality_data_Vila_Moema.
 ↪csv'
reference_column_name = 'Ozônio'
```

0.3 Load reference and sensor data

[ ]:
```
import pandas as pd

reference_data = pd.read_csv(reference_data_path)
reference_data['DateTime'] = (pd.to_datetime(reference_data['DateTime'],␣
 ↪infer_datetime_format=True))
reference_data = reference_data.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
reference_data.index = reference_data['DateTime']
reference_data = reference_data.drop(columns='DateTime')[reference_column_name]

reference_data
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_6358/2986665073.py:4:
```

```
UserWarning: The argument 'infer_datetime_format' is deprecated and will be
removed in a future version. A strict version of it is now the default, see
https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-parsing.html. You
can safely remove this argument.
  reference_data['DateTime'] = (pd.to_datetime(reference_data['DateTime'],
infer_datetime_format=True))
```

[ ]:
```
DateTime
2022-01-01 02:30:00    13.23
2022-01-01 03:30:00    12.07
2022-01-01 04:30:00    13.24
2022-01-01 05:30:00    14.42
2022-01-01 06:30:00    13.30
                       …
2023-02-08 12:30:00    50.01
2023-02-08 13:30:00    67.43
2023-02-08 14:30:00    72.46
2023-02-08 15:30:00    59.65
2023-02-08 16:30:00      NaN
Name: Ozônio, Length: 9687, dtype: float64
```

0.4 Merge sensor and reference data

[ ]:
```
import numpy as np

sensor_data = pd.concat([sensor_1_dataframe, sensor_2_dataframe], join='outer',␣
 ↪axis=1)
sensor_data['temperature'] = sensor_data[['temperature 1', 'temperature 2']].
 ↪apply(lambda df: df[1] if np.isnan(df[0]) else df[0], axis=1)
sensor_data = sensor_data.drop(columns=['temperature 1', 'Count 1', 'Tag',␣
 ↪'temperature 2', 'Count 2'])
sensor_data
```

[ ]:
```
                     measuring 1  measuring 1 no Temp  measuring 2  \
DateTime
2022-11-28 11:30:00          NaN                  NaN    53.759778
2022-11-28 12:30:00          NaN                  NaN    53.445666
2022-11-28 13:30:00          NaN                  NaN    54.100884
2022-11-28 14:30:00          NaN                  NaN    53.921742
2022-11-28 15:30:00          NaN                  NaN    53.494746
…                            …                    …           …
2023-04-21 17:30:00          NaN                  NaN    50.964672
2023-04-21 18:30:00          NaN                  NaN    42.360948
2023-04-21 19:30:00          NaN                  NaN    39.303264
2023-04-21 20:30:00          NaN                  NaN    37.688532
2023-04-21 21:30:00          NaN                  NaN    37.048038
```

```
                     measuring 2 no Temp  temperature
DateTime
2022-11-28 11:30:00            11.239952     30.10750
2022-11-28 12:30:00            11.476654     29.88250
2022-11-28 13:30:00            11.253630     30.24125
2022-11-28 14:30:00            11.340715     30.13250
2022-11-28 15:30:00            11.485953     29.89875
…                                    …          …
2023-04-21 17:30:00             7.395241     30.53625
2023-04-21 18:30:00             7.111857     27.13750
2023-04-21 19:30:00             8.485160     25.32750
2023-04-21 20:30:00             8.997179     24.45875
2023-04-21 21:30:00             9.709237     23.90625

[2616 rows x 5 columns]
```

[ ]:
```
sensor_data = pd.concat([sensor_data, reference_data], axis=1, join='inner')
sensor_data = sensor_data.rename(columns={'Ozônio': 'reference'})
sensor_data
```

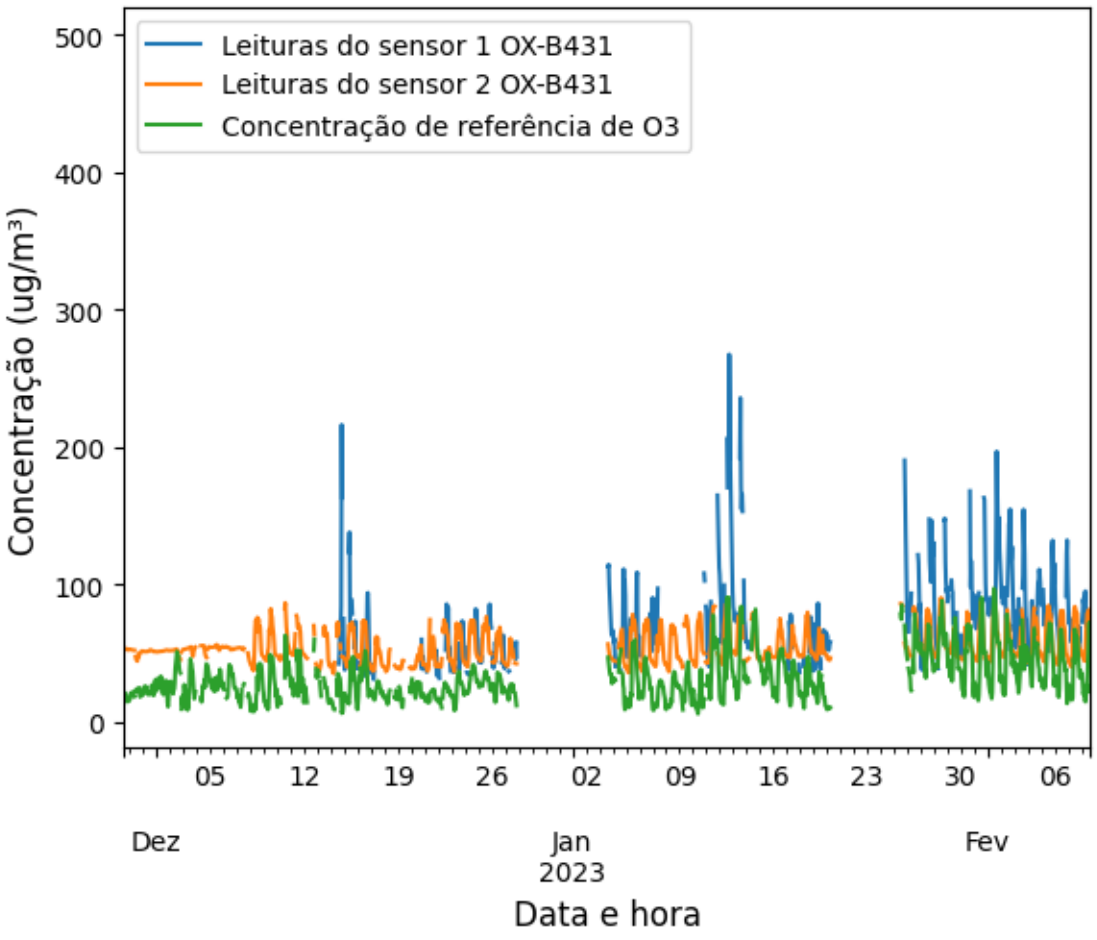
[ ]:
```
                     measuring 1  measuring 1 no Temp  measuring 2  \
DateTime
2022-11-28 11:30:00          NaN                  NaN    53.759778
2022-11-28 12:30:00          NaN                  NaN    53.445666
2022-11-28 13:30:00          NaN                  NaN    54.100884
2022-11-28 14:30:00          NaN                  NaN    53.921742
2022-11-28 15:30:00          NaN                  NaN    53.494746
…                            …                    …          …
2023-02-08 12:30:00          NaN                  NaN    80.255616
2023-02-08 13:30:00          NaN                  NaN    81.669120
2023-02-08 14:30:00          NaN                  NaN    80.498562
2023-02-08 15:30:00          NaN                  NaN    79.146408
2023-02-08 16:30:00          NaN                  NaN    71.617536

                     measuring 2 no Temp  temperature  reference
DateTime
2022-11-28 11:30:00            11.239952     30.10750      21.49
2022-11-28 12:30:00            11.476654     29.88250        NaN
2022-11-28 13:30:00            11.253630     30.24125        NaN
2022-11-28 14:30:00            11.340715     30.13250      20.56
2022-11-28 15:30:00            11.485953     29.89875      21.15
…                                    …          …         …
2023-02-08 12:30:00            15.204469     39.31125      50.01
2023-02-08 13:30:00            15.351102     39.82875      67.43
2023-02-08 14:30:00            13.929618     39.93125      72.46
2023-02-08 15:30:00            12.788609     39.84500      59.65
2023-02-08 16:30:00             9.874328     37.96000        NaN
```

```
[1345 rows x 6 columns]
```

0.4.1 Plot reference and sensor data

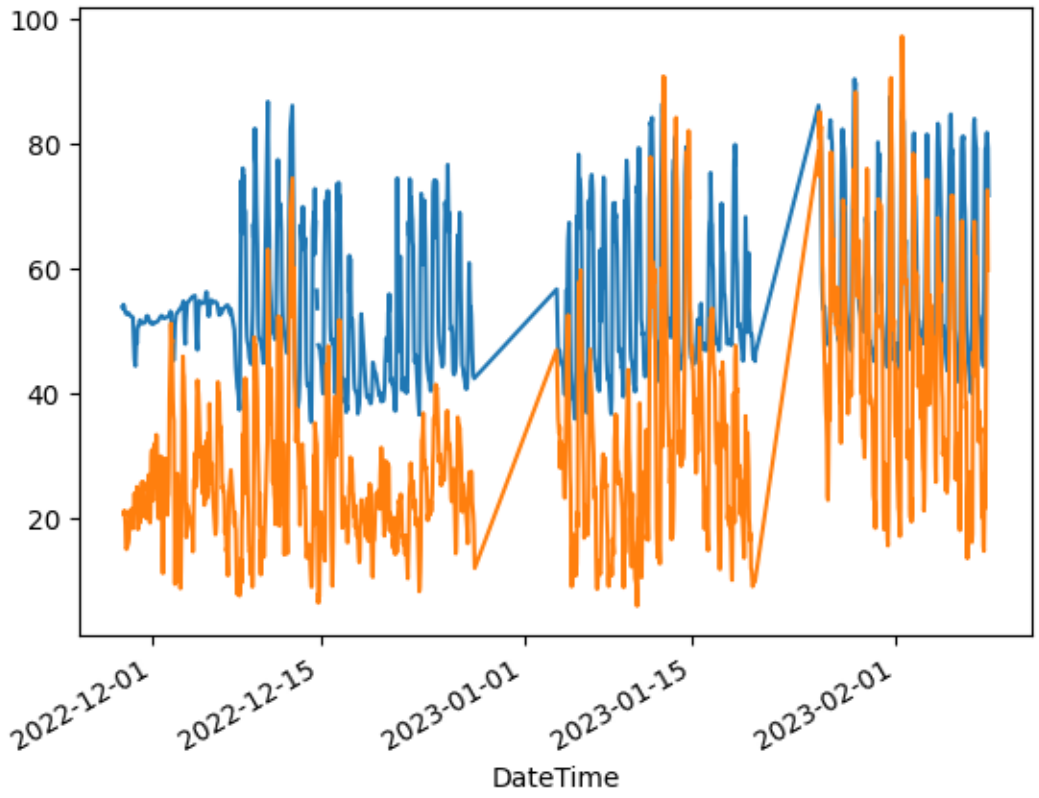
```
[ ]: import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(1.3*5,5))
sensor_data['measuring 1'].rename('Leituras do sensor 1 OX-B431').resample('H').
 ↪mean().plot()
sensor_data['measuring 2'].rename('Leituras do sensor 2 OX-B431').resample('H').
 ↪mean().plot()
sensor_data['reference'].rename('Concentração de referência de O3').
 ↪resample('H').mean().plot()
plt.legend()
ax.set_xlabel('Data e hora', fontsize=12)
ax.set_ylabel('Concentração (ug/m\N{SUPERSCRIPT THREE})', fontsize=12)
```

```
[ ]: Text(0, 0.5, 'Concentração (ug/m³)')
```



[ ]:
```
sensor_data['measuring 2'].plot()
sensor_data['reference'].plot()
```

[ ]:
```
<Axes: xlabel='DateTime'>
```



0.5 Plot sensor vs. reference

0.5.1 Sensor 1

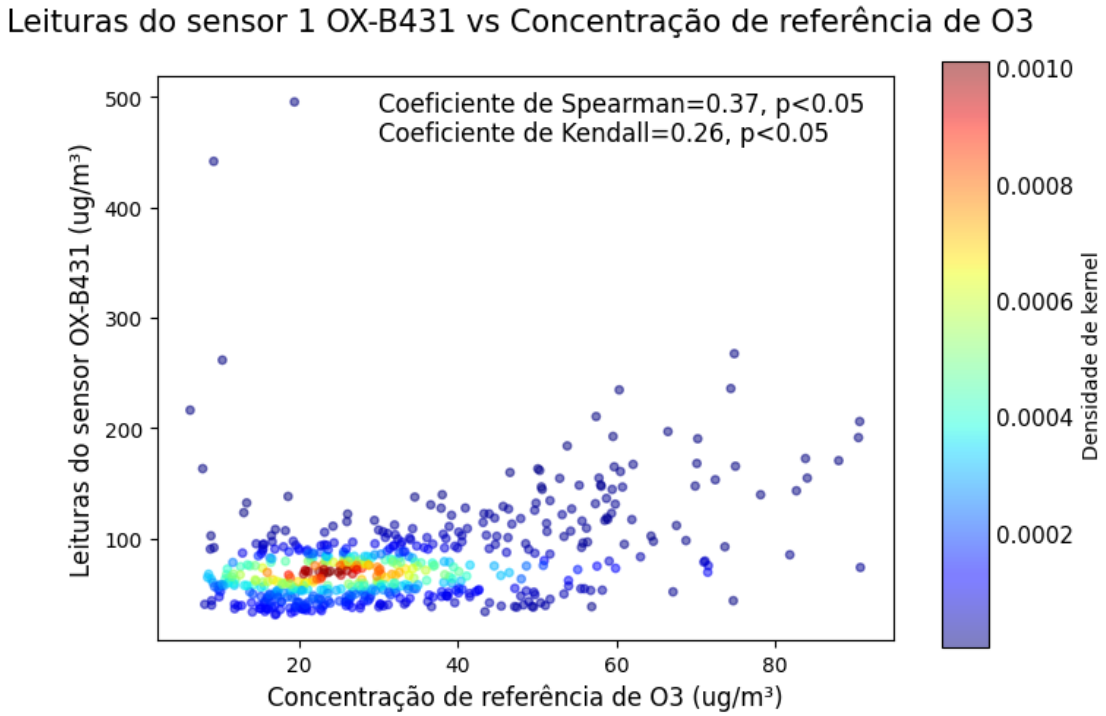
[ ]:
```
from scipy.stats import kendalltau, spearmanr, gaussian_kde
import matplotlib.pyplot as plt
import numpy as np

median_reference = sensor_data['reference'].median()
median_measuring = sensor_data['measuring 1'].median()

fig, ax = plt.subplots(figsize=(1.3*5,5))
xy = np.vstack([sensor_data['reference'].fillna(value=median_reference),␣
 ↪sensor_data['measuring 1'].fillna(value=median_measuring)])
```

```
z = gaussian_kde(xy)(xy)

plt.scatter(sensor_data['reference'], sensor_data['measuring 1'], c=z,␣
 ↪cmap='jet', s=15, alpha=.5)

plt.title('Leituras do sensor 1 OX-B431 vs Concentração de referência de O3 \n',
          fontdict={'fontsize':15})
plt.xlabel('Concentração de referência de O3 (ug/m\N{SUPERSCRIPT THREE})',
           fontsize=12)
plt.ylabel('Leituras do sensor OX-B431 (ug/m\N{SUPERSCRIPT THREE})',
           fontsize=12)
coef, p = spearmanr(sensor_data['reference'], sensor_data['measuring 1'],
                    nan_policy='omit')

print('Spearmans correlation coefficient: %.2f' % coef)
# interpret the significance
alpha = 0.05
if p > alpha:
        print('Samples are uncorrelated (fail to reject H0) p=%.2f' % p)
else:
        print('Samples are correlated (reject H0) p=%.2f' % p)

kendall, pken = kendalltau(sensor_data['reference'], sensor_data['measuring 1'],
                           nan_policy='omit')
print('Kendall correlation coefficient: %.2f' % coef)
# interpret the significance
alpha = 0.05
if p > alpha:
        print('Samples are uncorrelated (fail to reject H0) p=%.2f' % p)
else:
        print('Samples are correlated (reject H0) p=%.2f' % p)

cax = plt.axes([0.95, 0.1, 0.05, 0.8])
cbar = plt.colorbar(orientation='vertical', cax=cax, label="Densidade de␣
 ↪kernel")
cbar.ax.tick_params(labelsize=11, length=0)

string = 'Coeficiente de Spearman=%.2f, p<0.05' % coef
plt.text(0.3, 0.95, string, horizontalalignment='left', fontsize=12,
         verticalalignment='center', transform=ax.transAxes)
string = 'Coeficiente de Kendall=%.2f, p<0.05' % kendall
plt.text(0.3, 0.90, string, horizontalalignment='left', fontsize=12,
         verticalalignment='center', transform=ax.transAxes)
```

```
Spearmans correlation coefficient: 0.37
Samples are correlated (reject H0) p=0.00
Kendall correlation coefficient: 0.37
```

```
Samples are correlated (reject H0) p=0.00
```

[ ]:
```
Text(0.3, 0.9, 'Coeficiente de Kendall=0.26, p<0.05')
```



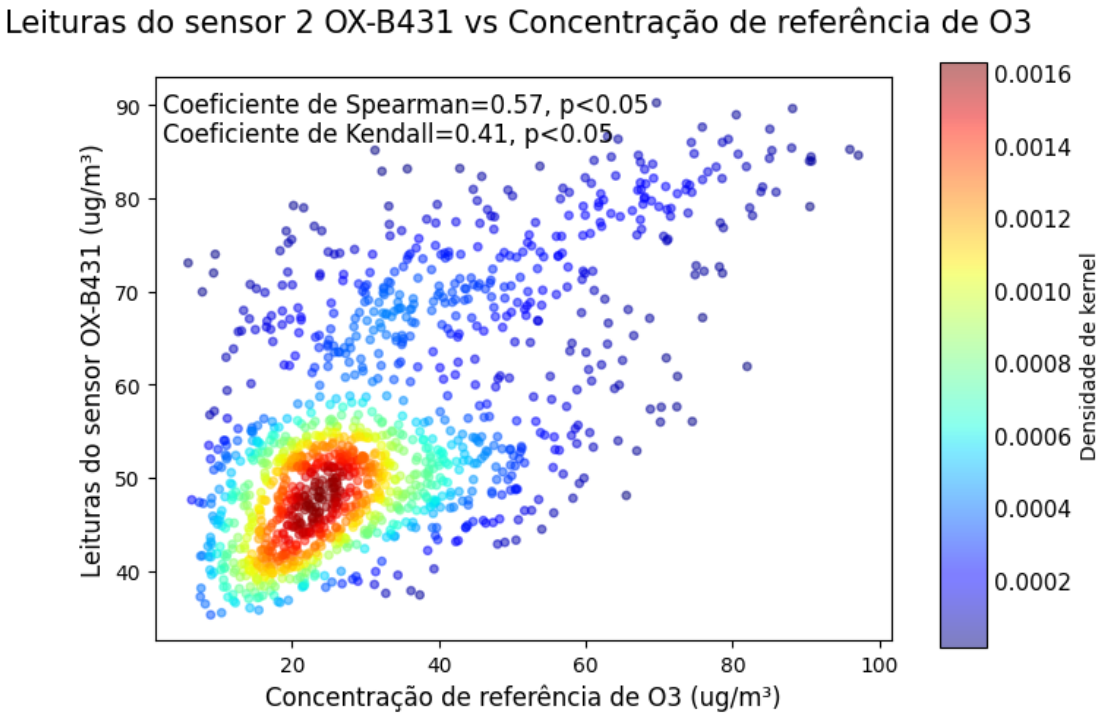
0.5.2 Sensor 2

[ ]:
```
from scipy.stats import kendalltau, spearmanr, gaussian_kde
import matplotlib.pyplot as plt
import numpy as np

median_reference = sensor_data['reference'].median()
median_measuring = sensor_data['measuring 2'].median()

fig, ax = plt.subplots(figsize=(1.3*5,5))
xy = np.vstack([sensor_data['reference'].fillna(value=median_reference),␣
 ↪sensor_data['measuring 2'].fillna(value=median_measuring)])
z = gaussian_kde(xy)(xy)

plt.scatter(sensor_data['reference'], sensor_data['measuring 2'], c=z,␣
 ↪cmap='jet', s=15, alpha=.5)

plt.title('Leituras do sensor 2 OX-B431 vs Concentração de referência de O3 \n',
          fontdict={'fontsize':15})
plt.xlabel('Concentração de referência de O3 (ug/m\N{SUPERSCRIPT THREE})',
```

```
           fontsize=12)
plt.ylabel('Leituras do sensor OX-B431 (ug/m\N{SUPERSCRIPT THREE})',
           fontsize=12)
coef, p = spearmanr(sensor_data['reference'], sensor_data['measuring 2'],
                    nan_policy='omit')

print('Spearmans correlation coefficient: %.2f' % coef)
# interpret the significance
alpha = 0.05
if p > alpha:
        print('Samples are uncorrelated (fail to reject H0) p=%.2f' % p)
else:
        print('Samples are correlated (reject H0) p=%.2f' % p)

kendall, pken = kendalltau(sensor_data['reference'], sensor_data['measuring 2'],
                           nan_policy='omit')
print('Kendall correlation coefficient: %.2f' % coef)
# interpret the significance
alpha = 0.05
if p > alpha:
        print('Samples are uncorrelated (fail to reject H0) p=%.2f' % p)
else:
        print('Samples are correlated (reject H0) p=%.2f' % p)

cax = plt.axes([0.95, 0.1, 0.05, 0.8])
cbar = plt.colorbar(orientation='vertical', cax=cax, label="Densidade de␣
 ↪kernel")
cbar.ax.tick_params(labelsize=11, length=0)

string = 'Coeficiente de Spearman=%.2f, p<0.05' % coef
plt.text(0.01, 0.95, string, horizontalalignment='left', fontsize=12,
         verticalalignment='center', transform=ax.transAxes)
string = 'Coeficiente de Kendall=%.2f, p<0.05' % kendall
plt.text(0.01, 0.90, string, horizontalalignment='left', fontsize=12,
         verticalalignment='center', transform=ax.transAxes)
```

```
Spearmans correlation coefficient: 0.57
Samples are correlated (reject H0) p=0.00
Kendall correlation coefficient: 0.57
Samples are correlated (reject H0) p=0.00
```

[ ]: Text(0.01, 0.9, 'Coeficiente de Kendall=0.41, p<0.05')



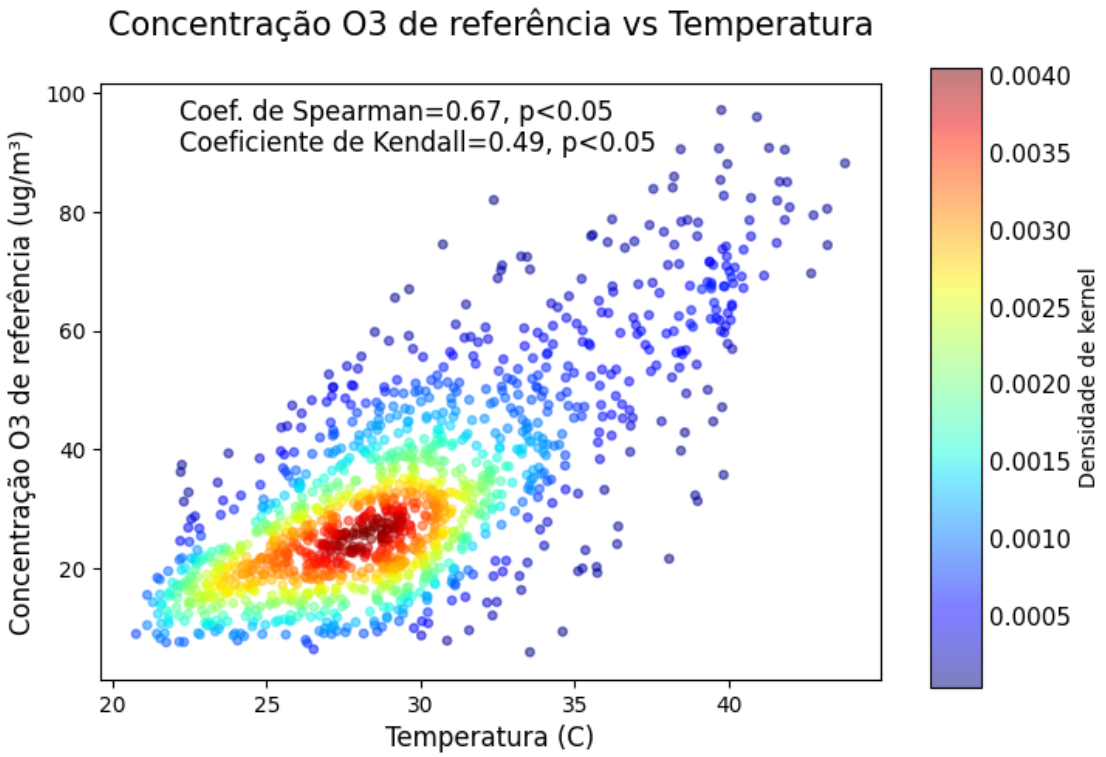
0.6 Plot reference vs. temperature

[ ]:
```
from scipy.stats import kendalltau, spearmanr, gaussian_kde
import matplotlib.pyplot as plt
import numpy as np

median_reference = sensor_data['reference'].median()
median_temperature = sensor_data['temperature'].median()

fig, ax = plt.subplots(figsize=(1.3*5,5))
xy = np.vstack([sensor_data['temperature'].fillna(value=median_temperature),␣
 ↪sensor_data['reference'].fillna(value=median_reference)])
z = gaussian_kde(xy)(xy)

plt.scatter(sensor_data['temperature'], sensor_data['reference'], c=z,␣
 ↪cmap='jet', s=15, alpha=.5)

plt.title('Concentração O3 de referência vs Temperatura\n',
          fontdict={'fontsize':15})
plt.xlabel('Temperatura (C)',
           fontsize=12)
plt.ylabel('Concentração O3 de referência (ug/m\N{SUPERSCRIPT THREE})',
           fontsize=12)
```

```
coef, p = spearmanr(sensor_data['temperature'], sensor_data['reference'],
                    nan_policy='omit')

print('Spearmans correlation coefficient: %.2f' % coef)
# interpret the significance
alpha = 0.05
if p > alpha:
        print('Samples are uncorrelated (fail to reject H0) p=%.2f' % p)
else:
        print('Samples are correlated (reject H0) p=%.2f' % p)

kendall, pken = kendalltau(sensor_data['temperature'], sensor_data['reference'],
                           nan_policy='omit')
print('Kendall correlation coefficient: %.2f' % coef)
# interpret the significance
alpha = 0.05
if p > alpha:
        print('Samples are uncorrelated (fail to reject H0) p=%.2f' % p)
else:
        print('Samples are correlated (reject H0) p=%.2f' % p)

cax = plt.axes([0.95, 0.1, 0.05, 0.8])
cbar = plt.colorbar(orientation='vertical', cax=cax, label="Densidade de␣
 ↪kernel")
cbar.ax.tick_params(labelsize=11, length=0)

string = 'Coef. de Spearman=%.2f, p<0.05' % coef
plt.text(0.1, 0.95, string, horizontalalignment='left', fontsize=12,
         verticalalignment='center', transform=ax.transAxes)
string = 'Coeficiente de Kendall=%.2f, p<0.05' % kendall
plt.text(0.1, 0.90, string, horizontalalignment='left', fontsize=12,
         verticalalignment='center', transform=ax.transAxes)
```

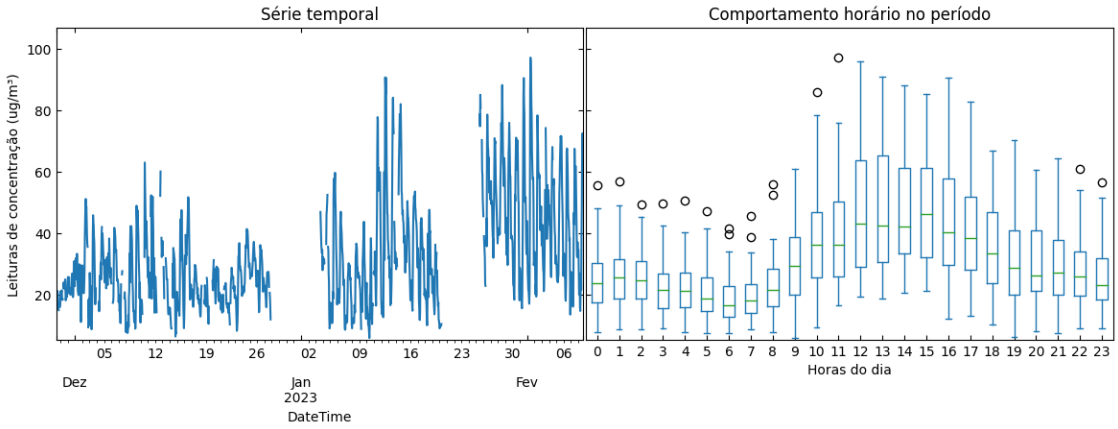
```
Spearmans correlation coefficient: 0.67
Samples are correlated (reject H0) p=0.00
Kendall correlation coefficient: 0.67
Samples are correlated (reject H0) p=0.00
```

[ ]: Text(0.1, 0.9, 'Coeficiente de Kendall=0.49, p<0.05')



[ ]:
```python
def plot_box(df):
    bottom, height = 0.1, 0.65
    left, width = bottom, height*1.3
    spacing = 0.005

    rect_ser = [left-width-spacing, bottom, width, height]
    rect_box = [left, bottom, width, height]

    plt.figure(figsize=(1.3*5,5))

    ax_ser  = plt.axes(rect_ser)
    ax_ser.tick_params(direction='in', top=True, right=True)
    ax_ser.set_title('Série temporal')
    ax_ser.set_xlabel("Data e hora")
    ax_ser.set_ylabel("Leituras de concentração (ug/m\N{SUPERSCRIPT THREE})")

    ax_box  = plt.axes(rect_box)
    ax_box.tick_params(direction='in', labelleft=False)

    lim_max = df['reference'].max()+df['reference'].max()*10/100
    lim_min = df['reference'].min()-df['reference'].min()*10/100
```

```
    df['reference'].plot(ax=ax_ser)
    ax_ser.set_ylim(lim_min, lim_max)

    df = df.dropna(axis='index', how='all', subset=['Hour'])
    df['Hour'] = df['Hour'].astype('int64')
    df.pivot(columns='Hour')['reference'].dropna(
            axis='columns', how='all').plot.box(
                ax=ax_box,title='Comportamento horário no período')
    ax_box.set_ylim(lim_min, lim_max)
    ax_box.set_xlabel("Horas do dia")

valid_dataframe = sensor_data.resample('H').mean()
valid_dataframe['Hour'] = valid_dataframe.index.hour
plot_box(df=valid_dataframe)
```



0.7 Calibrate data

0.7.1 Prepare training and test sets

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate

reference_median = sensor_data['reference'].median()
sensor_1_median = sensor_data['measuring 1'].median()
sensor_2_median = sensor_data['measuring 2'].median()
temperature_mean = sensor_data['temperature'].mean()
trend_1_median = sensor_data['measuring 1 no Temp'].median()
trend_2_median = sensor_data['measuring 2 no Temp'].median()

variables_names = ['measuring 1','measuring 2', 'temperature']

y = sensor_data['reference'].fillna(value=reference_median)
```

```
X = (sensor_data[variables_names].fillna(value={
                        'measuring 1': sensor_1_median,
                        'measuring 2': sensor_2_median,
                        'temperature': temperature_mean}).values.reshape(-1,3))

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

0.7.2 Grid search with different models and variables combinations

[ ]:
```
from itertools import combinations

def check_if_list_contains(list1, list2):
    return [element for element in list1 if element in list2]

indexes = []
reference_indexes = [0, 1]
num_variables = len(variables_names)
for num_combinations in list(range(num_variables)):
    contains_reference = False
    index_list = [list(index_tuple) for index_tuple in␣
 ↪list(combinations(list(range(num_variables)), r=num_combinations+1))]
    for sublist in index_list:
        contains_reference = check_if_list_contains(sublist, reference_indexes)
        if contains_reference:
            indexes.append(sublist)
            contains_reference = False

feature_subsets = { }
for index_list in indexes:
    key = ""
    new_index_list = []
    for index in index_list:
        if len(key) < 1:
            new_index_list.append(index)
            key = key + variables_names[index] + " | "
        elif not ('measuring 1' in key and 'measuring 1' in␣
 ↪variables_names[index]) and not ('measuring 2' in key and 'measuring 2' in␣
 ↪variables_names[index]):
            new_index_list.append(index)
            key = key + variables_names[index] + " | "
    feature_subsets[key] = new_index_list

feature_subsets
```

[ ]:
```
{'measuring 1 | ': [0],
 'measuring 2 | ': [1],
 'measuring 1 | measuring 2 | ': [0, 1],
```

```
 'measuring 1 | temperature | ': [0, 2],
 'measuring 2 | temperature | ': [1, 2],
 'measuring 1 | measuring 2 | temperature | ': [0, 1, 2]}
```

Function for plotting observations vs. predictions

[ ]:
```
import matplotlib.pyplot as plt
from scipy.stats import spearmanr, kendalltau, gaussian_kde
import numpy as np
import os

def plot_predictions_and_observations(X, y, r2, rmse, mae, file_name):
    fig, ax = plt.subplots(figsize=(1.3*5,5))
    xy = np.vstack([X, y])
    z = gaussian_kde(xy)(xy)
    ax.scatter(X, y, c=z,s=15,alpha=.5)
    spear_corr, p_value = spearmanr(y, X)
    spearman_text = ''
    alpha = 0.05
    if p_value > alpha:
        spearman_text = 'Coeficiente de Spearman: {:.2f}'.format(spear_corr) +␣
↪', p>0.05'
    else:
        spearman_text = 'Coeficiente de Spearman: {:.2f}'.format(spear_corr) +␣
↪', p<0.05'

    kendall_corr, p_value = kendalltau(y, X)
    alpha = 0.05
    kendall_text = ''
    if p_value > alpha:
        kendall_text = 'Coeficiente de Kendall: {:.2f}'.format(kendall_corr) +␣
↪', p>0.05'
    else:
        kendall_text = 'Coeficiente de Kendall: {:.2f}'.format(kendall_corr) +␣
↪', p<0.05'

    plt.text(0.02, 0.95, spearman_text, ha='left', va='center', transform=plt.
↪gca().transAxes, fontsize=12)
    plt.text(0.02, 0.90, kendall_text, ha='left', va='center', transform=plt.
↪gca().transAxes, fontsize=12)
    r2_text = 'R\N{SUPERSCRIPT TWO} = {:.2f} ± {:.2f}'.format(r2.mean(), r2.
↪std())
    rmse_text = 'RMSE = {:.2f} ± {:.2f}'.format(rmse.mean(), rmse.std())
    mae_text = 'MAE = {:.2f} ± {:.2f}'.format(mae.mean(), mae.std())
    plt.text(0.02, 0.85, r2_text, ha='left', va='center', transform=plt.gca().
↪transAxes, fontsize=12)
```

```
    plt.text(0.02, 0.80, rmse_text, ha='left', va='center', transform=plt.gca().
↪transAxes, fontsize=12)
    plt.text(0.02, 0.75, mae_text, ha='left', va='center', transform=plt.gca().
↪transAxes, fontsize=12)

    ax.set_xlim([np.min([y,X]),np.max([y,X])])
    ax.set_ylim([np.min([y,X]),np.max([y,X])])
    ax.set_aspect('equal')

    ax.plot([xy.min(), xy.max()], [xy.min(), xy.max()], 'k-', lw=1,dashes=[2,␣
↪2])
    ax.fill_between(np.linspace(xy.min(), xy.max(),y.shape[0]),
                    np.linspace(xy.min(), xy.max(),y.shape[0])*0.5,
                    alpha=0.2,facecolor='gray',edgecolor=None)
    ax.fill_between(np.linspace(xy.min(),xy.max(),y.shape[0]),
                    np.linspace(xy.max(),xy.max(),y.shape[0]),
                    np.linspace(xy.min(),xy.max(),y.shape[0])*2,
                    alpha=0.2,facecolor='gray',edgecolor=None)

    ax.set_xlabel('Concentração de O3 observada (ug/m\N{SUPERSCRIPT THREE})',␣
↪fontsize=12)
    ax.set_ylabel('Concentração de O3 inferida (ug/m\N{SUPERSCRIPT␣
↪THREE})',fontsize=12)

    if not os.path.exists('images/'):
        os.makedirs('images/')

    plt.savefig('images/' + 'O3_UNI_' + file_name + '.png')
```

[ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import  MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
import numpy as np

models = {
    'MLP Regression': (
        ('mlp_regressor', MLPRegressor(solver="lbfgs", max_iter=1000,␣
↪random_state=42)), {
            'mlp_regressor__hidden_layer_sizes': [
                (4,),   (10,),   (50,),   (100,),  (200,),
                (4,4),  (4,10),  (4,50),  (4,100), (4,200),
                (10,4), (10,10), (10,50), (10,100),(10,200),
```

```
                (50,4), (50,10), (50,50), (50,100),(50,200),
                (100,4),(100,10),(100,50),(100,100),(100,200),
                (200,4),(200,10),(200,50),(200,100),(200,200)],
            'mlp_regressor__alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10]
        }
    ),
    'Multilinear Regression': (
        ('linear_regressor', LinearRegression()), { }
    ),
    'KNN Regression': (
        ('knn_regressor', KNeighborsRegressor()), {
            'knn_regressor__n_neighbors': [3, 5, 7, 9, 11, 13, 15, 17, 20],
            'knn_regressor__weights': ['uniform', 'distance'],
            'knn_regressor__p': [1, 2]  # 1 for Manhattan distance, 2 for␣
↪Euclidean distance'
        }
    ),
    'Random Forests Regression': (
        ('random_forest_regressor', RandomForestRegressor()), {
            'random_forest_regressor__n_estimators': [50, 100, 150],
            'random_forest_regressor__max_depth': [None, 10, 20, 30],
            'random_forest_regressor__min_samples_split': [2, 5, 10],
            'random_forest_regressor__min_samples_leaf': [1, 2, 4]
        }
    )
}

# Perform grid search for each feature subset
results = {}
rmse_by_features = {}
r2_by_features = {}
mae_by_features = {}
for features_set, subset in feature_subsets.items():
    X_subset = X[:, subset]
    X_train_subset = X_train[:, subset]
    X_test_subset = X_test[:, subset]

    model_results = {}
    model_rmse = {}
    model_r2 = {}
    model_mae = {}
    for model_name, (model, param_grid) in models.items():
        print(f"Grid search for features: {features_set} with model:␣
↪{model_name}...")

        pipeline = Pipeline([
            ('scaler', StandardScaler()),
```

```
            model
        ])

        # Perform grid search with cross-validation
        grid_search = GridSearchCV(pipeline, param_grid, cv=3,␣
↪scoring='neg_root_mean_squared_error', n_jobs=-1)
        grid_search.fit(X_train_subset, y_train)

        # Print the best parameters and best score
        best_params = grid_search.best_params_

        # Evaluate the best model on the test set
        best_model = grid_search.best_estimator_
        cross_validation = cross_validate(best_model, X_subset, y, cv=3,␣
↪scoring=['r2', 'neg_root_mean_squared_error', 'neg_mean_absolute_error'])
        y_pred = best_model.predict(X_test_subset)

        # Evaluate the model
        r2 = cross_validation['test_r2']
        rmse = cross_validation['test_neg_root_mean_squared_error']
        mae = cross_validation['test_neg_mean_absolute_error']

        plot_predictions_and_observations(y_test, y_pred, r2=r2, rmse=rmse,␣
↪mae=mae, file_name=model_name+features_set)

        model_results[model_name] = {
            'Best Model': best_model,
            'Best Parameters': best_params,
            'Test R2': r2,
            'Test RMSE': rmse,
            'Test MAE': mae
        }
        model_rmse[model_name] = {
            'Mean': rmse.mean(),
            'Std': rmse.std()
        }
        model_r2[model_name] = {
            'Mean': r2.mean(),
            'Std': r2.std()
        }
        model_mae[model_name] = {
            'Mean': mae.mean(),
            'Std': mae.std()
        }

    results[features_set] = model_results
    rmse_by_features[features_set] = model_rmse
```

```
    r2_by_features[features_set] = model_r2
    mae_by_features[features_set] = model_mae

for feature_set, models in results.items():
    for model_name, result in models.items():
        print(f"\nResults for features: {feature_set} with model: {model_name}:
 ↪")
        print(f"Best Parameters: {result['Best Parameters']}")
        print(f"Test RMSE: {result['Test RMSE'].mean()} +/- {result['Test␣
 ↪RMSE'].std()}")
        print(f"Test R2: {result['Test R2'].mean()} +/- {result['Test R2'].
 ↪std()}")
        print(f"Test MAE: {result['Test MAE'].mean()} +/- {result['Test MAE'].
 ↪std()}")
```

1 Save Results

[ ]:
```
output_directory_path = 'output/'
rmse_file_name = output_directory_path + sensor_name + '_rmse.csv'
r2_file_name = output_directory_path + sensor_name + '_r2.csv'
mae_file_name = output_directory_path + sensor_name + '_mae.csv'
results_file_name = output_directory_path + sensor_name + '_results.csv'

pd.DataFrame(rmse_by_features).transpose().to_csv(rmse_file_name)
pd.DataFrame(r2_by_features).transpose().to_csv(r2_file_name)
pd.DataFrame(mae_by_features).transpose().to_csv(mae_file_name)
pd.DataFrame(results).transpose().to_csv(results_file_name)
```

1.1 Plot Results

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

def plot_metrics(features, r2_list, r2_error_list, rmse_list, rmse_error_list,␣
 ↪mae_list, mae_error_list):
    bottom, height = 0.1, 0.65
    left, width = bottom, height*1.3
    spacing = 0.03

    rect_r2 = [left-width-spacing, bottom, width, height]
    rect_rmse = [left, bottom, width, height]
    rect_mae = [left + width + spacing, bottom, height/1.3, height]

    plt.figure(figsize=(1.3*5,8))

    ax_r2  = plt.axes(rect_r2)
```

```
    ax_r2.tick_params(direction='in', top=True, right=True, labelsize=14)
    ax_r2.set_title('R2')

    ax_rmse  = plt.axes(rect_rmse)
    ax_rmse.tick_params(direction='in', labelleft=False, labelsize=14)
    ax_rmse.set_title('RMSE')

    ax_mae = plt.axes(rect_mae)
    ax_mae.tick_params(direction='in', labelleft=False, labelsize=14)
    ax_mae.set_title('MAE')

    y_pos = np.arange(len(features))

    ax_r2.barh(y_pos, r2_list, xerr=r2_error_list, align='center')
    min_r2 = r2_list.min() - r2_error_list.max()
    ax_r2.set_xlim([min_r2 - 0.05, 1.0 + 0.05])
    ax_r2.set_yticks(y_pos, labels=features, fontsize=14)
    ax_r2.invert_yaxis()  # labels read top-to-bottom
    ax_r2.set_xlabel('R2', fontsize=14)

    ax_rmse.barh(y_pos, rmse_list, xerr=rmse_error_list, align='center')
    max_rmse = rmse_list.max() + rmse_error_list.max()
    min_rmse = rmse_list.min() - rmse_error_list.max()
    if max_rmse <= 0: max_rmse = -min_rmse
    ax_rmse.set_xlim([min_rmse - 0.05, max_rmse + 0.05])
    ax_rmse.set_yticks(y_pos, labels=features, fontsize=14)
    ax_rmse.invert_yaxis()  # labels read top-to-bottom
    ax_rmse.set_xlabel('RMSE', fontsize=14)

    ax_mae.barh(y_pos, mae_list, xerr=mae_error_list, align='center')
    max_mae = mae_list.max() + mae_error_list.max()
    min_mae = mae_list.min() - mae_error_list.max()
    if max_mae <= 0: max_mae = -min_mae
    ax_mae.set_xlim([min_mae - 0.05, max_mae + 0.05])
    ax_mae.set_yticks(y_pos, labels=features, fontsize=14)
    ax_mae.invert_yaxis()  # labels read top-to-bottom
    ax_mae.set_xlabel('MAE', fontsize=14)
```

[ ]:
```
mean_r2_by_features_dataframe = pd.DataFrame()
std_r2_by_features_dataframe = pd.DataFrame()

mean_rmse_by_features_dataframe = pd.DataFrame()
std_rmse_by_features_dataframe = pd.DataFrame()

mean_mae_by_features_dataframe = pd.DataFrame()
std_mae_by_features_dataframe = pd.DataFrame()
```

```
for key in list(feature_subsets.keys()):
    feature_dict = r2_by_features[key]
    for model in list(feature_dict.keys()):
        colum_name = key.replace('measuring', 'O3')
        colum_name = colum_name.replace(' |', ',')
        colum_name += f': {model[:-11]}'
        mean_r2_by_features_dataframe[colum_name] =␣
 ↪[feature_dict[model]['Mean']]
        std_r2_by_features_dataframe[colum_name] = [feature_dict[model]['Std']]

for key in list(feature_subsets.keys()):
    feature_dict = rmse_by_features[key]
    for model in list(feature_dict.keys()):
        colum_name = key.replace('measuring', 'O3')
        colum_name = colum_name.replace(' |', ',')
        colum_name += f': {model[:-11]}'
        mean_rmse_by_features_dataframe[colum_name] =␣
 ↪[feature_dict[model]['Mean']]
        std_rmse_by_features_dataframe[colum_name] =␣
 ↪[feature_dict[model]['Std']]

for key in list(feature_subsets.keys()):
    feature_dict = mae_by_features[key]
    for model in list(feature_dict.keys()):
        colum_name = key.replace('measuring', 'O3')
        colum_name = colum_name.replace(' |', ',')
        colum_name += f': {model[:-11]}'
        mean_mae_by_features_dataframe[colum_name] =␣
 ↪[feature_dict[model]['Mean']]
        std_mae_by_features_dataframe[colum_name] = [feature_dict[model]['Std']]
```

[ ]:
```
r2_sorted_dataframe = (mean_r2_by_features_dataframe.
 ↪sort_values(by=mean_r2_by_features_dataframe.index[0], axis=1,␣
 ↪ascending=False))
features = r2_sorted_dataframe.columns

mean_r2 = r2_sorted_dataframe.values.flatten()
error_r2 = std_r2_by_features_dataframe[r2_sorted_dataframe.columns].values.
 ↪flatten()

mean_rmse = mean_rmse_by_features_dataframe[r2_sorted_dataframe.columns].values.
 ↪flatten()
error_rmse = std_rmse_by_features_dataframe[r2_sorted_dataframe.columns].values.
 ↪flatten()
```

```
mean_mae = mean_mae_by_features_dataframe[r2_sorted_dataframe.columns].values.
 ↪flatten()
error_mae = std_mae_by_features_dataframe[r2_sorted_dataframe.columns].values.
 ↪flatten()

plot_metrics(features, r2_list=mean_r2, r2_error_list=error_r2,␣
 ↪rmse_list=mean_rmse,
             rmse_error_list=error_rmse, mae_list=mean_mae,␣
 ↪mae_error_list=error_mae)
```