O3_reference_and_all_sensor_data_with_temp

February 14, 2024

0.0.1 Sensor Constants

[ ]:
```
sensor_name = 'o3_all_sensors'
sensor_co_name = 'alpha_co_conc'
sensor_no2_name = 'alpha_no2_conc'
sensor_o3_1_name = 'alpha_o3_1_conc'
sensor_o3_2_name = 'alpha_o3_2_conc'
sensor_so2_1_name = 'alpha_so2_1_conc'
sensor_so2_2_name = 'alpha_so2_2_conc'
sensor_pm_10_name = 'alpha_pm_10_conc'
```

0.1 Upload Data from File

0.1.1 CO Sensor

[ ]:
```
import pandas as pd

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

[ ]:
```
              DateTime  measuring  Hour  temperature  measuring no Temp  \
0  2022-11-27 17:30:00   0.095855    17     29.78500           0.159694
1  2022-11-27 18:30:00   0.091372    18     30.13125           0.153195
2  2022-11-27 19:30:00   0.088210    19     30.09375           0.150251
3  2022-11-27 20:30:00   0.087858    20     30.03750           0.150226
4  2022-11-27 21:30:00   0.090610    21     29.96875           0.153379

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

0.1.2 Create Sensor Dataframe as Pandas Series

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

# Resample data with 15 mins period and create sensor dataframe
sensor_co_dataframe = dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_co_dataframe.index = sensor_co_dataframe['DateTime']
sensor_co_dataframe = sensor_co_dataframe.drop(columns=['DateTime', 'Hour',␣
 ↪'Count', 'Tag'])
sensor_co_dataframe = sensor_co_dataframe.rename(columns={'measuring':␣
 ↪'measuring CO', 'measuring no Temp': 'measuring no Temp CO', 'temperature':␣
 ↪'temperature CO'})
sensor_co_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_8846/554760737.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_co['DateTime'],
infer_datetime_format=True))
```

[ ]:

DateTime	measuring CO	temperature CO	measuring no Temp CO
2022-11-27 17:30:00	0.095855	29.78500	0.159694
2022-11-27 18:30:00	0.091372	30.13125	0.153195
2022-11-27 19:30:00	0.088210	30.09375	0.150251
2022-11-27 20:30:00	0.087858	30.03750	0.150226
2022-11-27 21:30:00	0.090610	29.96875	0.153379
…	…	…	…
2023-02-04 03:30:00	0.030431	27.14750	0.109628
2023-02-04 04:30:00	0.028936	27.26000	0.107478
2023-02-04 05:30:00	0.033254	27.31750	0.111460
2023-02-04 06:30:00	0.044496	27.24875	0.123103
2023-02-04 07:30:00	0.050885	27.93250	0.125510

[1010 rows x 3 columns]

0.1.3 NO2 Sensor

[ ]:
```
import pandas as pd

directory_path = 'input/'
file_name = sensor_no2_name + '_and_temp_valid_1HR.csv'
```

```
df_no2 = pd.read_csv(directory_path + file_name)
df_no2.head()
```

[ ]:

	DateTime	measuring	Hour	temperature	Count	Tag
0	2022-11-27 17:30:00	182.373362	17	29.78500	3	VALID
1	2022-11-27 18:30:00	188.127215	18	30.13125	4	VALID
2	2022-11-27 19:30:00	175.393318	19	30.09375	4	VALID
3	2022-11-27 20:30:00	185.269497	20	30.03750	4	VALID
4	2022-11-27 21:30:00	179.436459	21	29.96875	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_no2.drop(df_no2.columns[0], axis='columns')
dataframe['DateTime'] = (pd.to_datetime(df_no2['DateTime'],␣
 ↪infer_datetime_format=True))

# Resample data with 15 mins period and create sensor dataframe
sensor_no2_dataframe = dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_no2_dataframe.index = sensor_no2_dataframe['DateTime']
sensor_no2_dataframe = sensor_no2_dataframe.drop(columns=['DateTime', 'Hour',␣
 ↪'Count', 'Tag'])
sensor_no2_dataframe = sensor_no2_dataframe.rename(columns={'measuring':␣
 ↪'measuring NO2', 'measuring no Temp': 'measuring no Temp NO2', 'temperature':
 ↪ 'temperature NO2'})
sensor_no2_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_8846/4266076299.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_no2['DateTime'],
infer_datetime_format=True))
```

[ ]:

DateTime	measuring NO2	temperature NO2
2022-11-27 17:30:00	182.373362	29.785000
2022-11-27 18:30:00	188.127215	30.131250
2022-11-27 19:30:00	175.393318	30.093750
2022-11-27 20:30:00	185.269497	30.037500
2022-11-27 21:30:00	179.436459	29.968750
…	…	…
2022-12-23 16:30:00	34.320839	31.335000
2022-12-23 18:30:00	51.619244	29.111667
2022-12-23 19:30:00	153.853416	26.977500

```
2022-12-23 20:30:00     164.976172         25.810000
2022-12-23 21:30:00     124.629429         25.195000

[278 rows x 2 columns]
```

0.1.5 O3

Sensor 1

[ ]:
```
import pandas as pd

directory_path = 'input/'
file_name = sensor_o3_1_name + '_and_temp_valid_1HR.csv'
df_o3_1 = pd.read_csv(directory_path + file_name)
df_o3_1.head()
```

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

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

0.1.6 Create Sensor Dataframe as Pandas Series

[ ]:
```
# Remove the first column with the indexes and save data into web dataframe
dataframe = df_o3_1.drop(df_o3_1.columns[0], axis='columns')
dataframe['DateTime'] = (pd.to_datetime(df_o3_1['DateTime'],␣
 ↪infer_datetime_format=True))

# Resample data with 15 mins period and create sensor dataframe
sensor_o3_1_dataframe = dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_o3_1_dataframe.index = sensor_o3_1_dataframe['DateTime']
sensor_o3_1_dataframe = sensor_o3_1_dataframe.drop(columns=['DateTime', 'Hour',␣
 ↪'Count 1', 'Tag'])
sensor_o3_1_dataframe = sensor_o3_1_dataframe.rename(columns={'measuring 1':␣
 ↪'measuring O3 1', 'measuring 1 no Temp': 'measuring no Temp O3 1',
                                                    'temperature':␣
 ↪'temperature O3 1'})
sensor_o3_1_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_8846/4189083599.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_o3_1['DateTime'],
infer_datetime_format=True))
```

[ ]:

DateTime	measuring O3 1	temperature O3 1	measuring no Temp O3 1
2022-12-14 14:30:00	42.267696	31.52000	56.369783
2022-12-14 15:30:00	50.822340	30.56750	69.328964
2022-12-14 16:30:00	67.516902	28.82875	94.063829
2022-12-14 17:30:00	68.069052	27.91125	98.858670
2022-12-14 18:30:00	84.294900	27.16250	118.546877
…	…	…	…
2023-04-19 16:30:00	50.962218	28.84750	77.422442
2023-04-19 17:30:00	56.130342	27.83625	87.266774
2023-04-19 18:30:00	40.233330	25.86125	80.502529
2023-04-19 19:30:00	51.222342	23.64875	101.722552
2023-04-20 20:30:00	38.675040	22.74000	93.377479

[1021 rows x 3 columns]

0.1.7 Sensor 2

[ ]:
```
import pandas as pd

directory_path = 'input/'
file_name = sensor_o3_2_name + '_and_temp_valid_1HR.csv'
df_o3_2 = pd.read_csv(directory_path + file_name)
df_o3_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

0.1.8 Create Sensor Dataframe as Pandas Series

[ ]:
```
# Remove the first column with the indexes and save data into web dataframe
dataframe = df_o3_2.drop(df_o3_2.columns[0], axis='columns')
dataframe['DateTime'] = (pd.to_datetime(df_o3_2['DateTime'],␣
 ↪infer_datetime_format=True))

# Resample data with 15 mins period and create sensor dataframe
sensor_o3_2_dataframe = dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_o3_2_dataframe.index = sensor_o3_2_dataframe['DateTime']
sensor_o3_2_dataframe = sensor_o3_2_dataframe.drop(columns=['DateTime', 'Hour',␣
 ↪'Count 2', 'Tag'])
sensor_o3_2_dataframe = sensor_o3_2_dataframe.rename(columns={'measuring 2':␣
 ↪'measuring O3 2', 'measuring 2 no Temp': 'measuring no Temp O3 2',
                                                    'temperature':␣
 ↪'temperature O3 2'})
sensor_o3_2_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_8846/2290086274.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_o3_2['DateTime'],
infer_datetime_format=True))
```

[ ]:

DateTime	measuring O3 2	temperature O3 2	measuring no Temp O3 2
2022-11-28 11:30:00	53.759778	30.10750	11.239952
2022-11-28 12:30:00	53.445666	29.88250	11.476654
2022-11-28 13:30:00	54.100884	30.24125	11.253630
2022-11-28 14:30:00	53.921742	30.13250	11.340715
2022-11-28 15:30:00	53.494746	29.89875	11.485953
…	…	…	…
2023-04-21 17:30:00	50.964672	30.53625	7.395241
2023-04-21 18:30:00	42.360948	27.13750	7.111857
2023-04-21 19:30:00	39.303264	25.32750	8.485160
2023-04-21 20:30:00	37.688532	24.45875	8.997179
2023-04-21 21:30:00	37.048038	23.90625	9.709237

[2603 rows x 3 columns]

0.1.9 SO2

Sensor 1

[ ]:
```
import pandas as pd
```

```
directory_path = 'input/'
file_name = sensor_so2_1_name + '_and_temp_valid_1HR.csv'
df_so2_1 = pd.read_csv(directory_path + file_name)
df_so2_1.head()
```

[ ]:

	DateTime	measuring 1	temperature	Hour	measuring 1 no Temp	\
0	2022-12-07 19:30:00	2334.044792	27.39125	19	2603.851233	
1	2022-12-08 13:30:00	408.498126	34.00500	13	1809.151682	
2	2022-12-10 08:30:00	1925.520463	32.23500	8	3023.531889	
3	2022-12-13 20:30:00	3721.797934	24.45875	20	3490.193048	
4	2022-12-13 21:30:00	4488.438582	23.87375	21	4156.807906	

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

0.1.10 Create Sensor Dataframe as Pandas Series

[ ]:
```
# Remove the first column with the indexes and save data into web dataframe
dataframe = df_so2_1.drop(df_so2_1.columns[0], axis='columns')
dataframe['DateTime'] = (pd.to_datetime(df_so2_1['DateTime'],␣
 ↪infer_datetime_format=True))

# Resample data with 15 mins period and create sensor dataframe
sensor_so2_1_dataframe = dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_so2_1_dataframe.index = sensor_so2_1_dataframe['DateTime']
sensor_so2_1_dataframe = sensor_so2_1_dataframe.drop(columns=['DateTime',␣
 ↪'Hour', 'Count 1', 'Tag'])
sensor_so2_1_dataframe = sensor_so2_1_dataframe.rename(columns={'measuring 1':␣
 ↪'measuring SO2 1', 'measuring 1 no Temp': 'measuring no Temp SO2 1',
                                                    'temperature':␣
 ↪'temperature SO2 1'})
sensor_so2_1_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_8846/3688823847.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_so2_1['DateTime'],
infer_datetime_format=True))
```

[ ]:
```
                     measuring SO2 1  temperature SO2 1  \
DateTime
2022-12-07 19:30:00      2334.044792           27.39125
2022-12-08 13:30:00       408.498126           34.00500
2022-12-10 08:30:00      1925.520463           32.23500
2022-12-13 20:30:00      3721.797934           24.45875
2022-12-13 21:30:00      4488.438582           23.87375
…                                 …                  …
2023-04-16 01:30:00      3447.282268           26.69000
2023-04-16 02:30:00      3646.048354           26.38125
2023-04-16 03:30:00      4043.577251           25.72250
2023-04-16 09:30:00      2506.453942           30.98125
2023-04-16 10:30:00      1947.160861           32.96375

                     measuring no Temp SO2 1
DateTime
2022-12-07 19:30:00              2603.851233
2022-12-08 13:30:00              1809.151682
2022-12-10 08:30:00              3023.531889
2022-12-13 20:30:00              3490.193048
2022-12-13 21:30:00              4156.807906
…                                        …
2023-04-16 01:30:00              3597.186000
2023-04-16 02:30:00              3743.160697
2023-04-16 03:30:00              4028.053717
2023-04-16 09:30:00              3390.093858
2023-04-16 10:30:00              3169.777062

[570 rows x 3 columns]
```

0.1.11 Sensor 2

[ ]:
```
import pandas as pd

directory_path = 'input/'
file_name = sensor_so2_2_name + '_and_temp_valid_1HR.csv'
df_so2_2 = pd.read_csv(directory_path + file_name)
df_so2_2.head()
```

[ ]:
```
              DateTime  measuring 2  temperature  Hour  measuring 2 no Temp  \
0  2022-12-01 21:30:00   203.905149     29.25375    21           155.913165
1  2022-12-01 22:30:00   205.143240     29.23250    22           156.961135
2  2022-12-01 23:30:00   205.382342     29.23250    23           157.200237
3  2022-12-02 00:30:00   205.765561     29.26375     0           157.863047
4  2022-12-02 01:30:00   205.179269     29.28125     1           157.433325

   Count 2    Tag
```

```
0        4  VALID
1        4  VALID
2        4  VALID
3        4  VALID
4        4  VALID
```

0.1.12 Create Sensor Dataframe as Pandas Series

[ ]:
```
# Remove the first column with the indexes and save data into web dataframe
dataframe = df_so2_2.drop(df_so2_2.columns[0], axis='columns')
dataframe['DateTime'] = (pd.to_datetime(df_so2_2['DateTime'],␣
 ↪infer_datetime_format=True))

# Resample data with 15 mins period and create sensor dataframe
sensor_so2_2_dataframe = dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_so2_2_dataframe.index = sensor_so2_2_dataframe['DateTime']
sensor_so2_2_dataframe = sensor_so2_2_dataframe.drop(columns=['DateTime',␣
 ↪'Hour', 'Count 2', 'Tag'])
sensor_so2_2_dataframe = sensor_so2_2_dataframe.rename(columns={'measuring 2':␣
 ↪'measuring SO2 2', 'measuring 2 no Temp': 'measuring no Temp SO2 2',
                                                    'temperature':␣
 ↪'temperature SO2 2'})
sensor_so2_2_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_8846/377440947.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_so2_2['DateTime'],
infer_datetime_format=True))
```

[ ]:
```
                     measuring SO2 2  temperature SO2 2  \
DateTime
2022-12-01 21:30:00       203.905149           29.25375
2022-12-01 22:30:00       205.143240           29.23250
2022-12-01 23:30:00       205.382342           29.23250
2022-12-02 00:30:00       205.765561           29.26375
2022-12-02 01:30:00       205.179269           29.28125
…                                …                  …
2023-03-31 13:30:00        85.397741           41.20625
2023-03-31 15:30:00        99.663088           41.13750
2023-03-31 16:30:00       109.607124           39.82500
2023-03-31 17:30:00       137.731671           36.72125
2023-03-31 19:30:00       213.030341           30.76375
```

```
                     measuring no Temp SO2 2
DateTime
2022-12-01 21:30:00                155.913165
2022-12-01 22:30:00                156.961135
2022-12-01 23:30:00                157.200237
2022-12-02 00:30:00                157.863047
2022-12-02 01:30:00                157.433325
…                                         …
2023-03-31 13:30:00                143.870195
2023-03-31 15:30:00                157.993882
2023-03-31 16:30:00                156.195106
2023-03-31 17:30:00                160.222656
2023-03-31 19:30:00                178.548184

[2034 rows x 3 columns]
```

0.1.13 PM Sensor

[ ]:
```
import pandas as pd

directory_path = 'input/'
file_name = sensor_pm_10_name + '_and_temp_valid_1HR.csv'
df_pm_10 = pd.read_csv(directory_path + file_name)
df_pm_10.head()
```

[ ]:
```
              DateTime  measuring  Hour  temperature  measuring no Temp  \
0  2022-11-23 13:30:00   2.588333    13    28.635000           3.261152
1  2022-11-23 15:30:00   1.341250    15    26.161250           2.150700
2  2022-11-23 16:30:00   1.335000    16    26.441250           2.128985
3  2022-11-23 18:30:00   2.338333    18    28.901667           2.996423
4  2022-11-23 20:30:00   1.025000    20    29.287500           1.661779

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

0.1.14 Create Sensor Dataframe as Pandas Series

[ ]:
```
# Remove the first column with the indexes and save data into web dataframe
dataframe = df_pm_10.drop(df_pm_10.columns[0], axis='columns')
dataframe['DateTime'] = (pd.to_datetime(df_pm_10['DateTime'],␣
 ↪infer_datetime_format=True))

# Resample data with 15 mins period and create sensor dataframe
```

```
sensor_pm_10_dataframe = dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_pm_10_dataframe.index = sensor_pm_10_dataframe['DateTime']
sensor_pm_10_dataframe = sensor_pm_10_dataframe.drop(columns=['DateTime',␣
 ↪'Hour', 'Count', 'Tag'])
sensor_pm_10_dataframe = sensor_pm_10_dataframe.rename(columns={'measuring':␣
 ↪'measuring PM10', 'measuring no Temp': 'measuring no Temp PM10',␣
 ↪'temperature': 'temperature PM10'})
sensor_pm_10_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_8846/2521444390.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_pm_10['DateTime'],
infer_datetime_format=True))
```

[ ]:

DateTime	measuring PM10	temperature PM10	measuring no Temp PM10
2022-11-23 13:30:00	2.588333	28.635000	3.261152
2022-11-23 15:30:00	1.341250	26.161250	2.150700
2022-11-23 16:30:00	1.335000	26.441250	2.128985
2022-11-23 18:30:00	2.338333	28.901667	2.996423
2022-11-23 20:30:00	1.025000	29.287500	1.661779
…	…	…	…
2023-04-19 19:30:00	0.375000	23.648750	1.323222
2023-04-20 14:30:00	0.698750	35.017500	1.019047
2023-04-20 15:30:00	1.735000	35.582500	2.024091
2023-04-20 16:30:00	2.346250	34.355000	2.703139
2023-04-20 17:30:00	2.101667	32.158333	2.579883

[1229 rows x 3 columns]

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_8846/2986665073.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

def merge_temperatures(df):
    # df[0]: CO, df[1]: NO2, df[2]: O31
    # df[3]: O32, df[4]: SO21, df[5]: SO22
    if not np.isnan(df[0]): return df[0]
    elif not np.isnan(df[1]): return df[1]
    elif not np.isnan(df[2]): return df[2]
    elif not np.isnan(df[3]): return df[3]
    elif not np.isnan(df[4]): return df[4]
    elif not np.isnan(df[5]): return df[5]
    return df[6]

sensor_data = pd.concat([sensor_co_dataframe, sensor_no2_dataframe,
```

```
                        sensor_o3_1_dataframe, sensor_o3_2_dataframe,
                        sensor_so2_1_dataframe, sensor_so2_2_dataframe,
                        sensor_pm_10_dataframe], join='outer', axis=1)

sensor_data['temperature'] = (sensor_data[['temperature CO', 'temperature␣
 ↪NO2','temperature O3 1',
                                        'temperature O3 2','temperature SO2 1',␣
 ↪'temperature SO2 2', 'temperature PM10']]
                                        .apply(lambda df:␣
 ↪merge_temperatures(df), axis=1))
sensor_data = sensor_data.drop(columns=['temperature CO', 'temperature␣
 ↪NO2','temperature O3 1',
                                        'temperature O3 2','temperature SO2 1',␣
 ↪'temperature SO2 2', 'temperature PM10'])
sensor_data
```

[ ]:

DateTime	measuring CO	measuring no Temp CO	measuring NO2
2022-11-23 13:30:00	NaN	NaN	NaN
2022-11-23 15:30:00	NaN	NaN	NaN
2022-11-23 16:30:00	NaN	NaN	NaN
2022-11-23 18:30:00	NaN	NaN	NaN
2022-11-23 20:30:00	NaN	NaN	NaN
…	…	…	…
2023-04-21 17:30:00	NaN	NaN	NaN
2023-04-21 18:30:00	NaN	NaN	NaN
2023-04-21 19:30:00	NaN	NaN	NaN
2023-04-21 20:30:00	NaN	NaN	NaN
2023-04-21 21:30:00	NaN	NaN	NaN

DateTime	measuring O3 1	measuring no Temp O3 1	measuring O3 2
2022-11-23 13:30:00	NaN	NaN	NaN
2022-11-23 15:30:00	NaN	NaN	NaN
2022-11-23 16:30:00	NaN	NaN	NaN
2022-11-23 18:30:00	NaN	NaN	NaN
2022-11-23 20:30:00	NaN	NaN	NaN
…	…	…	…
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

DateTime	measuring no Temp O3 2	measuring SO2 1
2022-11-23 13:30:00	NaN	NaN

```
2022-11-23 15:30:00                    NaN              NaN
2022-11-23 16:30:00                    NaN              NaN
2022-11-23 18:30:00                    NaN              NaN
2022-11-23 20:30:00                    NaN              NaN
…                                      …                …
2023-04-21 17:30:00               7.395241              NaN
2023-04-21 18:30:00               7.111857              NaN
2023-04-21 19:30:00               8.485160              NaN
2023-04-21 20:30:00               8.997179              NaN
2023-04-21 21:30:00               9.709237              NaN

                    measuring no Temp SO2 1  measuring SO2 2  \
DateTime
2022-11-23 13:30:00                    NaN              NaN
2022-11-23 15:30:00                    NaN              NaN
2022-11-23 16:30:00                    NaN              NaN
2022-11-23 18:30:00                    NaN              NaN
2022-11-23 20:30:00                    NaN              NaN
…                                      …                …
2023-04-21 17:30:00                    NaN              NaN
2023-04-21 18:30:00                    NaN              NaN
2023-04-21 19:30:00                    NaN              NaN
2023-04-21 20:30:00                    NaN              NaN
2023-04-21 21:30:00                    NaN              NaN

                    measuring no Temp SO2 2  measuring PM10  \
DateTime
2022-11-23 13:30:00                    NaN        2.588333
2022-11-23 15:30:00                    NaN        1.341250
2022-11-23 16:30:00                    NaN        1.335000
2022-11-23 18:30:00                    NaN        2.338333
2022-11-23 20:30:00                    NaN        1.025000
…                                      …                …
2023-04-21 17:30:00                    NaN              NaN
2023-04-21 18:30:00                    NaN              NaN
2023-04-21 19:30:00                    NaN              NaN
2023-04-21 20:30:00                    NaN              NaN
2023-04-21 21:30:00                    NaN              NaN

                    measuring no Temp PM10  temperature
DateTime
2022-11-23 13:30:00               3.261152    28.635000
2022-11-23 15:30:00               2.150700    26.161250
2022-11-23 16:30:00               2.128985    26.441250
2022-11-23 18:30:00               2.996423    28.901667
2022-11-23 20:30:00               1.661779    29.287500
…                                      …           …
```

```
2023-04-21 17:30:00                    NaN    30.536250
2023-04-21 18:30:00                    NaN    27.137500
2023-04-21 19:30:00                    NaN    25.327500
2023-04-21 20:30:00                    NaN    24.458750
2023-04-21 21:30:00                    NaN    23.906250

[2711 rows x 14 columns]
```

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

[ ]:
```
                     measuring CO  measuring no Temp CO  measuring NO2  \
DateTime
2022-11-23 13:30:00           NaN                   NaN            NaN
2022-11-23 15:30:00           NaN                   NaN            NaN
2022-11-23 16:30:00           NaN                   NaN            NaN
2022-11-23 18:30:00           NaN                   NaN            NaN
2022-11-23 20:30:00           NaN                   NaN            NaN
…                             …                     …              …
2023-02-08 12:30:00           NaN                   NaN            NaN
2023-02-08 13:30:00           NaN                   NaN            NaN
2023-02-08 14:30:00           NaN                   NaN            NaN
2023-02-08 15:30:00           NaN                   NaN            NaN
2023-02-08 16:30:00           NaN                   NaN            NaN

                     measuring O3 1  measuring no Temp O3 1  measuring O3 2  \
DateTime
2022-11-23 13:30:00             NaN                     NaN             NaN
2022-11-23 15:30:00             NaN                     NaN             NaN
2022-11-23 16:30:00             NaN                     NaN             NaN
2022-11-23 18:30:00             NaN                     NaN             NaN
2022-11-23 20:30:00             NaN                     NaN             NaN
…                               …                       …               …
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 no Temp O3 2  measuring SO2 1  \
DateTime
2022-11-23 13:30:00                     NaN             NaN
2022-11-23 15:30:00                     NaN             NaN
2022-11-23 16:30:00                     NaN             NaN
2022-11-23 18:30:00                     NaN             NaN
2022-11-23 20:30:00                     NaN             NaN
```

```
…                                    …                …
2023-02-08 12:30:00               15.204469              NaN
2023-02-08 13:30:00               15.351102              NaN
2023-02-08 14:30:00               13.929618              NaN
2023-02-08 15:30:00               12.788609              NaN
2023-02-08 16:30:00                9.874328              NaN

                     measuring no Temp SO2 1  measuring SO2 2  \
DateTime
2023-02-08 12:30:00                      NaN              NaN
2023-02-08 15:30:00                      NaN              NaN
2023-02-08 16:30:00                      NaN              NaN
2023-02-08 18:30:00                      NaN              NaN
2023-02-08 20:30:00                      NaN              NaN
…                                        …                …
2023-02-08 12:30:00                      NaN       105.037977
2023-02-08 13:30:00                      NaN       101.837936
2023-02-08 14:30:00                      NaN       106.043517
2023-02-08 15:30:00                      NaN       110.596287
2023-02-08 16:30:00                      NaN       136.193337

                     measuring no Temp SO2 2  measuring PM10  \
DateTime
2023-02-08 12:30:00                      NaN        2.588333
2023-02-08 15:30:00                      NaN        1.341250
2023-02-08 16:30:00                      NaN        1.335000
2023-02-08 18:30:00                      NaN        2.338333
2023-02-08 20:30:00                      NaN        1.025000
…                                        …                …
2023-02-08 12:30:00               147.029487        1.288333
2023-02-08 13:30:00               148.459469        1.476250
2023-02-08 14:30:00               153.582107        1.775000
2023-02-08 15:30:00               157.363207        2.096250
2023-02-08 16:30:00               166.095342        2.875000

                     measuring no Temp PM10  temperature  reference
DateTime
2023-02-08 12:30:00                3.261152    28.635000      37.76
2023-02-08 15:30:00                2.150700    26.161250      33.53
2023-02-08 16:30:00                2.128985    26.441250      30.82
2023-02-08 18:30:00                2.996423    28.901667      21.18
2023-02-08 20:30:00                1.661779    29.287500      20.67
…                                        …          …        …
2023-02-08 12:30:00                1.362800    39.311250      50.01
2023-02-08 13:30:00                1.530810    39.828750      67.43
2023-02-08 14:30:00                1.823899    39.931250      72.46
2023-02-08 15:30:00                2.149913    39.845000      59.65
```

```
2023-02-08 16:30:00                3.032776     37.960000        NaN

[1430 rows x 15 columns]
```

0.5 Calibrate data

0.5.1 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()
temperature_mean = sensor_data['temperature'].mean()
sensor_co_median = sensor_data['measuring CO'].median()
trend_co_median = sensor_data['measuring no Temp CO'].median()
sensor_no2_median = sensor_data['measuring NO2'].median()
sensor_o3_1_median = sensor_data['measuring O3 1'].median()
trend_o3_1_median = sensor_data['measuring no Temp O3 1'].median()
sensor_o3_2_median = sensor_data['measuring O3 2'].median()
trend_o3_2_median  = sensor_data['measuring no Temp O3 2'].median()
sensor_so2_1_median = sensor_data['measuring SO2 1'].median()
trend_so2_1_median = sensor_data['measuring no Temp SO2 1'].median()
sensor_so2_2_median = sensor_data['measuring SO2 2'].median()
trend_so2_2_median = sensor_data['measuring no Temp SO2 2'].median()
sensor_pm_10_median = sensor_data['measuring PM10'].median()

variables_names = ['measuring CO', 'measuring NO2', 'measuring O3 1',
                   'measuring O3 2', 'measuring PM10', 'temperature']

y = sensor_data['reference'].fillna(value=reference_median)
X = (sensor_data[variables_names].fillna(value={
                        variables_names[0]: sensor_co_median,
                        variables_names[1]: sensor_no2_median,
                        variables_names[2]: sensor_o3_1_median,
                        variables_names[3]: sensor_o3_2_median,
                        variables_names[4]: sensor_pm_10_median,
                        variables_names[5]: temperature_mean}).values.
 ↪reshape(-1,6))

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

0.5.2 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 = [2, 3]
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 = ""
    trends_subset = ""
    for index in index_list:
        key = key + variables_names[index] + ","
    feature_subsets[key] = index_list
feature_subsets
```

[ ]:
```
{'measuring O3 1,': [2],
 'measuring O3 2,': [3],
 'measuring CO,measuring O3 1,': [0, 2],
 'measuring CO,measuring O3 2,': [0, 3],
 'measuring NO2,measuring O3 1,': [1, 2],
 'measuring NO2,measuring O3 2,': [1, 3],
 'measuring O3 1,measuring O3 2,': [2, 3],
 'measuring O3 1,measuring PM10,': [2, 4],
 'measuring O3 1,temperature,': [2, 5],
 'measuring O3 2,measuring PM10,': [3, 4],
 'measuring O3 2,temperature,': [3, 5],
 'measuring CO,measuring NO2,measuring O3 1,': [0, 1, 2],
 'measuring CO,measuring NO2,measuring O3 2,': [0, 1, 3],
 'measuring CO,measuring O3 1,measuring O3 2,': [0, 2, 3],
 'measuring CO,measuring O3 1,measuring PM10,': [0, 2, 4],
 'measuring CO,measuring O3 1,temperature,': [0, 2, 5],
 'measuring CO,measuring O3 2,measuring PM10,': [0, 3, 4],
 'measuring CO,measuring O3 2,temperature,': [0, 3, 5],
 'measuring NO2,measuring O3 1,measuring O3 2,': [1, 2, 3],
 'measuring NO2,measuring O3 1,measuring PM10,': [1, 2, 4],
 'measuring NO2,measuring O3 1,temperature,': [1, 2, 5],
 'measuring NO2,measuring O3 2,measuring PM10,': [1, 3, 4],
 'measuring NO2,measuring O3 2,temperature,': [1, 3, 5],
 'measuring O3 1,measuring O3 2,measuring PM10,': [2, 3, 4],
```

```
'measuring O3 1,measuring O3 2,temperature,': [2, 3, 5],
'measuring O3 1,measuring PM10,temperature,': [2, 4, 5],
'measuring O3 2,measuring PM10,temperature,': [3, 4, 5],
'measuring CO,measuring NO2,measuring O3 1,measuring O3 2,': [0, 1, 2, 3],
'measuring CO,measuring NO2,measuring O3 1,measuring PM10,': [0, 1, 2, 4],
'measuring CO,measuring NO2,measuring O3 1,temperature,': [0, 1, 2, 5],
'measuring CO,measuring NO2,measuring O3 2,measuring PM10,': [0, 1, 3, 4],
'measuring CO,measuring NO2,measuring O3 2,temperature,': [0, 1, 3, 5],
'measuring CO,measuring O3 1,measuring O3 2,measuring PM10,': [0, 2, 3, 4],
'measuring CO,measuring O3 1,measuring O3 2,temperature,': [0, 2, 3, 5],
'measuring CO,measuring O3 1,measuring PM10,temperature,': [0, 2, 4, 5],
'measuring CO,measuring O3 2,measuring PM10,temperature,': [0, 3, 4, 5],
'measuring NO2,measuring O3 1,measuring O3 2,measuring PM10,': [1, 2, 3, 4],
'measuring NO2,measuring O3 1,measuring O3 2,temperature,': [1, 2, 3, 5],
'measuring NO2,measuring O3 1,measuring PM10,temperature,': [1, 2, 4, 5],
'measuring NO2,measuring O3 2,measuring PM10,temperature,': [1, 3, 4, 5],
'measuring O3 1,measuring O3 2,measuring PM10,temperature,': [2, 3, 4, 5],
'measuring CO,measuring NO2,measuring O3 1,measuring O3 2,measuring PM10,': [0,
 1,
 2,
 3,
 4],
'measuring CO,measuring NO2,measuring O3 1,measuring O3 2,temperature,': [0,
 1,
 2,
 3,
 5],
'measuring CO,measuring NO2,measuring O3 1,measuring PM10,temperature,': [0,
 1,
 2,
 4,
 5],
'measuring CO,measuring NO2,measuring O3 2,measuring PM10,temperature,': [0,
 1,
 3,
 4,
 5],
'measuring CO,measuring O3 1,measuring O3 2,measuring PM10,temperature,': [0,
 2,
 3,
 4,
 5],
'measuring NO2,measuring O3 1,measuring O3 2,measuring PM10,temperature,': [1,
 2,
 3,
 4,
 5],
```

```
 'measuring CO,measuring NO2,measuring O3 1,measuring O3 2,measuring
PM10,temperature,': [0,
  1,
  2,
  3,
  4,
  5]}
```

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, aic, bic, 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())
    aic_text = 'AIC = {:.2f} ± {:.2f}'.format(aic.mean(), aic.std())
    bic_text = 'BIC = {:.2f} ± {:.2f}'.format(bic.mean(), bic.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)
    plt.text(0.02, 0.70, aic_text, ha='left', va='center', transform=plt.gca().
↪transAxes, fontsize=12)
    plt.text(0.02, 0.65, bic_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_ALL_' + file_name + '.png')
```

[ ]:
```
def calculate_bic(n, mse, num_params):
    bic = n * np.log(mse) + num_params * np.log(n)
    return bic

def calculate_aic(n, mse, num_params):
    aic = n * np.log(mse) + 2 * num_params
    return aic
```

[ ]:
```
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
import numpy as np

models = {
    'MLP Regression': (
        ('mlp_regressor', MLPRegressor(solver="lbfgs", max_iter=1000,␣
 ↪random_state=42)), {
            'mlp_regressor__hidden_layer_sizes': [(4,50), (10,10), (200,),␣
 ↪(200,4), (200,10), (200,50)],
            'mlp_regressor__alpha': [0.001, 0.01, 0.1, 1, 10]
        }
    ),
    'Multilinear Regression': (
        ('linear_regressor', LinearRegression()), { }
    ),
    'KNN Regression': (
        ('knn_regressor', KNeighborsRegressor()), {
            'knn_regressor__n_neighbors': [13, 15, 17, 20]
        }
    ),
    'Random Forests Regression': (
        ('random_forest_regressor', RandomForestRegressor()), {
            'random_forest_regressor__n_estimators': [100, 150],
            'random_forest_regressor__max_depth': [None, 10],
            'random_forest_regressor__min_samples_split': [2, 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 = {}
mse_by_features = {}
aic_by_features = {}
bic_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 = {}
    model_mse = {}
    model_aic = {}
    model_bic = {}
    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', 'neg_mean_squared_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']
        mse = cross_validation['test_neg_mean_squared_error']
        num_params = best_model.n_features_in_ + 1
        print(f"Number of parameters: \{num_params}")
        aic = calculate_aic(len(y_pred), mse=abs(mse), num_params=num_params)
        bic = calculate_bic(len(y_pred), mse=abs(mse), num_params=num_params)

        plot_predictions_and_observations(y_test, y_pred, r2=r2, rmse=rmse,␣
↪mae=mae,
```

```
                                            aic=aic, bic=bic,␣
↪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,
            'Test MSE': mse,
            'AIC': aic,
            'BIC': bic
        }
        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()
        }
        model_mse[model_name] = {
            'Mean': mse.mean(),
            'Std': mse.std()
        }
        model_aic[model_name] = {
            'Mean': aic.mean(),
            'Std': aic.std()
        }
        model_bic[model_name] = {
            'Mean': bic.mean(),
            'Std': bic.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
    mse_by_features[features_set] = model_mse
    aic_by_features[features_set] = model_aic
    bic_by_features[features_set] = model_bic

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()}")
        print(f"Test MSE: {result['Test MSE'].mean()} +/- {result['Test MSE'].
↪std()}")
        print(f"Test AIC: {result['AIC'].mean()} +/- {result['AIC'].std()}")
        print(f"Test BIC: {result['BIC'].mean()} +/- {result['BIC'].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'
mse_file_name = output_directory_path + sensor_name + '_mse.csv'
aic_file_name = output_directory_path + sensor_name + '_aic.csv'
bic_file_name = output_directory_path + sensor_name + '_bic.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(mse_by_features).transpose().to_csv(mse_file_name)
pd.DataFrame(aic_by_features).transpose().to_csv(aic_file_name)
pd.DataFrame(bic_by_features).transpose().to_csv(bic_file_name)
pd.DataFrame(results).transpose().to_csv(results_file_name)
```

1.1 Plot Results

[ ]:
```
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()

mean_mse_by_features_dataframe = pd.DataFrame()
```

```
std_mse_by_features_dataframe = pd.DataFrame()

mean_aic_by_features_dataframe = pd.DataFrame()
std_aic_by_features_dataframe = pd.DataFrame()

mean_bic_by_features_dataframe = pd.DataFrame()
std_bic_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', '')
        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', '')
        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', '')
        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']]

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

```
        std_mse_by_features_dataframe[colum_name] = [feature_dict[model]['Std']]

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

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

[ ]:
```
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*5,5))

    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)
```

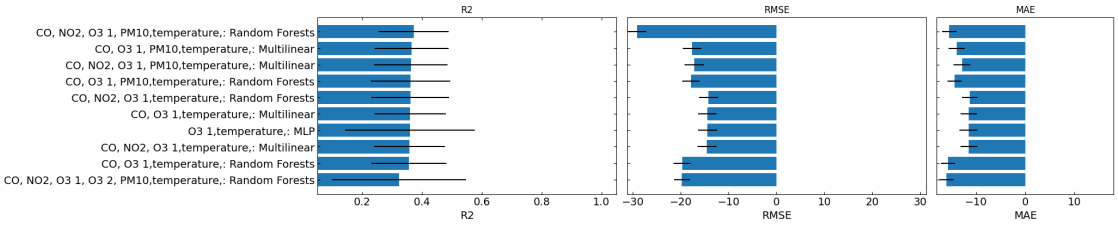
[ ]:
```
r2_sorted_dataframe = (mean_r2_by_features_dataframe.
 ↪sort_values(by=mean_r2_by_features_dataframe.index[0], axis=1,␣
 ↪ascending=False))
features = r2_sorted_dataframe.columns

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

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

```
plot_metrics(features=features[:10], r2_list=mean_r2[:10],␣
 ↪r2_error_list=error_r2[:10],
             rmse_list=mean_rmse[:10], rmse_error_list=error_rmse[:10],
             mae_list=mean_mae[:10], mae_error_list=error_mae[:10])
```



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

def plot_other_metrics(features, first_list, first_error_list, first_title,
                       second_list, second_error_list, second_title,
                       third_list, third_error_list, third_title):
    bottom, height = 0.1, 0.65
    left, width = bottom, height*1.3
    spacing = 0.005

    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*5,5))

    ax_r2  = plt.axes(rect_r2)
    ax_r2.tick_params(direction='in', top=True, right=True, labelsize=14)
    ax_r2.set_title(first_title)

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

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

    y_pos = np.arange(len(features))

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

```
    ax_r2.barh(y_pos, first_list, xerr=first_error_list, align='center')
    ax_r2.set_yticks(y_pos, labels=features, fontsize=14)
    ax_r2.invert_yaxis()  # labels read top-to-bottom
    ax_r2.set_xlabel(first_title, fontsize=14)

    ax_rmse.barh(y_pos, second_list, xerr=second_error_list, align='center')
    ax_rmse.set_yticks(y_pos, labels=features, fontsize=14)
    ax_rmse.invert_yaxis()  # labels read top-to-bottom
    ax_rmse.set_xlabel(second_title, fontsize=14)

    ax_mae.barh(y_pos, third_list, xerr=third_error_list, align='center')
    ax_mae.set_yticks(y_pos, labels=features, fontsize=14)
    ax_mae.invert_yaxis()  # labels read top-to-bottom
    ax_mae.set_xlabel(third_title, fontsize=14)
```

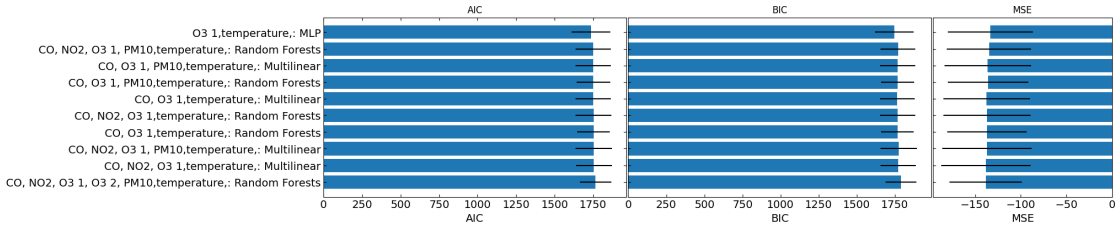
[ ]:
```
aic_sorted_dataframe = (mean_aic_by_features_dataframe.
 ↪sort_values(by=mean_aic_by_features_dataframe.index[0], axis=1,␣
 ↪ascending=True))
other_features = aic_sorted_dataframe.columns

mean_mse = mean_mse_by_features_dataframe[aic_sorted_dataframe.columns].values.
 ↪flatten()
error_mse = std_mse_by_features_dataframe[aic_sorted_dataframe.columns].values.
 ↪flatten()

mean_aic = mean_aic_by_features_dataframe[aic_sorted_dataframe.columns].values.
 ↪flatten()
error_aic = std_aic_by_features_dataframe[aic_sorted_dataframe.columns].values.
 ↪flatten()

mean_bic = mean_bic_by_features_dataframe[aic_sorted_dataframe.columns].values.
 ↪flatten()
error_bic = std_bic_by_features_dataframe[aic_sorted_dataframe.columns].values.
 ↪flatten()

plot_other_metrics(features=other_features[:10], first_list=mean_aic[:10],␣
 ↪first_error_list=error_aic[:10], first_title='AIC',
            second_list=mean_bic[:10], second_error_list=error_bic[:10],␣
 ↪second_title='BIC',
            third_list=mean_mse[:10], third_error_list=error_mse[:10],␣
 ↪third_title='MSE')
```


AIC
BIC
MSE
O3 1,temperature,: MLP
CO, NO2, O3 1, PM10,temperature,: Random Forests
CO, O3 1, PM10,temperature,: Multilinear
CO, O3 1, PM10,temperature,: Random Forests
CO, O3 1,temperature,: Multilinear
CO, NO2, O3 1,temperature,: Random Forests
CO, O3 1,temperature,: Random Forests
CO, NO2, O3 1, PM10,temperature,: Multilinear
CO, NO2, O3 1,temperature,: Multilinear
CO, NO2, O3 1, O3 2, PM10,temperature,: Random Forests
0
250
500
750
1000
1250
1500
1750
−150
−100
−50
0