O3_2-field-data-treatment-Diamante

February 13, 2024

1 Electrochemical O3 sensor data preprocessing

- Pollutant: Ozone
- Sensor: Alphasense OX-B4

1.1 Constants

1.1.1 Sensors IDs

1.1.2 API Constants

[ ]:
```
HOST = "renovar.lcqar.ufsc.br"
PORT = 8080
RAW_DATA_DIR = "data/raw-data-monit-fix-2022-2023-Diamante/"
RAW_FILE_NAME = "ISB_O32.CSV"

SENSOR_FILE_DIR = 'data/input/'
SENSOR_NAME = 'alpha_o3_2_conc'
SENSOR_FILE_NAME = SENSOR_NAME + 'web_dataframe.csv'
```

1.1.3 Sensor Constants

[ ]:
```
M = 48
lower_limit=15.0
upper_limit=20e3
t_90 = 80      # sensor takes 30 seconds to reach a value of 10e3
t_90_value = 1e3
sampling_period = 15 * 60
```

2 Alphasense O3 (2) Sensor Data

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

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

[ ]:
```
from GetSensorDataService import GetSensorDataService
```

```
get_sensor_data_service = GetSensorDataService(HOST, PORT)
sensor_data = get_sensor_data_service.
 ↪get_data_from_file(RAW_DATA_DIR+RAW_FILE_NAME, sensor_name=SENSOR_NAME)
sensor_data.head()
```

```
/Users/Fernando/Documents/Projects/Github/lcqar-low-cost-monit-proc/data-pre-
processing/GetSensorDataService.py:13: 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.
  df['DateTime'] = (pd.to_datetime(df[date_time_col],
infer_datetime_format=False, format='%d/%m/%Y/%H/%M/%S'))
```

[ ]:
```
    latitude  longitude  measuring            DateTime
0 -28.456899 -48.972999      27.48 2022-11-21 10:41:59
1 -28.456899 -48.972999      27.61 2022-11-21 10:57:45
2 -28.456899 -48.972999      27.92 2022-11-21 11:13:34
3 -28.456899 -48.972999      28.17 2022-11-21 11:29:19
4 -28.456899 -48.972999      28.34 2022-11-21 11:45:05
```

2.1 Upload Data from File

[ ]:
```
import pandas as pd

df = pd.read_csv(SENSOR_FILE_DIR + SENSOR_FILE_NAME)
df.head()
```

[ ]:
```
   Unnamed: 0   latitude  longitude  measuring             DateTime
0           0 -28.456899 -48.972999      27.48  2022-11-21 10:41:59
1           1 -28.456899 -48.972999      27.61  2022-11-21 10:57:45
2           2 -28.456899 -48.972999      27.92  2022-11-21 11:13:34
3           3 -28.456899 -48.972999      28.17  2022-11-21 11:29:19
4           4 -28.456899 -48.972999      28.34  2022-11-21 11:45:05
```

2.1.1 Create Sensor Dataframe as Pandas Series with a period of 15 mins

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

# Resample data with 15 mins period and create sensor dataframe
sensor_dataframe = web_dataframe.sort_values(by='DateTime', ascending=True).
 ↪reset_index().drop(columns='index')
sensor_dataframe.index = sensor_dataframe['DateTime']
sensor_dataframe = sensor_dataframe.drop(columns=['DateTime'])
sensor_dataframe = sensor_dataframe.resample('15T').mean()
sensor_dataframe
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_2814/166902210.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.
  web_dataframe['DateTime'] = (pd.to_datetime(df['DateTime'],
infer_datetime_format=True))
```

[ ]:

```
                      latitude  longitude  measuring
DateTime
2022-11-21 10:30:00 -28.456899 -48.972999      27.48
2022-11-21 10:45:00 -28.456899 -48.972999      27.61
2022-11-21 11:00:00 -28.456899 -48.972999      27.92
2022-11-21 11:15:00 -28.456899 -48.972999      28.17
2022-11-21 11:30:00        NaN        NaN        NaN
…                          …          …          …
2023-04-21 20:45:00 -28.456899 -48.972999      18.83
2023-04-21 21:00:00 -28.456899 -48.972999      18.74
2023-04-21 21:15:00 -28.456899 -48.972999      18.77
2023-04-21 21:30:00 -28.456899 -48.972999      19.23
2023-04-21 21:45:00 -28.456899 -48.972999      19.48

[14542 rows x 3 columns]
```

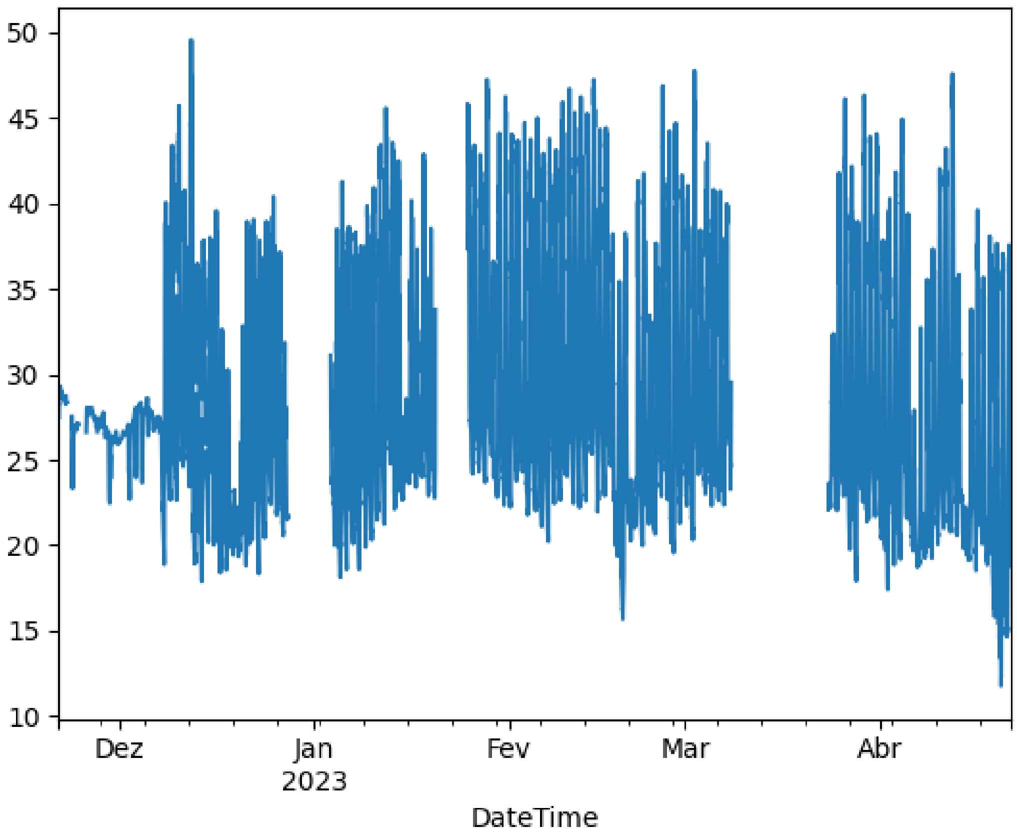
2.1.2 Plot raw data

[ ]:

```
sensor_dataframe['measuring'].plot()
```

[ ]:

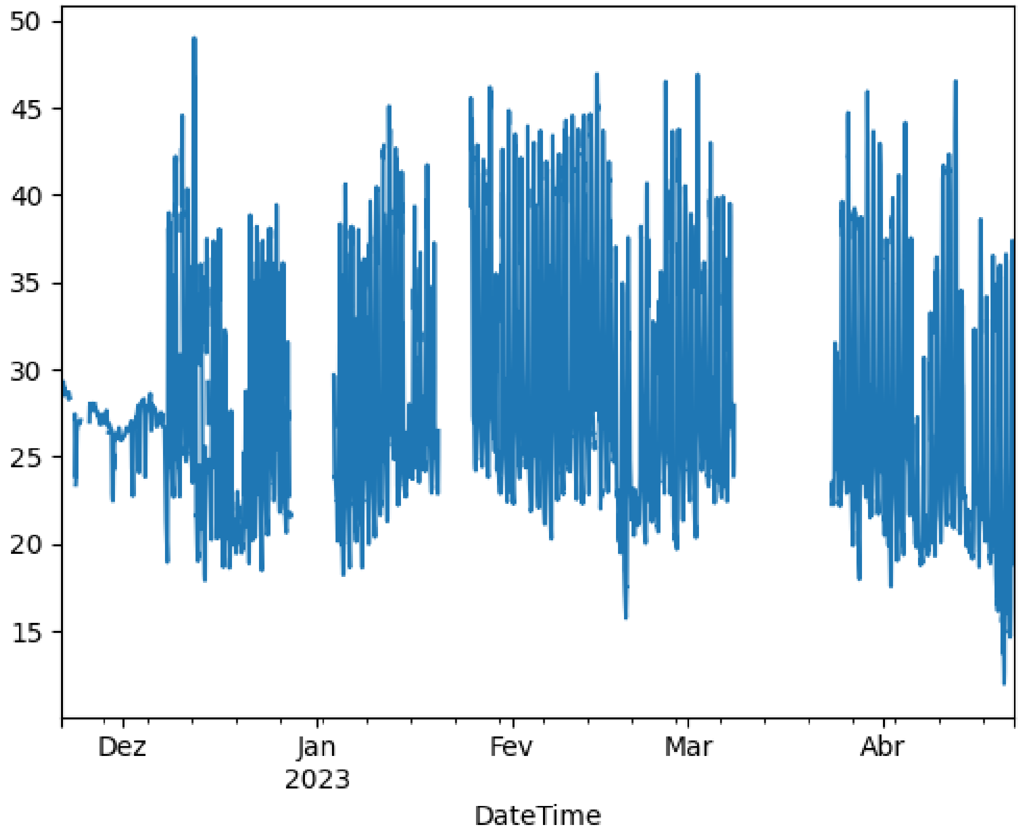
```
<Axes: xlabel='DateTime'>
```



2.1.3 Smooth data with a window of 4 samples, i.e.: an hour of data

```
[ ]: sensor_dataframe['raw measuring'] = sensor_dataframe['measuring']
     sensor_dataframe['measuring'] = sensor_dataframe['measuring'].rolling(window=4).
      ↪median()
     sensor_dataframe['Hour'] = sensor_dataframe.index.hour
     sensor_dataframe['measuring'].plot()
```

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



2.1.4 Convert to ug/m3

[ ]:
```
sensor_dataframe['value'] = sensor_dataframe['measuring'].map(lambda v: 0.
  ↪0409*v*M)
```

2.2 Tag data

Apply tags to the data according to the quality control processes described above

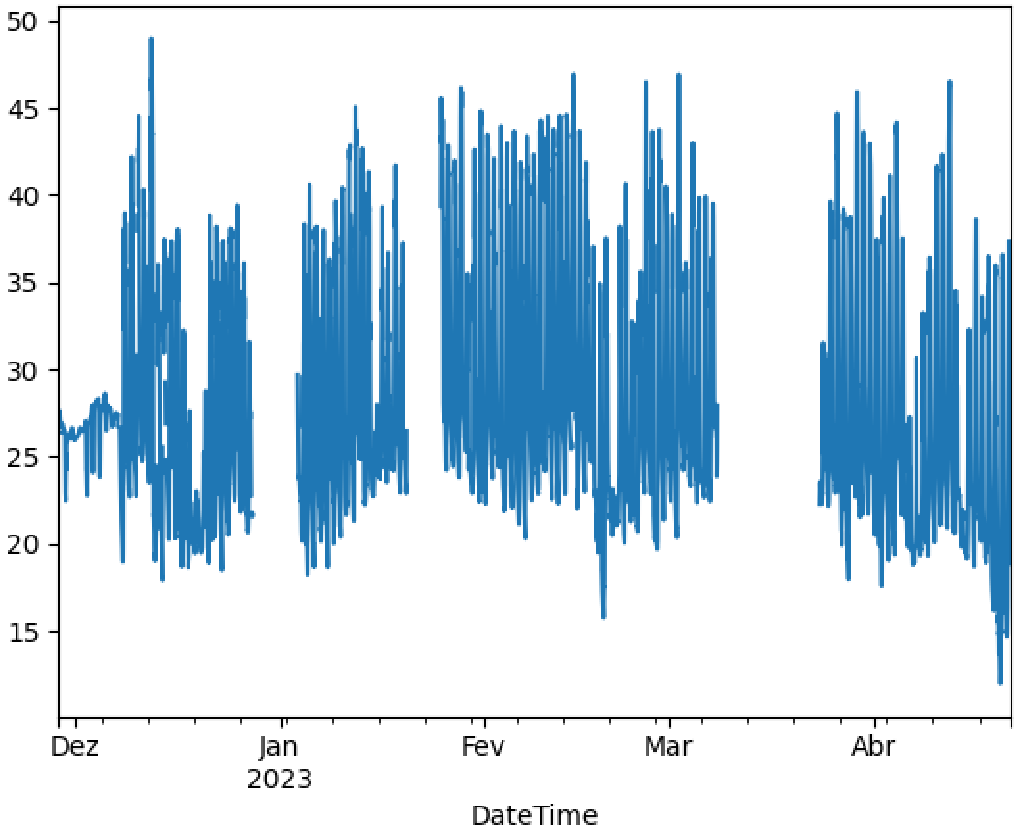
[ ]:
```
sensor_dataframe['Tag'] = 'VALID'
```

Tag data during stabilizing period AQMesh recomends to considering the first two days after installation as a period for stabilization. In our case, a period of seven days was selected in order to remove outliers candidates detected during that period. For that purpose we detect the first 7 days of uninterrupted measurements

[ ]:
```
# Identify consecutive measurements (15 mins between each)
consecutive_periods = sensor_dataframe.index.to_series().diff().dt.seconds ==␣
  ↪900
```

```
# Select the first seven days of consecutive measurements (96 = 24 * 60 mins /␣
 ↪15 mins, amount of 15 mins periods in a day)
last_stabilizing_index = sensor_dataframe[consecutive_periods].head(96 * 7).
 ↪index[-1] # 7 days
sensor_dataframe.loc[sensor_dataframe.index[0]:last_stabilizing_index,'Tag'] =␣
 ↪'STABILIZING'
sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['measuring'].
 ↪resample('15T').mean().plot()
sensor_dataframe.head()
```

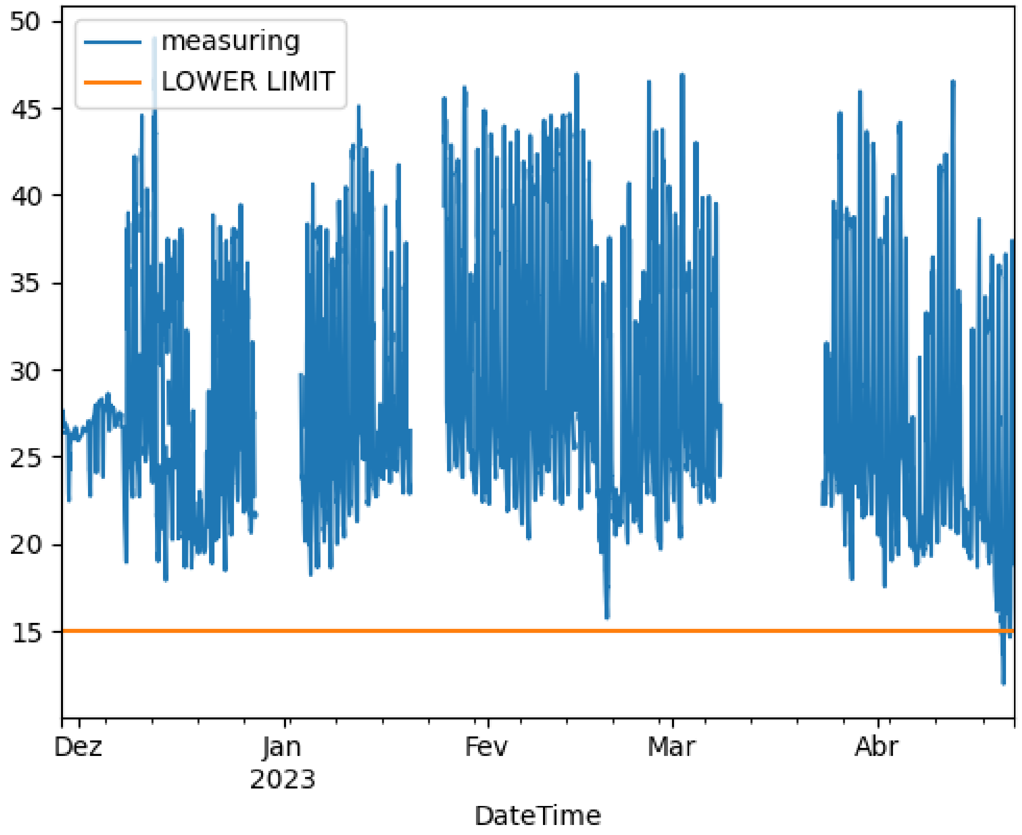
```
[ ]:                       latitude  longitude  measuring  raw measuring  Hour  \
     DateTime
     2022-11-21 10:30:00 -28.456899 -48.972999        NaN          27.48    10
     2022-11-21 10:45:00 -28.456899 -48.972999        NaN          27.61    10
     2022-11-21 11:00:00 -28.456899 -48.972999        NaN          27.92    11
     2022-11-21 11:15:00 -28.456899 -48.972999     27.765          28.17    11
     2022-11-21 11:30:00        NaN        NaN        NaN            NaN    11

                              value          Tag
     DateTime
     2022-11-21 10:30:00        NaN  STABILIZING
     2022-11-21 10:45:00        NaN  STABILIZING
     2022-11-21 11:00:00        NaN  STABILIZING
     2022-11-21 11:15:00  54.508248  STABILIZING
     2022-11-21 11:30:00        NaN  STABILIZING
```



Tag values that are missing, are greater than sensor span or are lower than sensor resolution The upper limit was too high that affected the graph resolution, therefore it is not plotted

```
[ ]: sensor_dataframe['UPPER LIMIT'] = upper_limit
     sensor_dataframe['LOWER LIMIT'] = lower_limit
     sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'][['measuring', 'LOWER␣
      ↪LIMIT']].resample('15T').mean().plot()
```

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



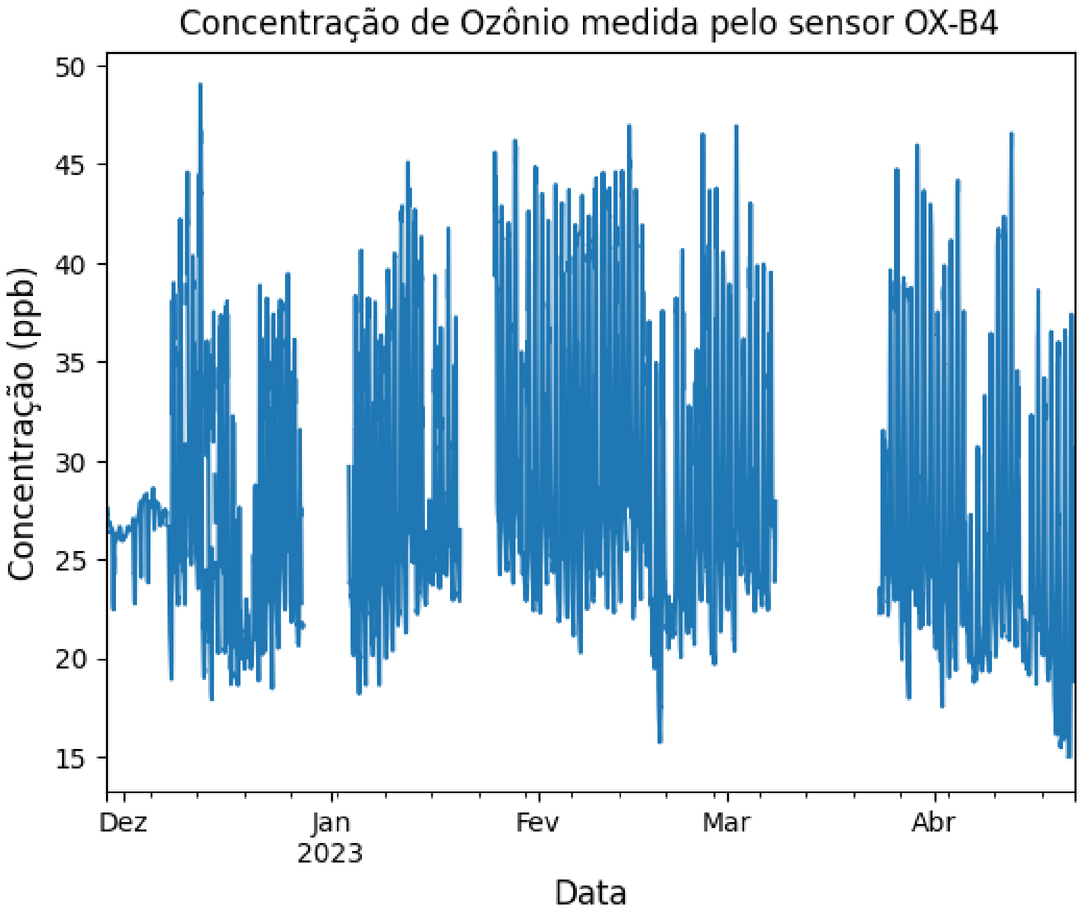
[ ]:

```
import numpy as np
from SensorDataAnalysisService import SensorDataAnalysisService as␣
 ↪sensor_analysis

def tag_by_sensor_limits(value, tag, lower_limit, upper_limit):
 if (tag != 'VALID'): return tag
 return sensor_analysis.get_tags_from_series(value=value,
                                             lower_limit=lower_limit,
                                             upper_limit=upper_limit)

sensor_dataframe['Tag'] = (sensor_dataframe[['measuring', 'Tag']]
                                        .apply(lambda df:␣
 ↪tag_by_sensor_limits(value=df[0], tag=df[1],
                                                                             ␣
 ↪lower_limit=lower_limit,
                                                                             ␣
 ↪upper_limit=upper_limit),
                                                axis=1))
```

Plot valid data after removing samples bellow lower limit

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

     fig, ax = plt.subplots(figsize=(1.3*5,5))
     sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['measuring'].
      ↪rename('Concentração de Ozônio').resample('15T').mean().plot()
     plt.title('Concentração de Ozônio medida pelo sensor OX-B4')
     ax.set_xlabel('Data', fontsize=12)
     ax.set_ylabel('Concentração (ppb)', fontsize=12)
```

```
[ ]: Text(0, 0.5, 'Concentração (ppb)')
```



2.2.1 Data Tag contabilization

```
[ ]: tags = ['MISSING','LTLL', 'GTUL', 'STABILIZING', 'BADSPIKE', 'VALID']
     data_contabilization = sensor_analysis.count_tags(tags, sensor_dataframe)
     data_contabilization
```

```
[ ]:                  #          %
     MISSING       2734  18.800715
     LTLL            49   0.336955
     GTUL             0        0.0
     STABILIZING    673   4.627974
     BADSPIKE         0        0.0
     VALID        11086  76.234356
     TOTAL        14542      100.0
```

2.3 Analyse valid data

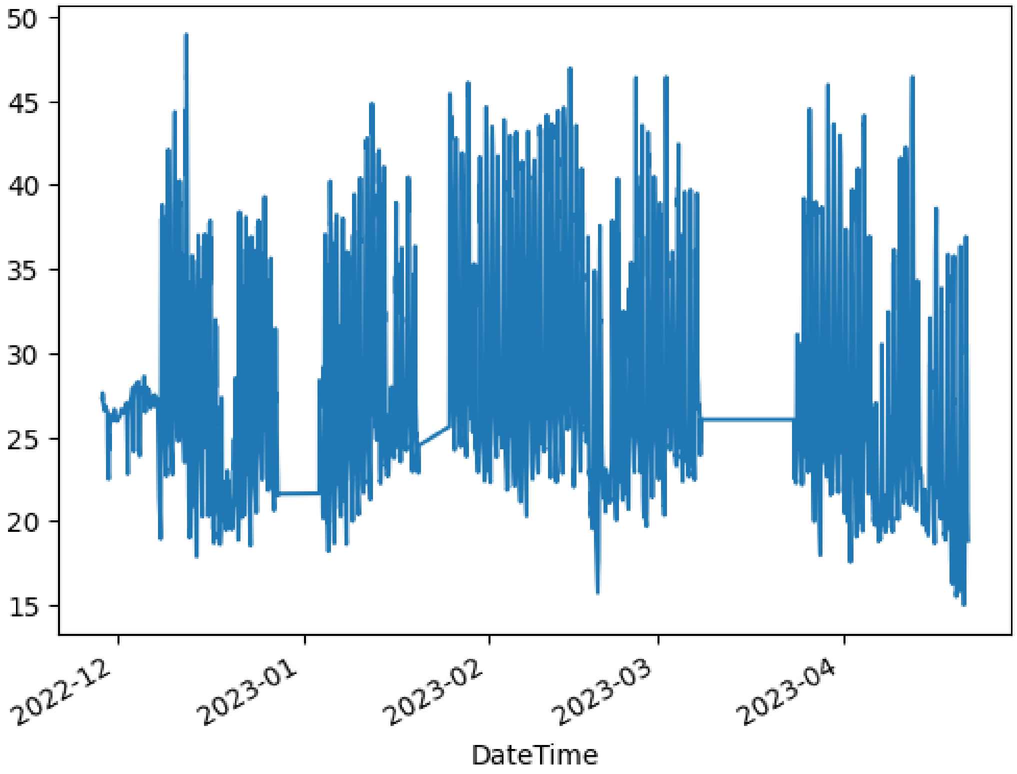
The graph shows the time series of the valid data, box plots of the data grouped by hour of the day, and histogram of the data.

```
[ ]: valid_dataframe = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'].
       ↪drop(columns=['Tag'])
```

2.3.1 Smooth data with a window of 4 samples, i.e.: an hour of data

```
[ ]: valid_dataframe['raw measuring'] = valid_dataframe['measuring']
     valid_dataframe['measuring'] = valid_dataframe['measuring'].rolling(window=4).
       ↪median()
     valid_dataframe['measuring'].plot()
```

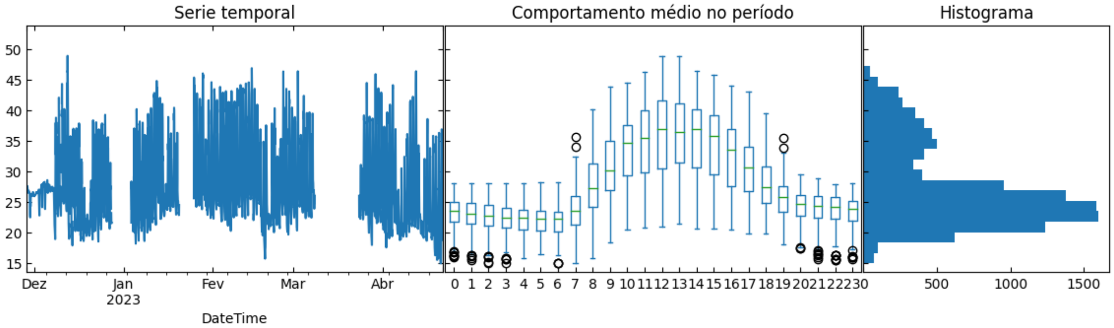
```
[ ]: <Axes: xlabel='DateTime'>
```



2.3.2 Analyse data

```
[ ]: valid_dataframe = valid_dataframe.resample('15T').mean()
     sensor_analysis.plot_box_hist(df=valid_dataframe, bins=20)
```

```
/Users/Fernando/Documents/Projects/Github/lcqar-low-cost-monit-proc/data-pre-
processing/SensorDataAnalysisService.py:111: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Hour'] = df['Hour'].astype('int64')
```

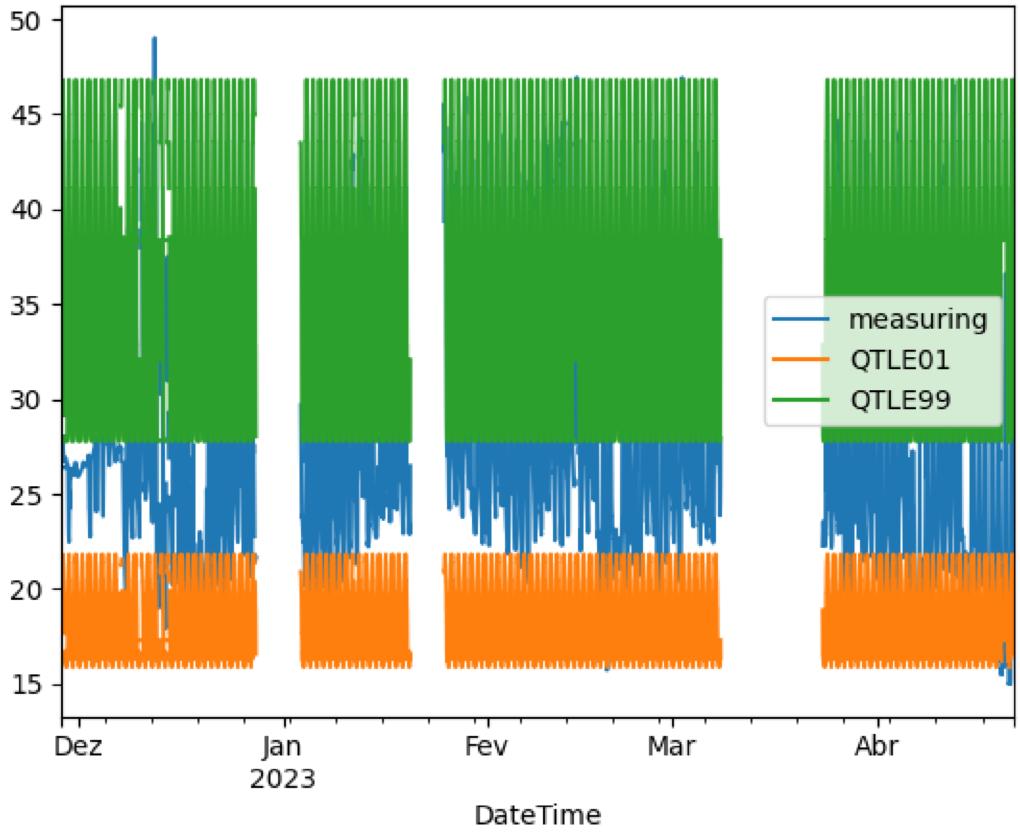


2.3.3 Calculate quantiles

We calculated the 1% and 99 % quantiles of every hour of the day. The values greater outside the 1 % - 99 % where tagged as Greater that Quantile 99 (GTQTLE99) and Lower than Quantile 1 (LTQTLE01)

```
[ ]: import numpy as np

     global_qtle_01 = valid_dataframe.pivot(columns='Hour')['measuring'].
      ↪quantile(q=0.01, axis='index', interpolation='lower').dropna()
     global_qtle_99 = valid_dataframe.pivot(columns='Hour')['measuring'].
      ↪quantile(q=0.99, axis='index', interpolation='higher').dropna()
     sensor_dataframe['QTLE01'] = sensor_dataframe['Hour'].map(lambda hr:␣
      ↪global_qtle_01[hr] if (not np.isnan(hr)) else np.nan)
     sensor_dataframe['QTLE99'] = sensor_dataframe['Hour'].map(lambda hr:␣
      ↪global_qtle_99[hr] if (not np.isnan(hr)) else np.nan)
     sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'][['measuring', 'QTLE01',␣
      ↪'QTLE99']].resample('15T').mean().plot()
```

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



2.3.4 Tag data according to quantiles

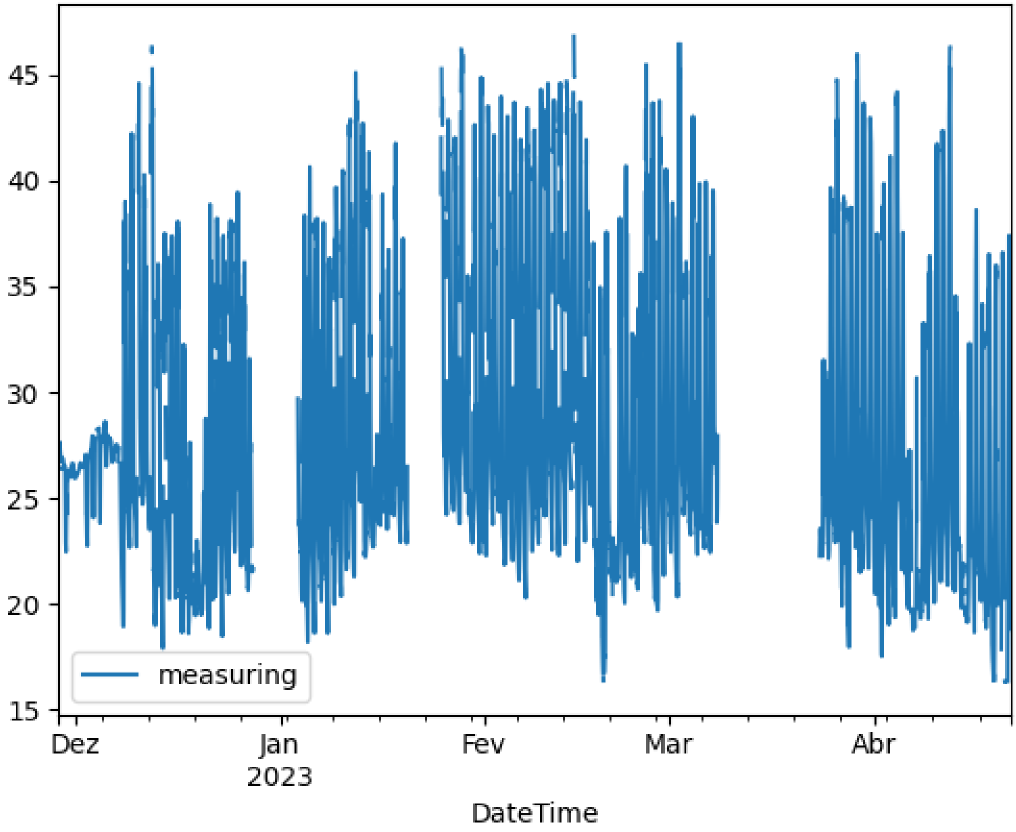
```
[ ]: sensor_dataframe['Tag'] = (sensor_dataframe[['Tag', 'measuring', 'QTLE01',␣
      ↪'QTLE99']]
                                     .apply(lambda df: sensor_analysis.
      ↪tag_by_quantiles(current_tag=df[0],
                                                                              ␣
      ↪value=df[1],
                                                                              ␣
      ↪quantile_01=df[2],
                                                                              ␣
      ↪quantile_99=df[3]),
                                            axis=1))
     tags = ['MISSING','LTLL', 'GTUL', 'BADSPIKE', 'VALID', 'LTQTLE01', 'GTQTLE99']
     data_contabilization = sensor_analysis.count_tags(tags, sensor_dataframe)
     data_contabilization
```

```
[ ]:                #            %
     MISSING     2734   18.800715
     LTLL          49    0.336955
     GTUL           0         0.0
     BADSPIKE       0         0.0
     VALID      10814   74.363911
     LTQTLE01     125    0.859579
     GTQTLE99     147    1.010865
     TOTAL      14542       100.0
```

Plot valid data

```
[ ]: sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'][['measuring']].
       ↪resample('15T').mean().plot()
```

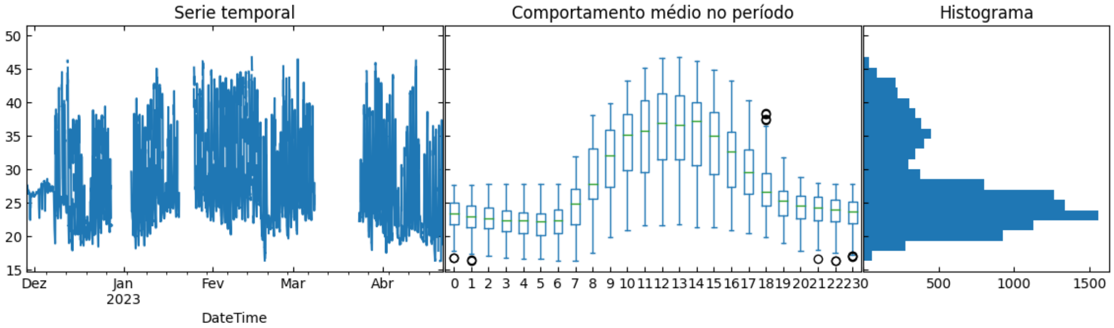
```
[ ]: <Axes: xlabel='DateTime'>
```



Analyse data after removing quantiles

```
[ ]: valid_dataframe = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'].
       ↪drop(columns=['Tag']).resample('15T').mean()
     sensor_analysis.plot_box_hist(df=valid_dataframe, bins=20)
```

```
/Users/Fernando/Documents/Projects/Github/lcqar-low-cost-monit-proc/data-pre-
processing/SensorDataAnalysisService.py:111: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Hour'] = df['Hour'].astype('int64')
```



2.4 Analyse data derivatives

Analyse the derivatives of the data for removing abrupt changes in the data (BADSPIKE)

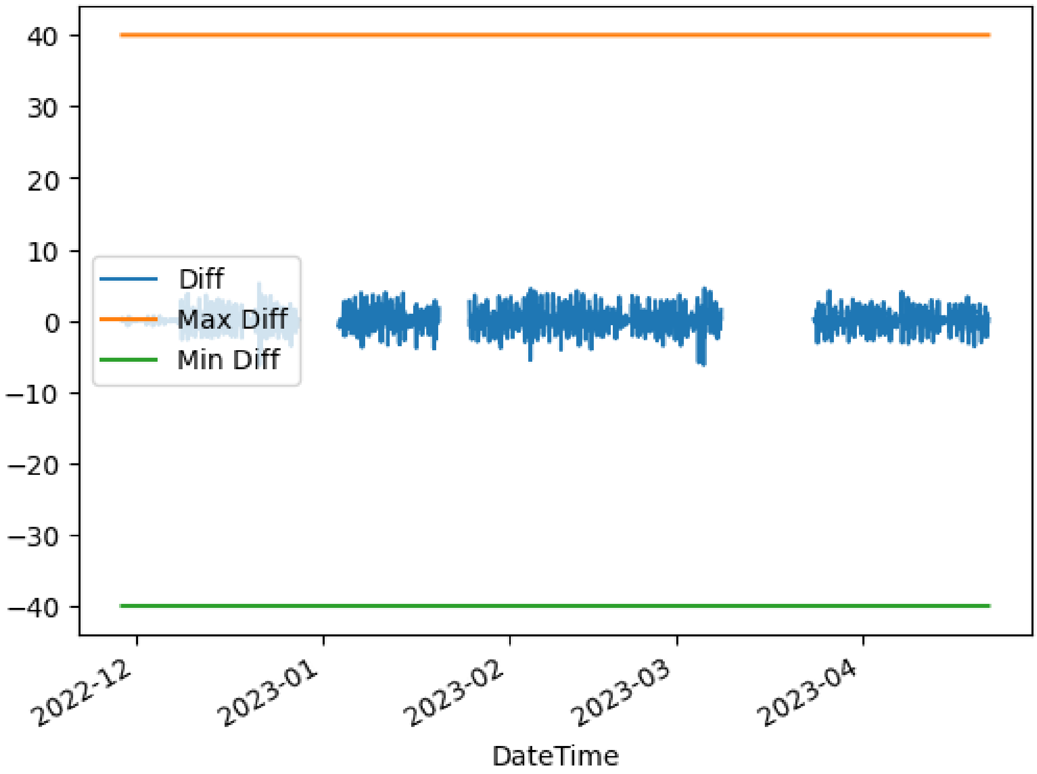
Tag values with derivatives faster than sensor response time The value of 40 ppb was selected based on the maximum derivative found in the reference data. The maximum derivative (with measuring period of 1 hour) found was -45 ppb. Therefore, a maximum of 40 ppb was considered appropriate for a 15 mins period.

[ ]:

```
max_diff_value = 40
sensor_dataframe['Diff'] = sensor_dataframe['measuring'].resample('15T').mean().
 ↪diff()
sensor_dataframe['Max Diff'] = max_diff_value
sensor_dataframe['Min Diff'] = -max_diff_value
sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'][['Diff', 'Max Diff', 'Min␣
 ↪Diff']].plot()
```

[ ]:

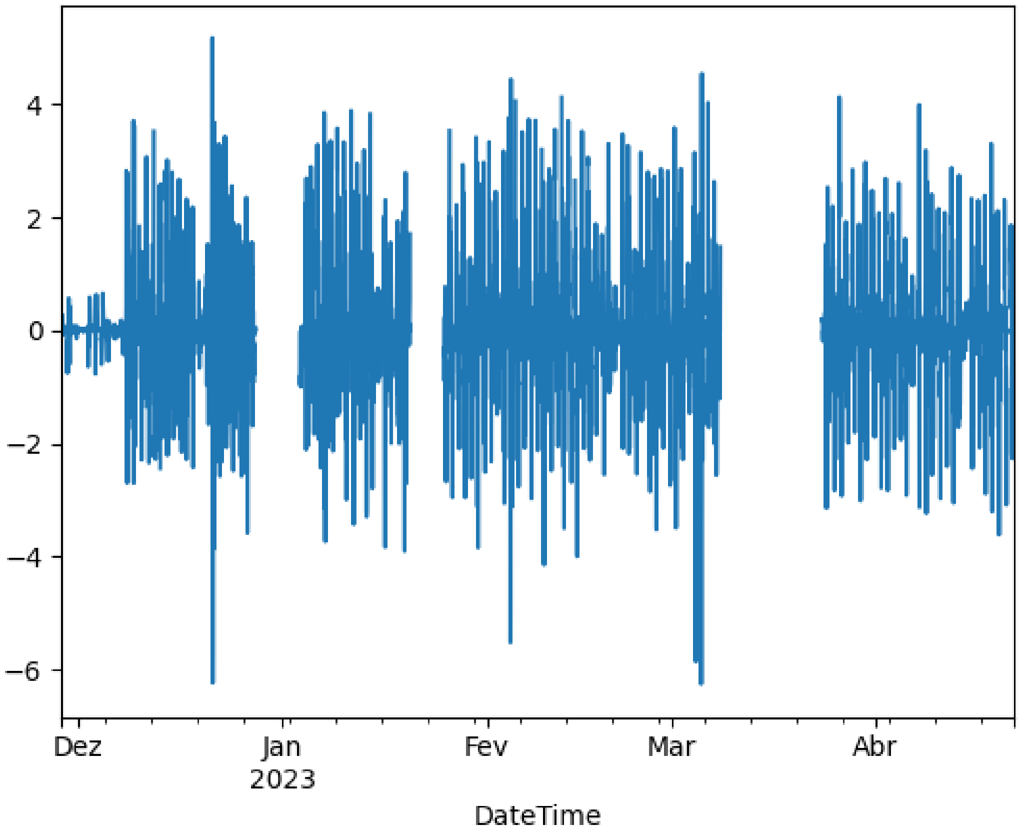
```
<Axes: xlabel='DateTime'>
```



[ ]:
```
import numpy as np

def tag_data_with_derivatives(tagged_df, max_diff_value):
        current_tag = tagged_df[0]
        value = tagged_df[1]
        if ((current_tag != 'VALID') or (np.isnan(value))): return current_tag
        if ((value > max_diff_value) or (value < -max_diff_value)): return␣
 ↪'BADSPIKE'
        return 'VALID'

sensor_dataframe['Tag'] = (sensor_dataframe[['Tag', 'Diff', 'Max Diff']]
                           .apply(lambda df:␣
 ↪tag_data_with_derivatives(tagged_df=df, max_diff_value=df[2]),
                                   axis=1))
```

[ ]:
```
sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['Diff'].resample('15T').
 ↪mean().plot()
```

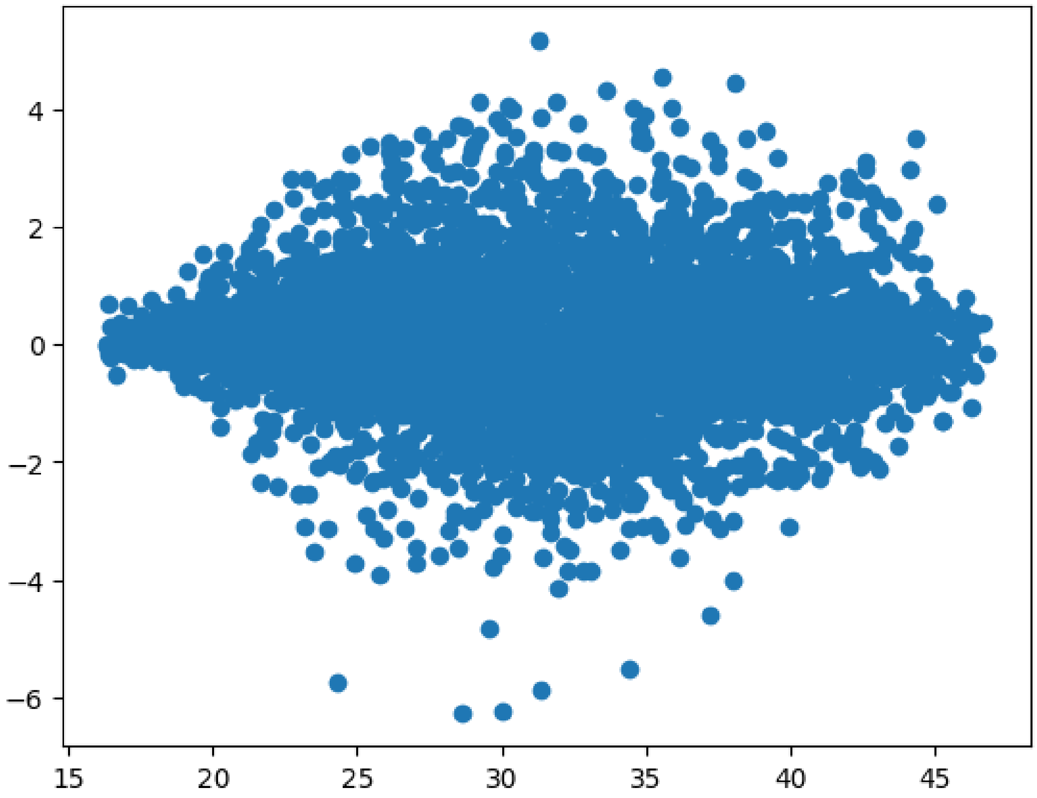
[ ]:
```
<Axes: xlabel='DateTime'>
```



Plot data derivatives vs data

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

     valid_dataframe = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']
     fig = plt.figure(figsize=(1.3*5,5))
     plt.scatter(valid_dataframe['measuring'], valid_dataframe['Diff'])
```

```
[ ]: <matplotlib.collections.PathCollection at 0x16a977df0>
```



Data Tag contabilization

```
[ ]: tags = ['MISSING','LTLL', 'GTUL', 'STABILIZING', 'BADSPIKE', 'VALID',␣
      ↪'LTQTLE01', 'GTQTLE99', 'REBASE']
     data_contabilization = sensor_analysis.count_tags(tags, sensor_dataframe)
     data_contabilization
```

```
[ ]:                  #          %
     MISSING       2734  18.800715
     LTLL            49   0.336955
     GTUL             0        0.0
     STABILIZING    673   4.627974
     BADSPIKE         0        0.0
     VALID        10814  74.363911
     LTQTLE01       125   0.859579
     GTQTLE99       147   1.010865
     REBASE           0        0.0
     TOTAL        14542      100.0
```

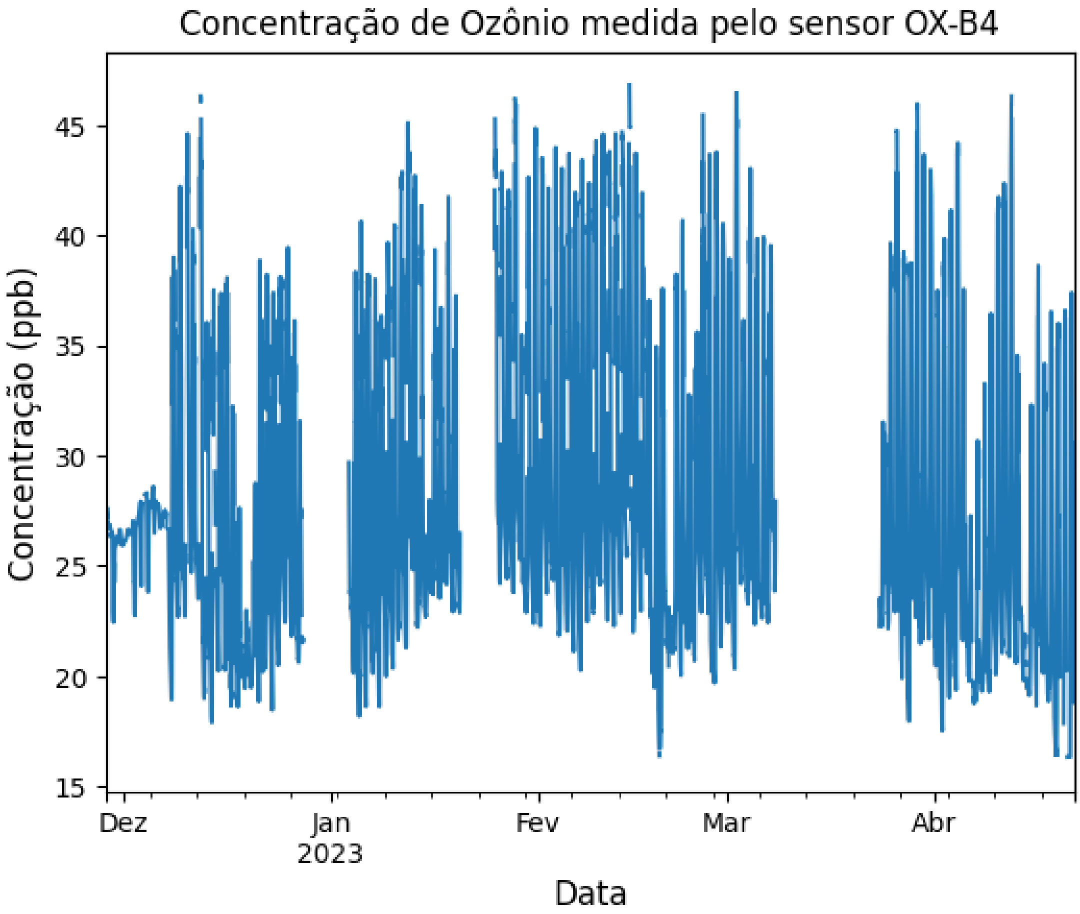
Plot valid data

[ ]:

```
fig, ax = plt.subplots(figsize=(1.3*5,5))
sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['measuring'].
 ↪rename('Concentração de Ozônio').resample('15T').mean().plot()
plt.title('Concentração de Ozônio medida pelo sensor OX-B4')
ax.set_xlabel('Data', fontsize=12)
ax.set_ylabel('Concentração (ppb)', fontsize=12)
```

[ ]:

```
Text(0, 0.5, 'Concentração (ppb)')
```

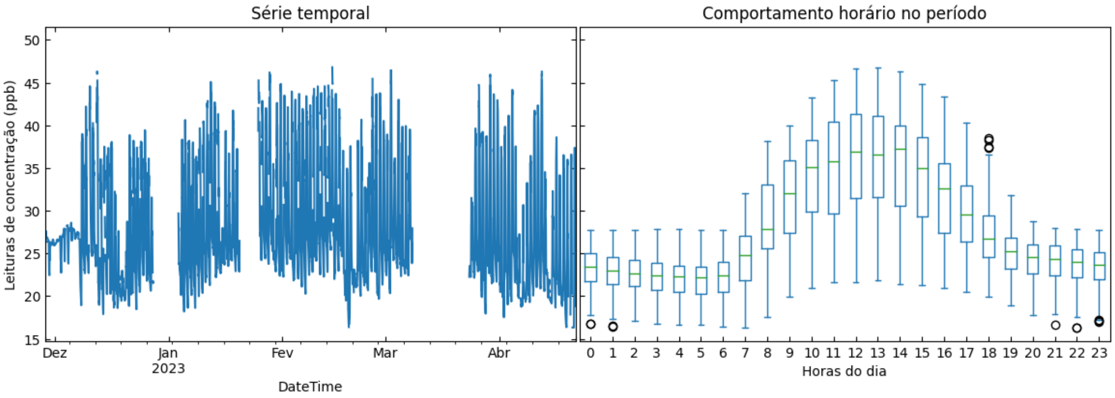


[ ]:

```
valid_dataframe = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'].
 ↪drop(columns=['Tag']).resample('15T').mean()
sensor_analysis.plot_box(df=valid_dataframe)
```

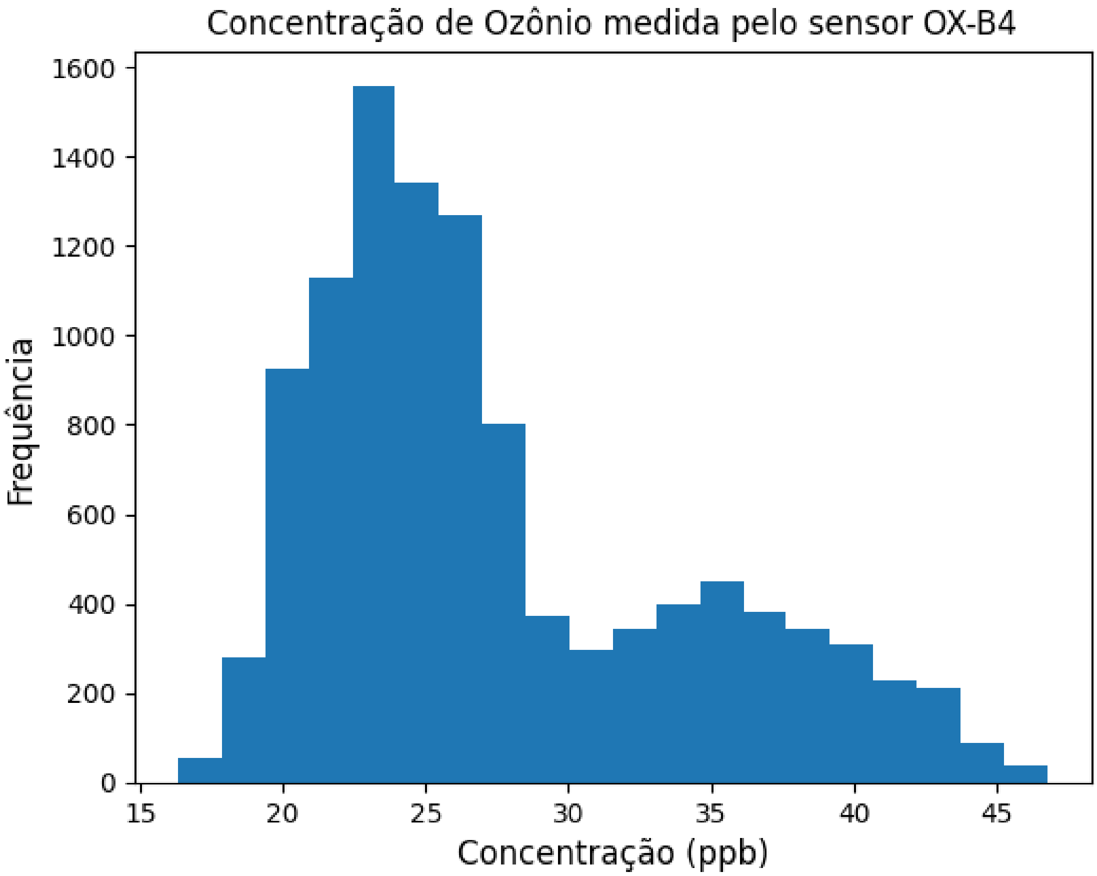
```
/Users/Fernando/Documents/Projects/Github/lcqar-low-cost-monit-proc/data-pre-
processing/SensorDataAnalysisService.py:143: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Hour'] = df['Hour'].astype('int64')
```



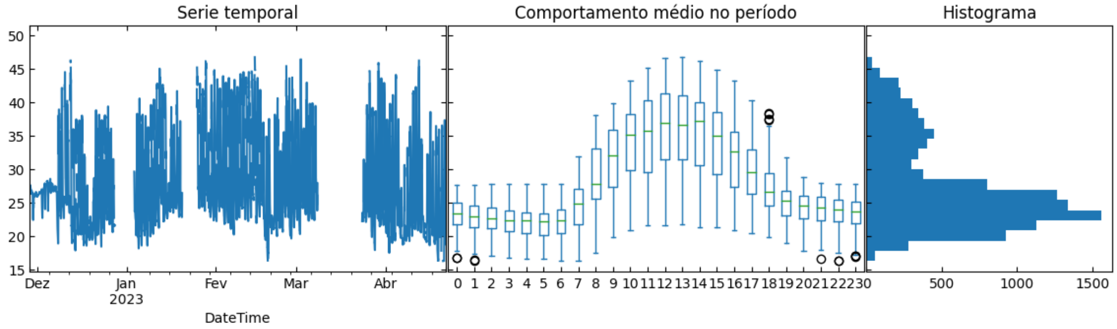
[ ]:

```
fig, ax = plt.subplots(figsize=(1.3*5,5))
(sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['measuring'].
 ↪rename('Concentração de Ozônio')
 .resample('15T').mean().hist(bins=20))
plt.title('Concentração de Ozônio medida pelo sensor OX-B4')
ax.set_xlabel('Concentração (ppb)', fontsize=12)
ax.set_ylabel('Frequência', fontsize=12)
ax.grid(False)
```



```
[ ]: valid_dataframe = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'].
       ↪drop(columns=['Tag']).resample('15T').mean()
     sensor_analysis.plot_box_hist(df=valid_dataframe, bins=20)
```

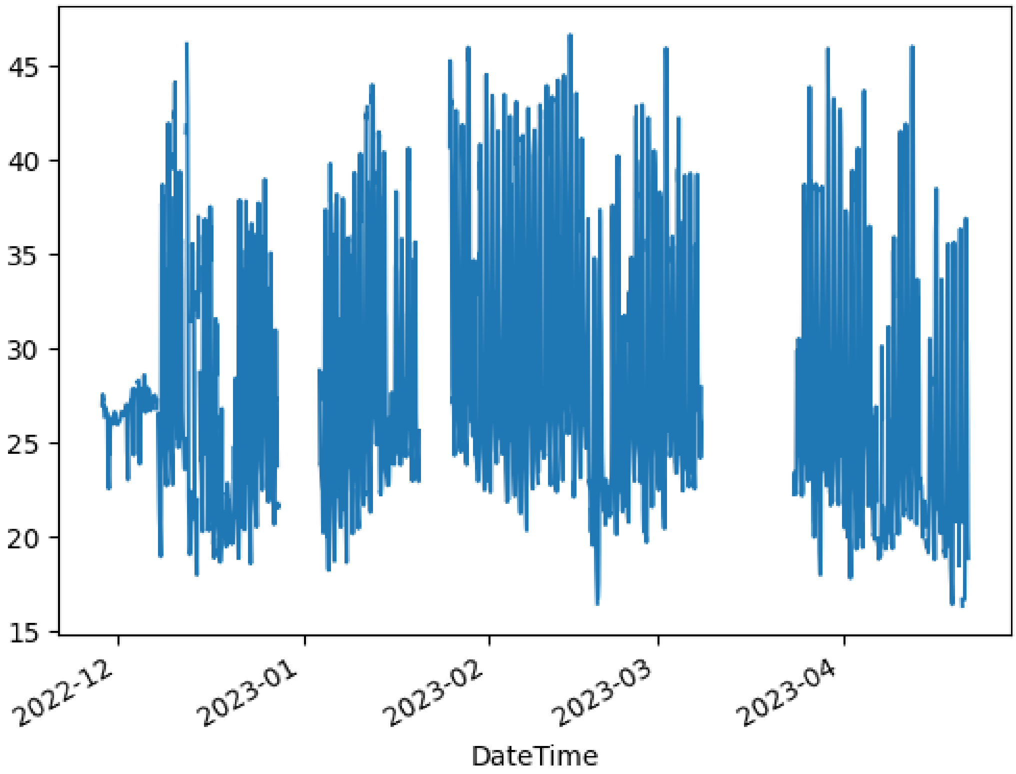
```
/Users/Fernando/Documents/Projects/Github/lcqar-low-cost-monit-proc/data-pre-
processing/SensorDataAnalysisService.py:111: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Hour'] = df['Hour'].astype('int64')
```



2.5 Resample valid data to 1 HR

```
[ ]: resampled_dataframe = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'].
       ↪drop(columns=['Tag']).resample('H').mean()
     resampled_dataframe['Hour'] = resampled_dataframe.index.hour
     resampled_dataframe['Count'] = (valid_dataframe['measuring'].resample('H').
       ↪count())
     resampled_dataframe['Std'] = (valid_dataframe['measuring'].resample('H').std())
     resampled_dataframe.index = resampled_dataframe.index.map(lambda t: t.
       ↪replace(minute=30, second=0))
     resampled_dataframe['measuring'].plot()
```

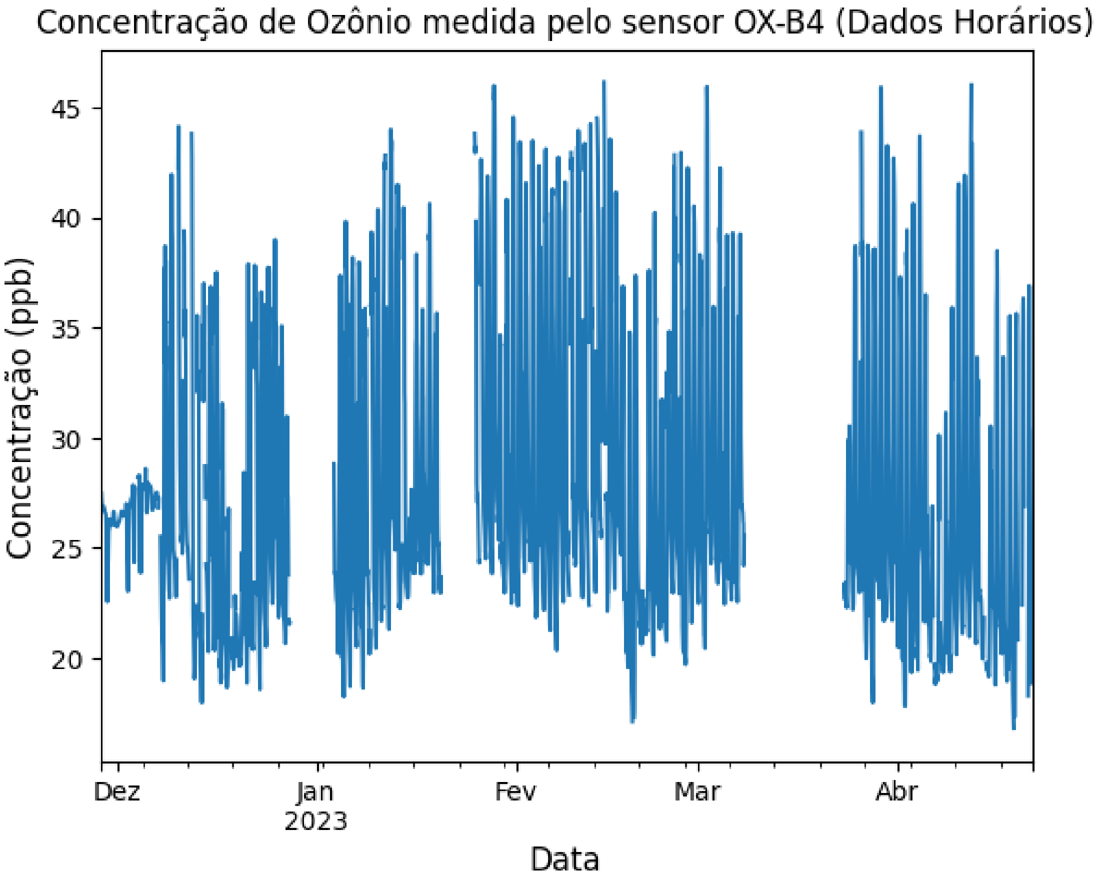
```
[ ]: <Axes: xlabel='DateTime'>
```



Tag hourly data according to the number of samples in an hour At least 3 samples must be valid in an hour (75 %) for the hourly data be considered as valid

```
[ ]: original_freq = sensor_dataframe.index.freq
     resampled_dataframe['% valid'] = (resampled_dataframe['Count']
                                        .map(lambda c:
                                             c / (pd.Timedelta("1 hour") /␣
       ↪original_freq) * 100))
     resampled_dataframe['Tag'] = (resampled_dataframe['% valid']
                                   .map(lambda c: 'VALID' if c >= 75 else␣
       ↪'LOWSAMPLES'))

     fig, ax = plt.subplots(figsize=(1.3*5,5))
     resampled_dataframe[resampled_dataframe['Tag'] == 'VALID']['measuring'].
       ↪rename('Concentração de Ozônio').resample('1H').mean().plot()
     plt.title('Concentração de Ozônio medida pelo sensor OX-B4 (Dados Horários)')
     ax.set_xlabel('Data', fontsize=12)
     ax.set_ylabel('Concentração (ppb)', fontsize=12)
```

```
[ ]: Text(0, 0.5, 'Concentração (ppb)')
```



Valid data contabilization

```
[ ]: tags = ['LOWSAMPLES','VALID']
     data_contabilization = sensor_analysis.count_tags(tags, resampled_dataframe)
     data_contabilization
```

```
[ ]:                #          %
     LOWSAMPLES   783  22.577855
     VALID       2685  77.422145
     TOTAL       3468      100.0
```

3 Save Data

```
[ ]: import os

     output_directory_path = 'data/output/'
     processing_directory_path = '../data-processing/input/'
     valid_1HR_filename = '_valid_data_1HR.csv'
     valid_filename = '_valid_data.csv'
```

```
dataframe_filename = '_dataframe.csv'
dataframe_1HR_filename = '_dataframe_1hr.csv'

valid_1HR_file_path_output = output_directory_path + SENSOR_NAME +␣
 ↪valid_1HR_filename
valid_file_path_output = output_directory_path + SENSOR_NAME + valid_filename

dataframe_path_1HR_output = output_directory_path + SENSOR_NAME +␣
 ↪dataframe_1HR_filename
dataframe_path_output = output_directory_path + SENSOR_NAME + dataframe_filename

valid_processing_1HR_file_path = processing_directory_path + SENSOR_NAME +␣
 ↪valid_1HR_filename
valid_processing_file_path = processing_directory_path + SENSOR_NAME +␣
 ↪valid_filename

processing_dataframe_path_1HR = processing_directory_path + SENSOR_NAME +␣
 ↪dataframe_1HR_filename
processing_dataframe_path = processing_directory_path + SENSOR_NAME +␣
 ↪dataframe_filename

if not os.path.exists(output_directory_path):
    os.makedirs(output_directory_path)

if not os.path.exists(processing_directory_path):
    os.makedirs(processing_directory_path)

sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'][['measuring', 'value']].
 ↪to_csv(valid_file_path_output)
resampled_dataframe[resampled_dataframe['Tag'] == 'VALID'][['measuring',␣
 ↪'value']].to_csv(valid_1HR_file_path_output)
sensor_dataframe.to_csv(dataframe_path_output)
resampled_dataframe.to_csv(dataframe_path_1HR_output)

sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'][['measuring', 'value']].
 ↪to_csv(valid_processing_file_path)
resampled_dataframe[resampled_dataframe['Tag'] == 'VALID'][['measuring',␣
 ↪'value']].to_csv(valid_processing_1HR_file_path)
sensor_dataframe.to_csv(processing_dataframe_path)
resampled_dataframe.to_csv(processing_dataframe_path_1HR)
```