NO2-field-data-treatment-Diamante

February 13, 2024

1 Electrochemical NO2 sensor data preprocessing

- Pollutant: Nitrogen Dioxide
- Sensor: Alphasense NO-B43F

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_NO2.CSV"

SENSOR_FILE_DIR = 'data/input/'
SENSOR_NAME = 'alpha_no2_conc'
SENSOR_FILE_NAME = SENSOR_NAME + 'web_dataframe.csv'
```

1.1.3 Sensor Constants

[ ]:
```
M = 46.0055
lower_limit=15.0
upper_limit=20e3
t_90 = 80      # sensor takes 30 seconds to reach a value of 10e3
t_90_value = 2e3
sampling_period = 15 * 60
```

2 Alphasense NO2 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/venv/lib/python3.9/site-packages/urllib3/__init__.py:34: NotOpenSSLWarning:
urllib3 v2.0 only supports OpenSSL 1.1.1+, currently the 'ssl' module is
compiled with 'LibreSSL 2.8.3'. See:
https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
/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	101.39	2022-11-20 13:46:27
1	-28.456899	-48.972999	103.08	2022-11-21 10:38:49
2	-28.456899	-48.972999	104.34	2022-11-21 10:54:36
3	-28.456899	-48.972999	100.55	2022-11-21 11:10:24
4	-28.456899	-48.972999	89.67	2022-11-21 11:26:10

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	101.39	2022-11-20 13:46:27
1	1	-28.456899	-48.972999	103.08	2022-11-21 10:38:49
2	2	-28.456899	-48.972999	104.34	2022-11-21 10:54:36
3	3	-28.456899	-48.972999	100.55	2022-11-21 11:10:24
4	4	-28.456899	-48.972999	89.67	2022-11-21 11:26:10

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.head()
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_4206/2771449038.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))
```

[ ]:

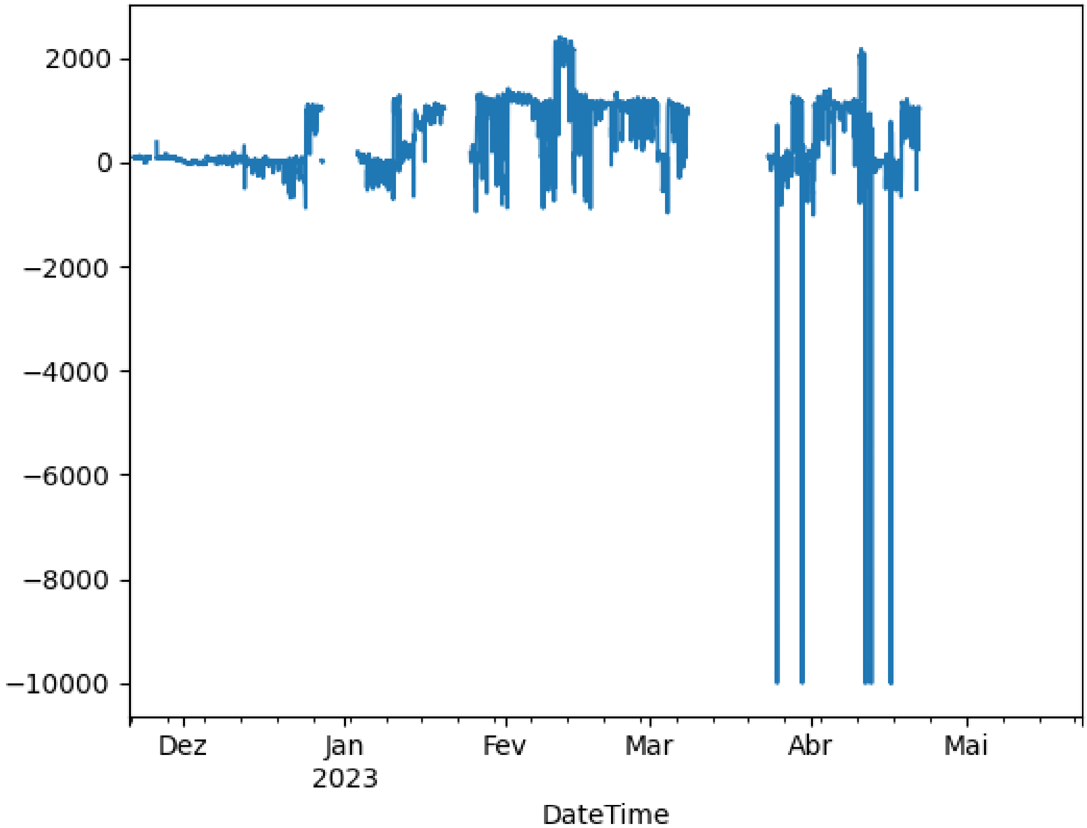
DateTime	latitude	longitude	measuring
2022-11-20 13:45:00	-28.456899	-48.972999	101.39
2022-11-20 14:00:00	NaN	NaN	NaN
2022-11-20 14:15:00	NaN	NaN	NaN
2022-11-20 14:30:00	NaN	NaN	NaN
2022-11-20 14:45:00	NaN	NaN	NaN

2.1.2 Plot raw data

[ ]:

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

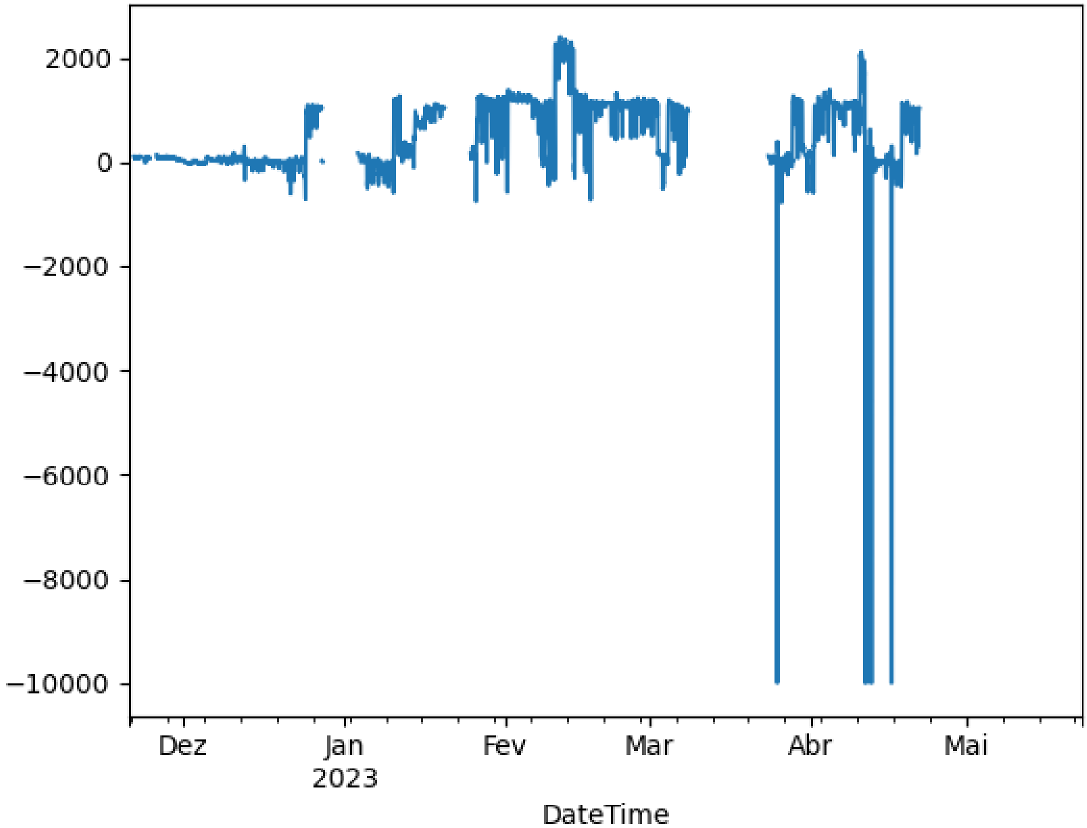
[ ]: <Axes: xlabel='DateTime'>



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'
```

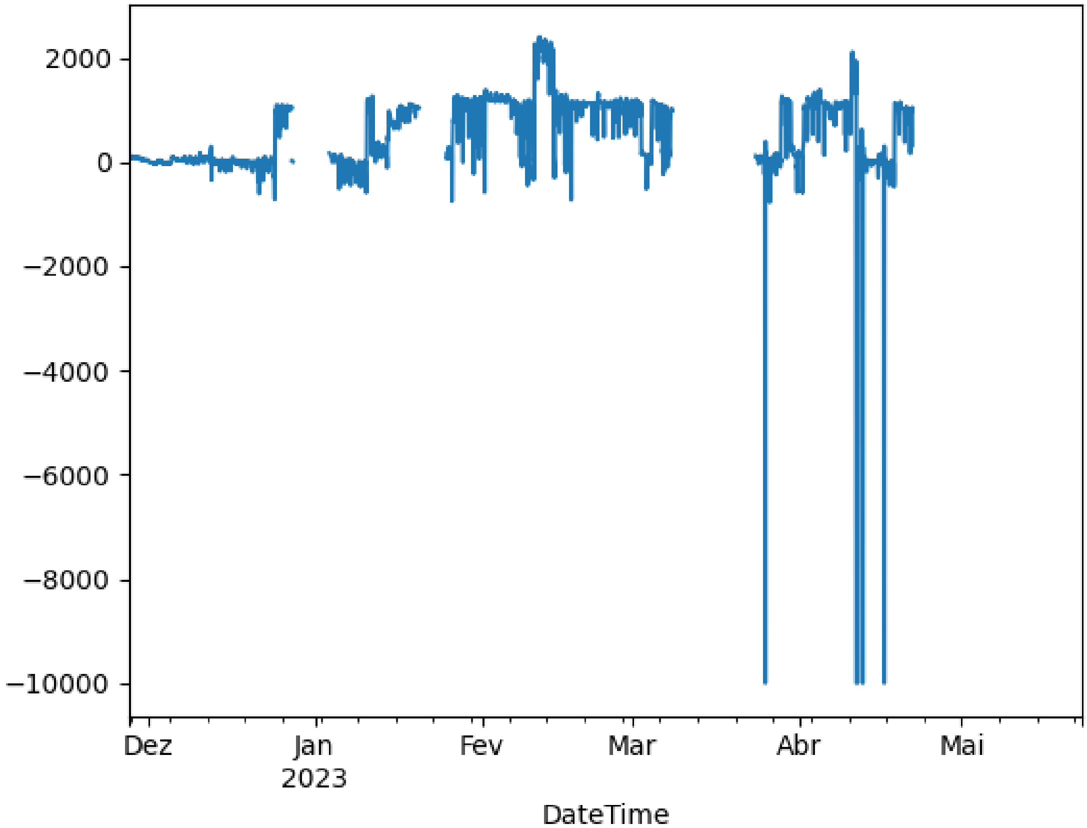
Tag data during stabilizing period AQMesh recomends to considering the first two days after installation as a period for stabilization. In our case, a period of seven days was selected in order to remove outliers candidates detected during that period. For that purpose we detect the first 7 days of uninterrupted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

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

```python
# 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()
```

[ ]:

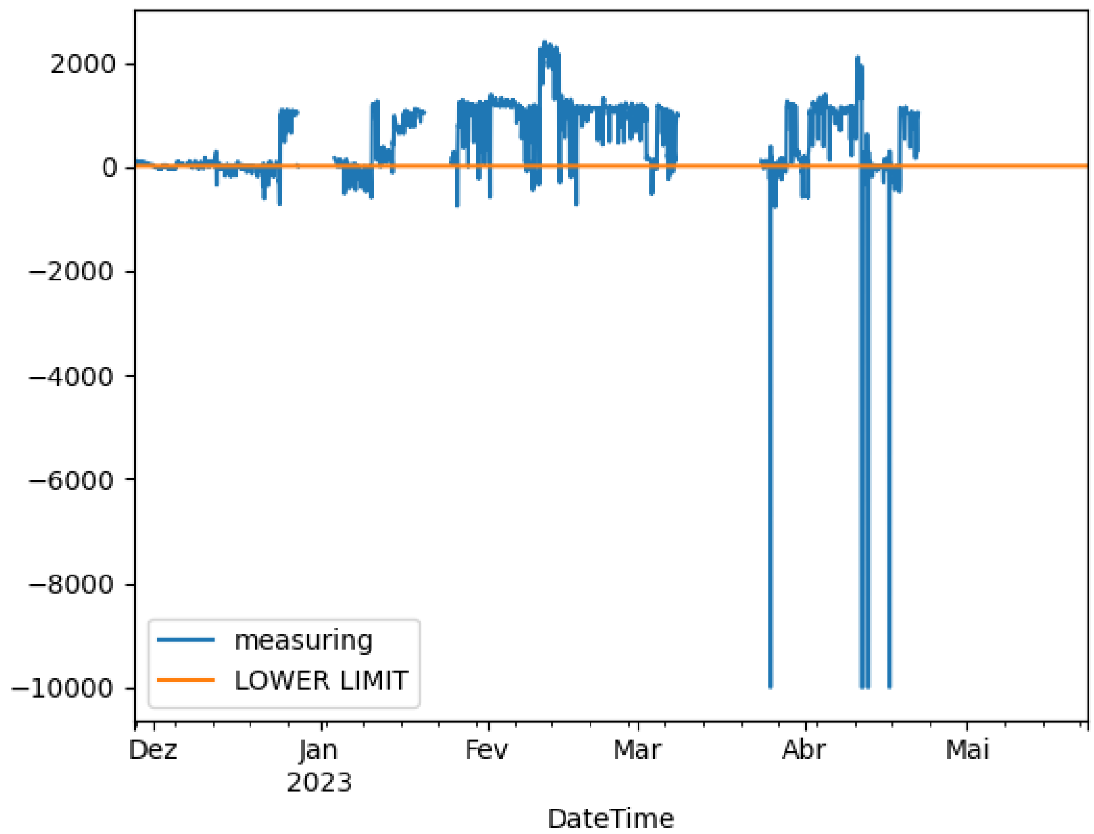
```
                      latitude  longitude  measuring  raw measuring  Hour  \
DateTime
2022-11-20 13:45:00 -28.456899 -48.972999        NaN         101.39    13
2022-11-20 14:00:00        NaN        NaN        NaN            NaN    14
2022-11-20 14:15:00        NaN        NaN        NaN            NaN    14
2022-11-20 14:30:00        NaN        NaN        NaN            NaN    14
2022-11-20 14:45:00        NaN        NaN        NaN            NaN    14

                     value          Tag
DateTime
2022-11-20 13:45:00    NaN  STABILIZING
2022-11-20 14:00:00    NaN  STABILIZING
2022-11-20 14:15:00    NaN  STABILIZING
2022-11-20 14:30:00    NaN  STABILIZING
2022-11-20 14:45:00    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'>
```

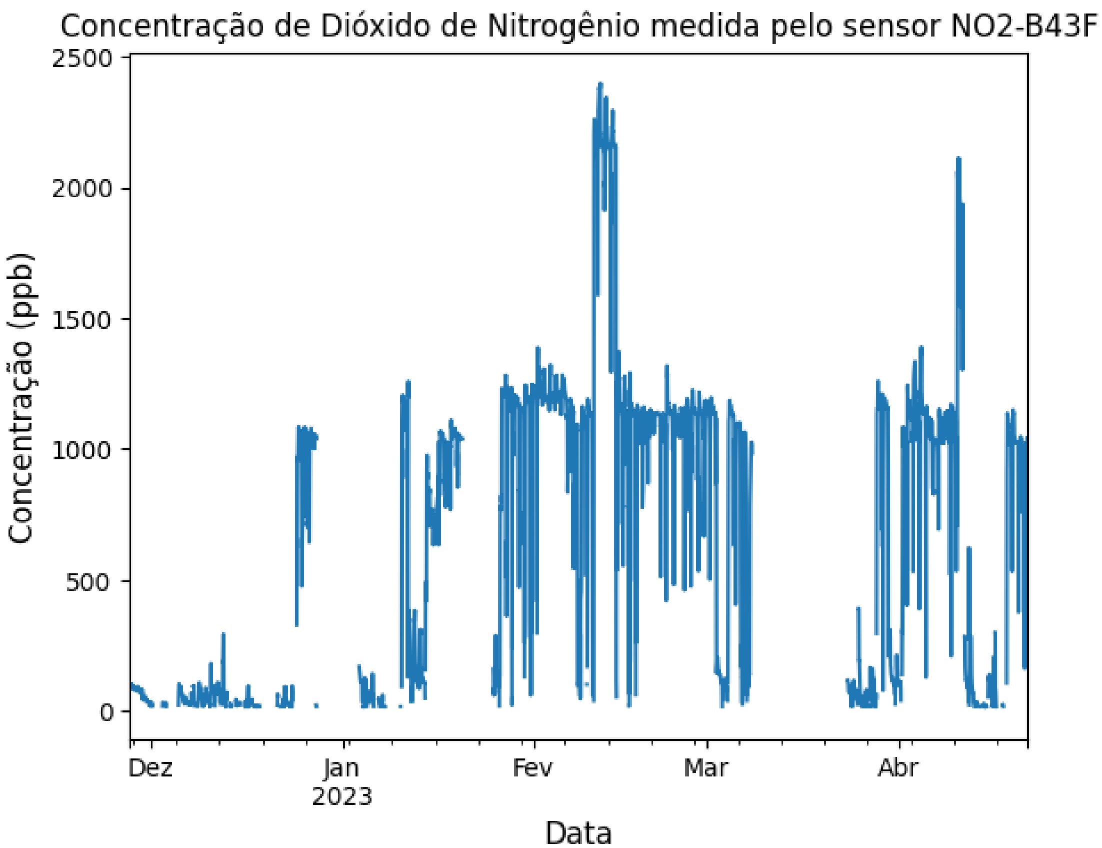



```
[ ]: import numpy as np
from SensorDataAnalysisService import SensorDataAnalysisService as␣
 ↪sensor_analysis

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 Dióxido de Nitrogênio').resample('15T').mean().plot()
plt.title('Concentração de Dióxido de Nitrogênio medida pelo sensor NO2-B43F')
ax.set_xlabel('Data', fontsize=12)
ax.set_ylabel('Concentração (ppb)', fontsize=12)
```

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



2.3 Change point Analysis

2.3.1 Change point analysis of 15 mins data

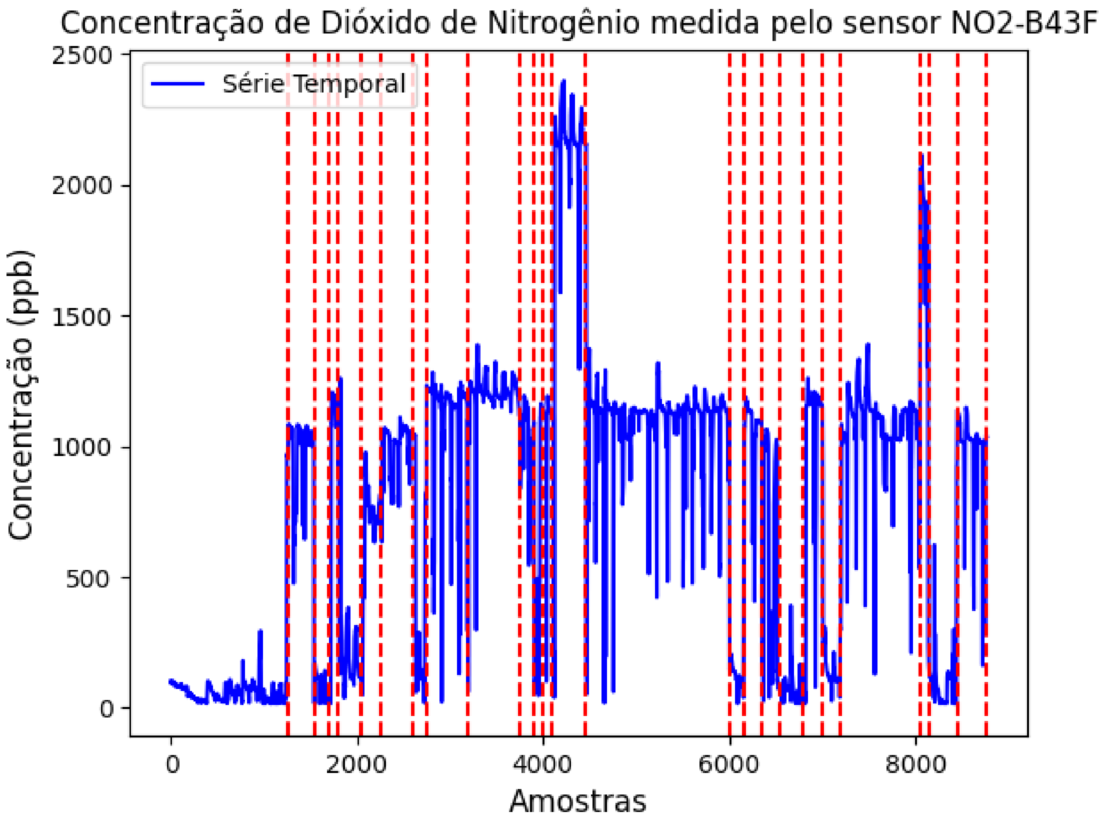
```
[ ]: import ruptures as rpt

series = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['measuring']
signal = np.array(series).reshape(-1, 1)
algo = rpt.Pelt(model="rbf", jump=50, min_size=100).fit(signal=signal)
```

```
result = algo.predict(pen=10)
```

[ ]:
```
change_point_index = series[[x - 1 for x in result]].index
sensor_dataframe['CHANGE POINT'] = False
sensor_dataframe['CHANGE POINT'].loc[change_point_index] = True
```

```
/var/folders/wc/_83zcrx913j1dqwg4g90kbhh0000gp/T/ipykernel_4206/717246363.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sensor_dataframe['CHANGE POINT'].loc[change_point_index] = True
```

[ ]:
```
fig, ax = plt.subplots(figsize=(1.3*5,5))
plt.plot(signal, label='Série Temporal', color='blue')
plt.title('Concentração de Dióxido de Nitrogênio medida pelo sensor NO2-B43F')
ax.set_xlabel('Amostras', fontsize=12)
ax.set_ylabel('Concentração (ppb)', fontsize=12)

# Plot change points
for point in result:
    plt.axvline(x=point, color='red', linestyle='--')

# Show the plot
plt.legend()
plt.show()
```



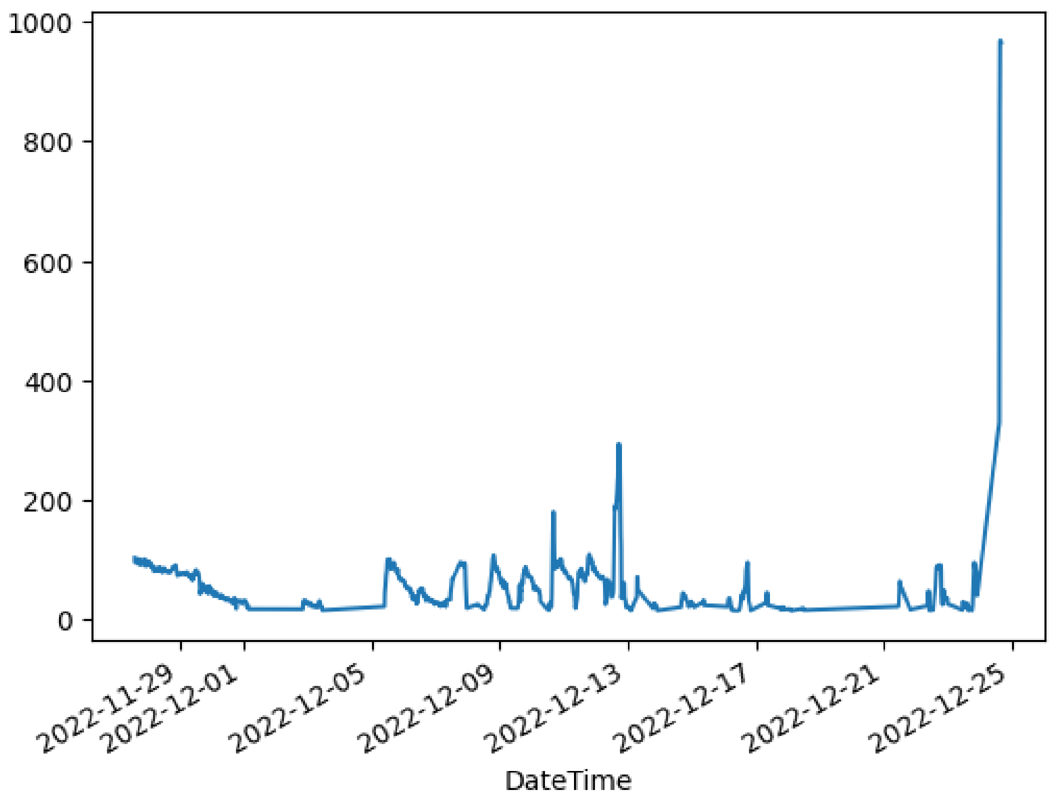
Tag dataframe with changepoints

[ ]:
```
sensor_dataframe.loc[change_point_index[0]:, 'Tag'] = (sensor_dataframe.
 ↪loc[change_point_index[0]:, 'Tag']
                                                            .apply(lambda t:␣
 ↪'REBASE' if t == 'VALID' else t))
sensor_dataframe[sensor_dataframe['Tag'] == 'REBASE'].head()
```

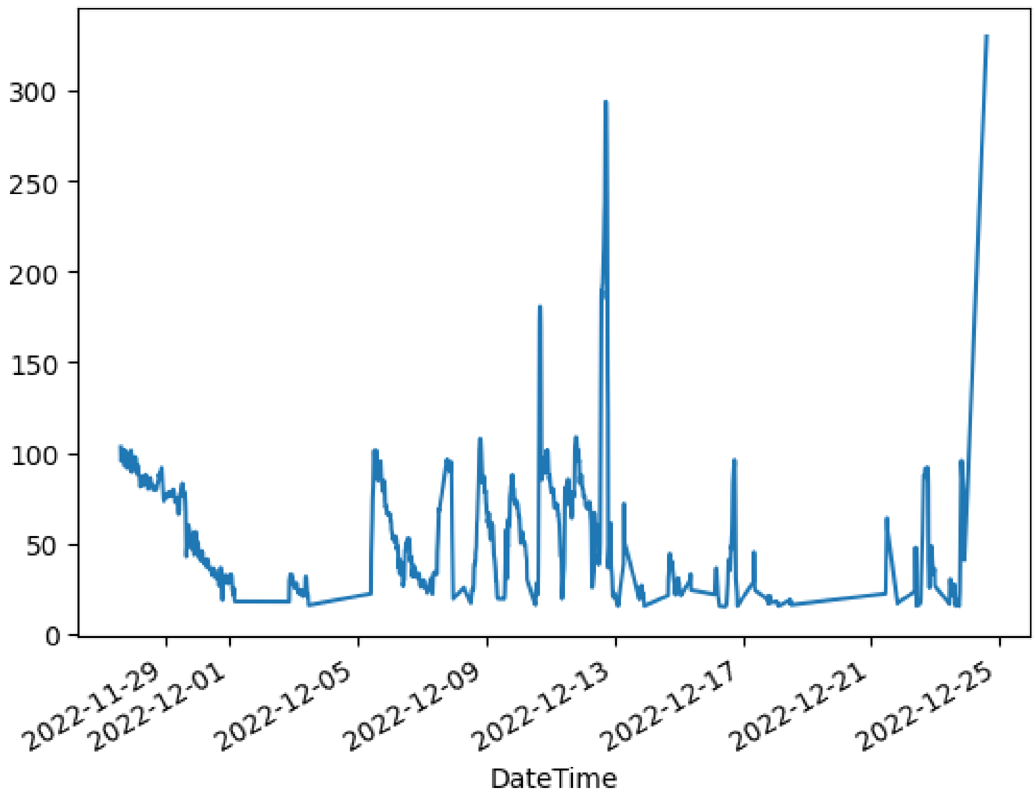
[ ]:
```
                     latitude  longitude  measuring  raw measuring  Hour  \
DateTime
2022-12-24 16:00:00 -28.456899 -48.972999    961.770         960.63    16
2022-12-24 16:15:00 -28.456899 -48.972999    961.770         970.24    16
2022-12-24 16:30:00 -28.456899 -48.972999    966.575         972.52    16
2022-12-24 16:45:00 -28.456899 -48.972999    968.470         966.70    16
2022-12-24 17:00:00 -28.456899 -48.972999    971.380         972.78    17

                           value     Tag  UPPER LIMIT  LOWER LIMIT  \
DateTime
2022-12-24 16:00:00  1809.690428  REBASE      20000.0         15.0
2022-12-24 16:15:00  1809.690428  REBASE      20000.0         15.0
2022-12-24 16:30:00  1818.731636  REBASE      20000.0         15.0
```

```
2022-12-24 16:45:00  1822.297315  REBASE      20000.0         15.0
2022-12-24 17:00:00  1827.772844  REBASE      20000.0         15.0

                     CHANGE POINT
DateTime
2022-12-24 16:00:00          True
2022-12-24 16:15:00         False
2022-12-24 16:30:00         False
2022-12-24 16:45:00         False
2022-12-24 17:00:00         False
```

[ ]:
```python
sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['measuring'].plot()
valid_dataframe = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']
```



[ ]:
```python
sensor_dataframe['Tag'] = (sensor_dataframe[['Tag', 'measuring']]
                           .apply(lambda df: 'REBASE' if df[0] == 'VALID' and␣
 ↪df[1] > 500 else df[0], axis=1))
sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['measuring'].plot()
valid_dataframe = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']
```



Data Tag contabilization

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

[ ]:

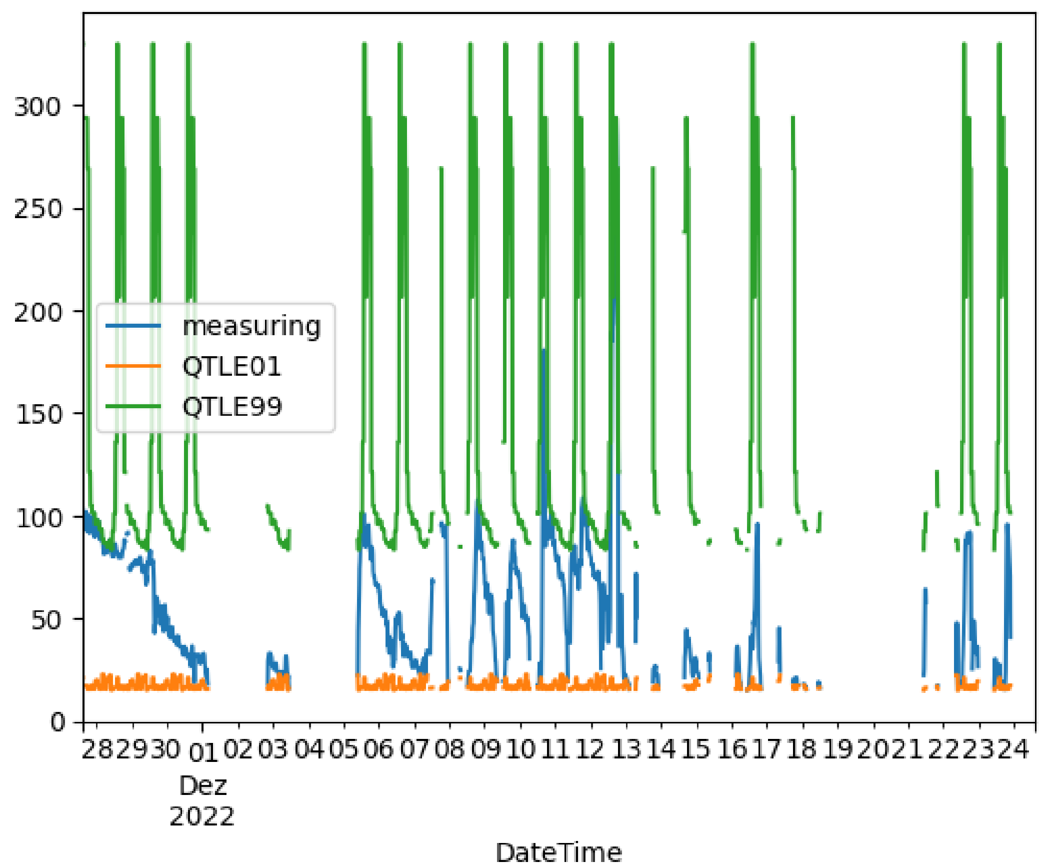
	#	%
MISSING	5767	32.679776
LTLL	2438	13.815379
GTUL	0	0.0
STABILIZING	673	3.813679
BADSPIKE	0	0.0
VALID	1244	7.049357
REBASE	7525	42.641809
TOTAL	17647	100.0

2.3.2 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'>



2.3.3 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
```

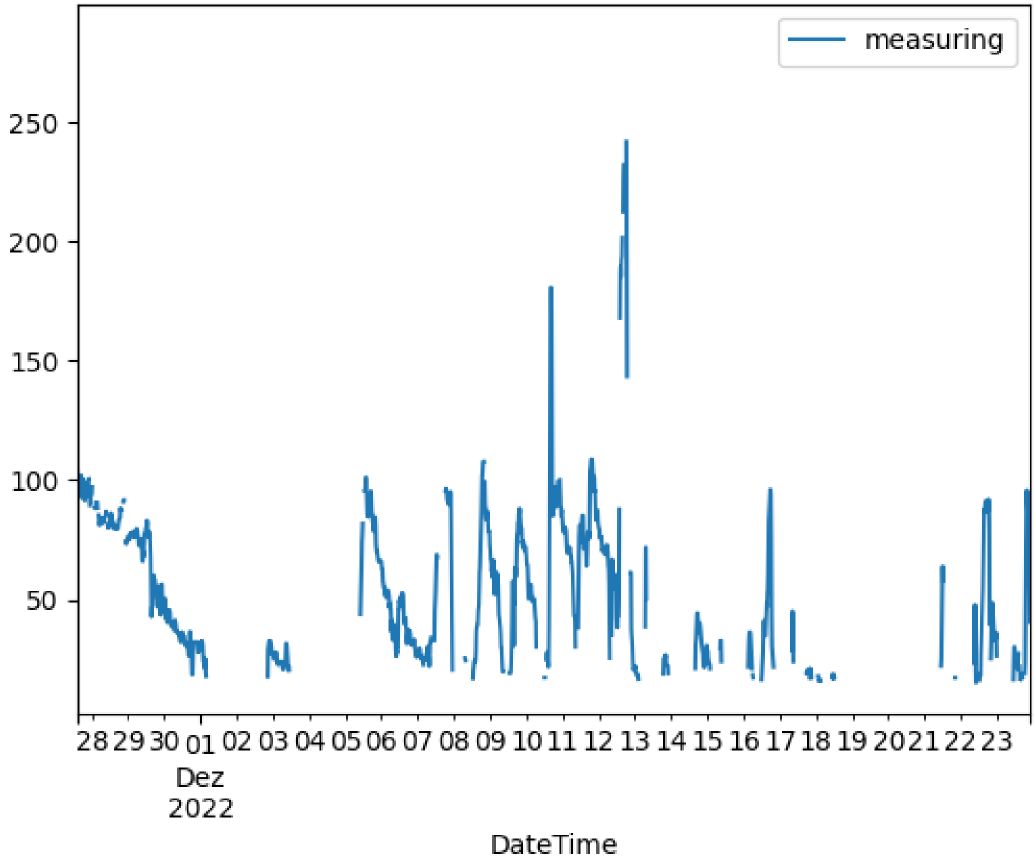
[ ]:

	#	%
MISSING	5767	32.679776
LTLL	2438	13.815379
GTUL	0	0.0
BADSPIKE	0	0.0
VALID	1176	6.664022
LTQTLE01	32	0.181334
GTQTLE99	36	0.204001
TOTAL	17647	100.0

Plot valid data

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

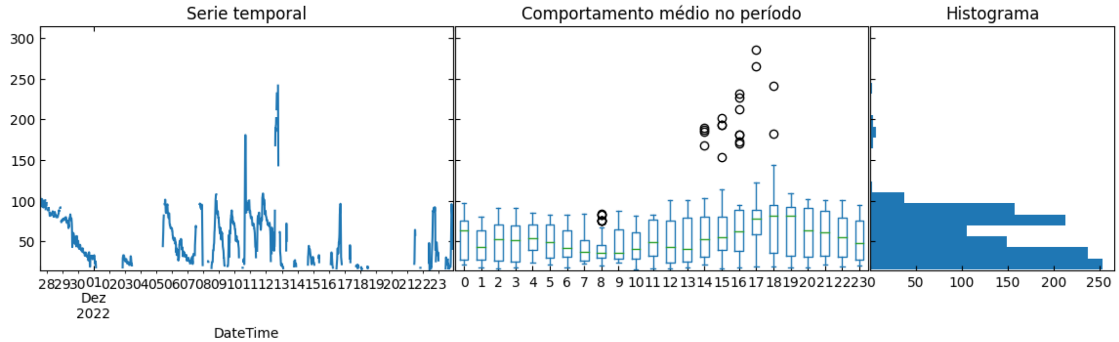
[ ]: <Axes: xlabel='DateTime'>



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 valid data

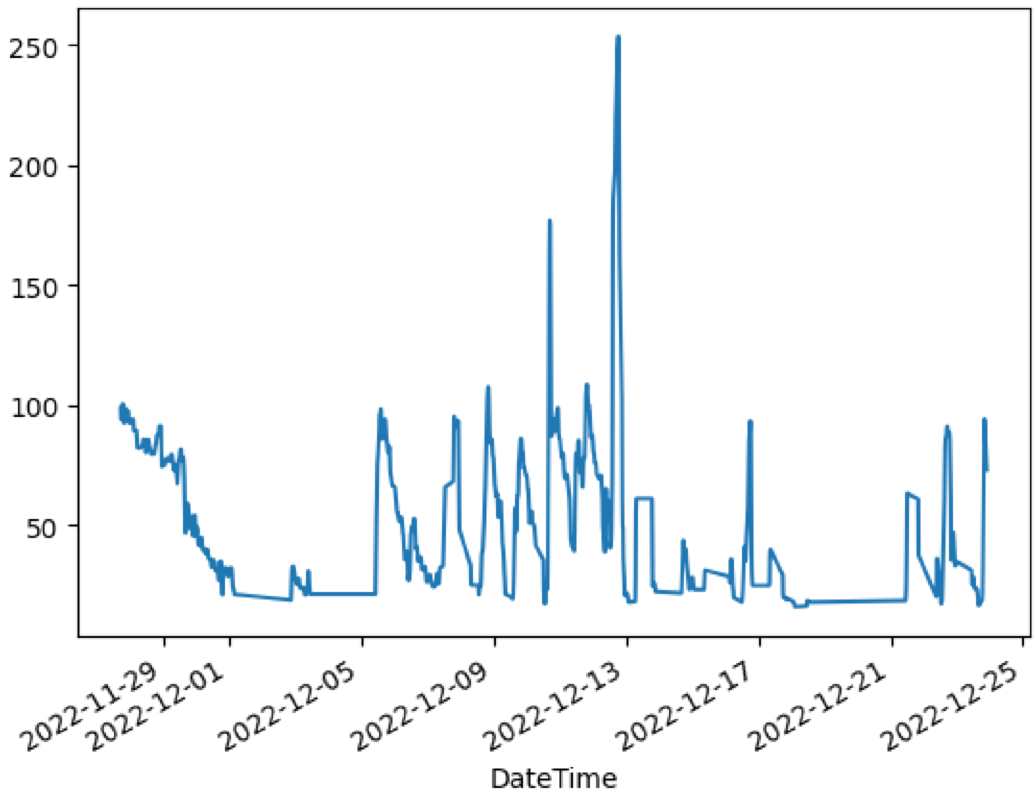
The graph shows the time series of the valid data, box plots of the data grouped by hour of the day, and histogram of the data.

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

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

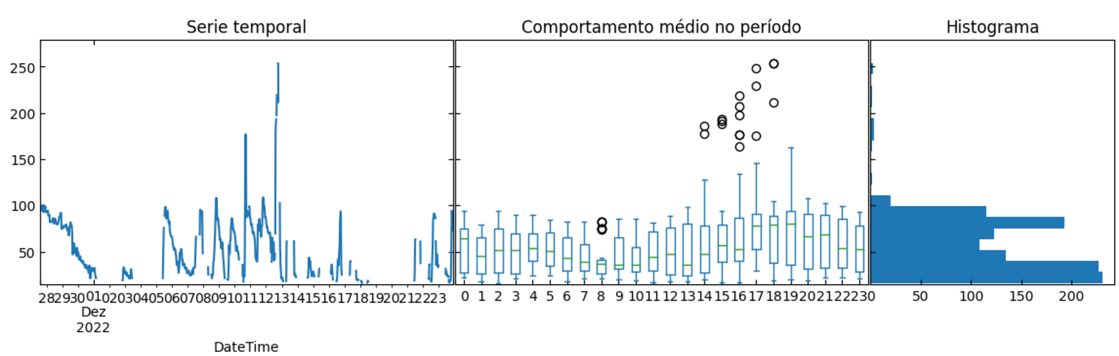
[ ]:
```
valid_dataframe['raw measuring'] = valid_dataframe['measuring']
valid_dataframe['measuring'] = valid_dataframe['measuring'].rolling(window=4).
  ↪median()
valid_dataframe['measuring'].plot()
```

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




```
[ ]: 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')
```



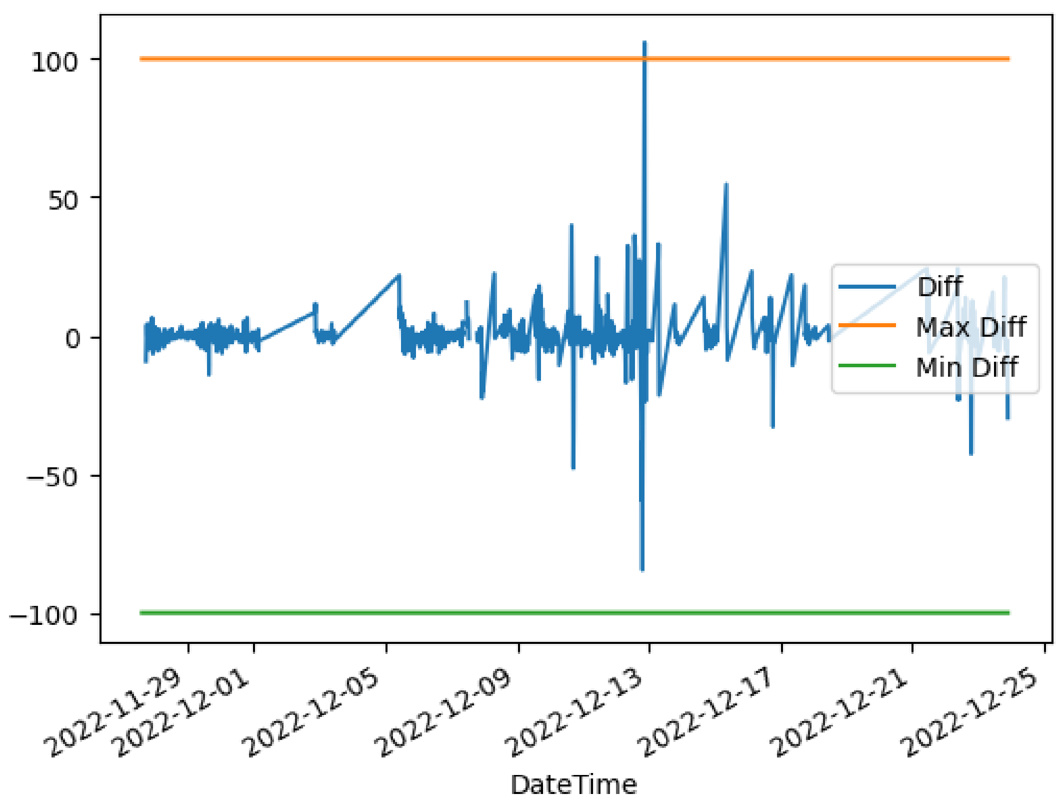
2.5 Analyse data derivatives

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 100 ppb was selected based on the maximum derivative found in the reference data. The maximum derivative (with measuring period of 1 hour) found was 30 ppb. Therefore, a maximum of 100 ppb was considered appropriate for a 15 mins period.

[ ]:
```
max_diff_value = 100
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))
```

2.5.1 Data Tag contabilization

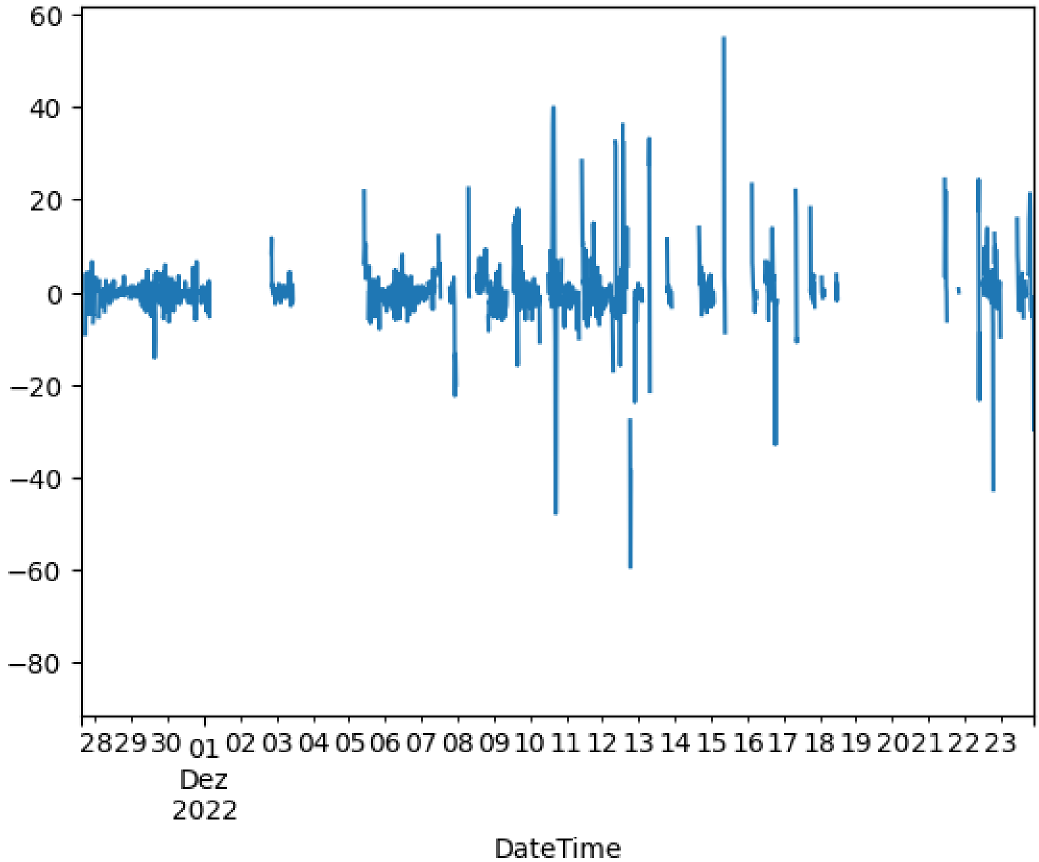
[ ]:
```
tags = ['MISSING','LTLL', 'GTUL', 'STABILIZING', 'BADSPIKE', 'VALID']
data_contabilization = sensor_analysis.count_tags(tags, sensor_dataframe)
data_contabilization
```

[ ]:

	#	%
MISSING	5767	32.679776
LTLL	2438	13.815379
GTUL	0	0.0
STABILIZING	673	3.813679
BADSPIKE	1	0.005667
VALID	1175	6.658356
TOTAL	17647	100.0

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

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

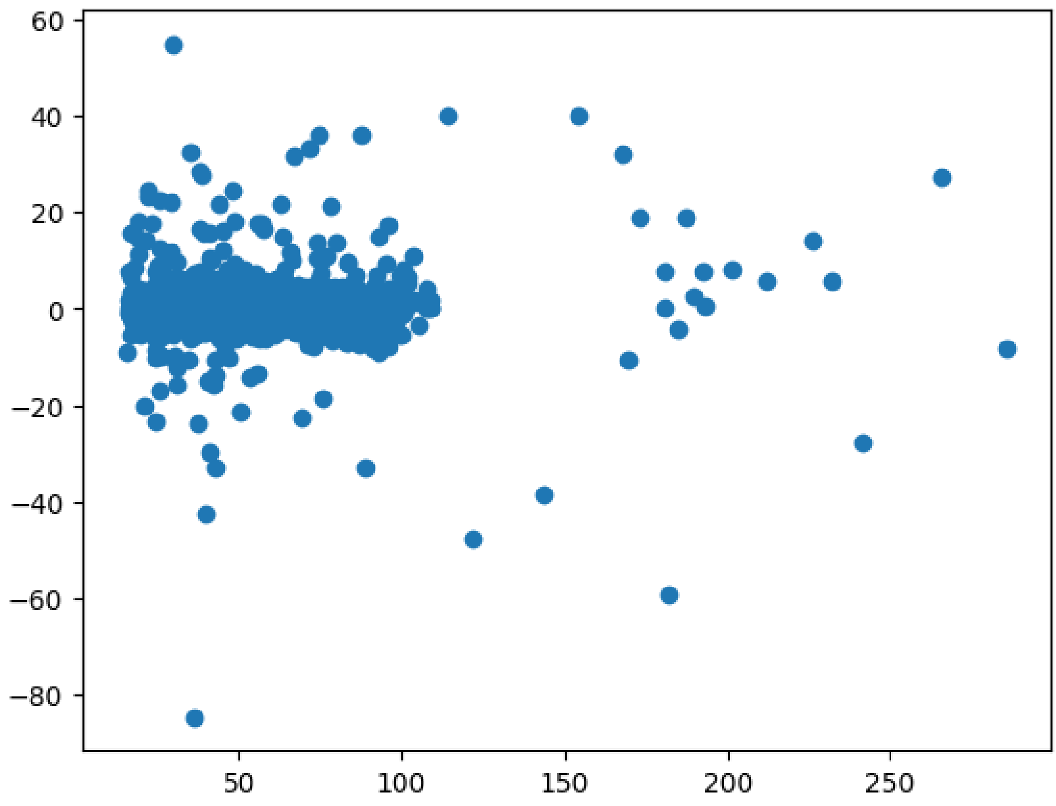


Plot data derivatives vs data

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

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


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

[ ]: <matplotlib.collections.PathCollection at 0x1477f6970>



Data Tag contabilization

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

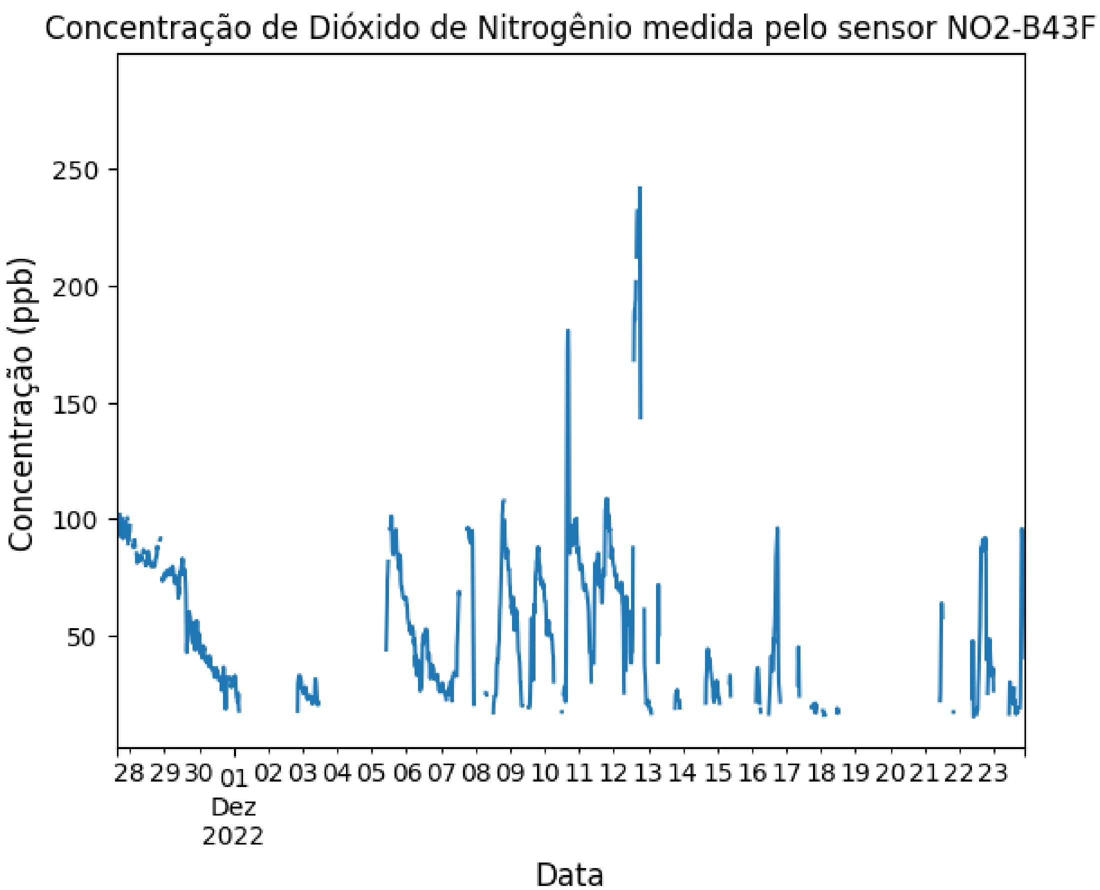
[ ]:

	#	%
MISSING	5767	32.679776
LTLL	2438	13.815379
GTUL	0	0.0
STABILIZING	673	3.813679
BADSPIKE	1	0.005667
VALID	1175	6.658356
LTQTLE01	32	0.181334
GTQTLE99	36	0.204001
REBASE	7525	42.641809
TOTAL	17647	100.0

Plot valid data

[ ]:
```
fig, ax = plt.subplots(figsize=(1.3*5,5))
(sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['measuring']
 .rename('Concentração de Dióxido de Nitrogênio').resample('15T').mean().plot())
plt.title('Concentração de Dióxido de Nitrogênio medida pelo sensor NO2-B43F')
ax.set_xlabel('Data', fontsize=12)
ax.set_ylabel('Concentração (ppb)', fontsize=12)
```

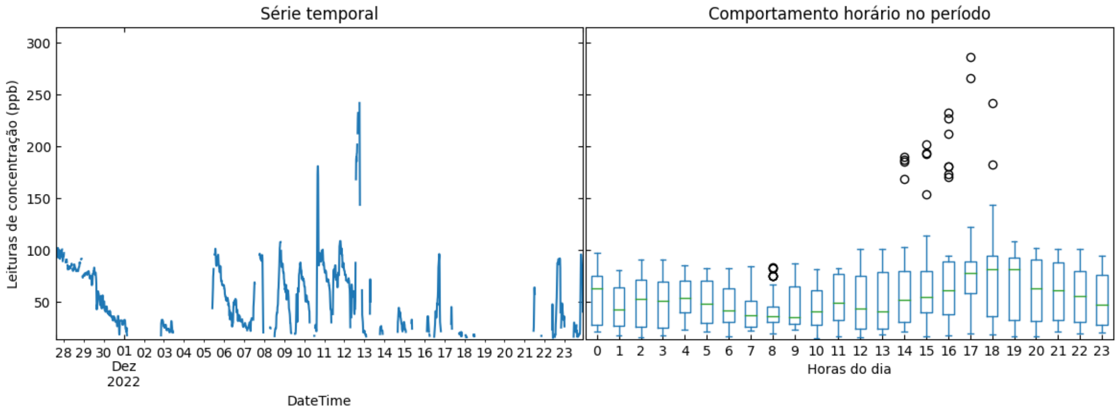
[ ]:
```
Text(0, 0.5, 'Concentração (ppb)')
```



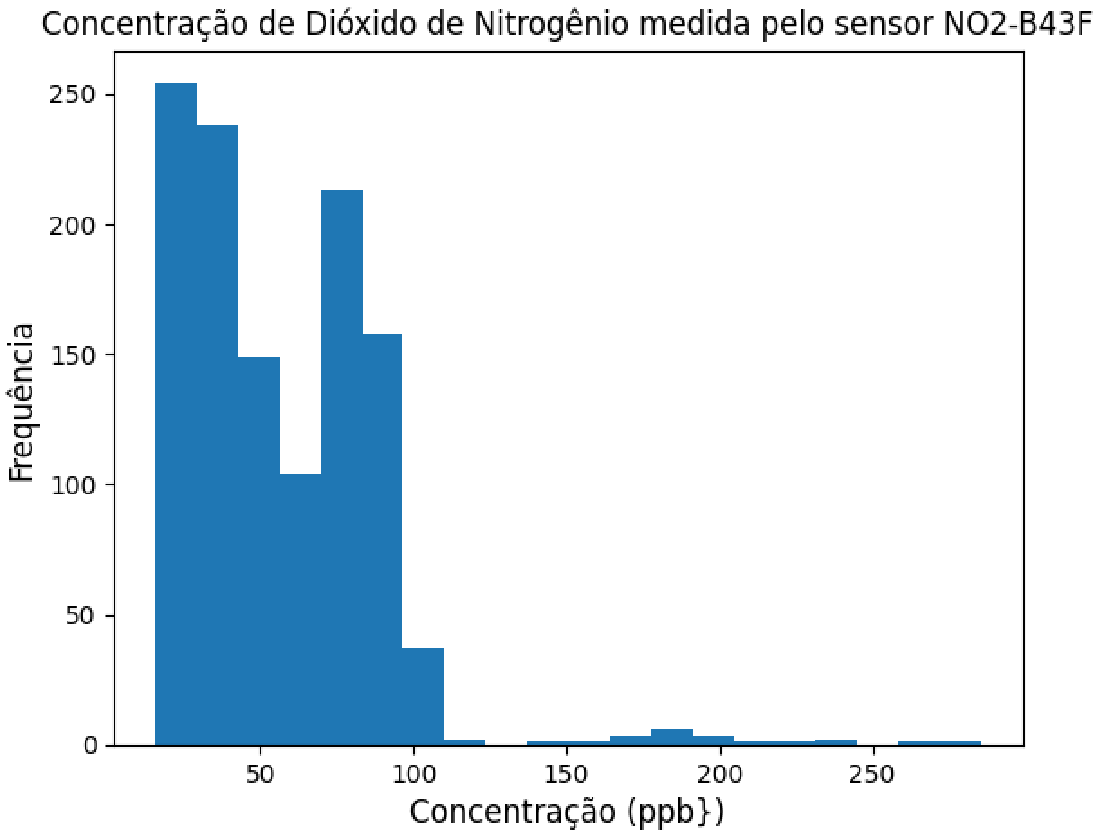
[ ]:
```
valid_dataframe = sensor_dataframe[sensor_dataframe['Tag'] == 'VALID'].
 ↪drop(columns=['Tag']).resample('15T').mean()
sensor_analysis.plot_box(df=valid_dataframe)
```

```
/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')
```



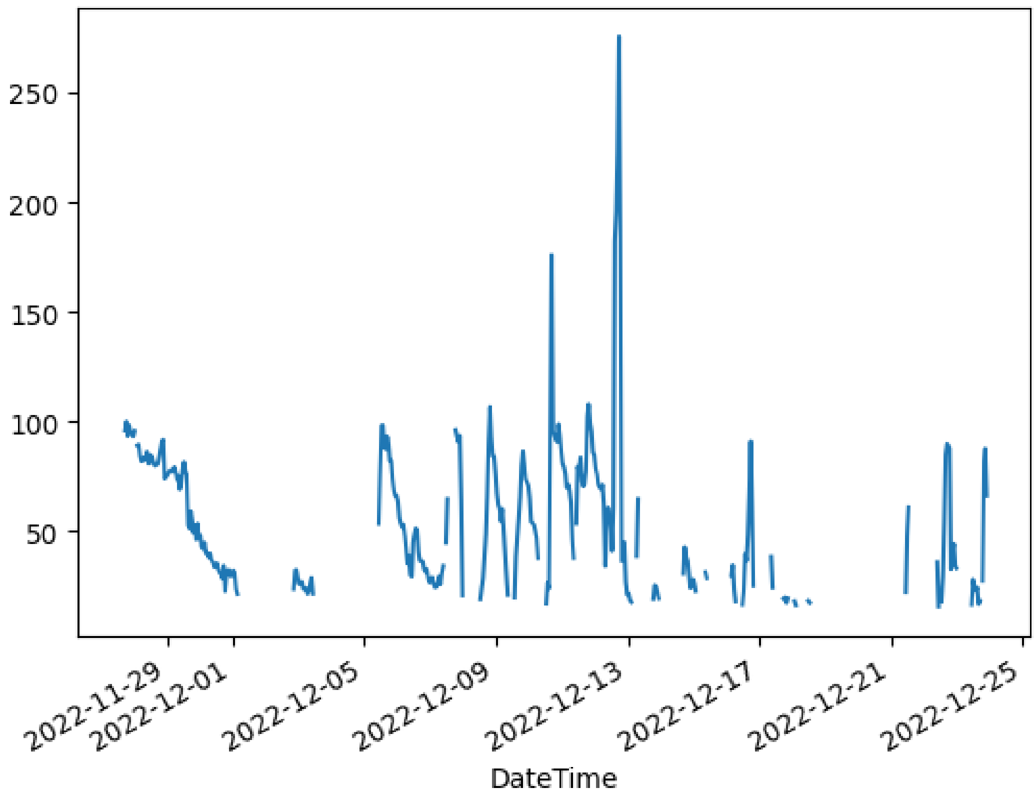
[ ]:
```
fig, ax = plt.subplots(figsize=(1.3*5,5))
(sensor_dataframe[sensor_dataframe['Tag'] == 'VALID']['measuring'].
 ↪rename('Concentração de Dióxido de Nitrogênio')
 .resample('15T').mean().hist(bins=20))
plt.title('Concentração de Dióxido de Nitrogênio medida pelo sensor NO2-B43F')
ax.set_xlabel('Concentração (ppb})', fontsize=12)
ax.set_ylabel('Frequência', fontsize=12)
ax.grid(False)
```



2.6 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()
```

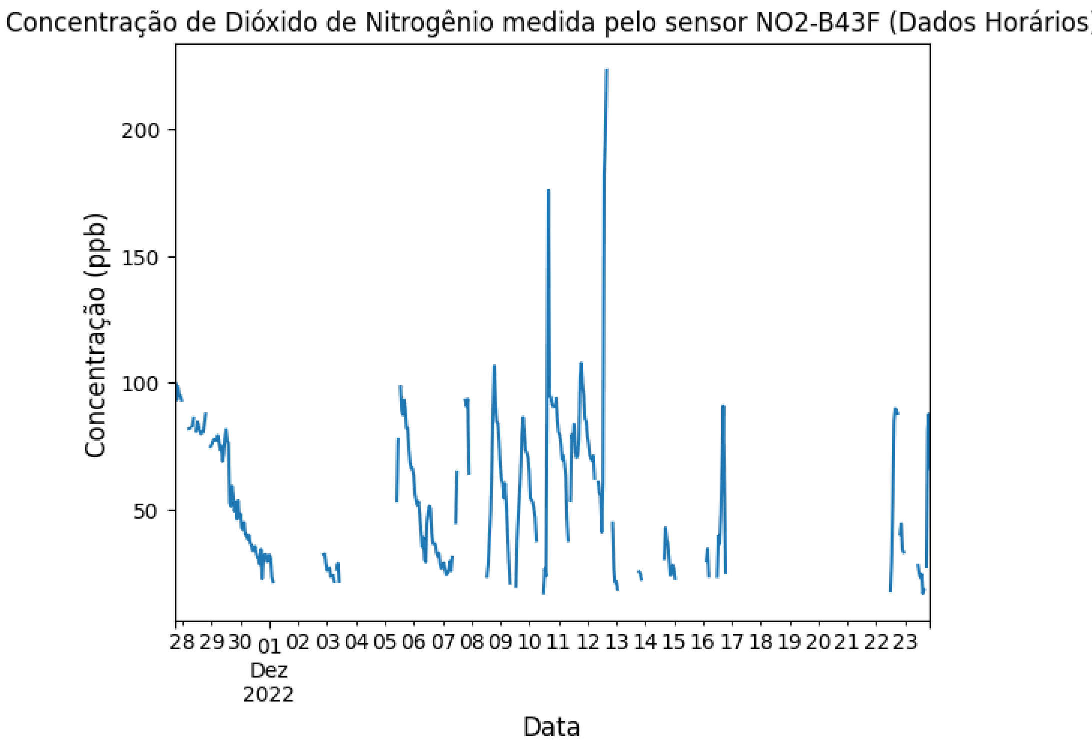
[ ]: <Axes: xlabel='DateTime'>



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 Dióxido de Nitrogênio').resample('1H').mean().plot())
plt.title('Concentração de Dióxido de Nitrogênio medida pelo sensor NO2-B43F␣
 ↪(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
```

[ ]:

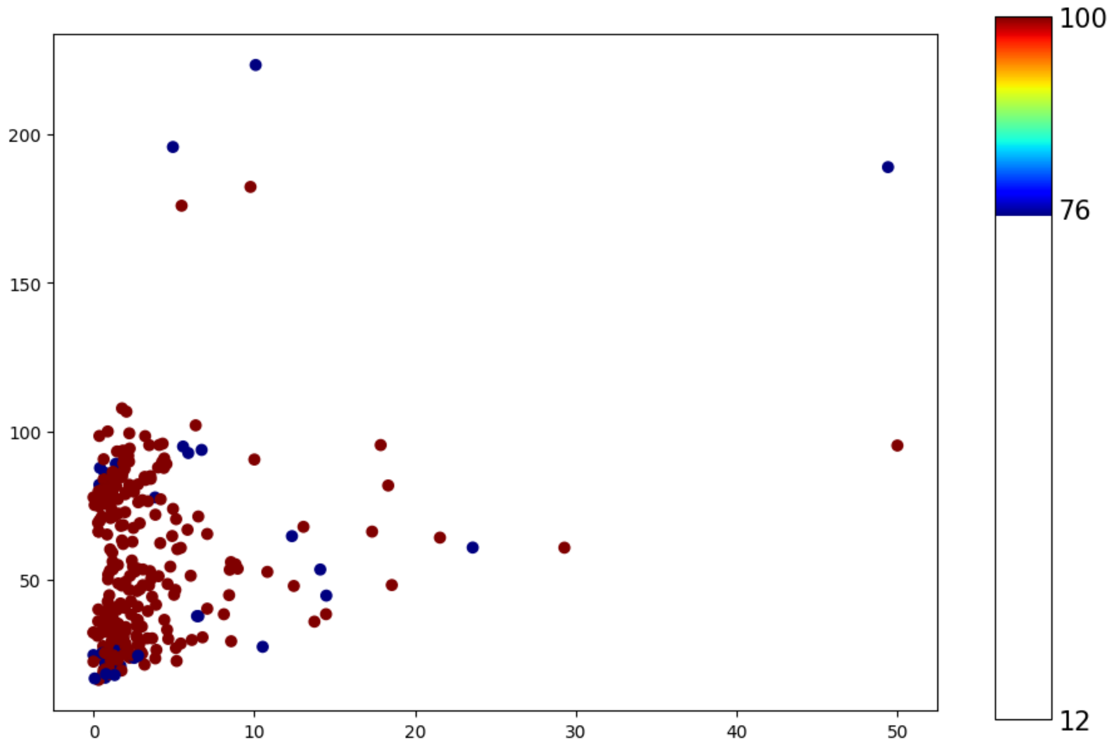
	#	%
LOWSAMPLES	347	54.905063
VALID	285	45.094937
TOTAL	632	100.0

2.7 Analyse the mean and standard deviation of the resampled data

The color of each data point represents the percentage of valid 15 mins samples found in an hour

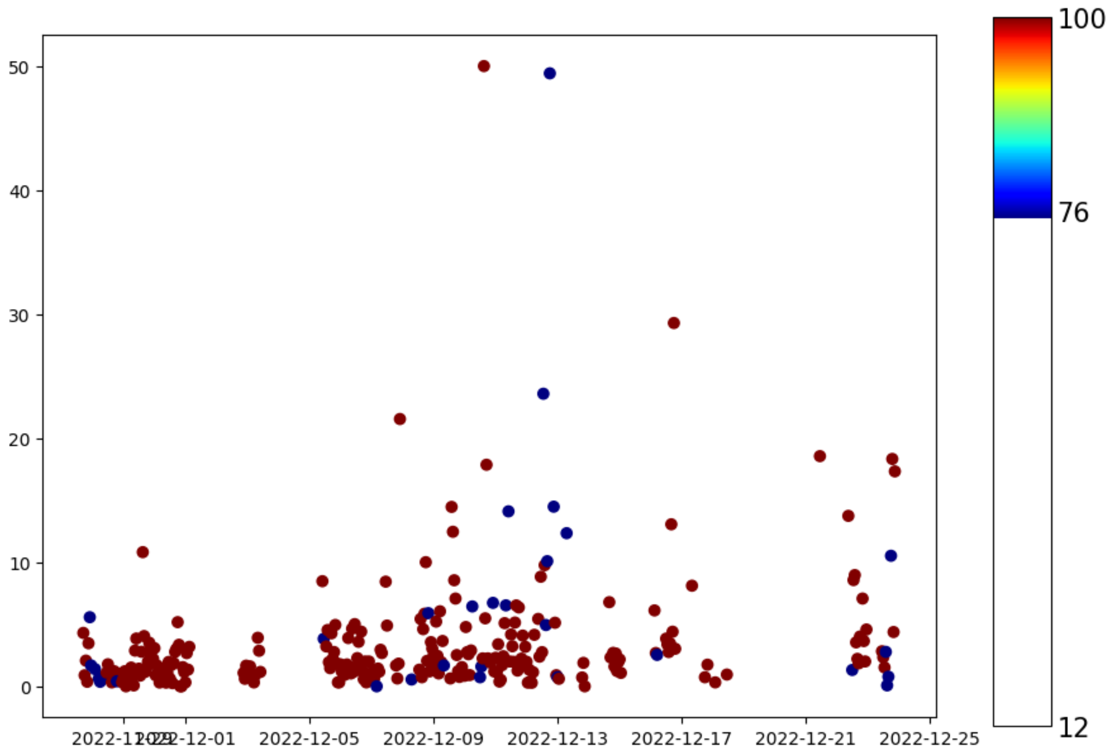
Plot mean (Y axis) vs. standard deviation (X axis) for valid hourly data colored by valid percentage of valid samples

[ ]:
```
valid_resampled_dataframe = resampled_dataframe[resampled_dataframe['Tag'] ==␣
↪'VALID']
sensor_analysis.plot_mean_vs_std(valid_resampled_dataframe)
```



2.7.1 Plot standard deviation in time colored according to the percentage of valid samples

[ ]:
```
sensor_analysis.plot_std_in_time(valid_resampled_dataframe)
```



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)
```