

FOUNDATION OF INFERENCEAL STATISTICS AND AUTOMATION - EE798Q

ASSIGNMENTS

NAME : ANUBHA TRIPATHI

ROLL NO. : 200163

DEPARTMENT : ELECTRICAL

CASE STUDY TOPIC : COAL INDIA OPEN-PIT BLASTING

Blasting from open-pit coal mines causing massive air pollution. The air pollution data set is obtained from the Singrauli Coalfield Pollution Control Board for coal India's (Singrauli Coalfield). The pollution is monitored during open-pit blasting.

We need to perform descriptive, exploratory and inferential statistical data analysis to analyze the give data. There are 13 columns overall in the air pollution data collection of pollutants that are available at intervals of 15 minutes. Coal India open-pit blasting effects on air pollution at time 13:45pm to 14:25pm.

Importing Relevant Libraries

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
# To get the plots on same page
import statsmodels.tsa.stattools as sts
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.seasonal import seasonal_decompose
import scipy.stats as stats
import seaborn as sns
sns.set()
from scipy.interpolate import CubicSpline
from scipy.interpolate import interp1d
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARMA
from scipy.stats import chi2
from statsmodels.tsa.api import VAR
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
import warnings
warnings.filterwarnings("ignore")
```

```
In [2]: path = "Open pit blasting 01-02-2023 000000 To 01-05-2023 235959.csv"
```

```
In [3]: df = pd.read_csv(path)
```

Data Preprocessing

```
In [4]: df.shape
```

```
Out[4]: (8643, 13)
```

In [5]: df.sample(n=5)

Out[5]:

#	From	To (Interval: 15M)	Singrauli, Surya Kiran Bhawan Dudhichua PM10 (µg/m3)	Singrauli, Surya Kiran Bhawan Dudhichua PM2.5 (µg/m3)	Singrauli, Surya Kiran Bhawan Dudhichua NO (µg/m3)	Singrauli, Surya Kiran Bhawan Dudhichua NO2 (µg/m3)	Singrauli, Surya Kiran Bhawan Dudhichua NOX (ppb)	Singrauli, Surya Kiran Bhawan Dudhichua CO (mg/m3)	Singrauli, Surya Kiran Bhawan Dudhichua SO2 (µg/m3)	Singrauli, Surya Kiran Bhawan Dudhichua NH3 (µg/m3)	Singrauli, Surya Kiran Bhawan Dudhichua Ozone (µg/m3)	Singrauli, Surya Kiran Bhawan Dudhichua Benzene (µg/m3)	
1021	1022	11-02-2023 15:15	11-02-2023 15:30	158.0	22.0	4.9	51.9	31.6	0.53	7.5	17.2	56.3	0.1
3585	3586	10-03-2023 08:15	10-03-2023 08:30	85.0	101.0	30.2	57.7	55.2	2.12	40.3	12.5	17.2	NaN
863	864	09-02-2023 23:45	10-02-2023 00:00	528.0	123.0	Nan	56.1	87.8	Nan	Nan	20.2	5.5	0.4
5472	5473	30-03-2023 00:00	30-03-2023 00:15	160.0	82.0	3.8	47.3	28.2	1.66	14.3	10.7	20.5	NaN
5046	5047	25-03-2023 13:30	25-03-2023 13:45	63.0	21.0	3.1	28.3	17.6	1.29	9.3	10.4	65.7	NaN

In [6]: df.columns

Out[6]: Index(['#', 'From', 'To (Interval: 15M)',
'Singrauli, Surya Kiran Bhawan Dudhichua PM10 (µg/m3)',
'Singrauli, Surya Kiran Bhawan Dudhichua PM2.5 (µg/m3)',
'Singrauli, Surya Kiran Bhawan Dudhichua NO (µg/m3)',
'Singrauli, Surya Kiran Bhawan Dudhichua NO2 (µg/m3)',
'Singrauli, Surya Kiran Bhawan Dudhichua NOX (ppb)',
'Singrauli, Surya Kiran Bhawan Dudhichua CO (mg/m3)',
'Singrauli, Surya Kiran Bhawan Dudhichua SO2 (µg/m3)',
'Singrauli, Surya Kiran Bhawan Dudhichua NH3 (µg/m3)',
'Singrauli, Surya Kiran Bhawan Dudhichua Ozone (µg/m3)',
'Singrauli, Surya Kiran Bhawan Dudhichua Benzene (µg/m3)'],
dtype='object')

```
In [7]: df.rename(columns=
    {'Singrauli, Surya Kiran Bhawan Dudhichua PM10 (\u00b5g/m3)': 'PM10 (\u00b5g/m3)',
     'To (Interval: 15M)': 'To',
     'Singrauli, Surya Kiran Bhawan Dudhichua PM2.5 (\u00b5g/m3)': 'PM2.5 (\u00b5g/m3)',
     'Singrauli, Surya Kiran Bhawan Dudhichua NO (\u00b5g/m3)': 'NO (\u00b5g/m3)',
     'Singrauli, Surya Kiran Bhawan Dudhichua NO2 (\u00b5g/m3)': 'NO2 (\u00b5g/m3)',
     'Singrauli, Surya Kiran Bhawan Dudhichua NOX (ppb)': 'NOX (ppb)',
     'Singrauli, Surya Kiran Bhawan Dudhichua CO (mg/m3)': 'CO (mg/m3)',
     'Singrauli, Surya Kiran Bhawan Dudhichua SO2 (\u00b5g/m3)': 'SO2 (\u00b5g/m3)',
     'Singrauli, Surya Kiran Bhawan Dudhichua NH3 (\u00b5g/m3)': 'NH3 (\u00b5g/m3)',
     'Singrauli, Surya Kiran Bhawan Dudhichua Ozone (\u00b5g/m3)': 'Ozone (\u00b5g/m3)',
     'Singrauli, Surya Kiran Bhawan Dudhichua Benzene (\u00b5g/m3)': 'Benzene (\u00b5g/m3)'}
    , inplace=True)
```

```
In [8]: # Tried removing unnecessarily Long column names
df.columns
```

```
Out[8]: Index(['#', 'From', 'To', 'PM10 (\u00b5g/m3)', 'PM2.5 (\u00b5g/m3)', 'NO (\u00b5g/m3)',
   'NO2 (\u00b5g/m3)', 'NOX (ppb)', 'CO (mg/m3)', 'SO2 (\u00b5g/m3)', 'NH3 (\u00b5g/m3)',
   'Ozone (\u00b5g/m3)', 'Benzene (\u00b5g/m3)'],
  dtype='object')
```

```
In [9]: df = df.drop(df.columns[[0]], axis = 1)
```

```
In [10]: df.tail()
```

Out[10]:

	From	To	PM10 (\u00b5g/m3)	PM2.5 (\u00b5g/m3)	NO (\u00b5g/m3)	NO2 (\u00b5g/m3)	NOX (ppb)	CO (mg/m3)	SO2 (\u00b5g/m3)	NH3 (\u00b5g/m3)	Ozone (\u00b5g/m3)	Benzene (\u00b5g/m3)
8638	01-05-2023 23:30	01-05-2023 23:45	19.00	11.00	20.80	100.20	70.20	0.58	9.50	10.80	30.00	0.10
8639	01-05-2023 23:45	02-05-2023 00:00	32.00	6.00	21.80	98.80	70.30	NaN	NaN	11.00	33.50	0.10
8640	Min	NaN	12.00	3.00	0.10	0.20	4.20	0.10	0.10	4.60	0.10	0.10
8641	Max	NaN	847.00	474.00	157.50	106.90	165.20	4.00	645.60	62.40	123.80	0.60
8642	Avg.	NaN	181.41	75.69	14.65	55.76	42.67	1.41	34.23	13.24	35.63	0.18

```
In [11]: # Removing last three unnecessary rows
```

```
df = df.iloc[:-3]
```

```
In [12]: df.describe(include="all")
```

Out[12]:

	From	To	PM10 (µg/m3)	PM2.5 (µg/m3)	NO (µg/m3)	NO2 (µg/m3)	NOX (ppb)	CO (mg/m3)	SO2 (µg/m3)	NH3 (µg/m3)	Ozone (µg/m3)	Benzene (µg/m3)
count	8640	8640	6959.000000	8414.000000	7271.000000	8224.000000	8225.000000	8144.000000	7189.000000	8314.000000	8187.000000	2445.000000
unique	8640	8640	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	01-02-2023 00:00	01-02-2023 00:15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	1	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	181.408679	75.690397	14.649636	55.757028	42.672219	1.408538	34.232731	13.242663	35.626530	0.177505
std	NaN	NaN	136.016142	55.245265	19.221385	20.231407	22.435262	0.631056	39.452131	6.151034	27.018693	0.098895
min	NaN	NaN	12.000000	3.000000	0.100000	0.200000	4.200000	0.100000	0.100000	4.600000	0.100000	0.100000
25%	NaN	NaN	84.000000	36.000000	3.900000	39.400000	25.000000	0.950000	16.100000	9.400000	10.500000	0.100000
50%	NaN	NaN	145.000000	61.000000	6.100000	53.200000	37.700000	1.420000	25.300000	11.000000	32.400000	0.100000
75%	NaN	NaN	238.000000	101.000000	16.500000	71.025000	53.800000	1.850000	35.200000	14.000000	58.800000	0.200000
max	NaN	NaN	847.000000	474.000000	157.500000	106.900000	165.200000	4.000000	645.600000	62.400000	123.800000	0.600000

```
In [13]: df.describe()
```

Out[13]:

	PM10 (µg/m3)	PM2.5 (µg/m3)	NO (µg/m3)	NO2 (µg/m3)	NOX (ppb)	CO (mg/m3)	SO2 (µg/m3)	NH3 (µg/m3)	Ozone (µg/m3)	Benzene (µg/m3)
count	6959.000000	8414.000000	7271.000000	8224.000000	8225.000000	8144.000000	7189.000000	8314.000000	8187.000000	2445.000000
mean	181.408679	75.690397	14.649636	55.757028	42.672219	1.408538	34.232731	13.242663	35.626530	0.177505
std	136.016142	55.245265	19.221385	20.231407	22.435262	0.631056	39.452131	6.151034	27.018693	0.098895
min	12.000000	3.000000	0.100000	0.200000	4.200000	0.100000	0.100000	4.600000	0.100000	0.100000
25%	84.000000	36.000000	3.900000	39.400000	25.000000	0.950000	16.100000	9.400000	10.500000	0.100000
50%	145.000000	61.000000	6.100000	53.200000	37.700000	1.420000	25.300000	11.000000	32.400000	0.100000
75%	238.000000	101.000000	16.500000	71.025000	53.800000	1.850000	35.200000	14.000000	58.800000	0.200000
max	847.000000	474.000000	157.500000	106.900000	165.200000	4.000000	645.600000	62.400000	123.800000	0.600000

```
In [14]: df.From.describe()
```

```
Out[14]: count          8640  
unique         8640  
top    01-02-2023 00:00  
freq             1  
Name: From, dtype: object
```

```
In [15]: # "From" column will be converted to a datetime data type. After setting as the index of the DataFrame  
# ,we can perform time series analysis using the time index.
```

```
df.From = pd.to_datetime(df.From, dayfirst = True)  
df.To = pd.to_datetime(df.To, dayfirst = True)
```

```
In [16]: df["To"].dt.time
```

```
Out[16]: 0      00:15:00  
1      00:30:00  
2      00:45:00  
3      01:00:00  
4      01:15:00  
      ...  
8635    23:00:00  
8636    23:15:00  
8637    23:30:00  
8638    23:45:00  
8639    00:00:00  
Name: To, Length: 8640, dtype: object
```

```
In [17]: df.From.describe()
```

```
Out[17]: count          8640  
unique         8640  
top    2023-02-01 00:00:00  
freq             1  
first   2023-02-01 00:00:00  
last    2023-05-01 23:45:00  
Name: From, dtype: object
```

```
In [18]: df.tail()
```

Out[18]:

	From	To	PM10 (µg/m3)	PM2.5 (µg/m3)	NO (µg/m3)	NO2 (µg/m3)	NOX (ppb)	CO (mg/m3)	SO2 (µg/m3)	NH3 (µg/m3)	Ozone (µg/m3)	Benzene (µg/m3)
8635	2023-05-01 22:45:00	2023-05-01 23:00:00	19.0	11.0	17.9	100.0	67.8	0.63	10.0	10.7	26.1	0.1
8636	2023-05-01 23:00:00	2023-05-01 23:15:00	19.0	11.0	17.9	100.0	67.7	0.57	10.0	10.4	30.9	0.1
8637	2023-05-01 23:15:00	2023-05-01 23:30:00	19.0	11.0	19.6	100.2	69.2	0.58	9.9	10.5	29.6	0.1
8638	2023-05-01 23:30:00	2023-05-01 23:45:00	19.0	11.0	20.8	100.2	70.2	0.58	9.5	10.8	30.0	0.1
8639	2023-05-01 23:45:00	2023-05-02 00:00:00	32.0	6.0	21.8	98.8	70.3	NaN	NaN	11.0	33.5	0.1

Setting the Index

```
In [19]: df["Time"] = df["From"]
```

```
In [20]: df.set_index("Time", inplace = True)
```

```
In [21]: # "From" column represents the starting time of each time interval, and since the time intervals have a constant gap  
# of 15 minutes, it provides a consistent and evenly spaced time sequence.
```

By setting the "From" column as the time index, it is ensured that each data point in the time series is aligned
with the corresponding time interval. This will enable us to perform various time-based operations and analyses.

```
In [22]: df.head()
```

Out[22]:

	From	To	PM10 (µg/m3)	PM2.5 (µg/m3)	NO (µg/m3)	NO2 (µg/m3)	NOX (ppb)	CO (mg/m3)	SO2 (µg/m3)	NH3 (µg/m3)	Ozone (µg/m3)	Benzene (µg/m3)
Time												
2023-02-01 00:00:00	2023-02-01 00:00:00	2023-02-01 00:15:00	95.0	35.0	NaN	90.1	56.2	0.31	NaN	17.7	28.1	0.4
2023-02-01 00:15:00	2023-02-01 00:15:00	2023-02-01 00:30:00	95.0	35.0	NaN	88.0	55.1	0.33	NaN	18.3	27.1	0.4
2023-02-01 00:30:00	2023-02-01 00:30:00	2023-02-01 00:45:00	95.0	35.0	NaN	87.7	55.2	0.38	NaN	19.7	24.9	0.4
2023-02-01 00:45:00	2023-02-01 00:45:00	2023-02-01 01:00:00	122.0	34.0	NaN	88.9	55.7	0.38	NaN	21.3	21.9	0.4
2023-02-01 01:00:00	2023-02-01 01:00:00	2023-02-01 01:15:00	122.0	34.0	NaN	90.0	55.8	0.38	NaN	22.3	16.7	0.4

Setting the Desired Frequency

```
In [23]: df = df.asfreq('15Min')
```

Descriptive Statistics

In [24]: df.describe()

Out[24]:

	PM10 (µg/m3)	PM2.5 (µg/m3)	NO (µg/m3)	NO2 (µg/m3)	NOX (ppb)	CO (mg/m3)	SO2 (µg/m3)	NH3 (µg/m3)	Ozone (µg/m3)	Benzene (µg/m3)
count	6959.000000	8414.000000	7271.000000	8224.000000	8225.000000	8144.000000	7189.000000	8314.000000	8187.000000	2445.000000
mean	181.408679	75.690397	14.649636	55.757028	42.672219	1.408538	34.232731	13.242663	35.626530	0.177505
std	136.016142	55.245265	19.221385	20.231407	22.435262	0.631056	39.452131	6.151034	27.018693	0.098895
min	12.000000	3.000000	0.100000	0.200000	4.200000	0.100000	0.100000	4.600000	0.100000	0.100000
25%	84.000000	36.000000	3.900000	39.400000	25.000000	0.950000	16.100000	9.400000	10.500000	0.100000
50%	145.000000	61.000000	6.100000	53.200000	37.700000	1.420000	25.300000	11.000000	32.400000	0.100000
75%	238.000000	101.000000	16.500000	71.025000	53.800000	1.850000	35.200000	14.000000	58.800000	0.200000
max	847.000000	474.000000	157.500000	106.900000	165.200000	4.000000	645.600000	62.400000	123.800000	0.600000

In [25]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8640 entries, 2023-02-01 00:00:00 to 2023-05-01 23:45:00
Freq: 15T
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   From              8640 non-null   datetime64[ns]
 1   To                8640 non-null   datetime64[ns]
 2   PM10 (µg/m3)      6959 non-null   float64
 3   PM2.5 (µg/m3)     8414 non-null   float64
 4   NO (µg/m3)        7271 non-null   float64
 5   NO2 (µg/m3)       8224 non-null   float64
 6   NOX (ppb)         8225 non-null   float64
 7   CO (mg/m3)        8144 non-null   float64
 8   SO2 (µg/m3)       7189 non-null   float64
 9   NH3 (µg/m3)        8314 non-null   float64
 10  Ozone (µg/m3)     8187 non-null   float64
 11  Benzene (µg/m3)    2445 non-null   float64
dtypes: datetime64[ns](2), float64(10)
memory usage: 877.5 KB
```

```
In [26]: # These values are null values and needed to be filled  
# Data for Benzene is highly inconsistent
```

```
df.isnull().sum()
```

```
Out[26]: From          0  
To            0  
PM10 (µg/m³)    1681  
PM2.5 (µg/m³)   226  
NO (µg/m³)      1369  
NO2 (µg/m³)     416  
NOX (ppb)        415  
CO (mg/m³)       496  
SO2 (µg/m³)     1451  
NH3 (µg/m³)      326  
Ozone (µg/m³)    453  
Benzene (µg/m³)  6195  
dtype: int64
```

```
In [27]: df.dtypes # Categorical datatypes show dtype object
```

```
Out[27]: From          datetime64[ns]  
To            datetime64[ns]  
PM10 (µg/m³)    float64  
PM2.5 (µg/m³)   float64  
NO (µg/m³)      float64  
NO2 (µg/m³)     float64  
NOX (ppb)        float64  
CO (mg/m³)       float64  
SO2 (µg/m³)     float64  
NH3 (µg/m³)      float64  
Ozone (µg/m³)    float64  
Benzene (µg/m³)  float64  
dtype: object
```

Histograms

It plots the distribution of a dataset by displaying the frequency or count of observations falling into different intervals.

provide a visual representation of the data's underlying pattern and help identify patterns, trends, and outliers within the dataset.

```
In [28]: # Interpolation needs to be done not for time columns  
df_n = df.iloc[:, 2:]
```

```
In [29]: def histogram_plot(df,parameter):  
    df[parameter].hist(figsize=(10,5))
```

Boxplots - Used to get outliers

insights into the spread, skewness, and presence of outliers in the data

particularly helpful in identifying any outliers that may be present in the dataset and understanding the overall shape of the distribution. used to get fair idea about central tendencies

```
In [30]: def boxplot_plot(df,parameter):  
    sns.boxplot(data=df[parameter])
```

Countplots

```
In [31]: def countplot_plot(df,parameter):  
    sns.countplot(x=parameter, data=df)
```

Pairplots

```
In [32]: def pairplot_plot(df):  
    sns.pairplot(df,diag_kind='kde', height=5)
```

Heatmaps

```
In [33]: def heatmap_plot(df):  
    corr = df.corr()  
    print(corr)  
    sns.heatmap(corr, annot=True)
```

Dealing with Missing Values

We can fill Nan values with mean, median, zero, front fill or back fill

```
In [34]: # Filling the missing values with median  
medianFiller = lambda x: x.fillna(x.median())  
df_median = df.apply(medianFiller, axis=0)
```

```
In [35]: # Filling the missing values with mean  
meanFiller = lambda x: x.fillna(x.mean())  
df_mean = df.apply(meanFiller, axis=0)
```

```
In [36]: # Filling the missing values with zero  
zeroFiller = lambda x: x.fillna(0)  
df_zero = df.apply(zeroFiller, axis=0)
```

```
In [37]: df_ffill = df.fillna(method = "ffill") # front filling
```

```
In [38]: df_bfill = df.fillna(method = "bfill") # back filling
```

Interpolation Techniques

Whenever we have time-series data, Then to deal with missing values, we cannot use mean imputation techniques.

Interpolation is a powerful method to fill in missing values in time-series data.

In such data, when next value depends on previous known values, we cannot consider the overall mean to replace the data

but the effect of the values close by which are more effecting it.

So, filling the missing values with mean is not a very good approach.

In [39]: # This function plots data points and interpolation data for different column names

```
def interpolation_plot(df_original , df_interpolate ,parameter, type_interpolate):  
  
    # Set the figure size  
    plt.figure(figsize=(20, 10))  
  
    # Plot the original data points  
    plt.scatter(df_original.index, df_original[parameter], label='Data Points of '+ parameter)  
  
    # Plot the interpolation  
    plt.plot(df_interpolate.index, df_interpolate[parameter], label=type_interpolate)  
  
    # Set plot title and Labels for the first plot  
    plt.title(type_interpolate + ' (Data Points Included)', fontsize=30)  
    plt.xlabel('Time', fontsize=20)  
    plt.ylabel(parameter, fontsize=20)  
  
    # Add Legend for the first plot  
    plt.legend()  
  
    # Show the first plot  
    plt.show()  
  
    # Set the figure size for the second plot  
    plt.figure(figsize=(20, 10))  
  
    # Plot the original data points (without markers)  
    plt.plot(df_original.index, df_original[parameter], label='Data Points')  
  
    # Plot the interpolation  
    plt.plot(df_interpolate.index, df_interpolate[parameter], label=type_interpolate)  
  
    # Set plot title and Labels for the second plot  
    plt.title(type_interpolate + ' (Data Points Excluded)', fontsize=30)  
    plt.xlabel('Time', fontsize=20)  
    plt.ylabel(parameter, fontsize=20)  
  
    # Add Legend for the second plot  
    plt.legend()  
  
    # Show the second plot  
    plt.show()
```

Linear Interpolation

Forward Filling

```
In [40]: df_linear_f = df_n.interpolate(method ='linear', limit_direction ='forward')
```

Backward Filling

```
In [41]: df_linear_b = df_n.interpolate(method ='linear', limit_direction ='backward')
```

Quadratic Interpolation

```
In [42]: df_quad = df_n.interpolate(method="polynomial", order=2)
```

Cubic interpolation

```
In [43]: df_cubic = df_n.interpolate(method="polynomial", order=3)
```

Here, cubic interpolation introduces negative values as well which is practically impossible.

Hence, cubic interpolation is not a very good choice

Spline Interpolation

We are ignoring all the NaN values for spline interpolations


```
In [44]: def quadraticspline(df, parameter):
    # Remove NaN values from the DataFrame
    df_valid = df.dropna(subset=[parameter])

    # Convert the index (Time) to timestamps
    timestamps = df_valid.index.to_series().apply(lambda x: x.timestamp())

    # Extract the timestamps and the column for interpolation
    x = timestamps.values
    y = df_valid[parameter].values

    # Perform quadratic spline interpolation
    quadratic_interp = interp1d(x, y, kind='quadratic')

    # Generate timestamps for the interpolated curve
    x_interp = np.linspace(min(x), max(x), 100)

    # Convert the interpolated timestamps back to datetime format
    x_interp_datetime = pd.to_datetime(x_interp, unit='s')

    # Generate the interpolated values
    y_interp = quadratic_interp(x_interp)

    # Set the figure size
    plt.figure(figsize=(20, 10))

    # Plot the original data points and the interpolated curve
    plt.scatter(df_valid.index, df_valid[parameter], label='Data Points')
    plt.plot(x_interp_datetime, y_interp, label='Quadratic Spline')

    plt.title('Quadratic Spline Interpolation (with datapoints)', fontsize=30)
    plt.xlabel('Time', fontsize=20)
    plt.ylabel(parameter, fontsize=20)

    plt.legend()
    plt.show()

    # Set the figure size
    plt.figure(figsize=(20, 10))

    # Plot the interpolated curve
    plt.plot(x_interp_datetime, y_interp, label='Quadratic Spline')

    plt.title('Quadratic Spline Interpolation (without datapoints)', fontsize=30)
    plt.xlabel('Time', fontsize=20)
    plt.ylabel(parameter, fontsize=20)
```

```
plt.legend()  
plt.show()
```



```
In [45]: def cubicspline(df, parameter):
    # Remove NaN values from the DataFrame
    df_valid = df.dropna(subset=[parameter])

    # Convert the index (Time) to timestamps
    timestamps = df_valid.index.to_series().apply(lambda x: x.timestamp())

    # Extract the timestamps and the column for interpolation
    x = timestamps.values
    y = df_valid[parameter].values

    # Create a cubic spline interpolation object
    cs = CubicSpline(x, y)

    # Generate timestamps for the interpolated curve
    x_interp = np.linspace(min(x), max(x), 100)

    # Convert the interpolated timestamps back to datetime format
    x_interp_datetime = pd.to_datetime(x_interp, unit='s')

    # Generate the interpolated values
    y_interp = cs(x_interp)

    # Set the figure size
    plt.figure(figsize=(20, 10))

    # Plot the original data points and the interpolated curve
    plt.scatter(df_valid.index, df_valid[parameter], label='Data Points')
    plt.plot(x_interp_datetime, y_interp, label='Cubic Spline')

    plt.title('Cubic Spline Interpolation (with datapoints)', fontsize=30)
    plt.xlabel('Time', fontsize=20)
    plt.ylabel(parameter, fontsize=20)

    plt.legend()
    plt.show()

    # Set the figure size
    plt.figure(figsize=(20, 10))

    # Plot the interpolated curve
    plt.plot(x_interp_datetime, y_interp, label='Cubic Spline')

    plt.title('Cubic Spline Interpolation (without datapoints)', fontsize=30)
    plt.xlabel('Time', fontsize=20)
    plt.ylabel(parameter, fontsize=20)
```

```
plt.legend()  
plt.show()
```

```
In [46]: def simple_time_plot(df,parameter,title):  
    # Set the figure size  
    plt.figure(figsize=(20, 10))  
  
    # Plotting the column with respect to the index  
    plt.plot(df.index, df[parameter])  
  
    # Set plot title and labels  
    plt.title(parameter + " " + title, fontsize = 30)  
    plt.xlabel('Time', fontsize = 20)  
    plt.ylabel(parameter, fontsize = 20)  
  
    # Show the plot  
    plt.show()
```

QQ Plot

```
In [47]: def qq_plot(df,parameter):  
    # Set the figure size for the second plot  
    plt.figure(figsize=(20, 10))  
  
    # Generate the QQ plot  
    sm.qqplot(df[parameter], line='s')  
  
    # Set plot title and labels  
    plt.title("QQ Plot for " + parameter, fontsize=30)  
    plt.xlabel("Theoretical Quantiles", fontsize=20)  
    plt.ylabel("Sample Quantiles", fontsize=20)  
  
    # Display the plot  
    plt.show()
```

We will create a new dataframe after filling in the missing values for time series analysis

```
In [48]: df_new = pd.DataFrame()
```

```
In [49]: df.columns
```

```
Out[49]: Index(['From', 'To', 'PM10 (\u00b5g/m\u00b3)', 'PM2.5 (\u00b5g/m\u00b3)', 'NO (\u00b5g/m\u00b3)',  
   'NO2 (\u00b5g/m\u00b3)', 'NOX (ppb)', 'CO (mg/m\u00b3)', 'SO2 (\u00b5g/m\u00b3)', 'NH3 (\u00b5g/m\u00b3)',  
   'Ozone (\u00b5g/m\u00b3)', 'Benzene (\u00b5g/m\u00b3')],  
  dtype='object')
```

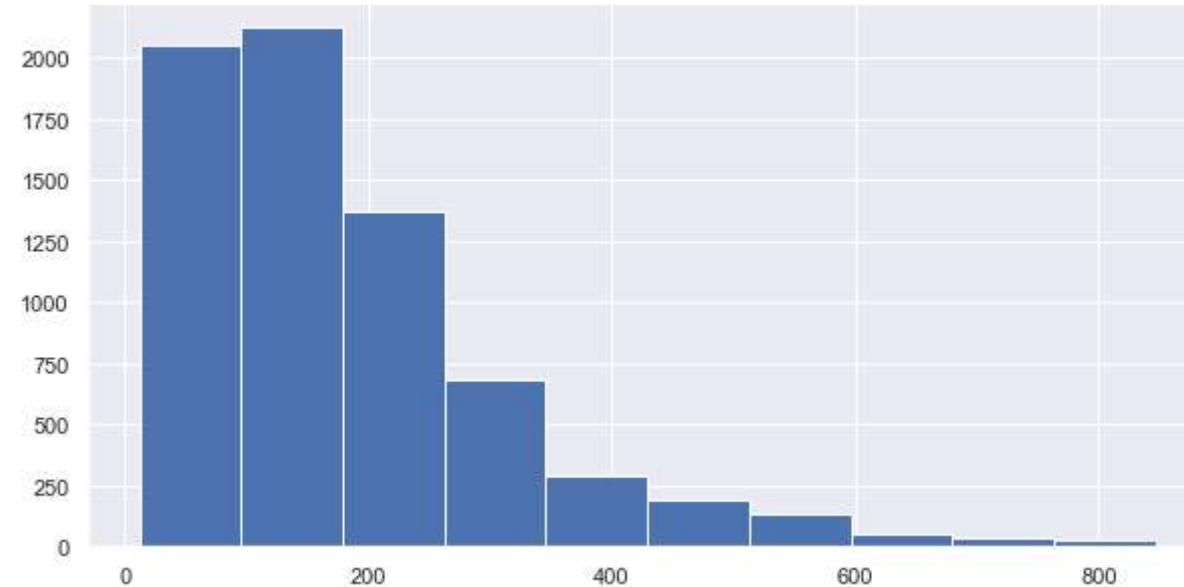
Analysis for "PM10 (\u00b5g/m\u00b3)"

```
In [50]: pollutant = 'PM10 (\u00b5g/m\u00b3)'
```

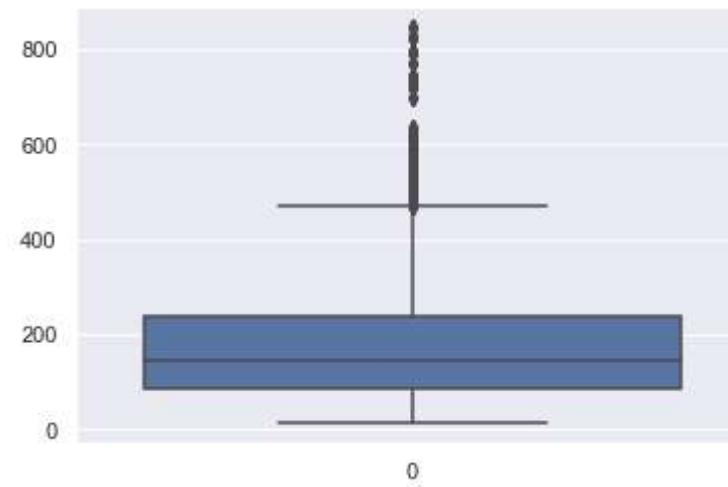
```
In [51]: df[pollutant].describe()
```

```
Out[51]: count    6959.000000  
mean     181.408679  
std      136.016142  
min      12.000000  
25%     84.000000  
50%     145.000000  
75%     238.000000  
max     847.000000  
Name: PM10 (\u00b5g/m\u00b3), dtype: float64
```

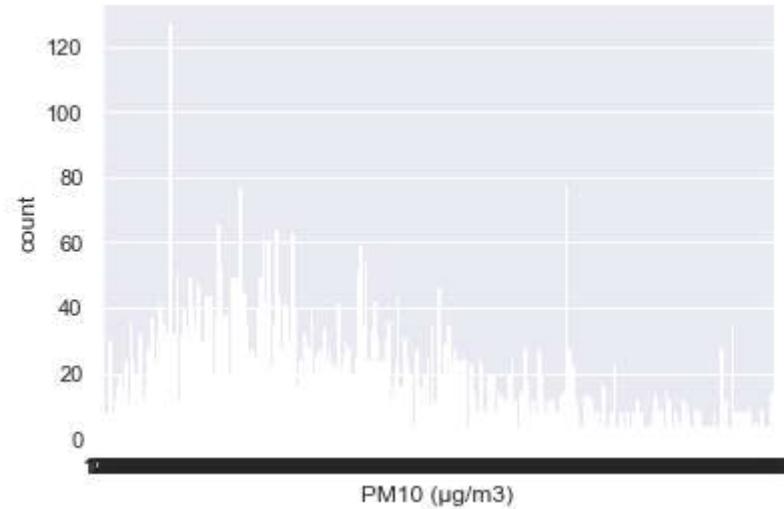
```
In [52]: histogram_plot(df_n,pollutant)
```



```
In [53]: boxplot_plot(df_n,pollutant)
```

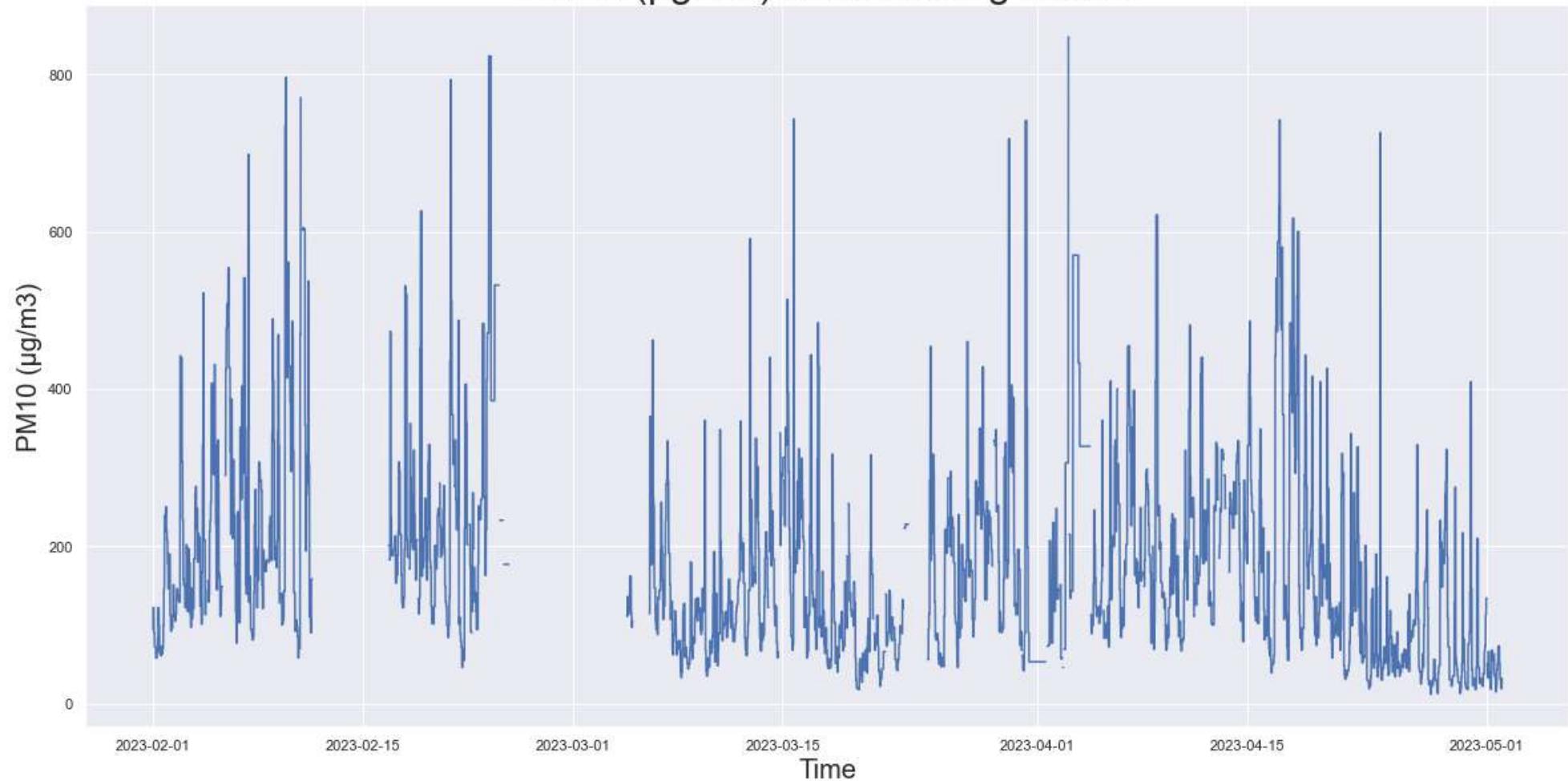


```
In [54]: countplot_plot(df_n,pollutant)
```



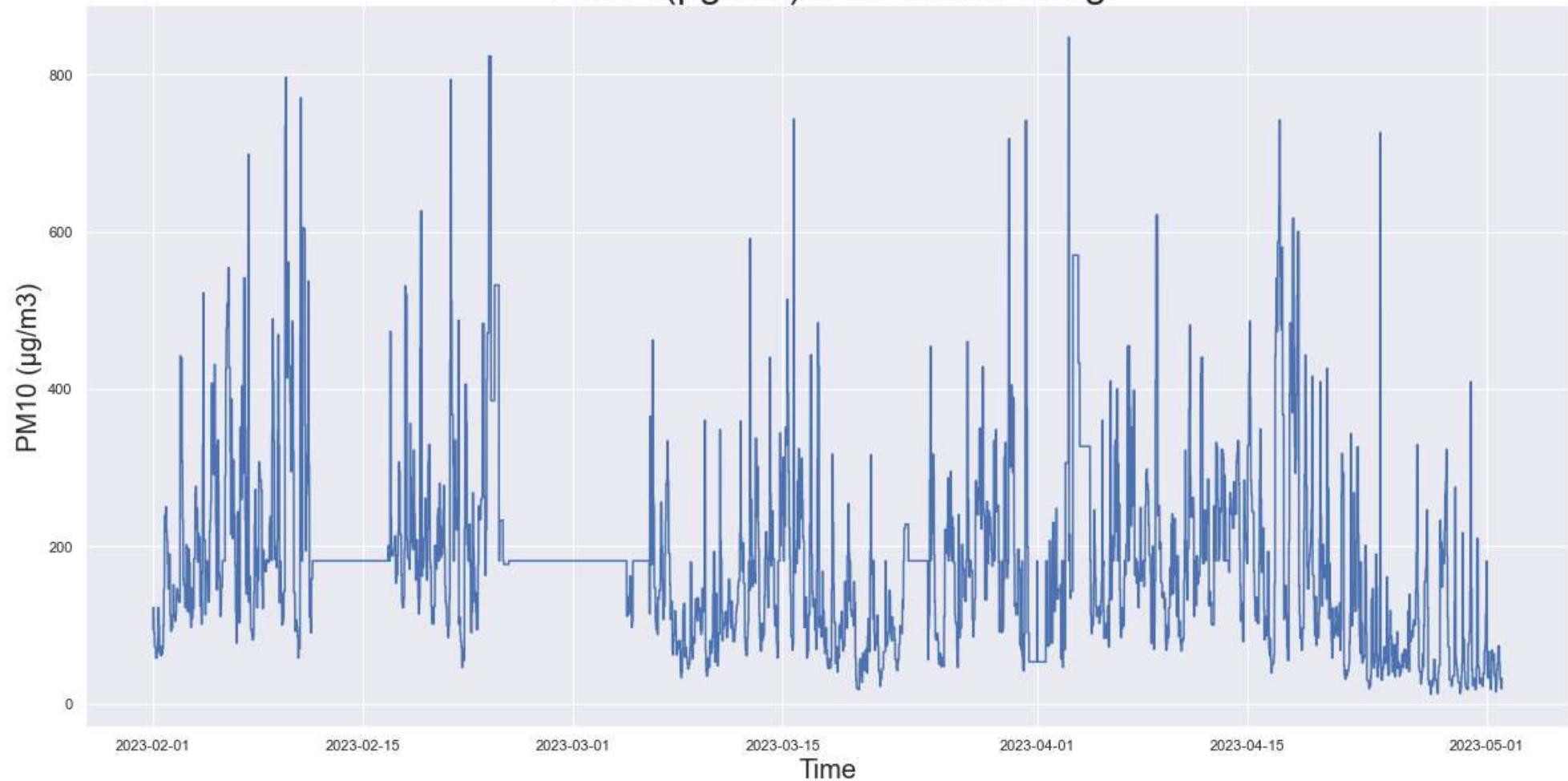
```
In [55]: simple_time_plot(df_n,pollutant,"With missing values")
```

PM10 ($\mu\text{g}/\text{m}^3$) With missing values



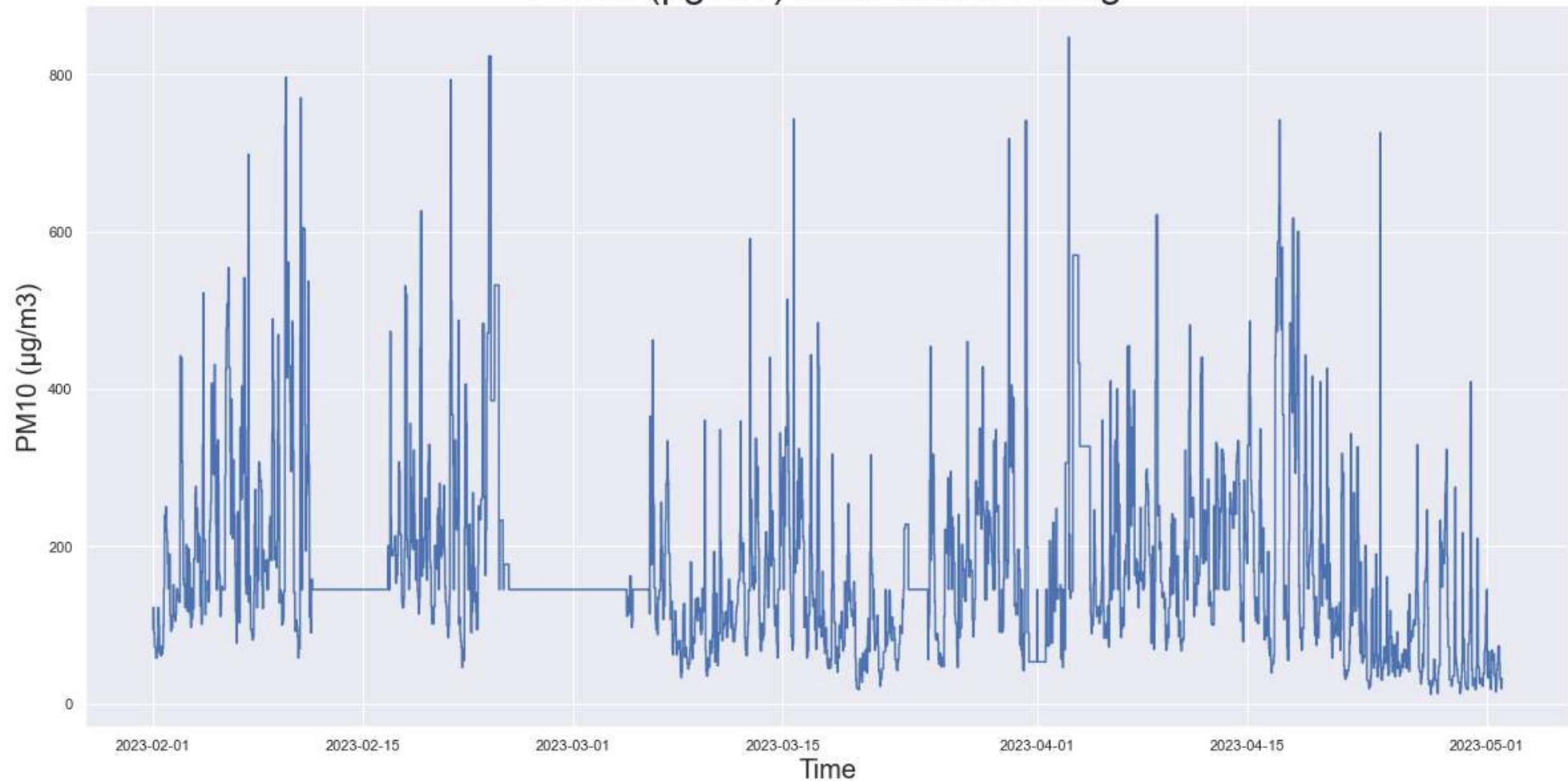
```
In [56]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

PM10 ($\mu\text{g}/\text{m}^3$) after mean filling



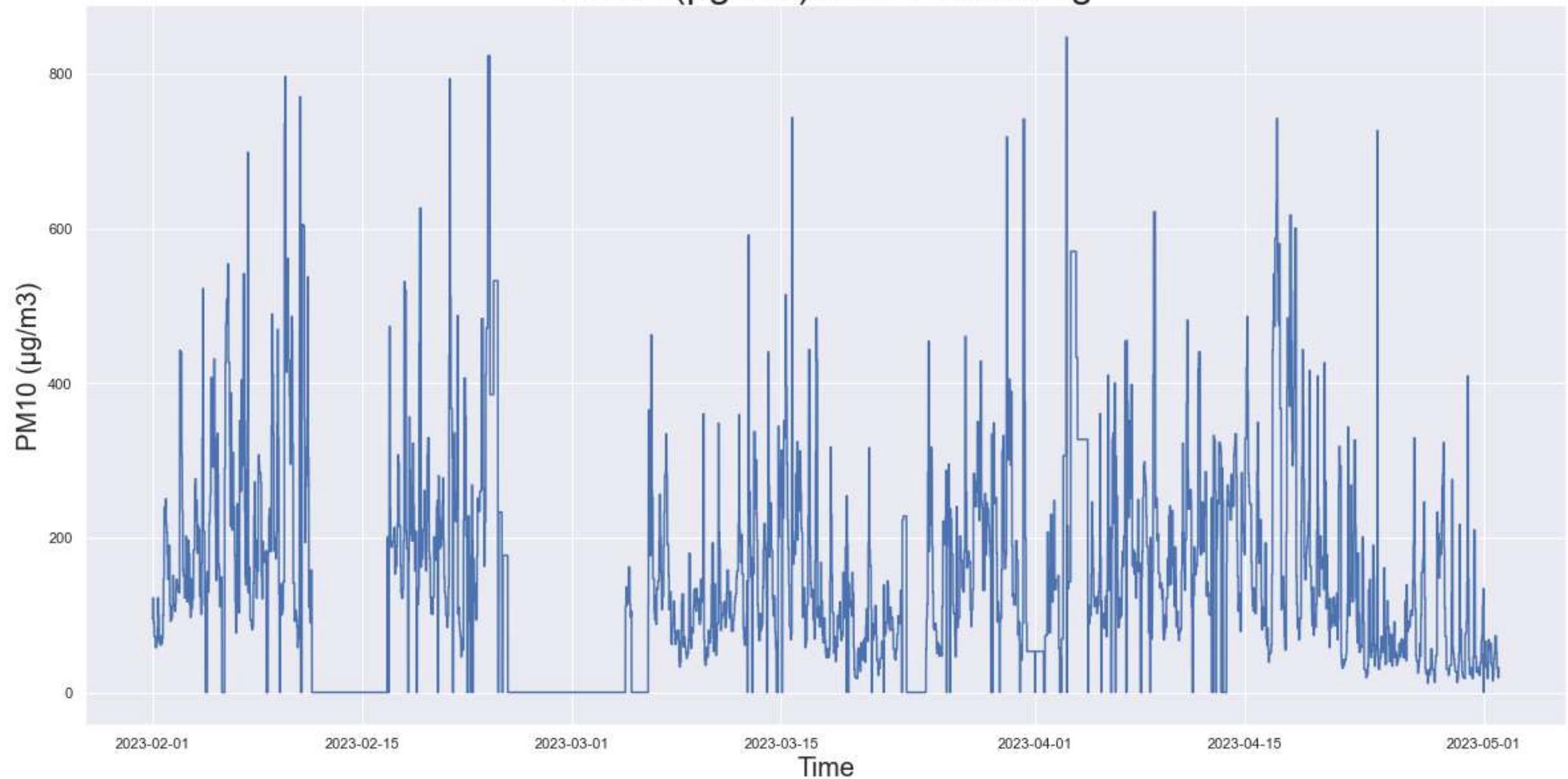
```
In [57]: simple_time_plot(df_median,pollutant,"after median filling")
```

PM10 ($\mu\text{g}/\text{m}^3$) after median filling

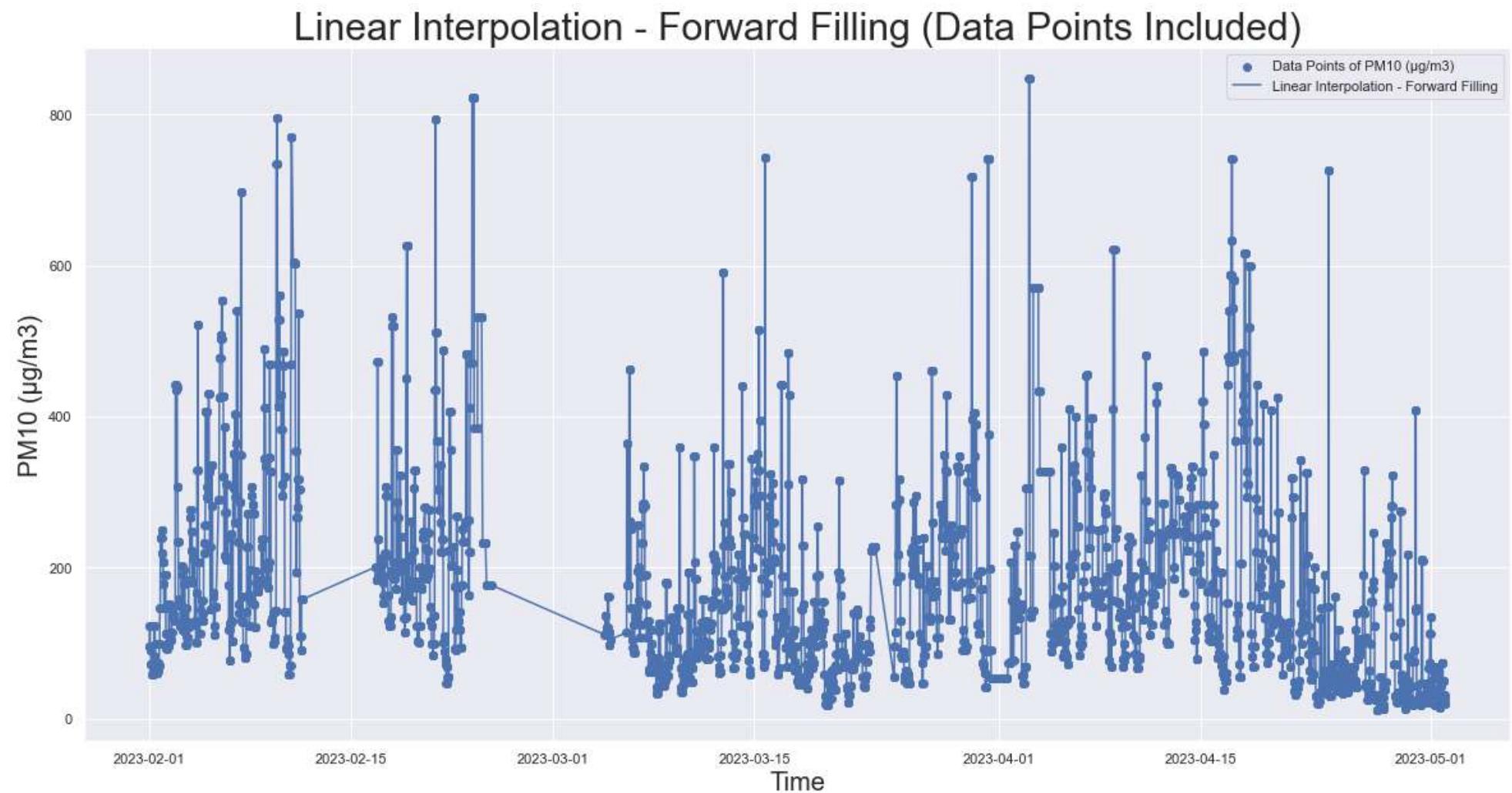


```
In [58]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

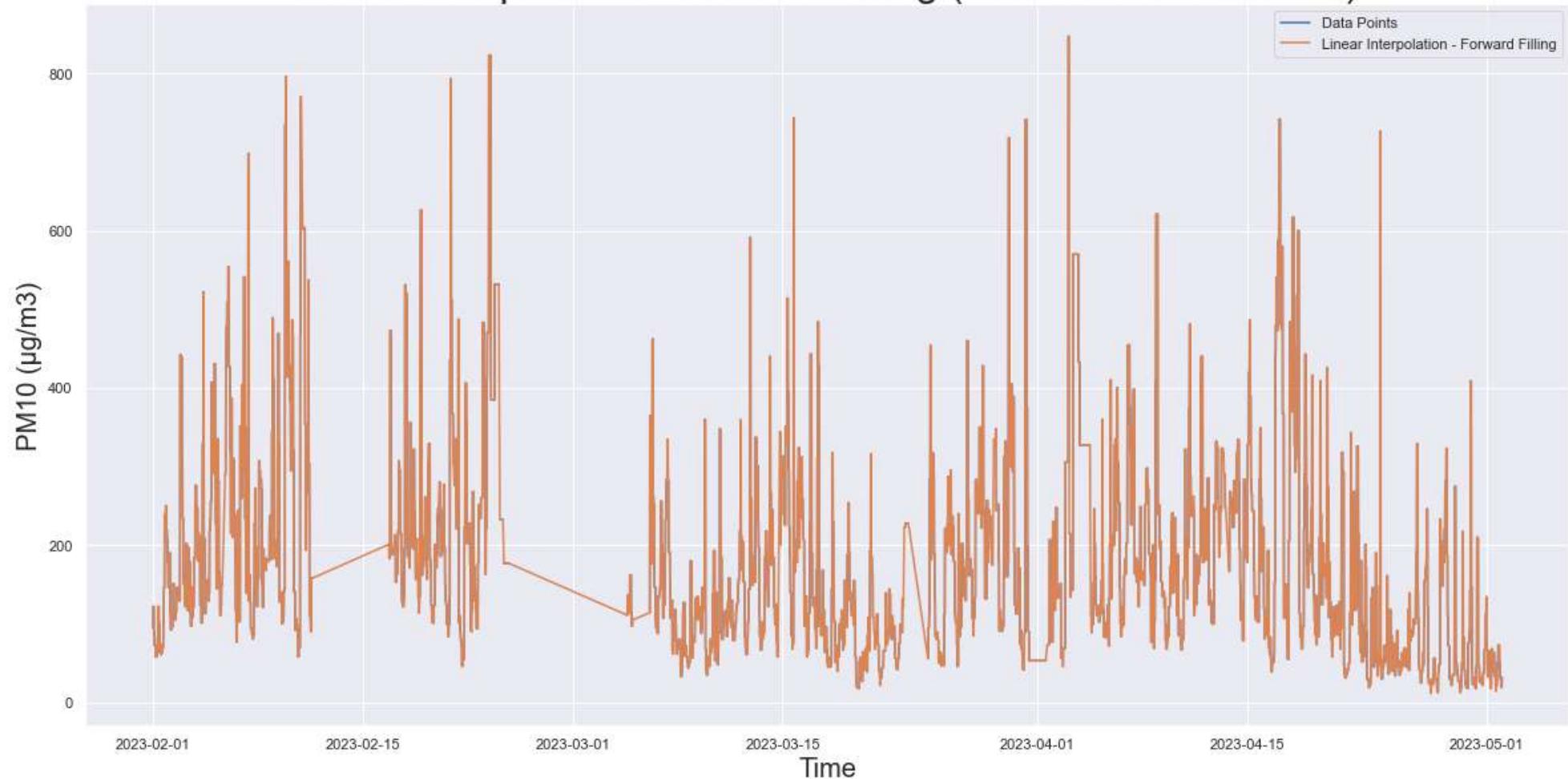
PM10 ($\mu\text{g}/\text{m}^3$) after zero filling



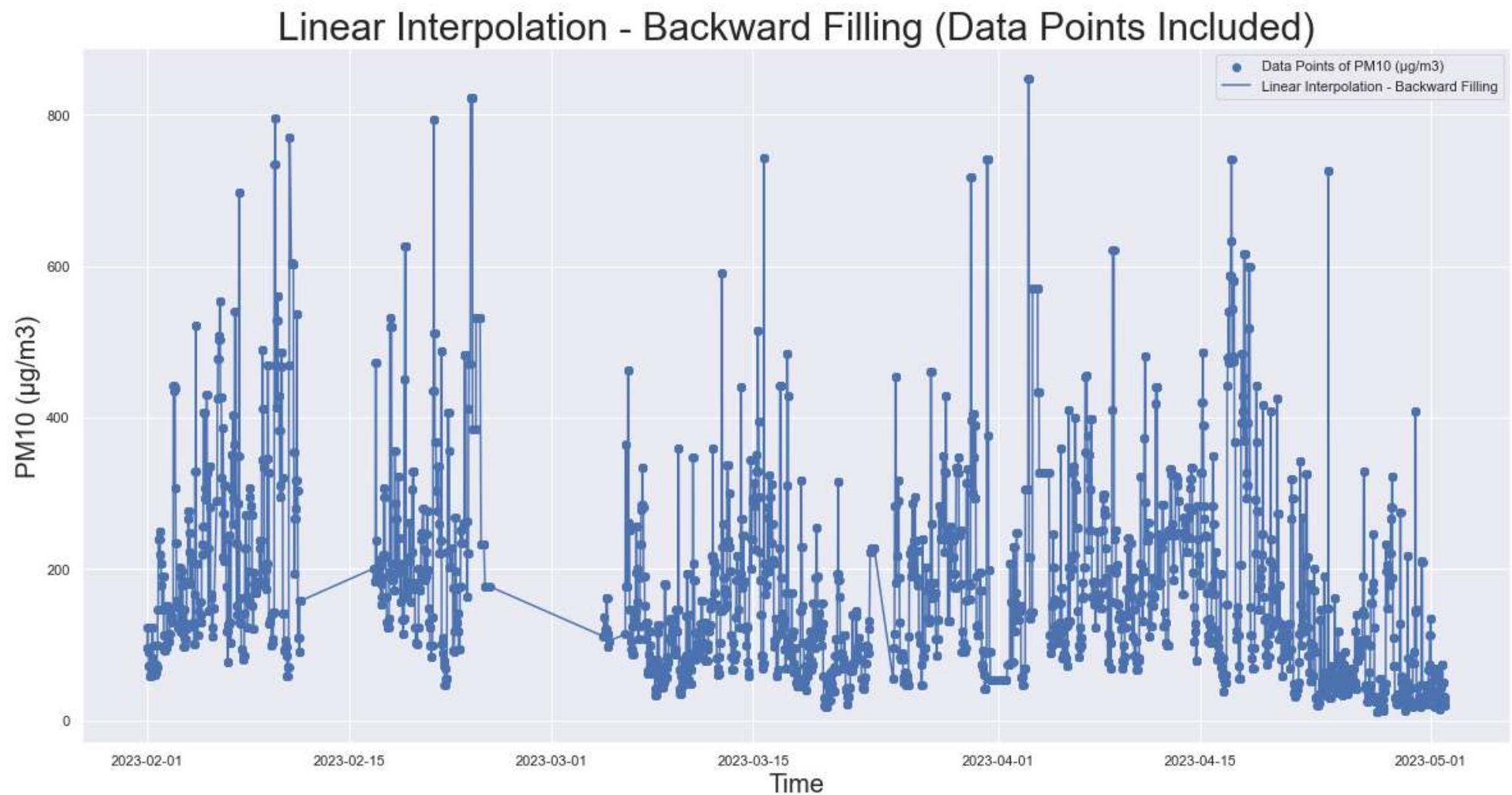
```
In [59]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```



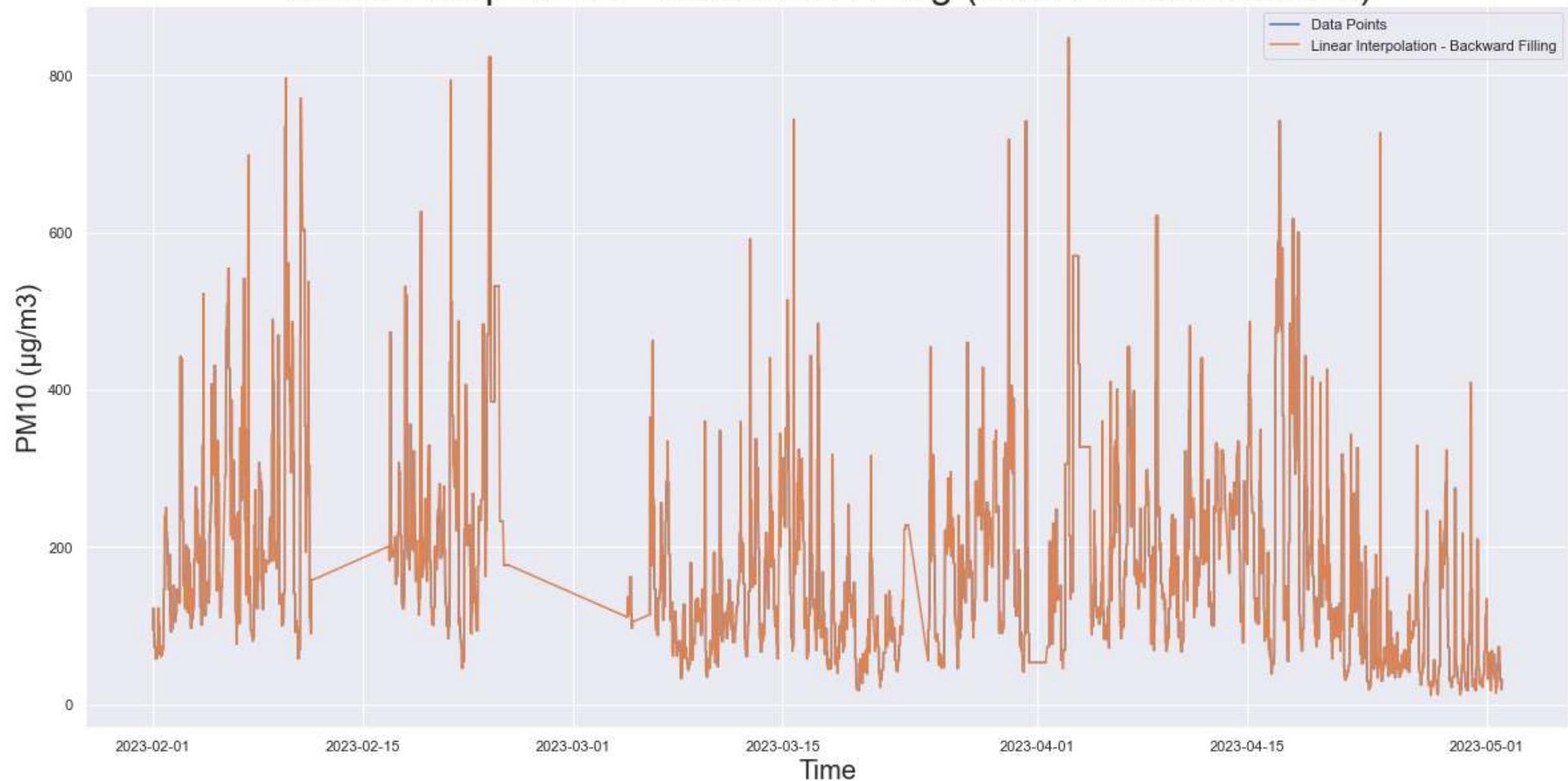
Linear Interpolation - Forward Filling (Data Points Excluded)



```
In [60]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

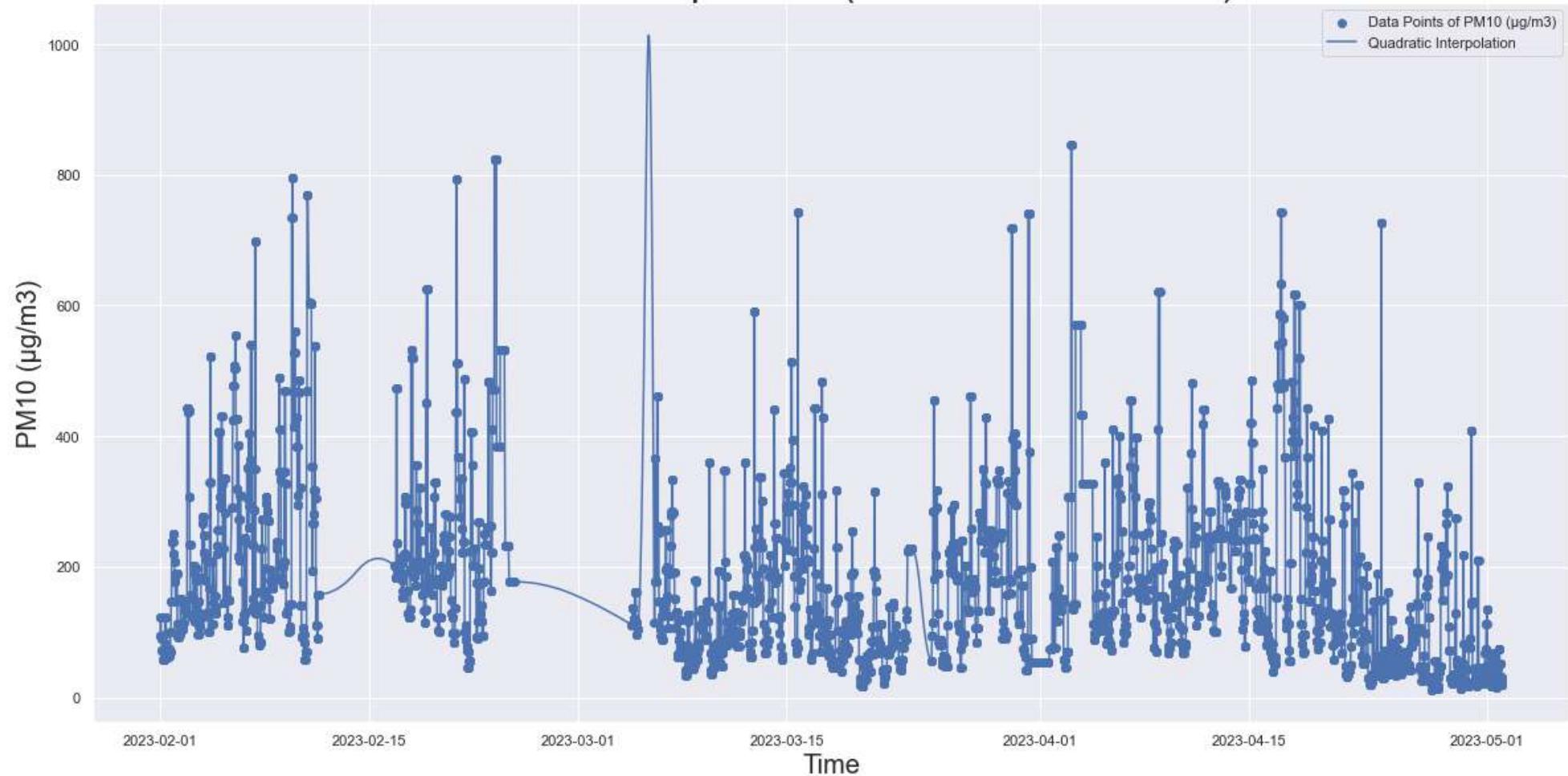


Linear Interpolation - Backward Filling (Data Points Excluded)

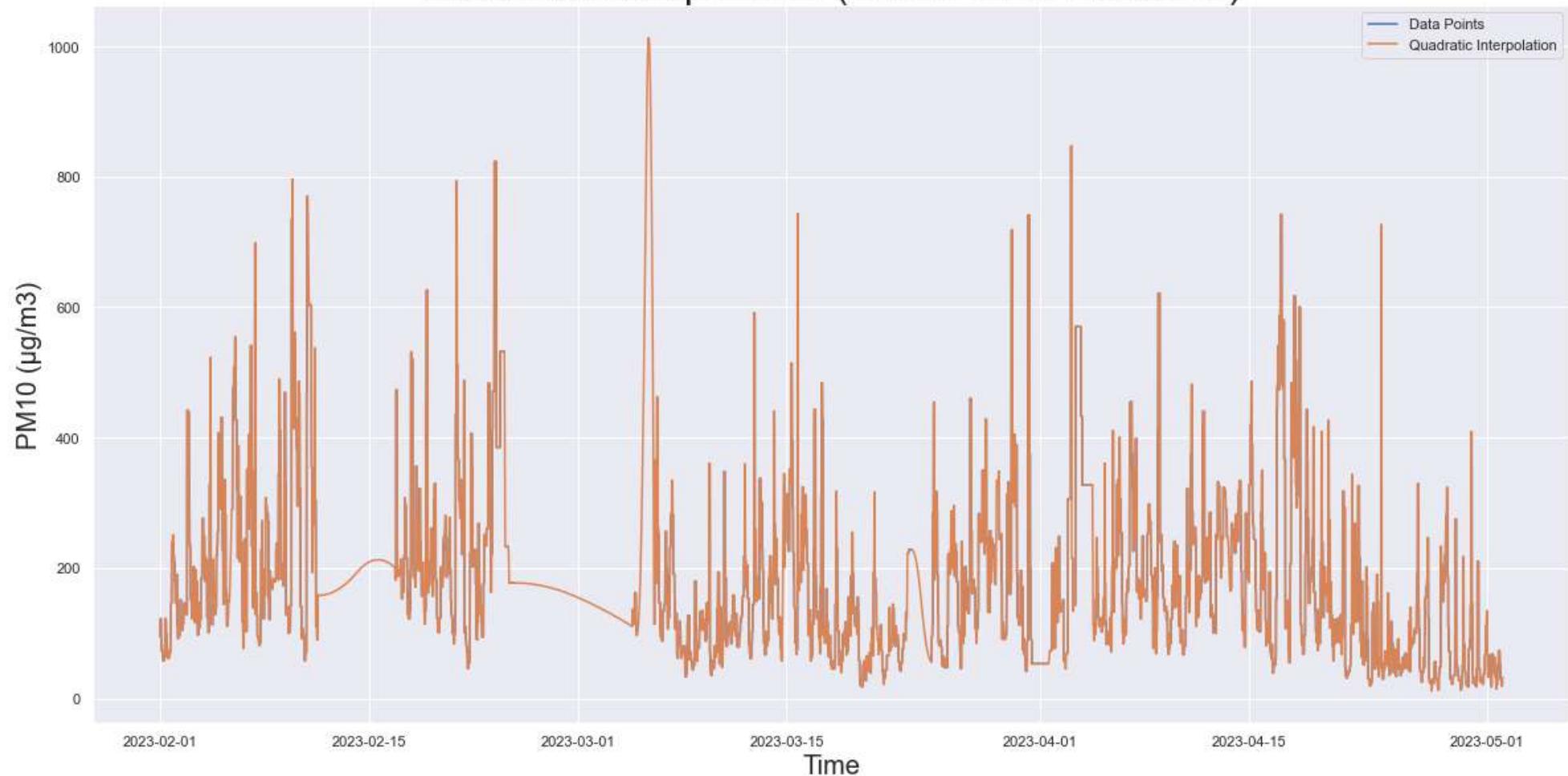


```
In [61]: interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")
```

Quadratic Interpolation (Data Points Included)

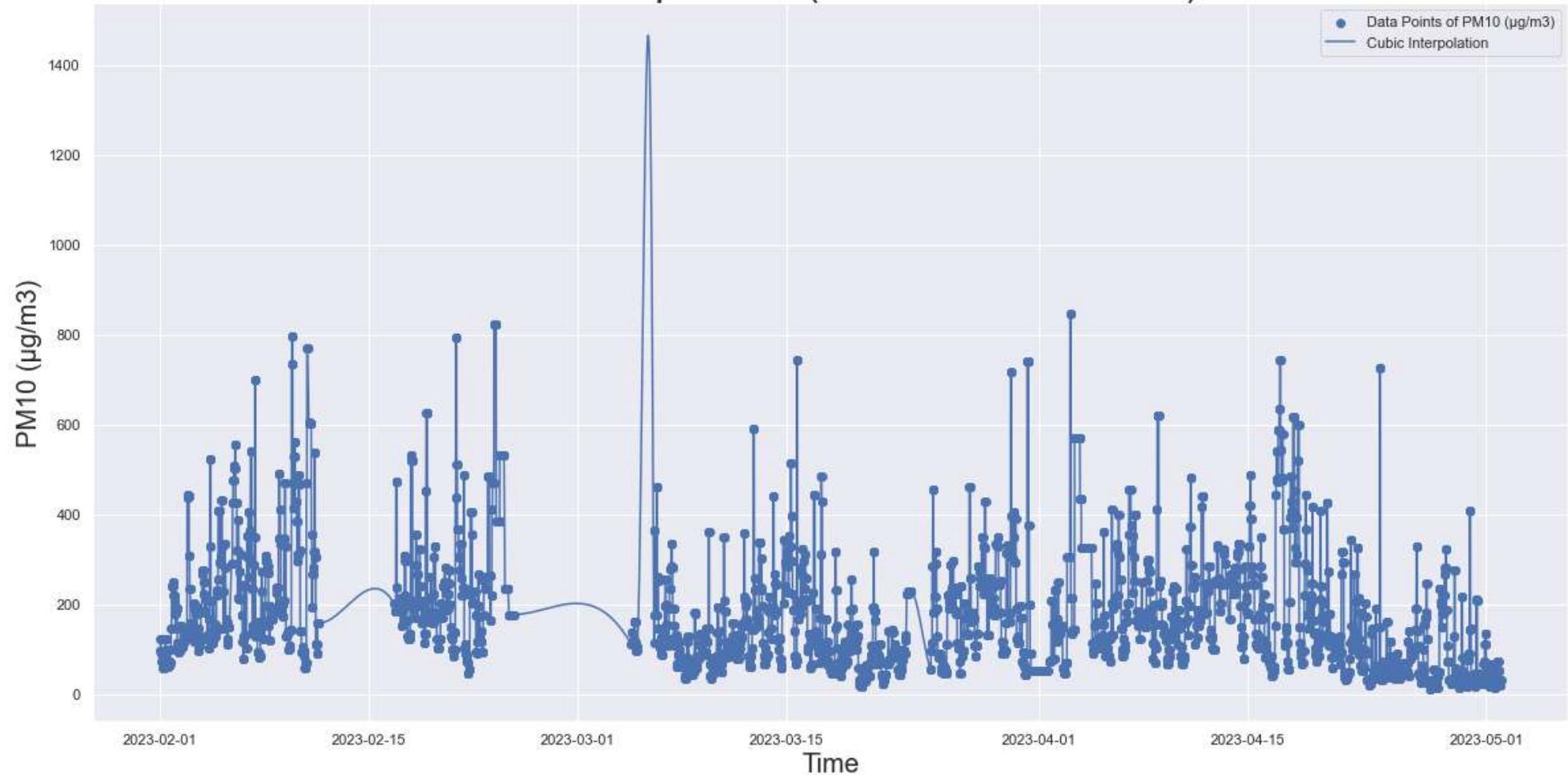


Quadratic Interpolation (Data Points Excluded)

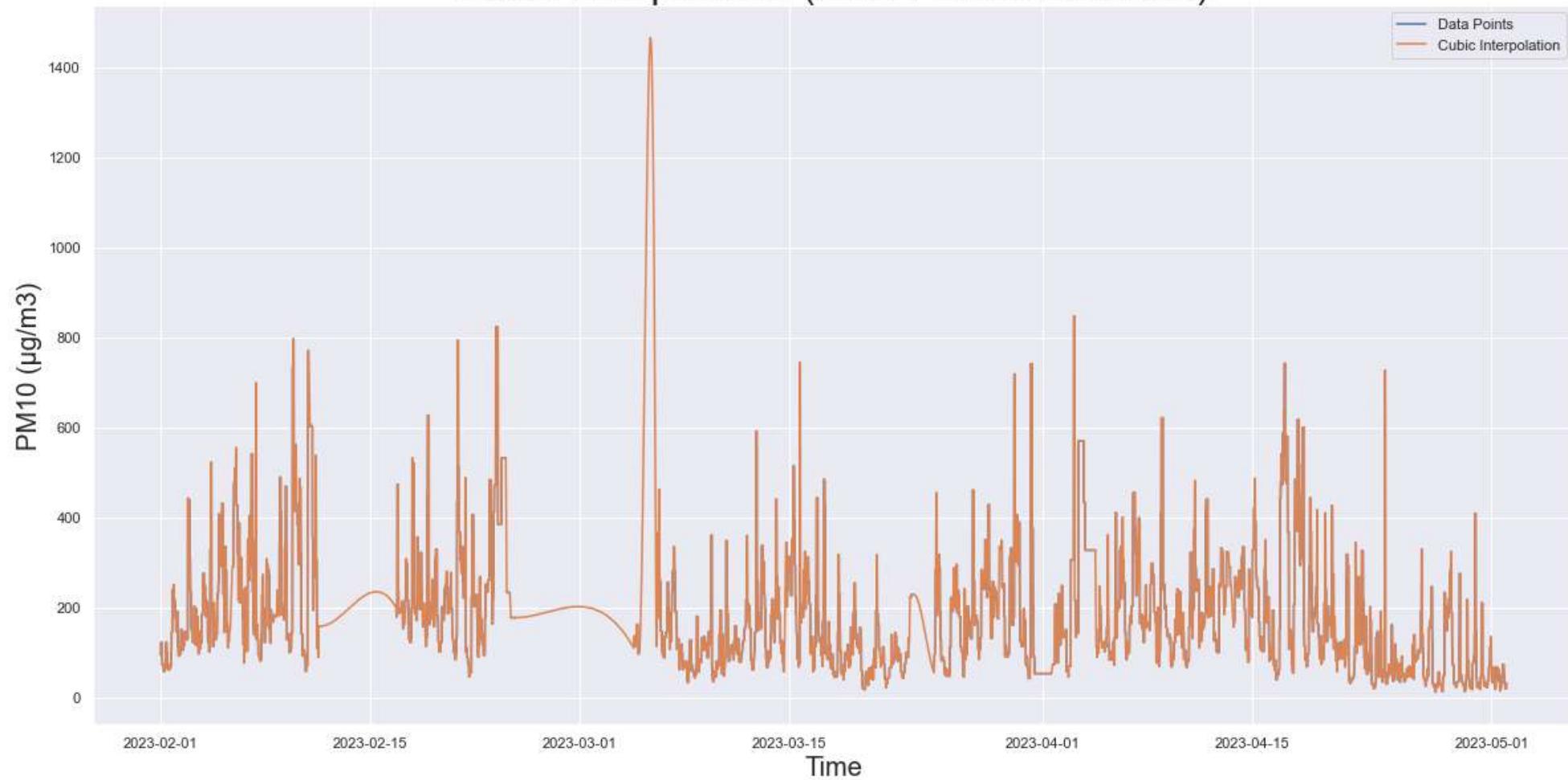


```
In [62]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```

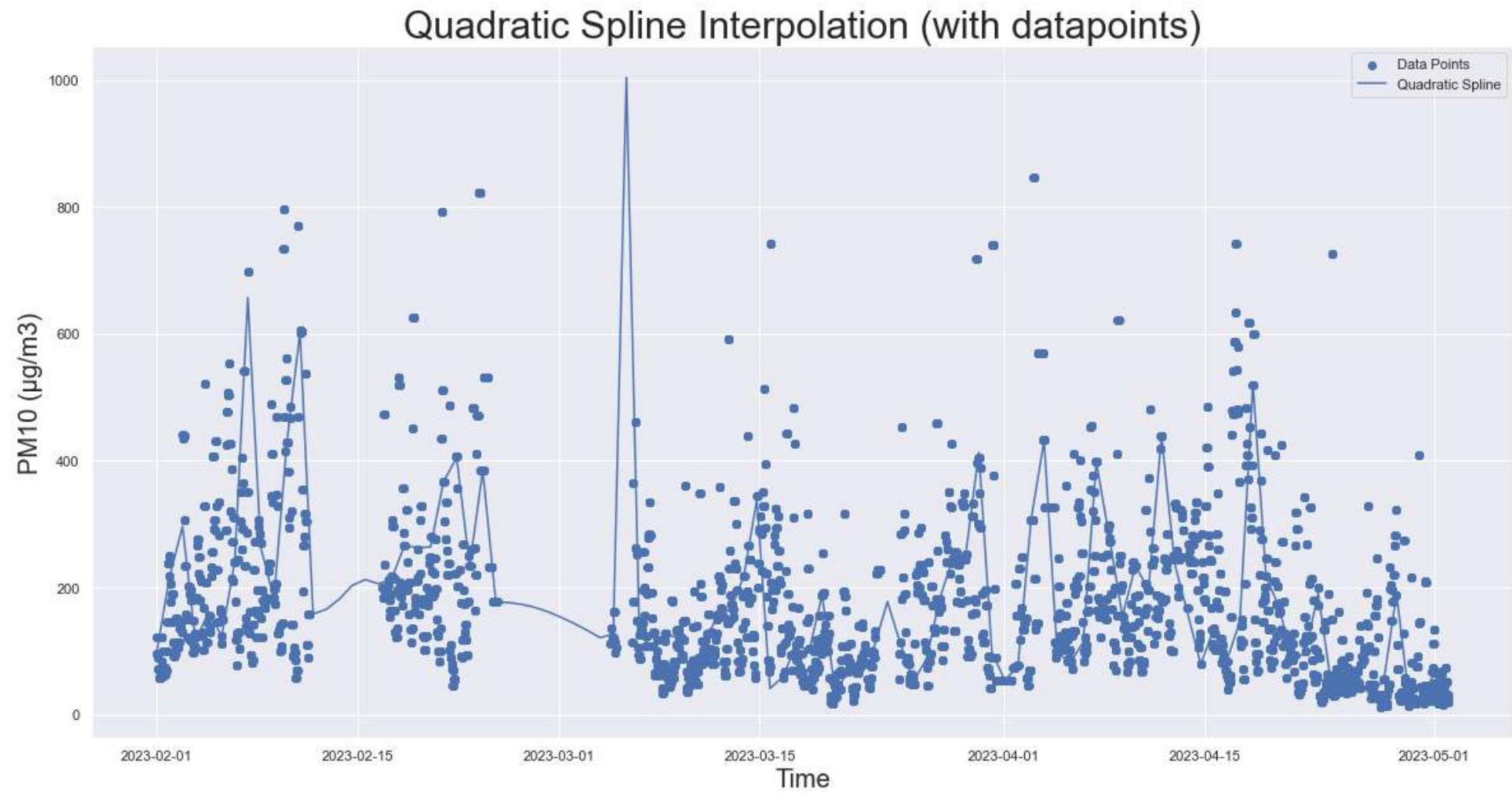
Cubic Interpolation (Data Points Included)



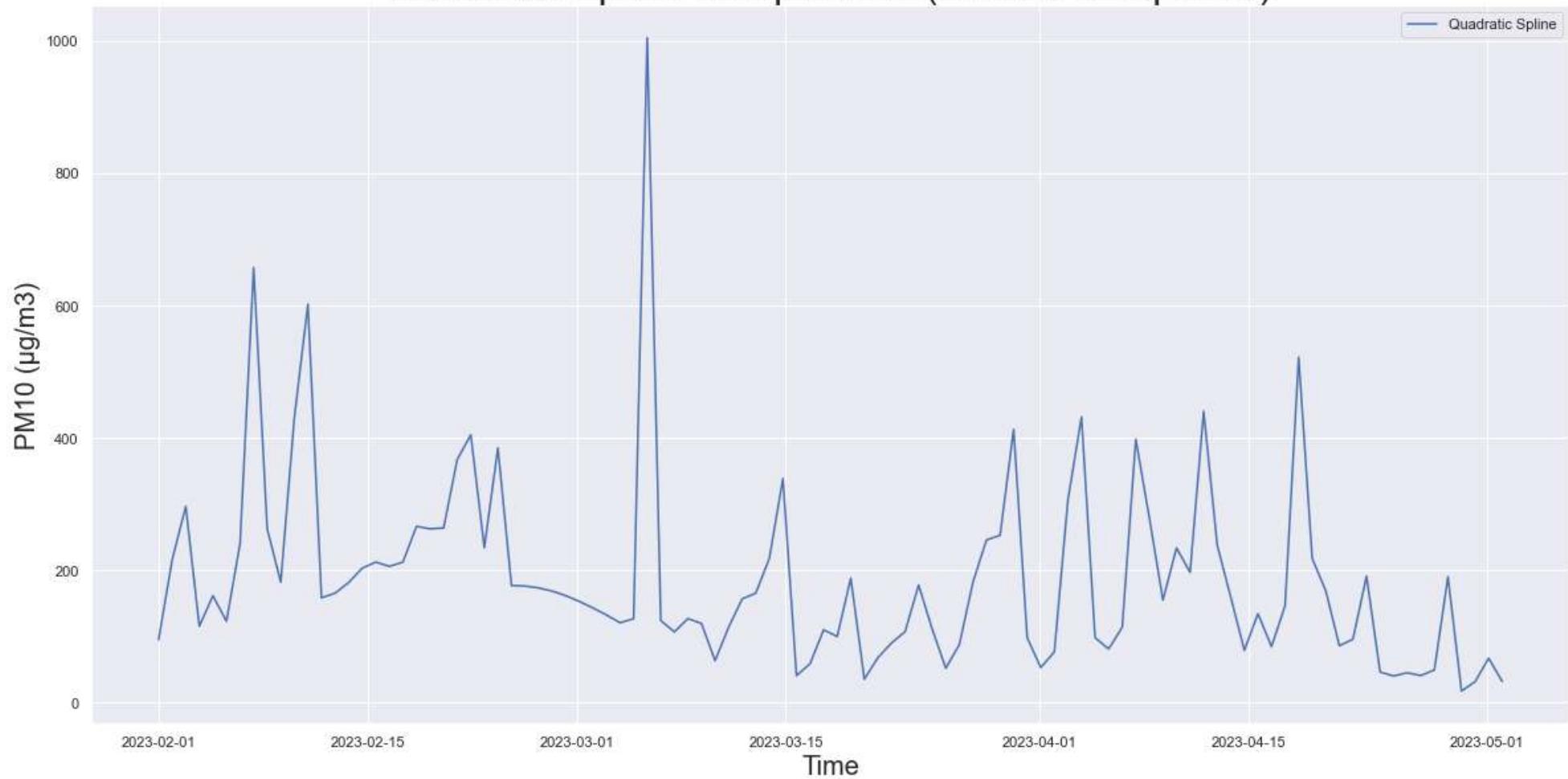
Cubic Interpolation (Data Points Excluded)



```
In [63]: quadraticspline(df_n,pollutant)
```

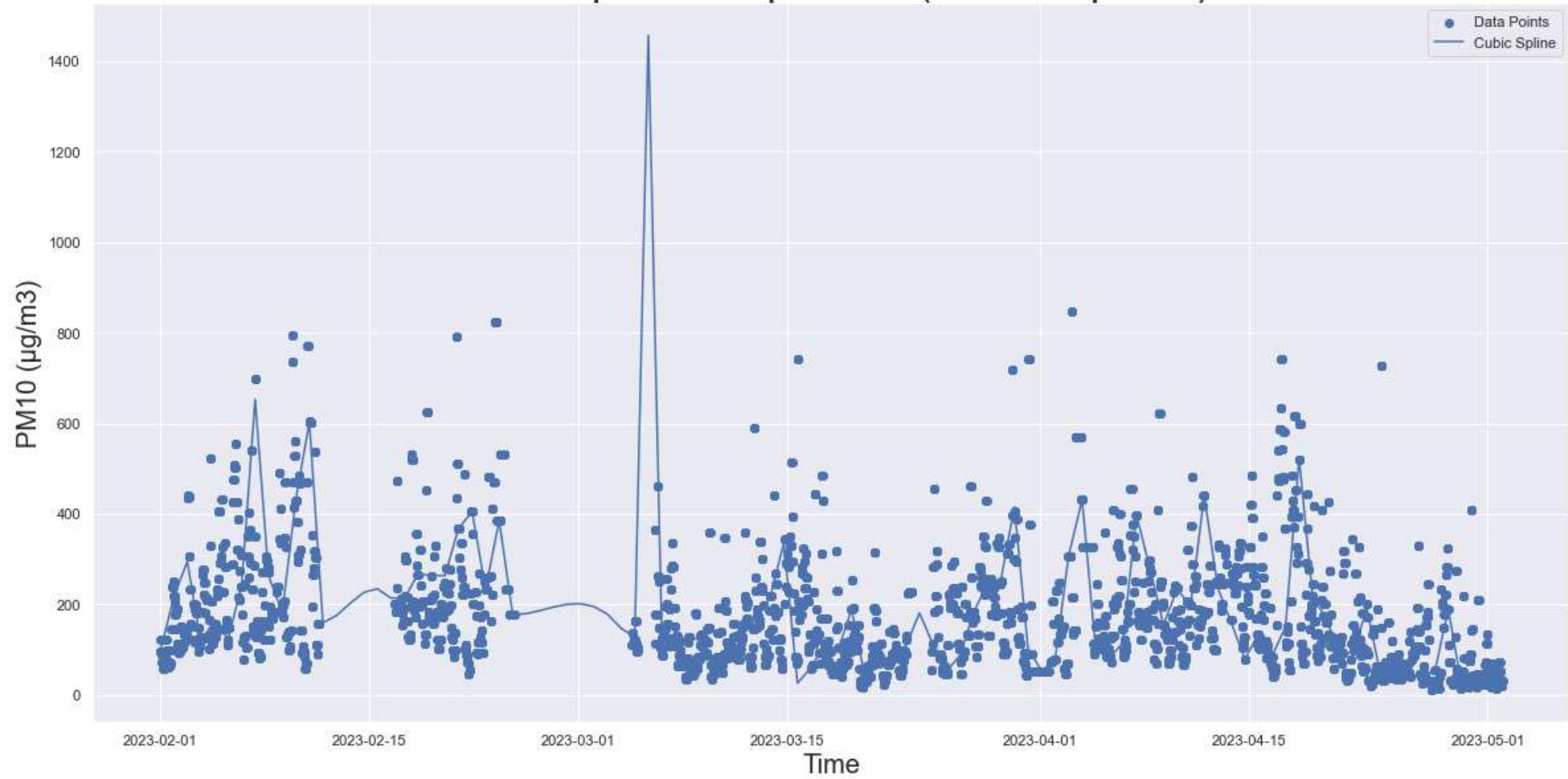


Quadratic Spline Interpolation (without datapoints)

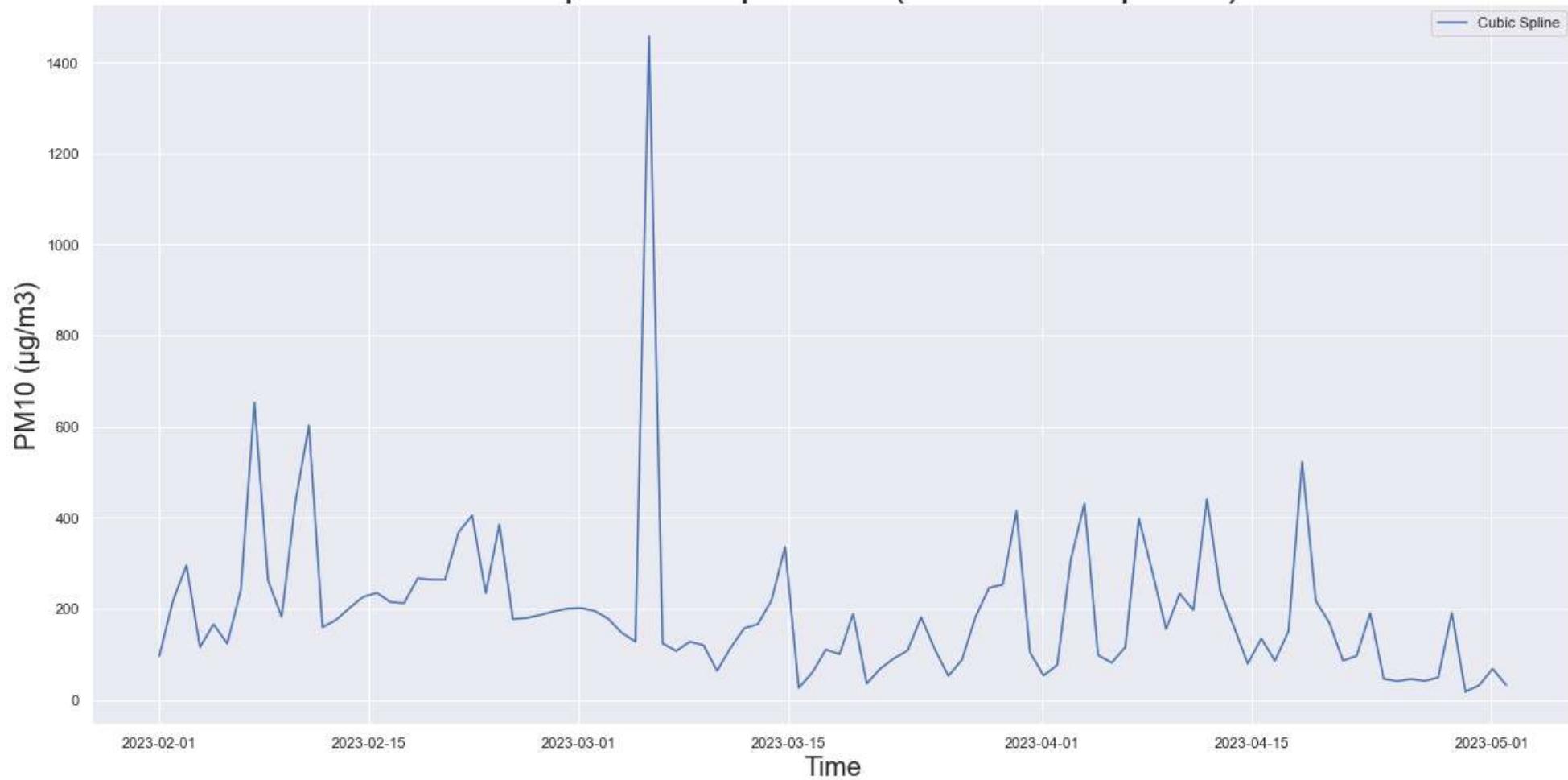


```
In [64]: cubicspline(df_n,pollutant)
```

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



Since mean value is 181.408679, and histogram suggests that most of the values lie around the mean i.e. mean is not affected by the outliers. Other plots like cubic and quadratic interpolations and splines suggest that data is overestimated for missing values which is ambiguous. Therefore, Linear Interpolation and mean filling seems to work best for filling missing values.

In [65]: `df_mean[pollutant].isnull().sum()`

Out[65]: 0

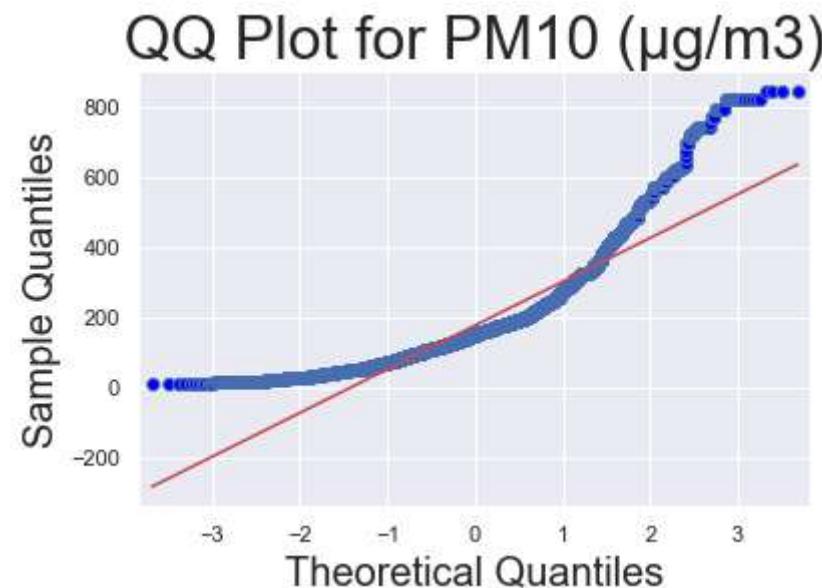
In [66]: `df_linear_f[pollutant].isnull().sum()`

Out[66]: 0

```
In [67]: df_new[pollutant] = df_linear_f[pollutant]
# Since non-null values are comparatively more, Linear seems to work best
```

```
In [68]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>



```
In [69]: df_new.head()
```

Out[69]:

PM10 ($\mu\text{g}/\text{m}^3$)

Time	PM10 ($\mu\text{g}/\text{m}^3$)
2023-02-01 00:00:00	95.0
2023-02-01 00:15:00	95.0
2023-02-01 00:30:00	95.0
2023-02-01 00:45:00	122.0
2023-02-01 01:00:00	122.0

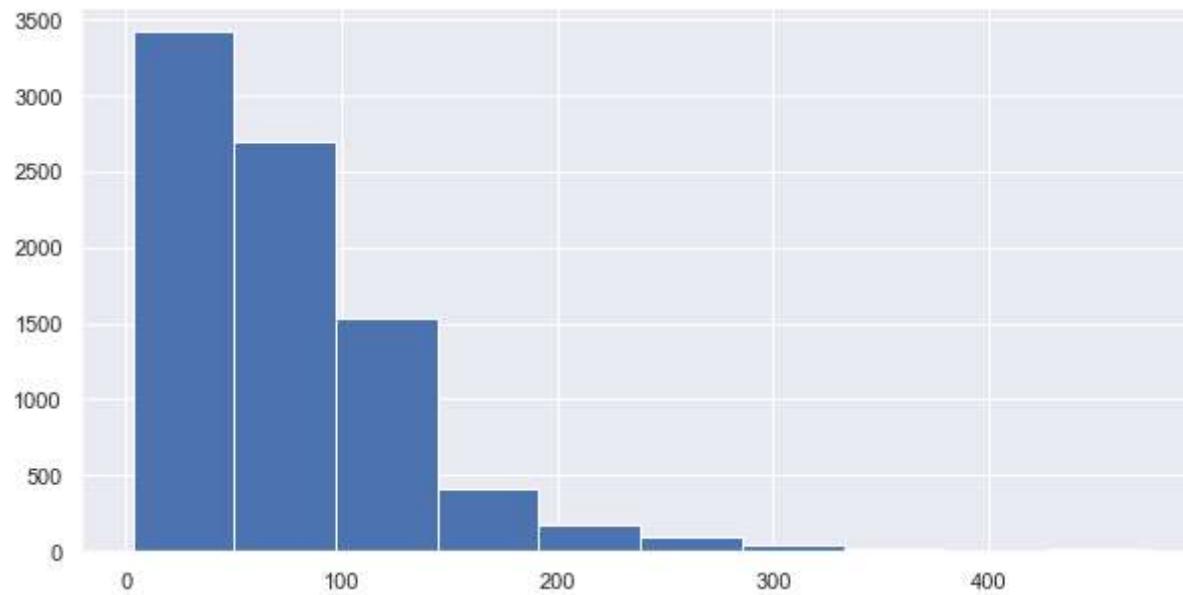
Analysis for "PM2.5 ($\mu\text{g}/\text{m}^3$)"

```
In [70]: pollutant = 'PM2.5 ( $\mu\text{g}/\text{m}^3$ )'
```

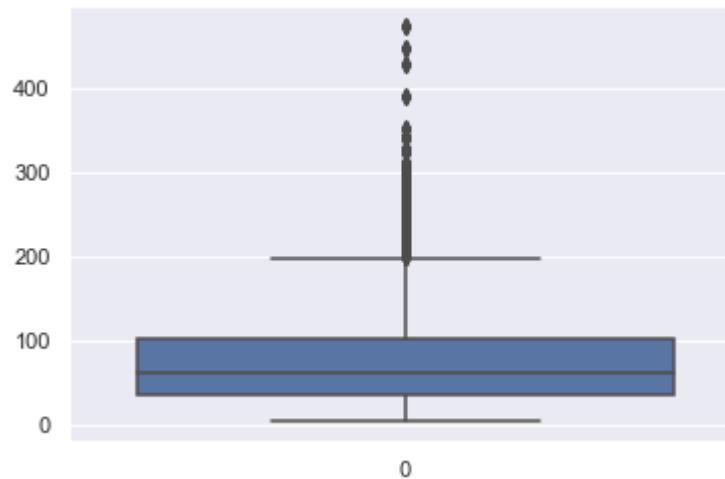
```
In [71]: df[pollutant].describe()
```

```
Out[71]: count    8414.000000
mean      75.690397
std       55.245265
min       3.000000
25%      36.000000
50%      61.000000
75%     101.000000
max     474.000000
Name: PM2.5 ( $\mu\text{g}/\text{m}^3$ ), dtype: float64
```

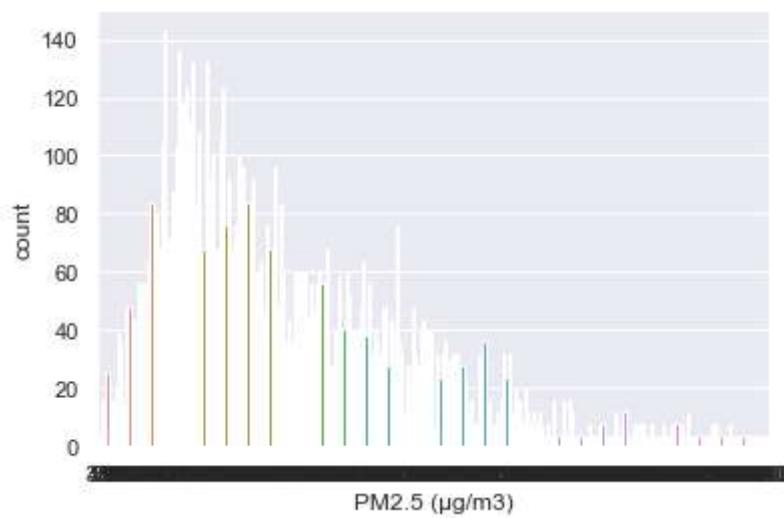
```
In [72]: histogram_plot(df_n,pollutant)
```



```
In [73]: boxplot_plot(df_n,pollutant)
```

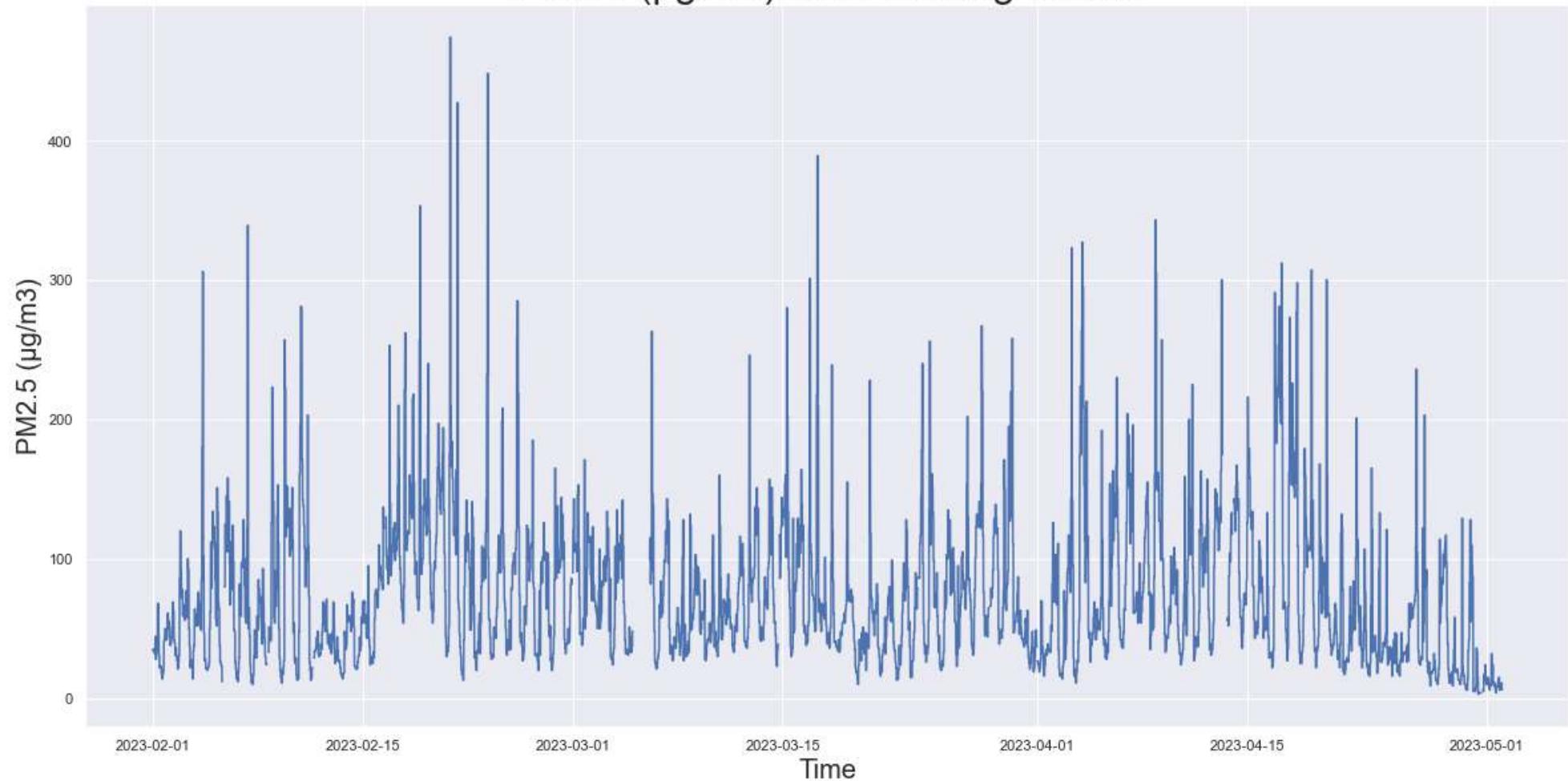


```
In [74]: countplot_plot(df_n,pollutant)
```



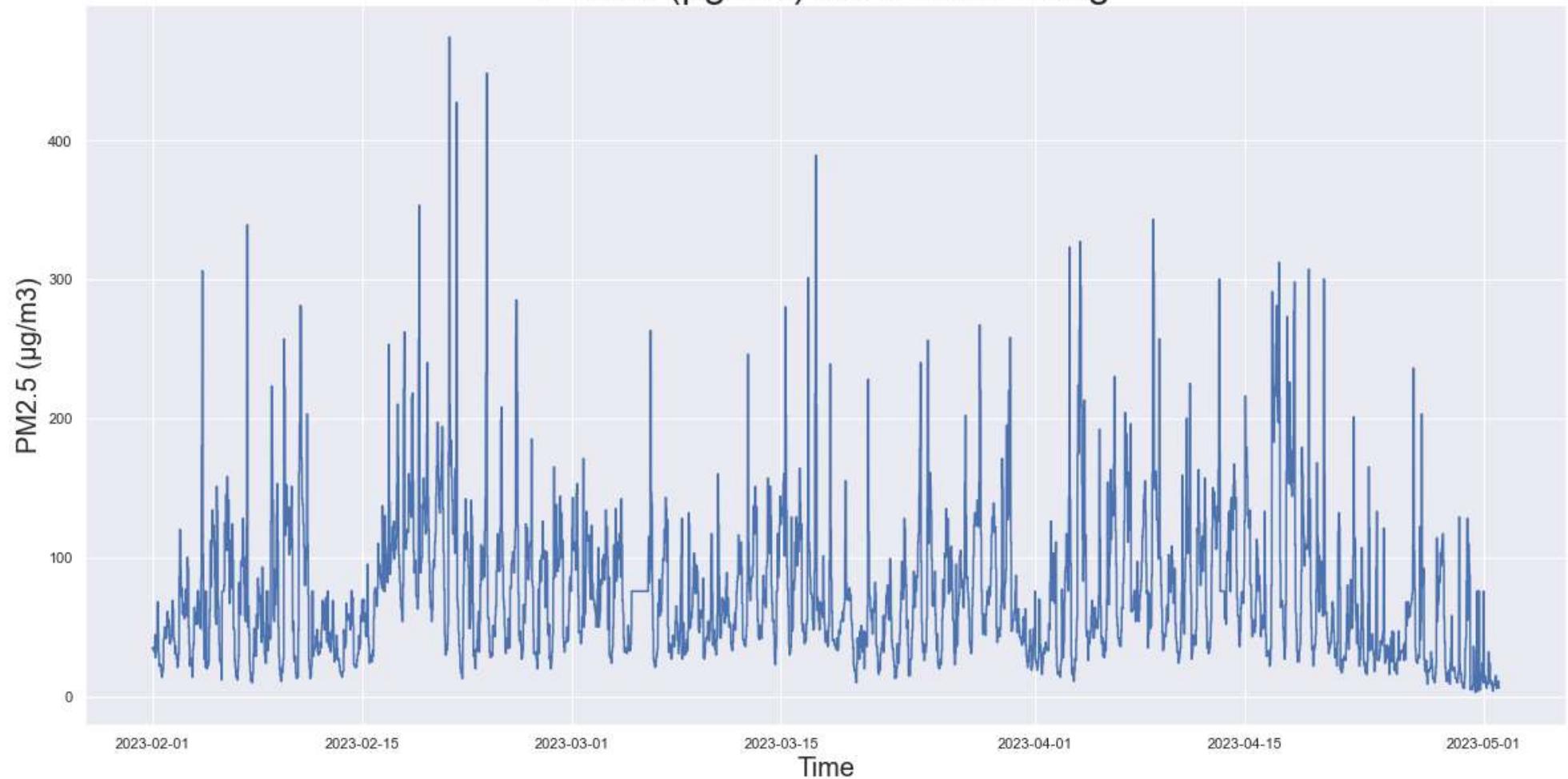
```
In [75]: simple_time_plot(df_n,pollutant,"With missing values")
```

PM2.5 ($\mu\text{g}/\text{m}^3$) With missing values



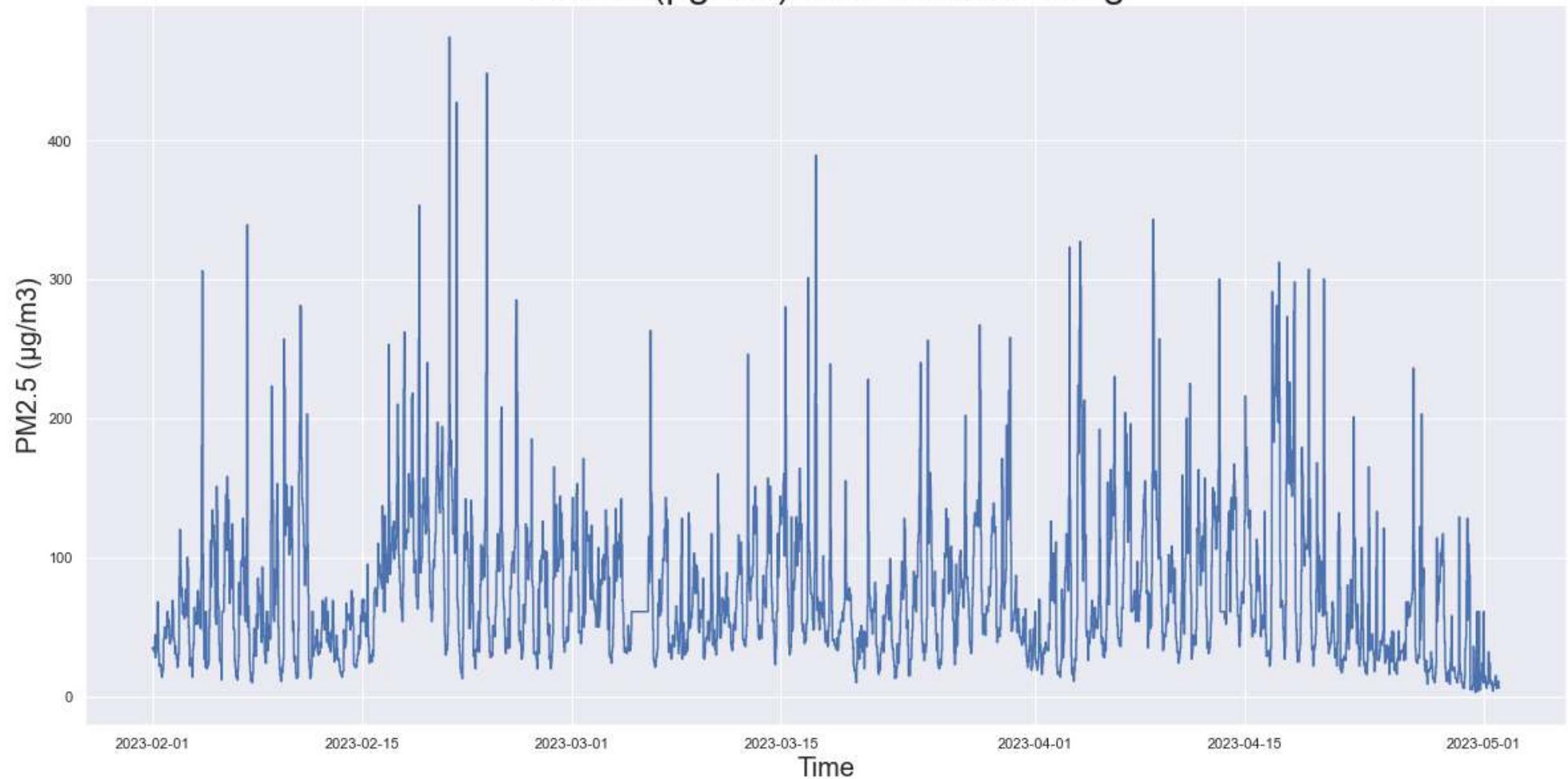
```
In [76]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

PM2.5 ($\mu\text{g}/\text{m}^3$) after mean filling



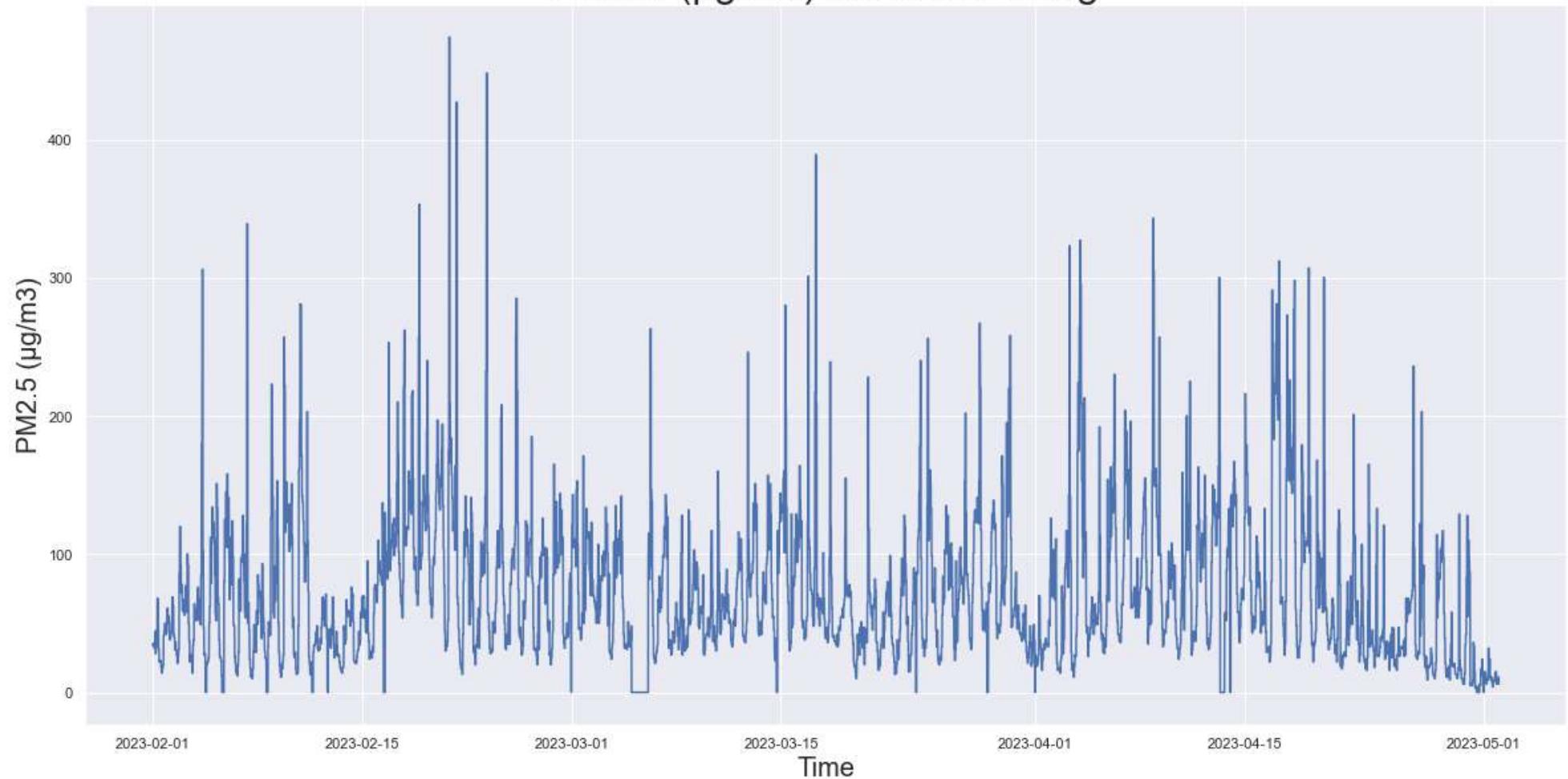
```
In [77]: simple_time_plot(df_median,pollutant,"after median filling")
```

PM2.5 ($\mu\text{g}/\text{m}^3$) after median filling



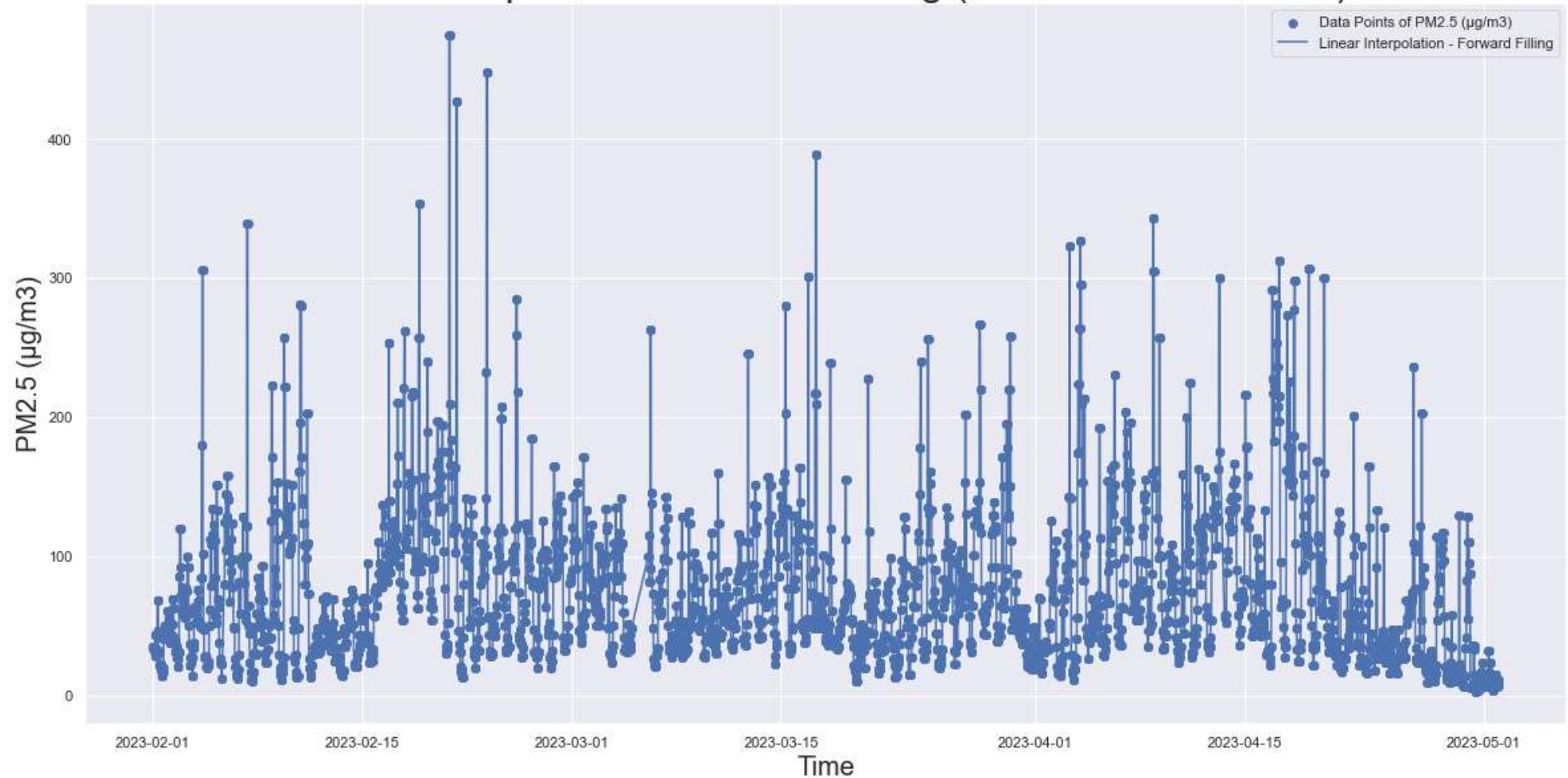
```
In [78]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

PM2.5 ($\mu\text{g}/\text{m}^3$) after zero filling

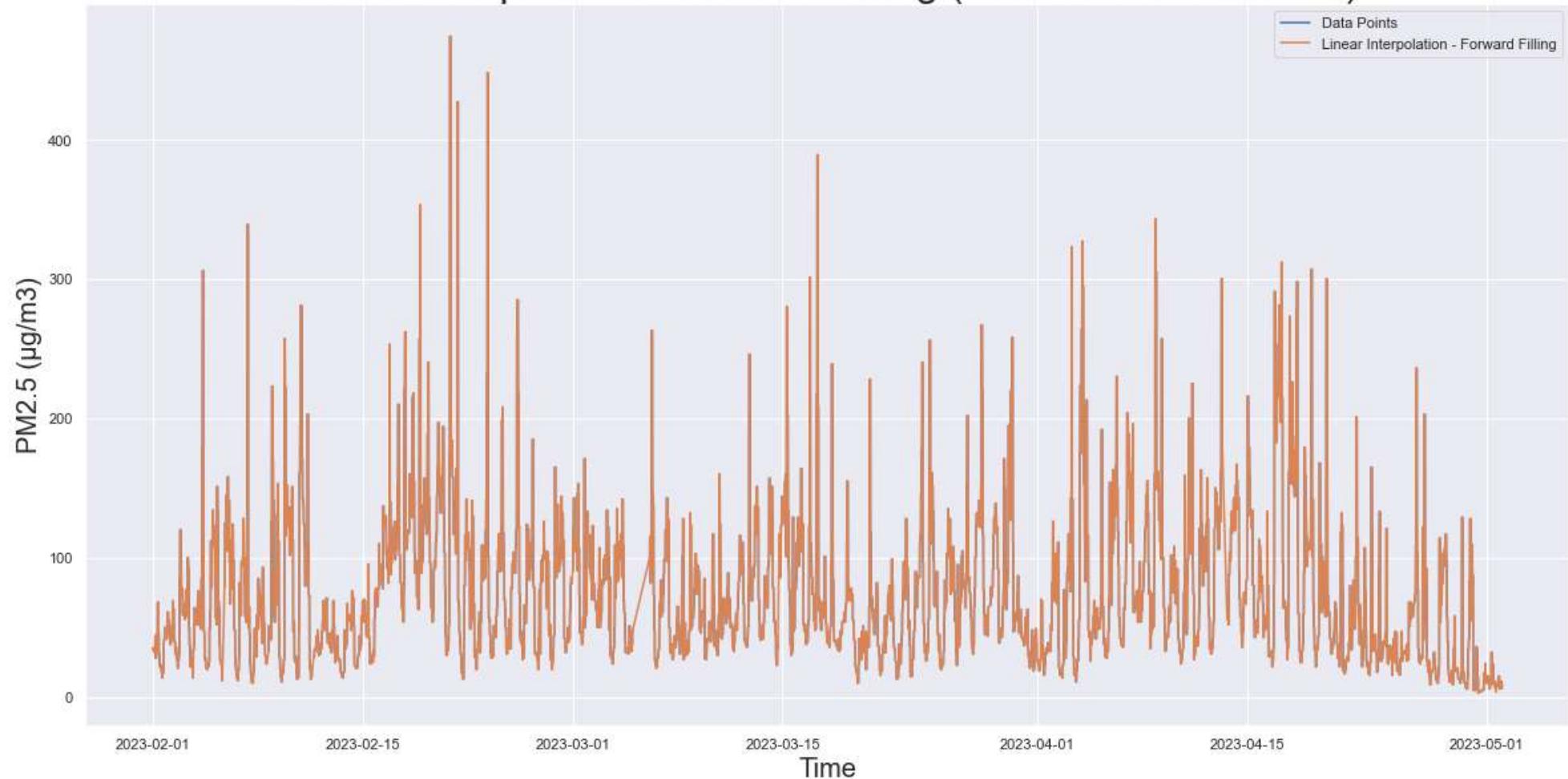


```
In [79]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```

Linear Interpolation - Forward Filling (Data Points Included)

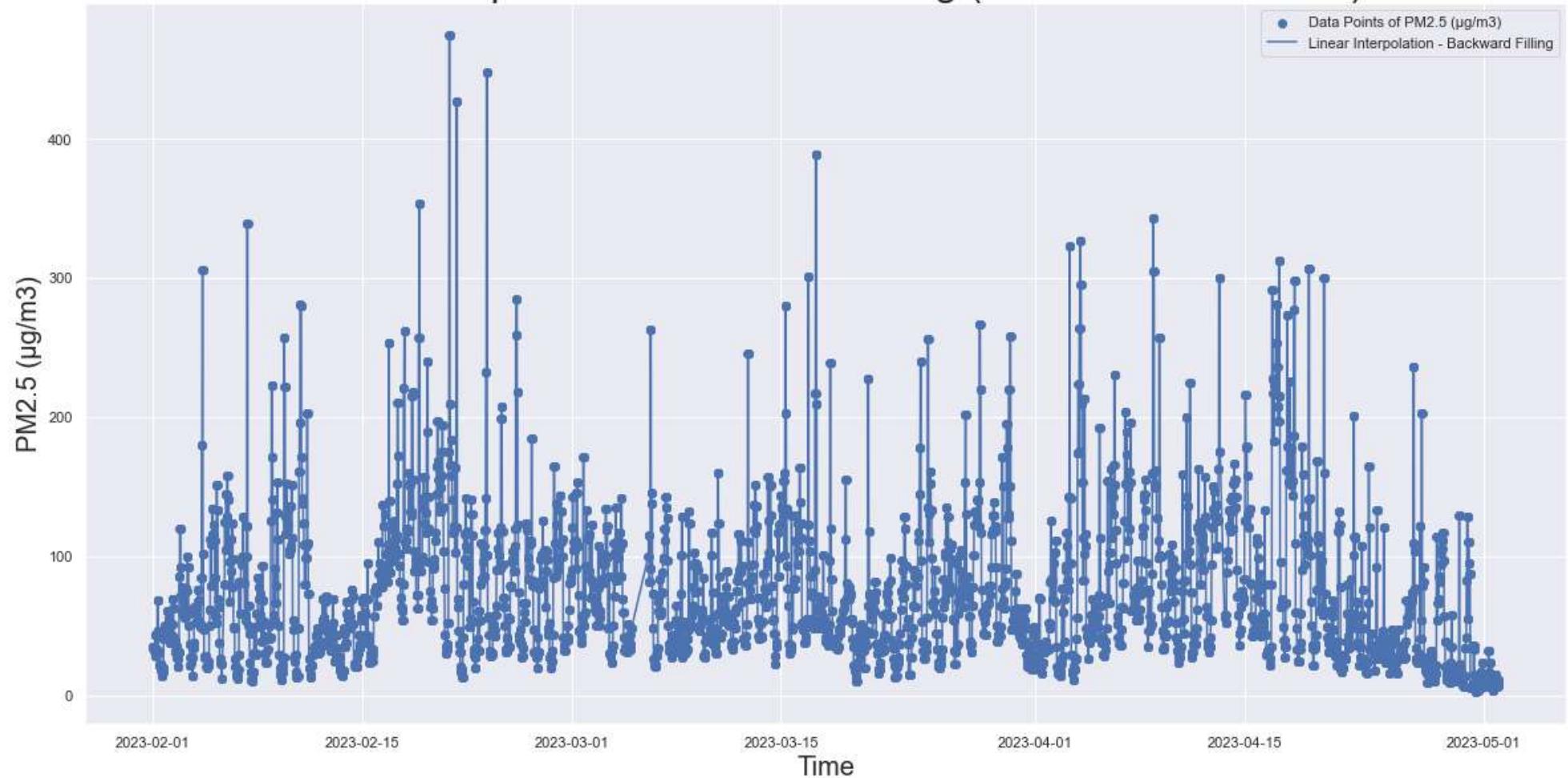


Linear Interpolation - Forward Filling (Data Points Excluded)

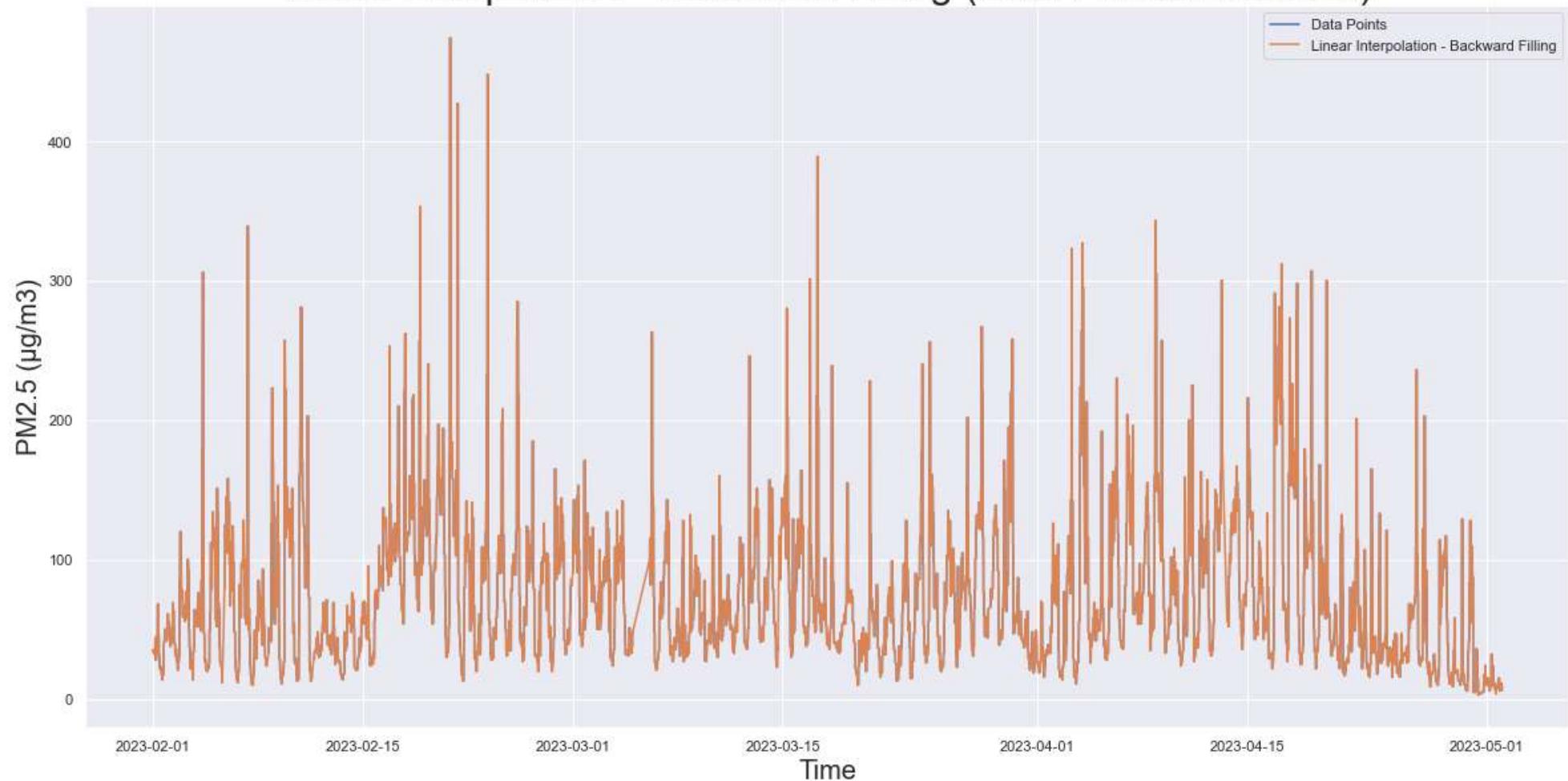


```
In [80]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

Linear Interpolation - Backward Filling (Data Points Included)

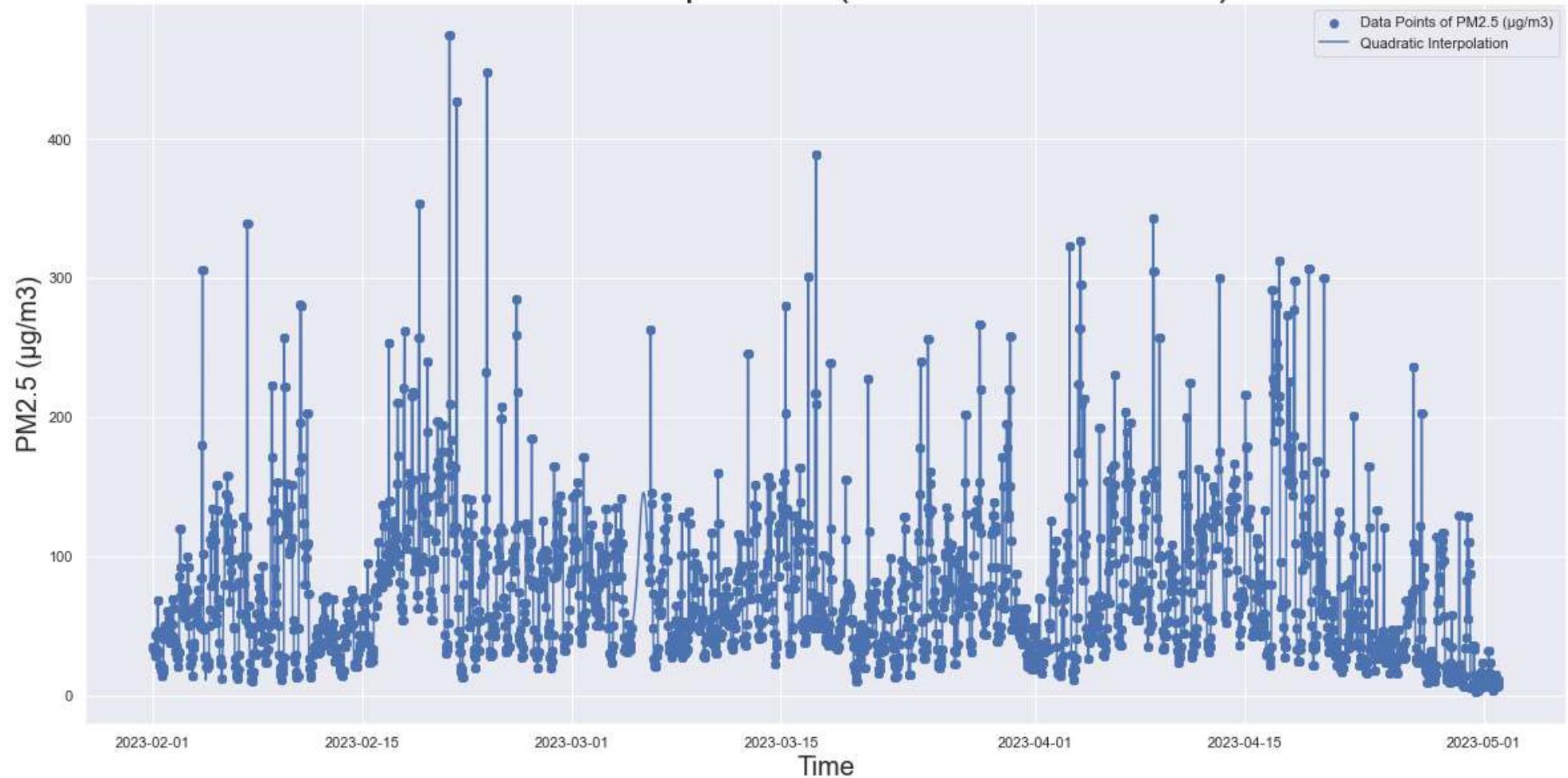


Linear Interpolation - Backward Filling (Data Points Excluded)

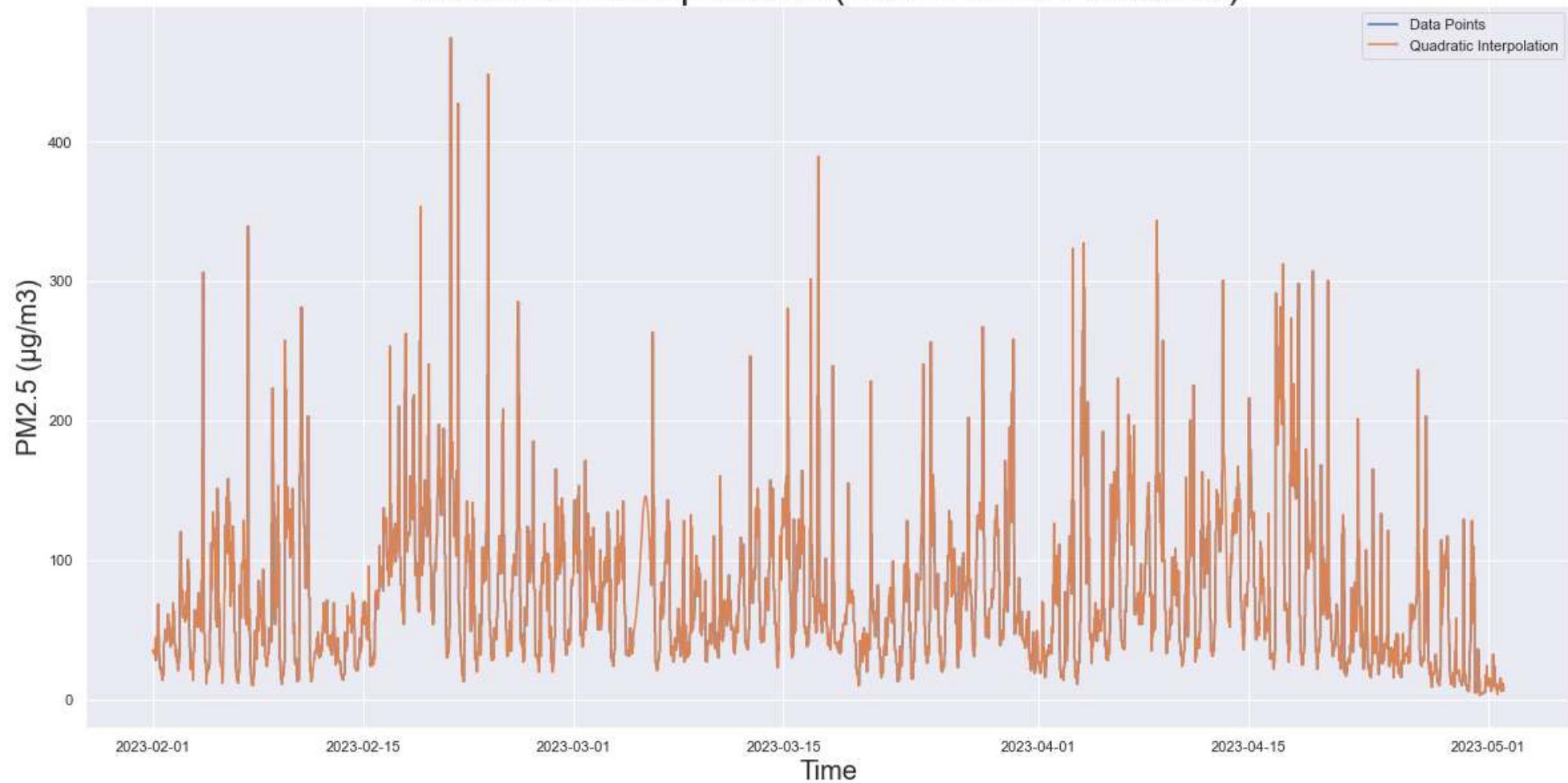


```
In [81]: interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")
```

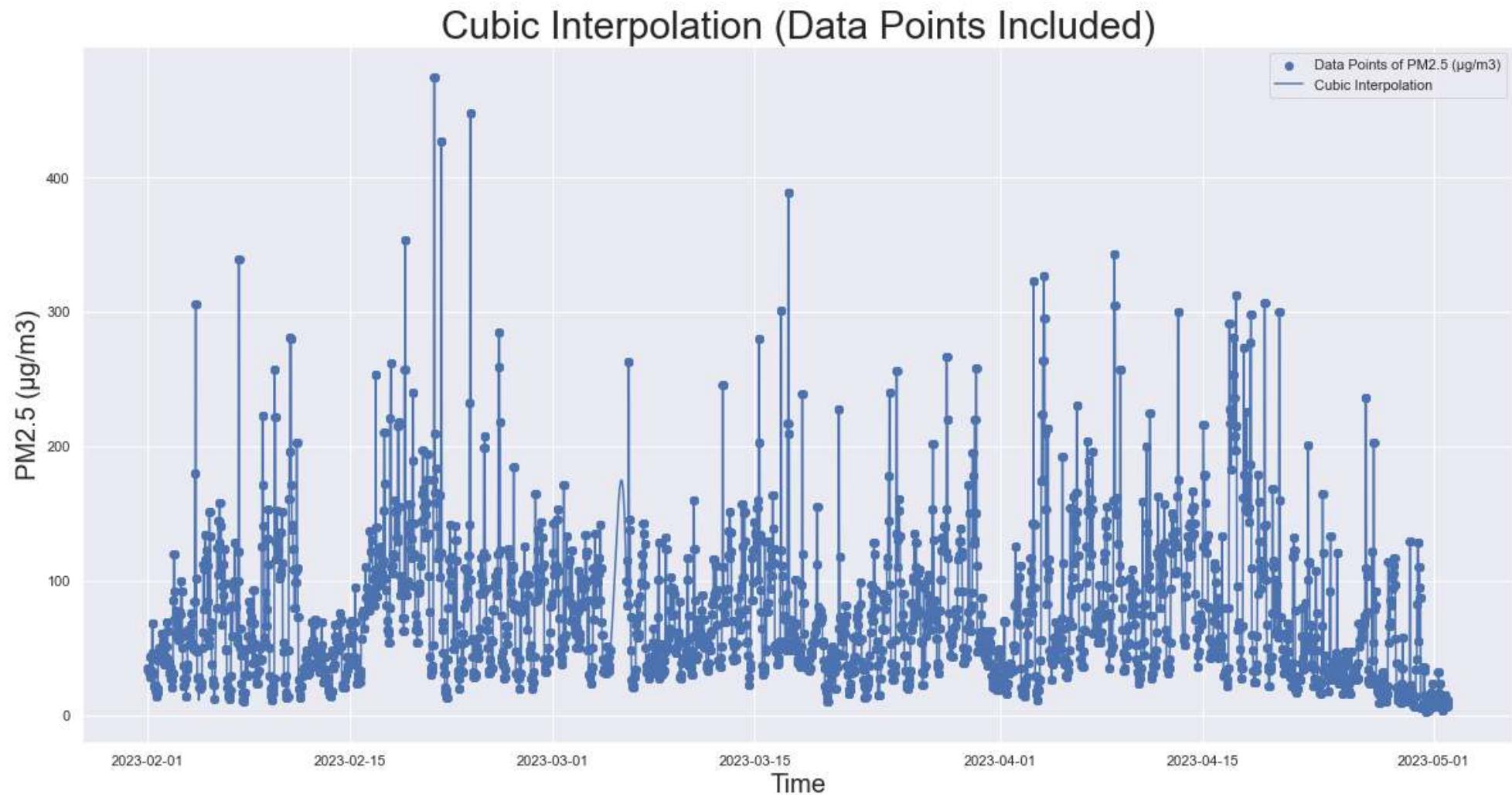
Quadratic Interpolation (Data Points Included)



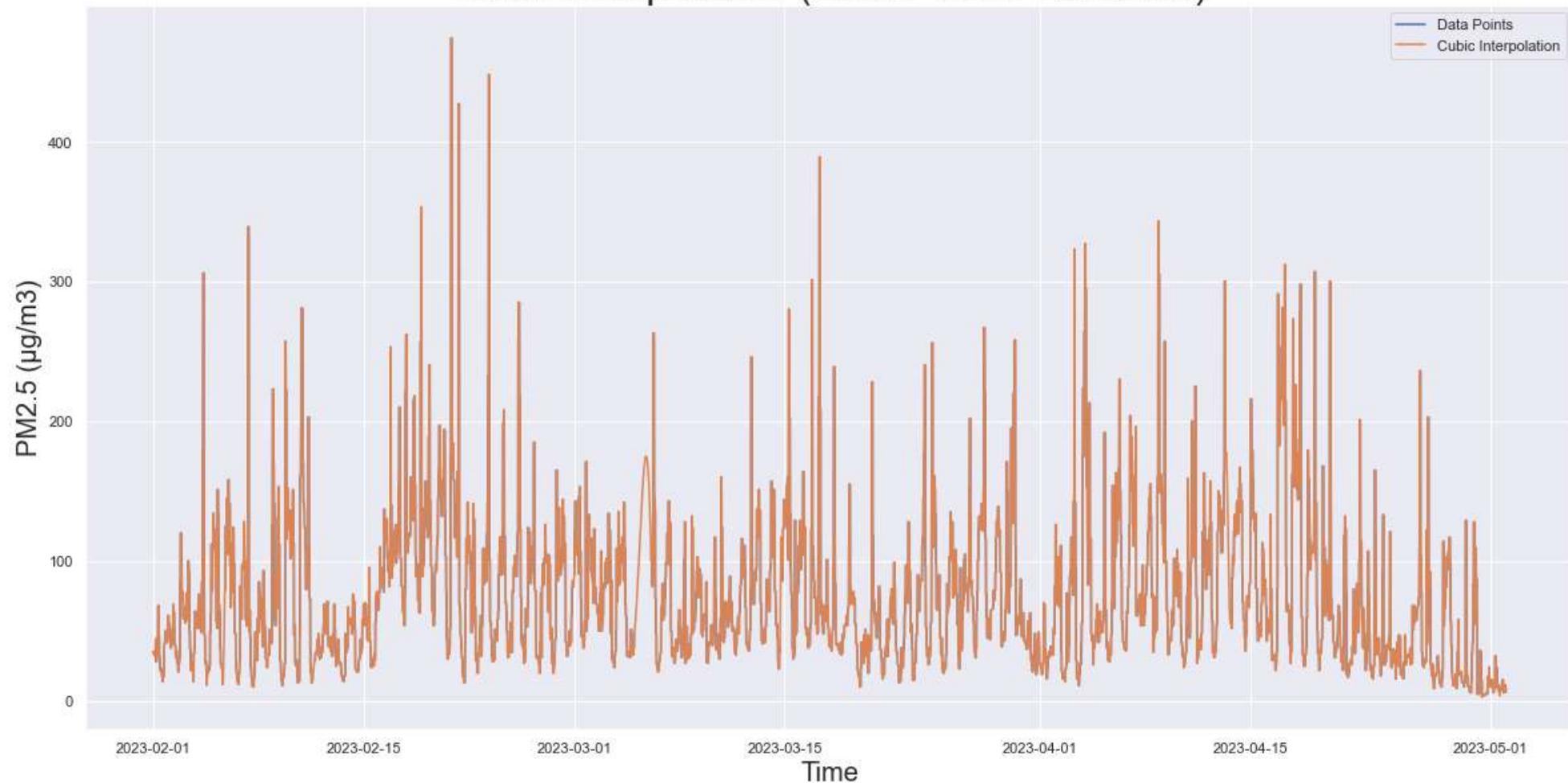
Quadratic Interpolation (Data Points Excluded)



```
In [82]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```

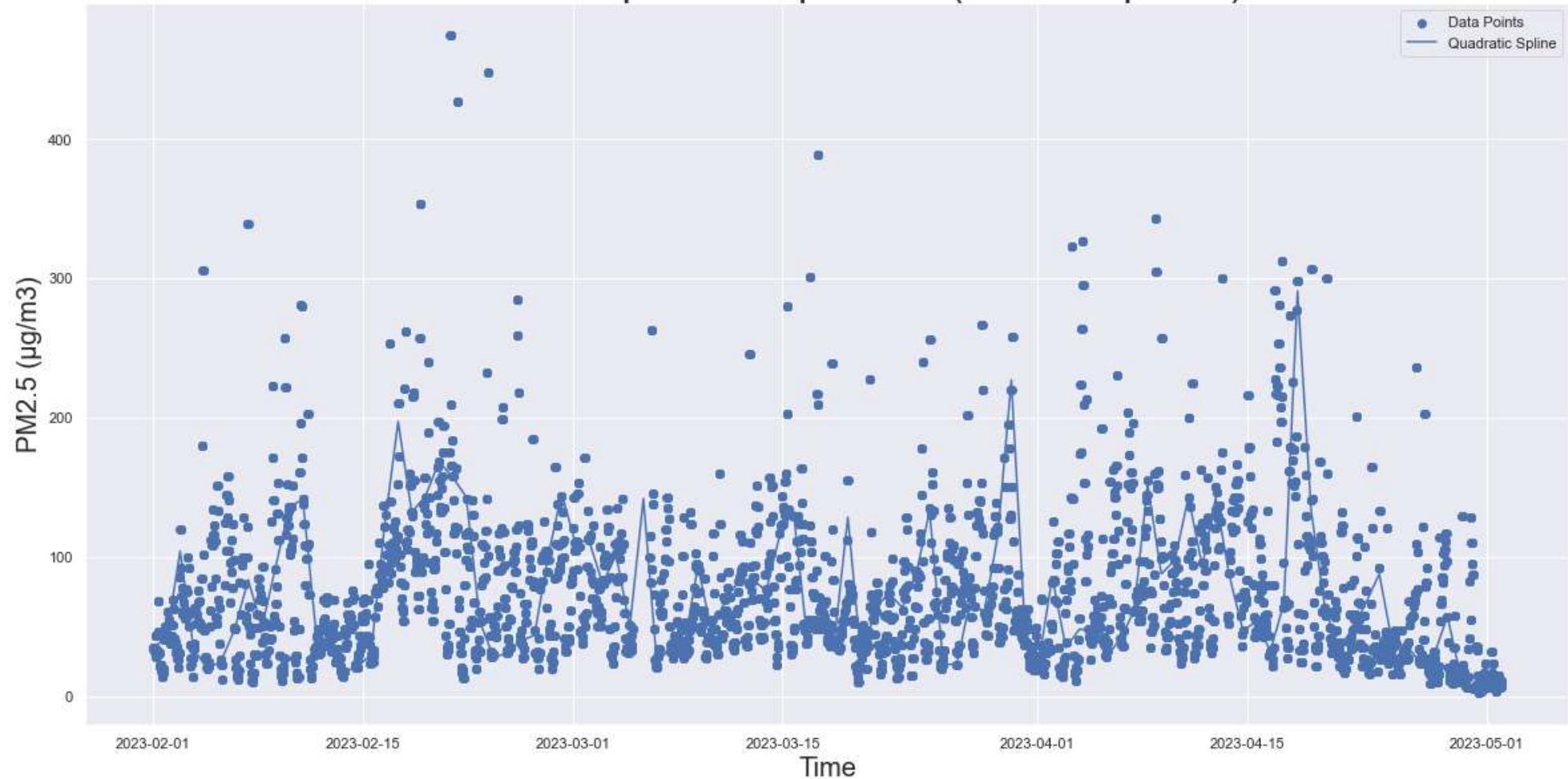


Cubic Interpolation (Data Points Excluded)

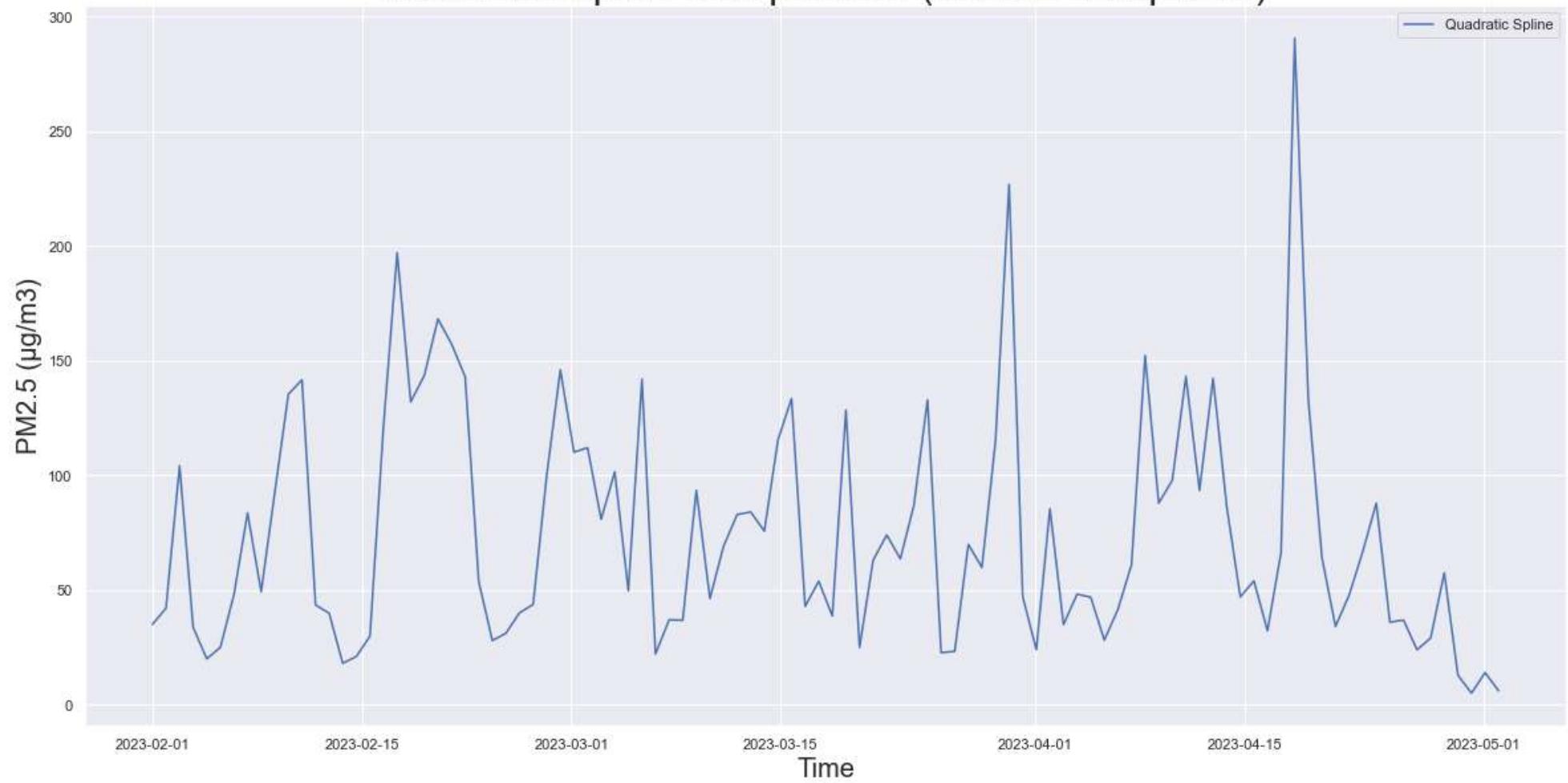


```
In [83]: quadraticspline(df_n,pollutant)
```

Quadratic Spline Interpolation (with datapoints)

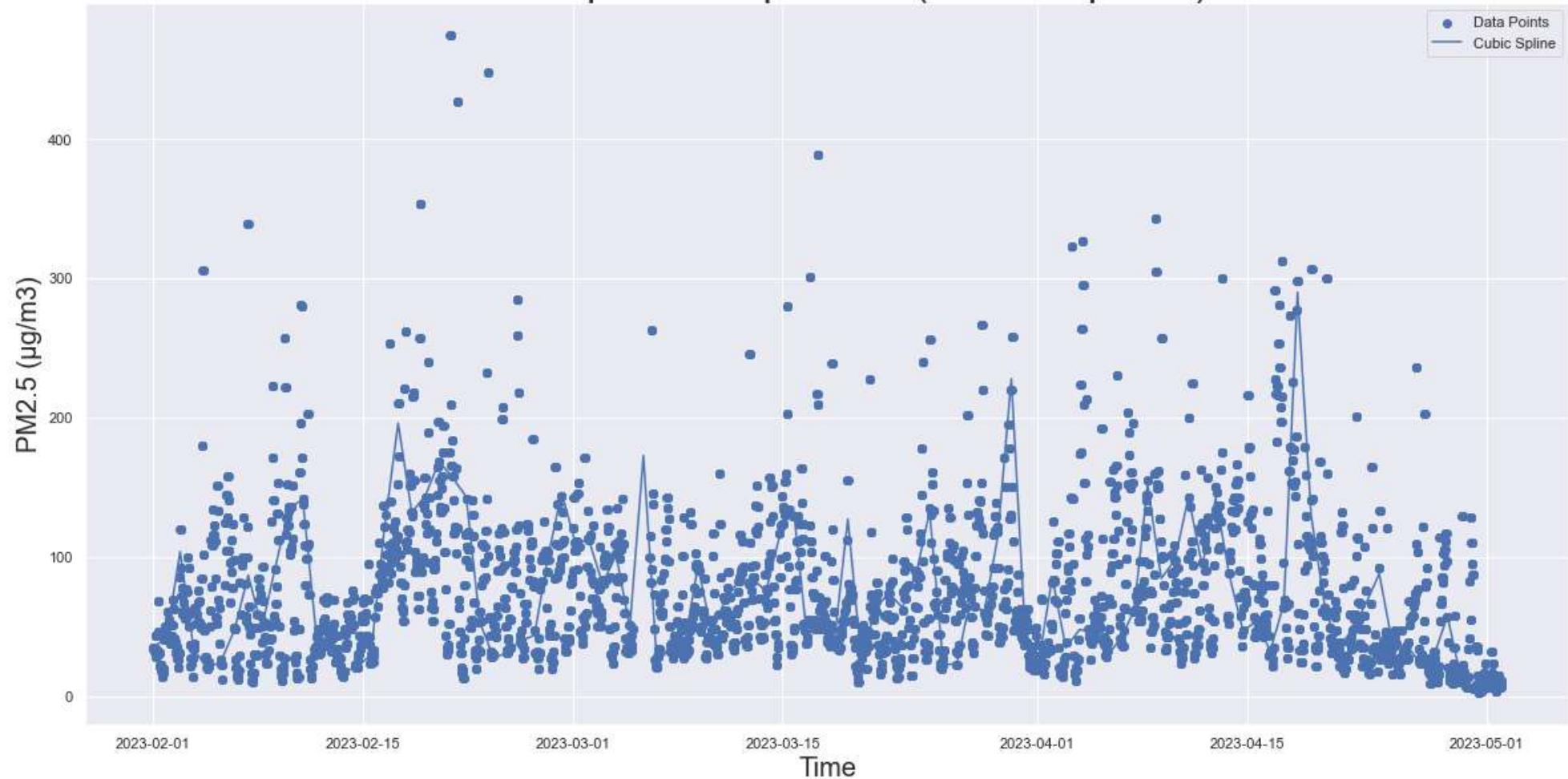


Quadratic Spline Interpolation (without datapoints)

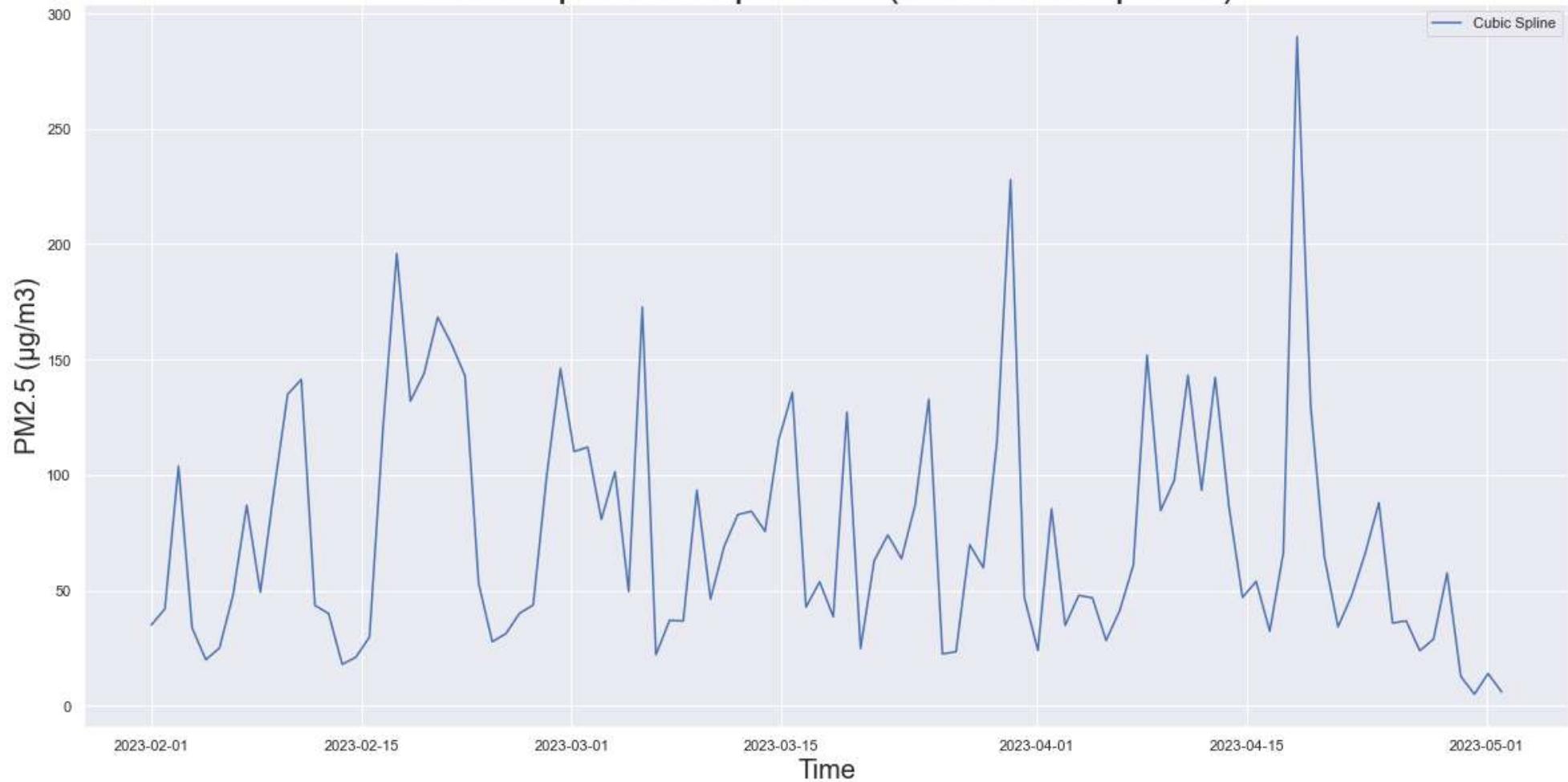


```
In [84]: cubicspline(df_n,pollutant)
```

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



Since outliers are less and boxplot and histogram suggest that most of the value is concentrated around mean which is 75.690397, linear interpolation seems to work best because data seems to vary more. Linear is able to account for trend in rise and fall of data as evident from the plots.

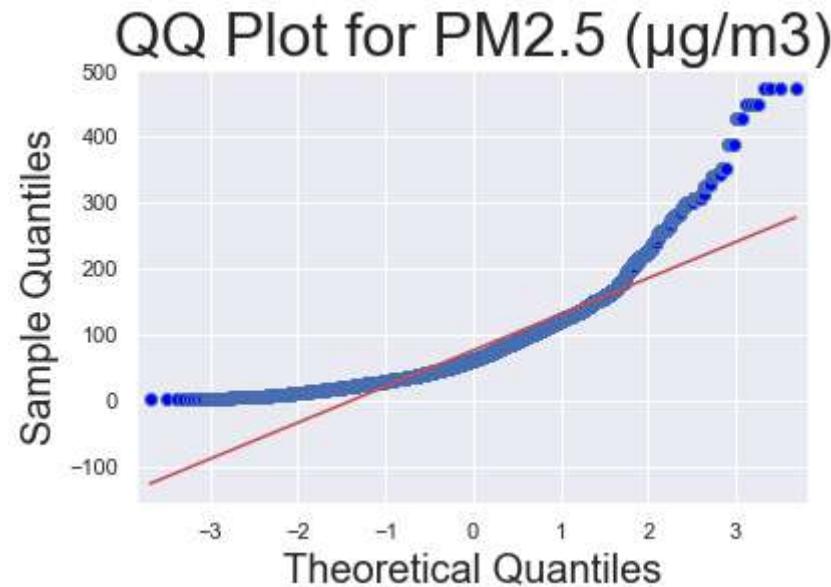
In [85]: `df_linear_f[pollutant].isnull().sum()`

Out[85]: 0

In [86]: `df_new[pollutant] = df_linear_f[pollutant]`

```
In [87]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>



```
In [88]: df_new.head()
```

Out[88]:

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)
Time		

Time	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)
2023-02-01 00:00:00	95.0	35.0
2023-02-01 00:15:00	95.0	35.0
2023-02-01 00:30:00	95.0	35.0
2023-02-01 00:45:00	122.0	34.0
2023-02-01 01:00:00	122.0	34.0

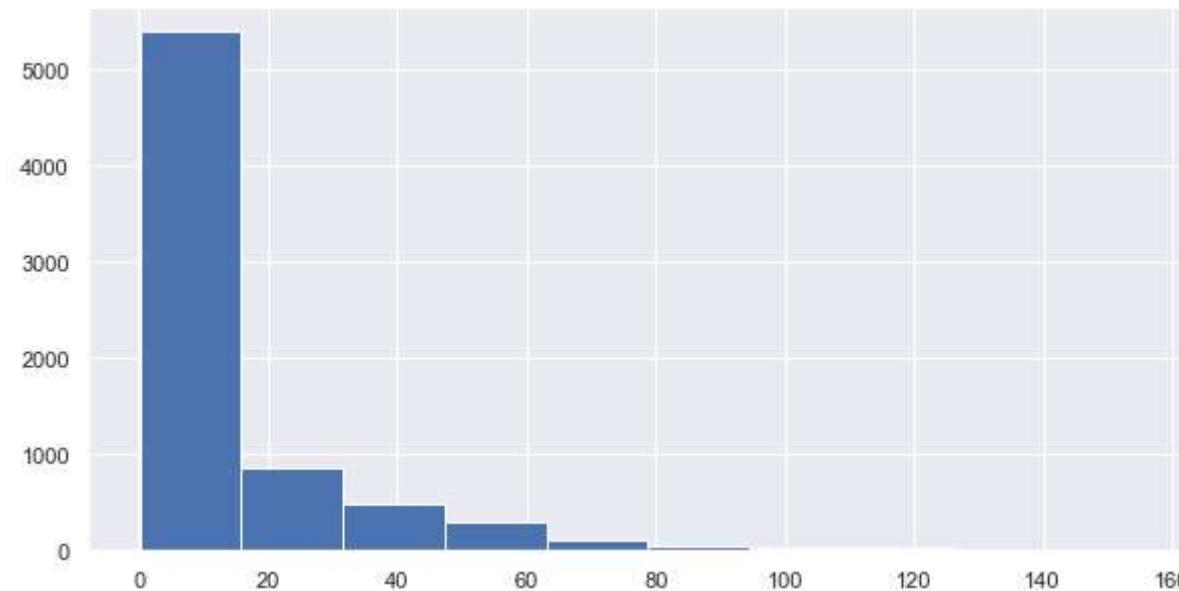
Analysis for "NO ($\mu\text{g}/\text{m}^3$)"

```
In [89]: pollutant = 'NO (\u03bcg/m\u00b3)'
```

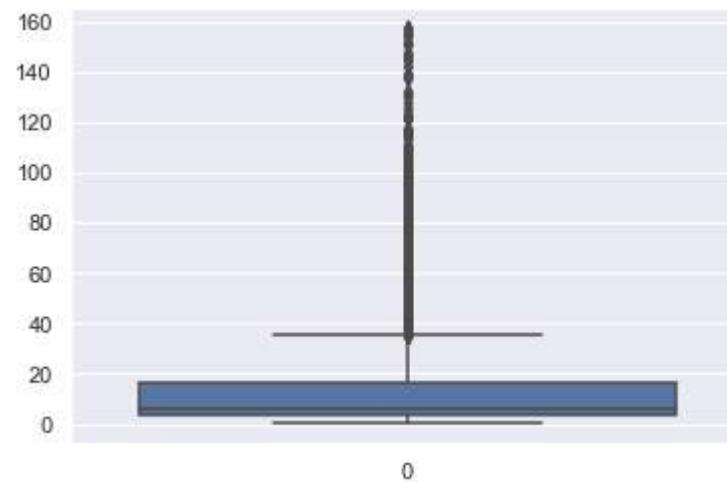
```
In [90]: df[pollutant].describe()
```

```
Out[90]: count    7271.000000
mean      14.649636
std       19.221385
min       0.100000
25%      3.900000
50%      6.100000
75%     16.500000
max     157.500000
Name: NO (µg/m³), dtype: float64
```

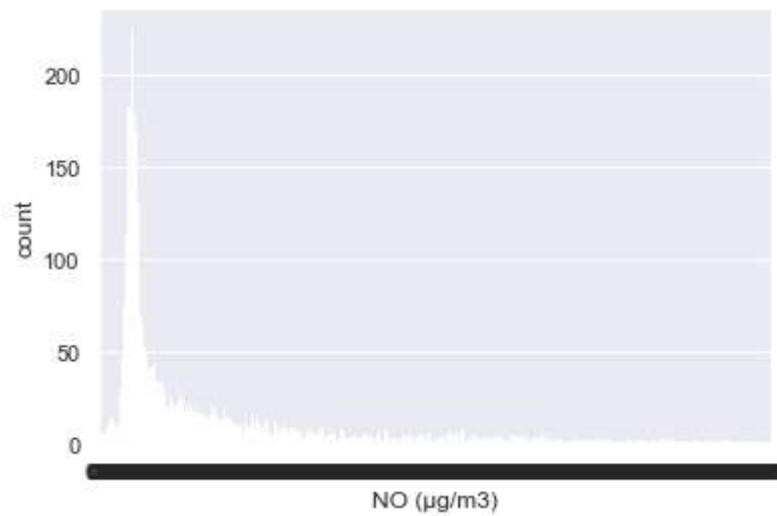
```
In [91]: histogram_plot(df_n,pollutant)
```



```
In [92]: boxplot_plot(df_n,pollutant)
```

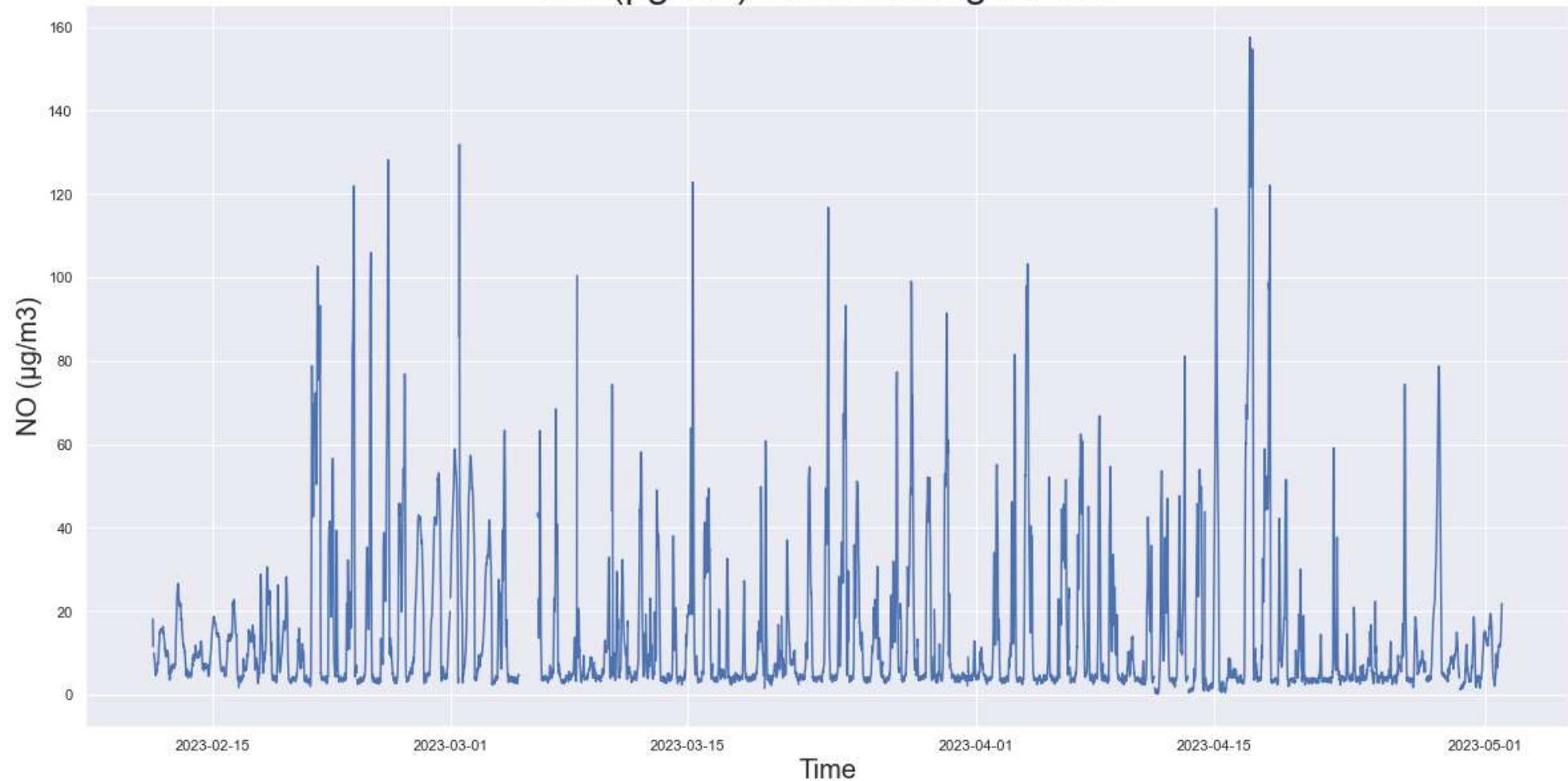


```
In [93]: countplot_plot(df_n,pollutant)
```



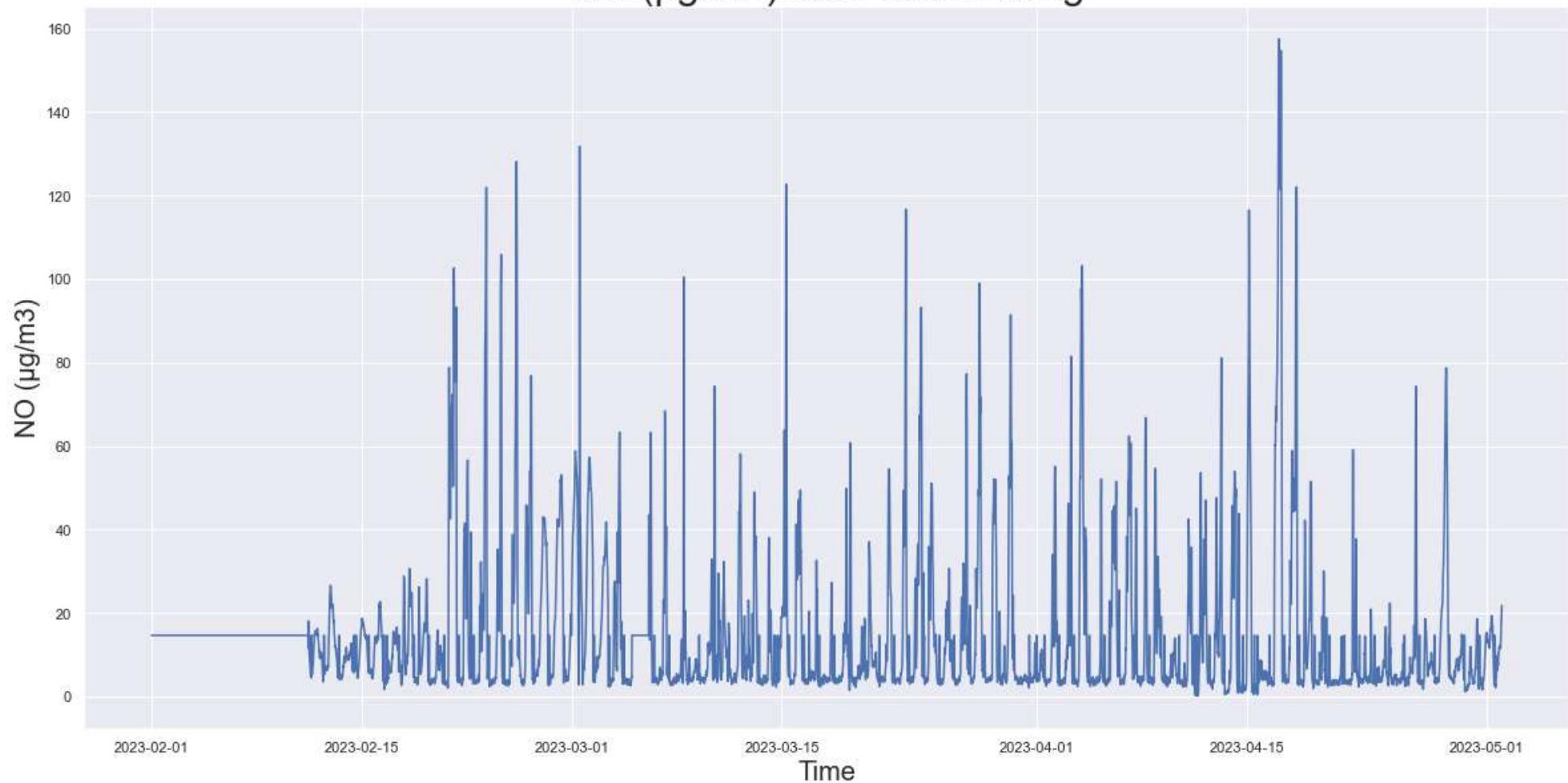
```
In [94]: simple_time_plot(df_n,pollutant,"With missing values")
```

NO ($\mu\text{g}/\text{m}^3$) With missing values



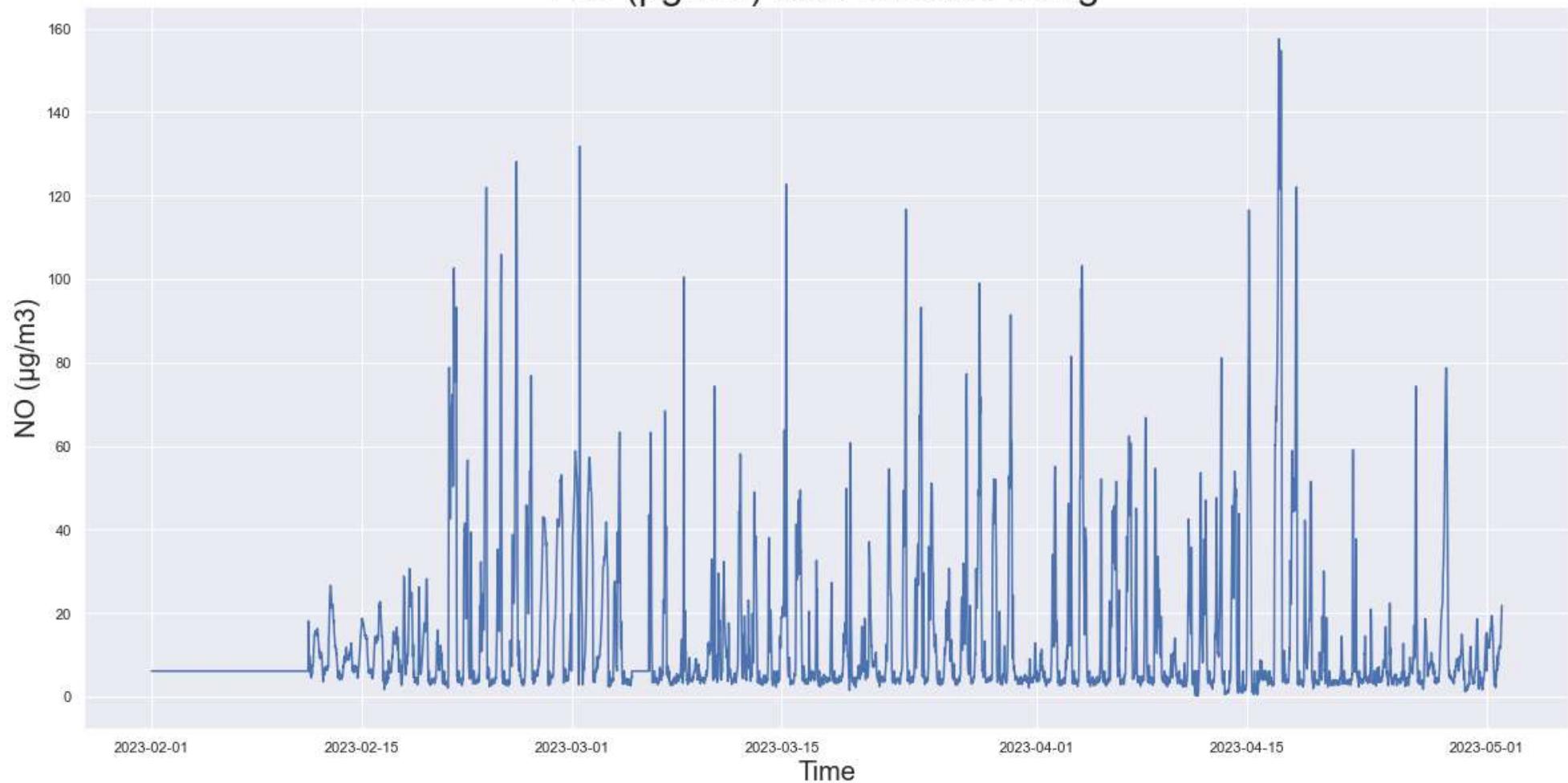
```
In [95]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

NO ($\mu\text{g}/\text{m}^3$) after mean filling



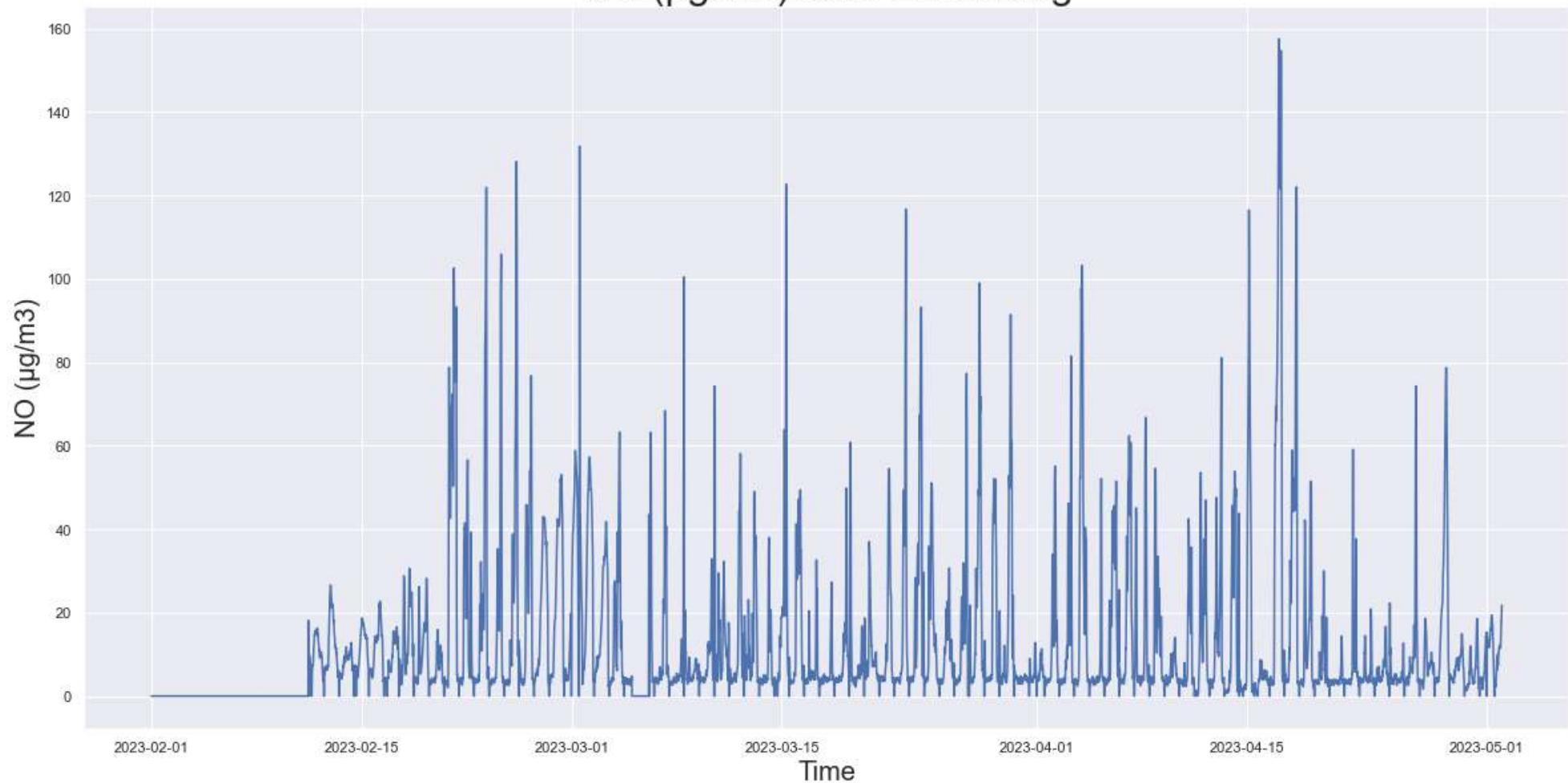
```
In [96]: simple_time_plot(df_median,pollutant,"after median filling")
```

NO ($\mu\text{g}/\text{m}^3$) after median filling

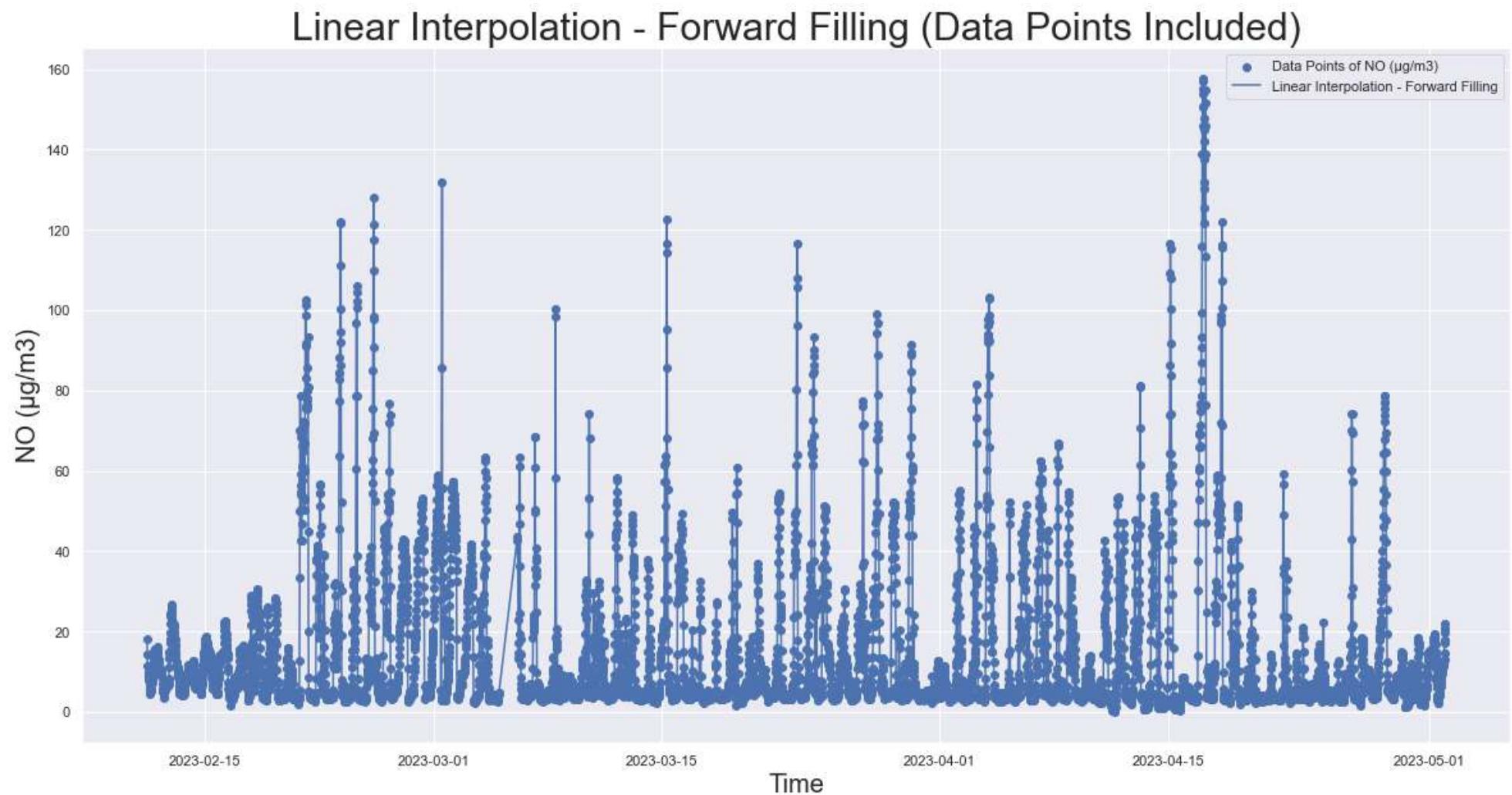


```
In [97]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

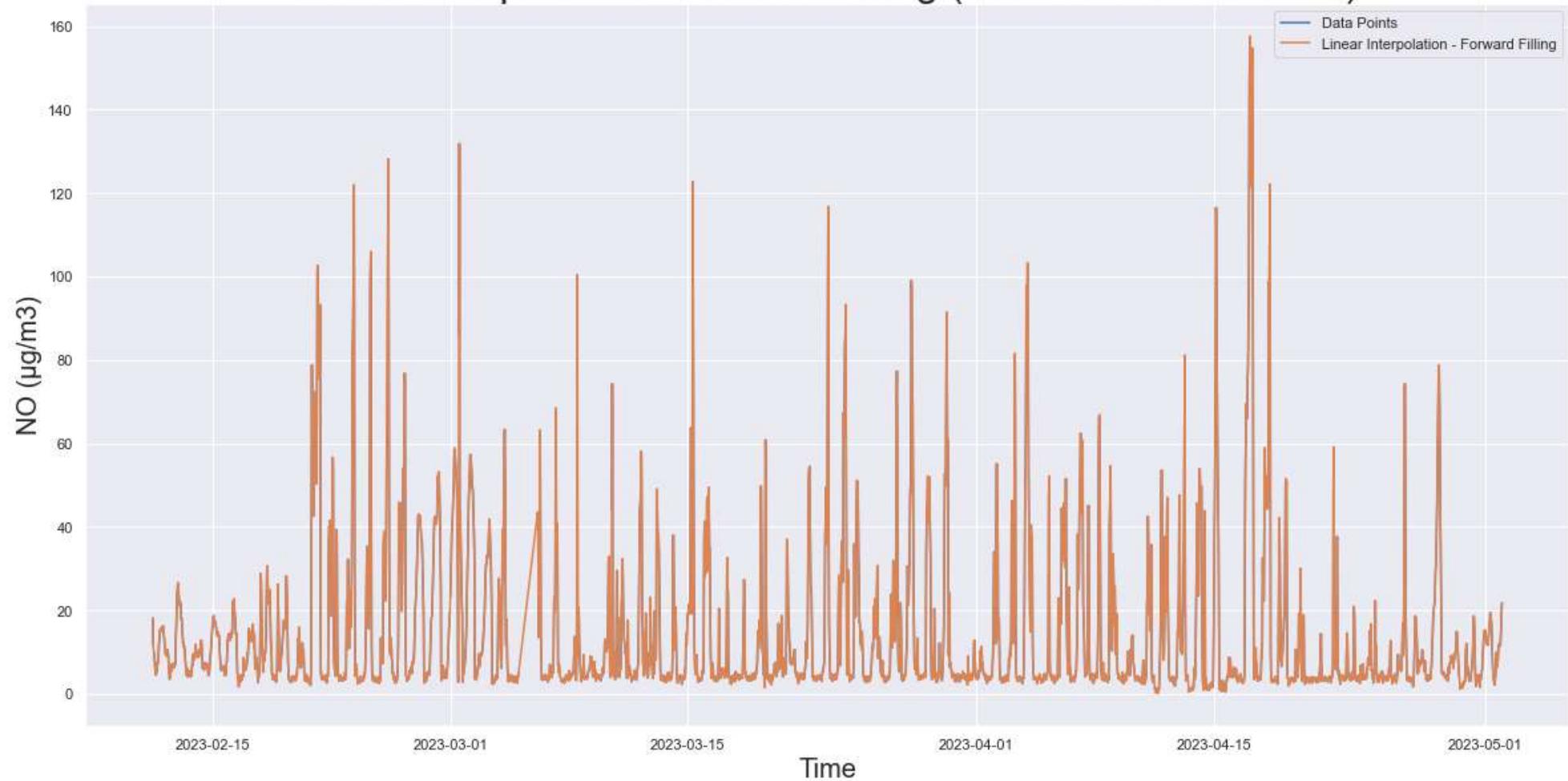
NO ($\mu\text{g}/\text{m}^3$) after zero filling



```
In [98]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```

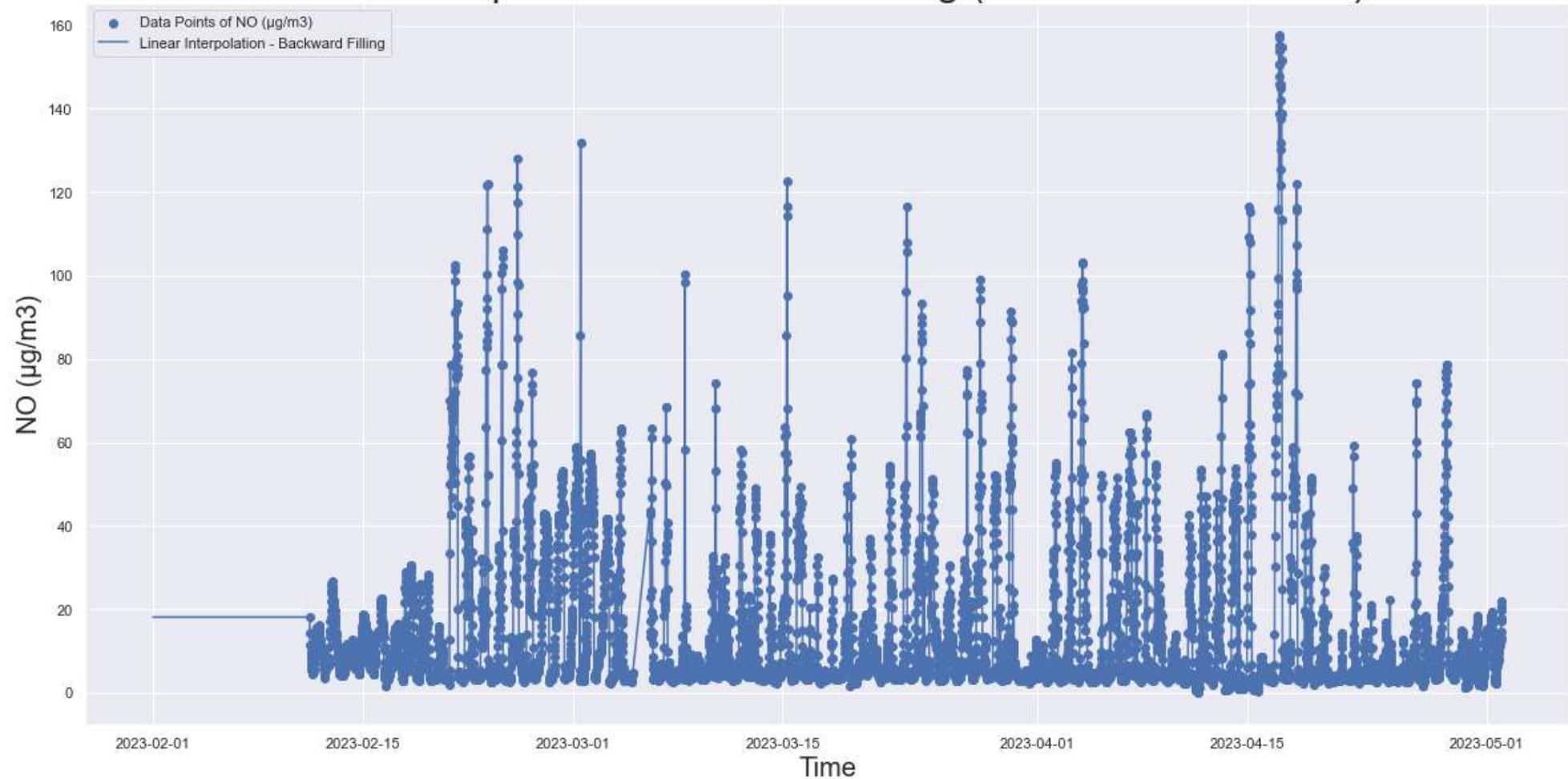


Linear Interpolation - Forward Filling (Data Points Excluded)

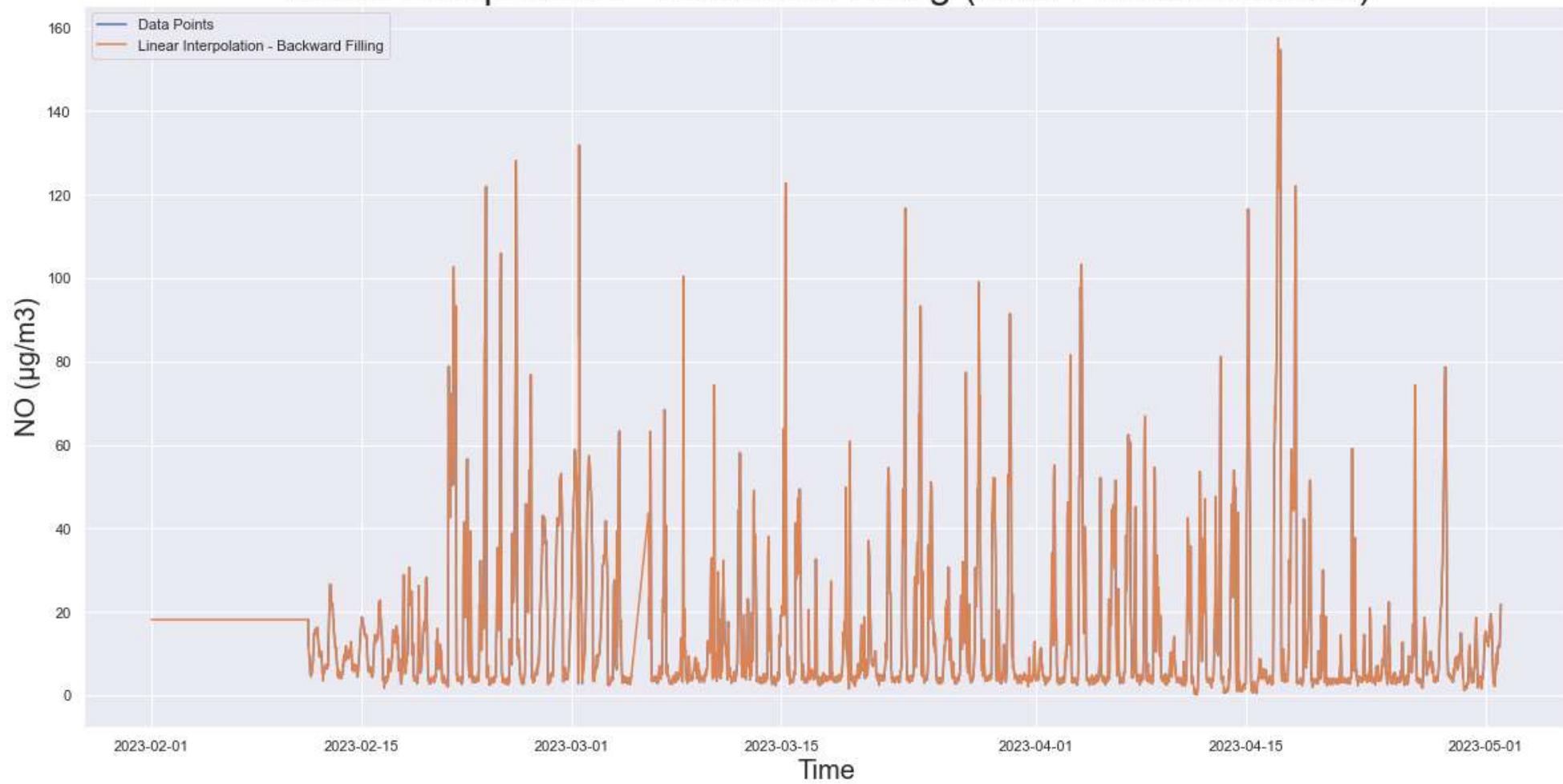


```
In [99]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

Linear Interpolation - Backward Filling (Data Points Included)

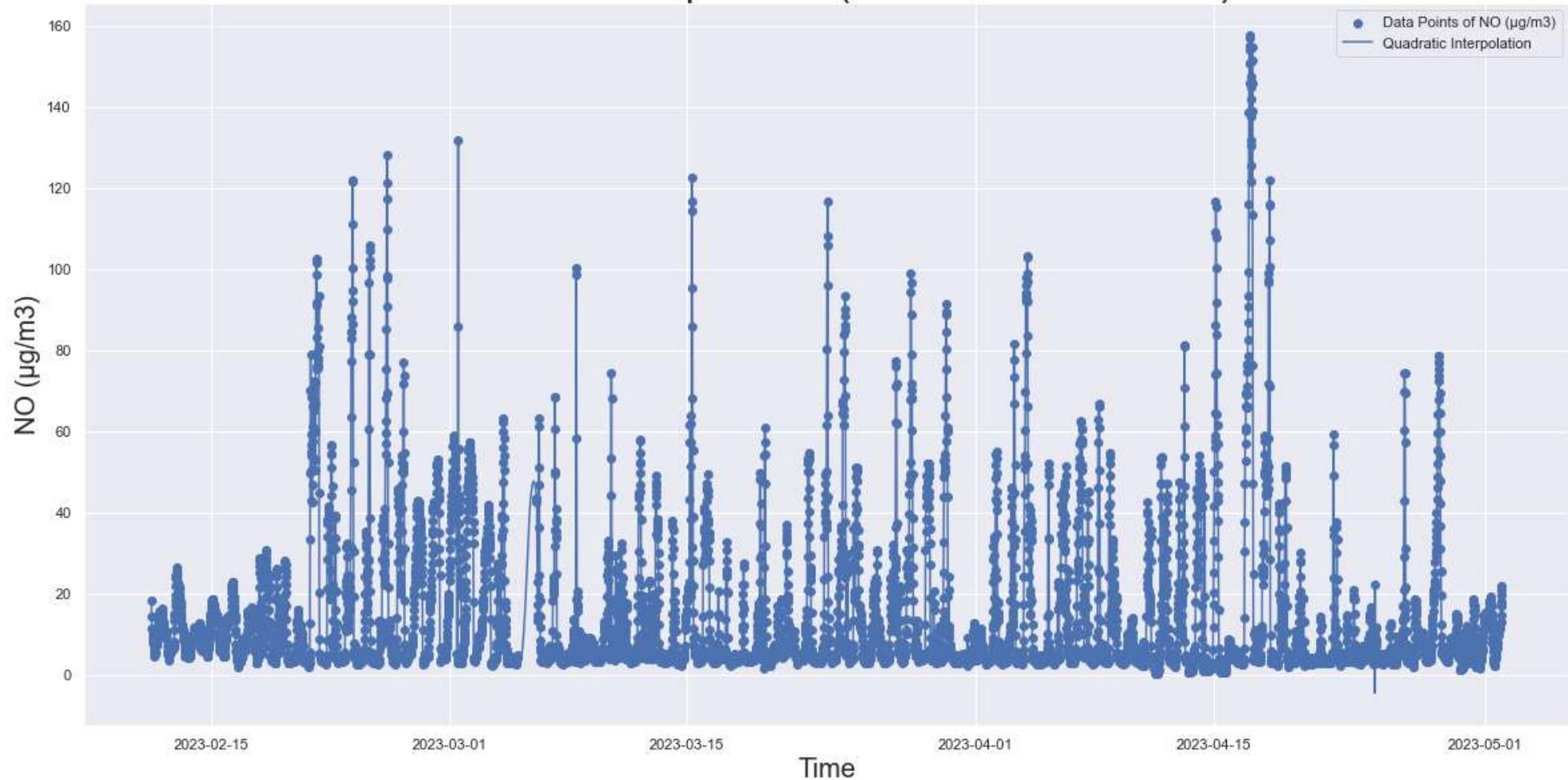


Linear Interpolation - Backward Filling (Data Points Excluded)

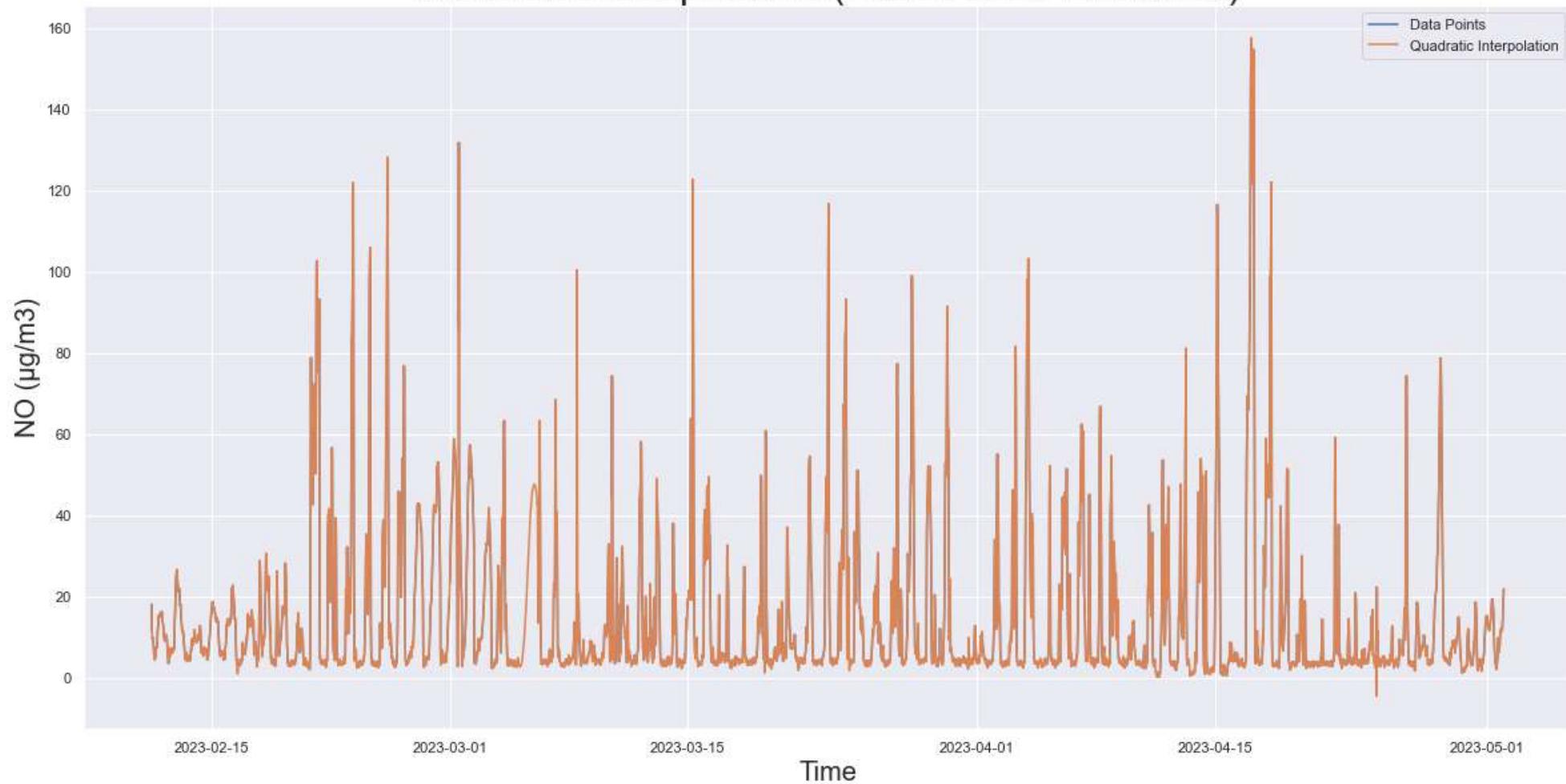


```
In [100]: interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")
```

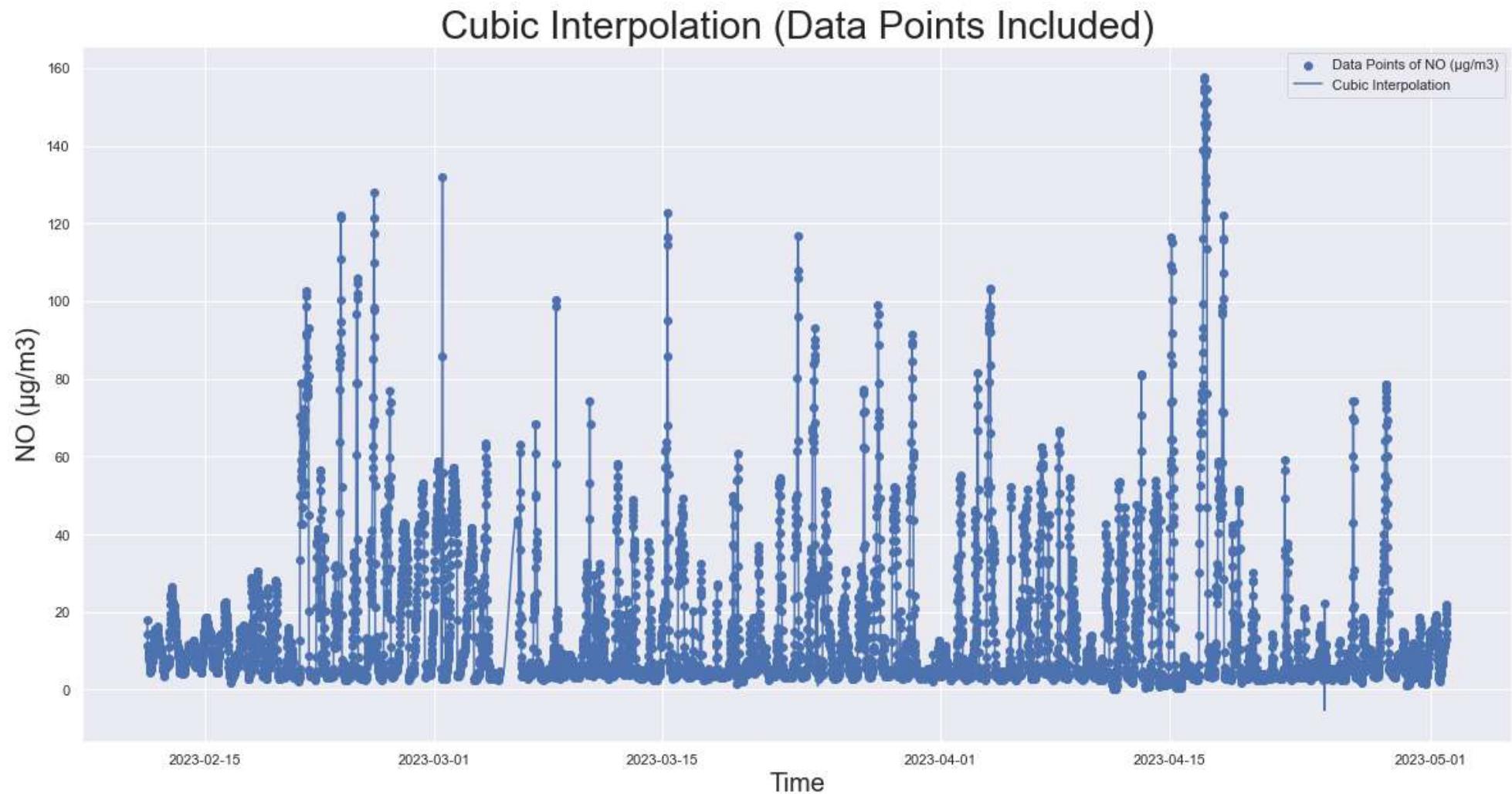
Quadratic Interpolation (Data Points Included)



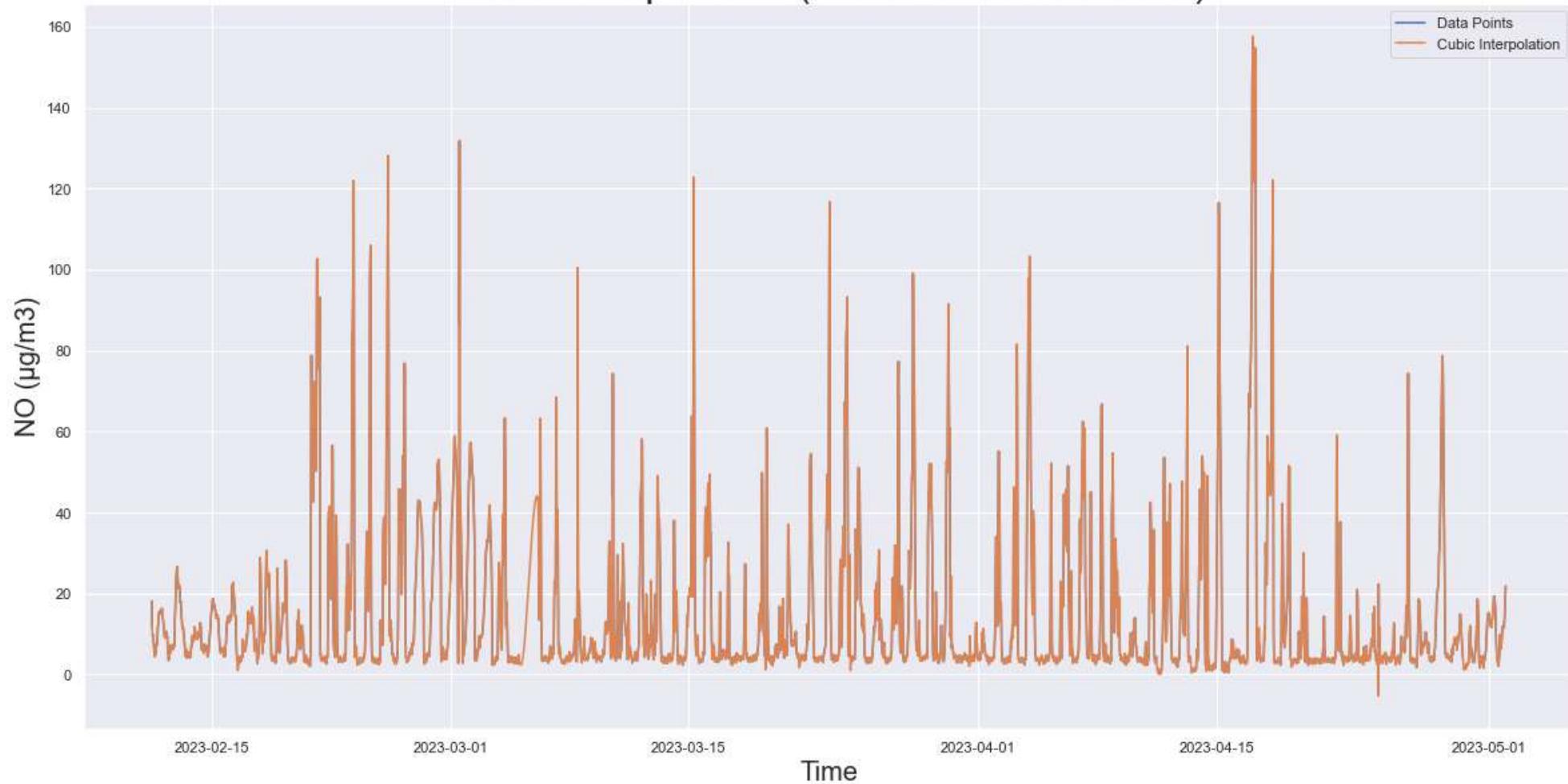
Quadratic Interpolation (Data Points Excluded)



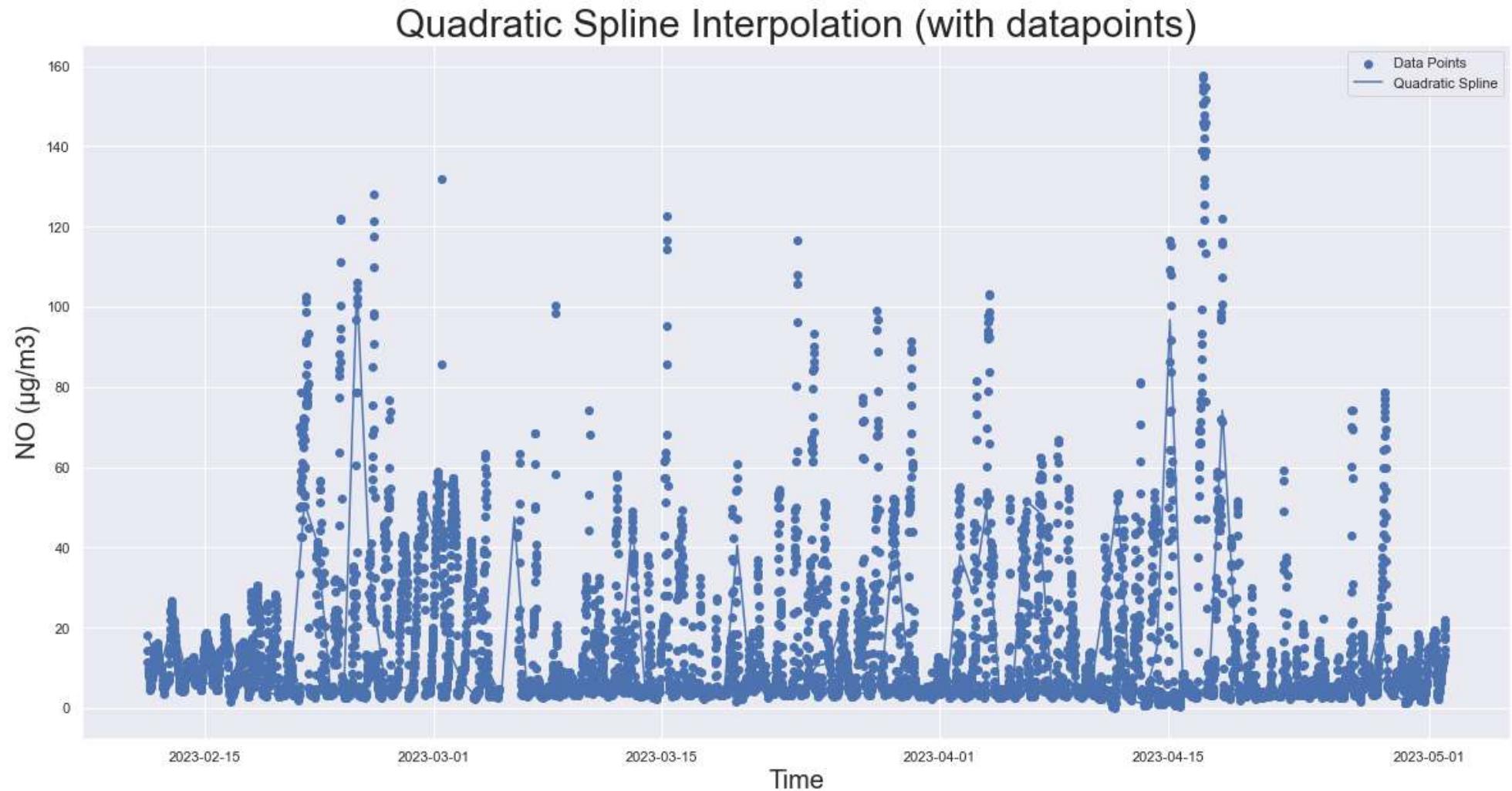
```
In [101]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```



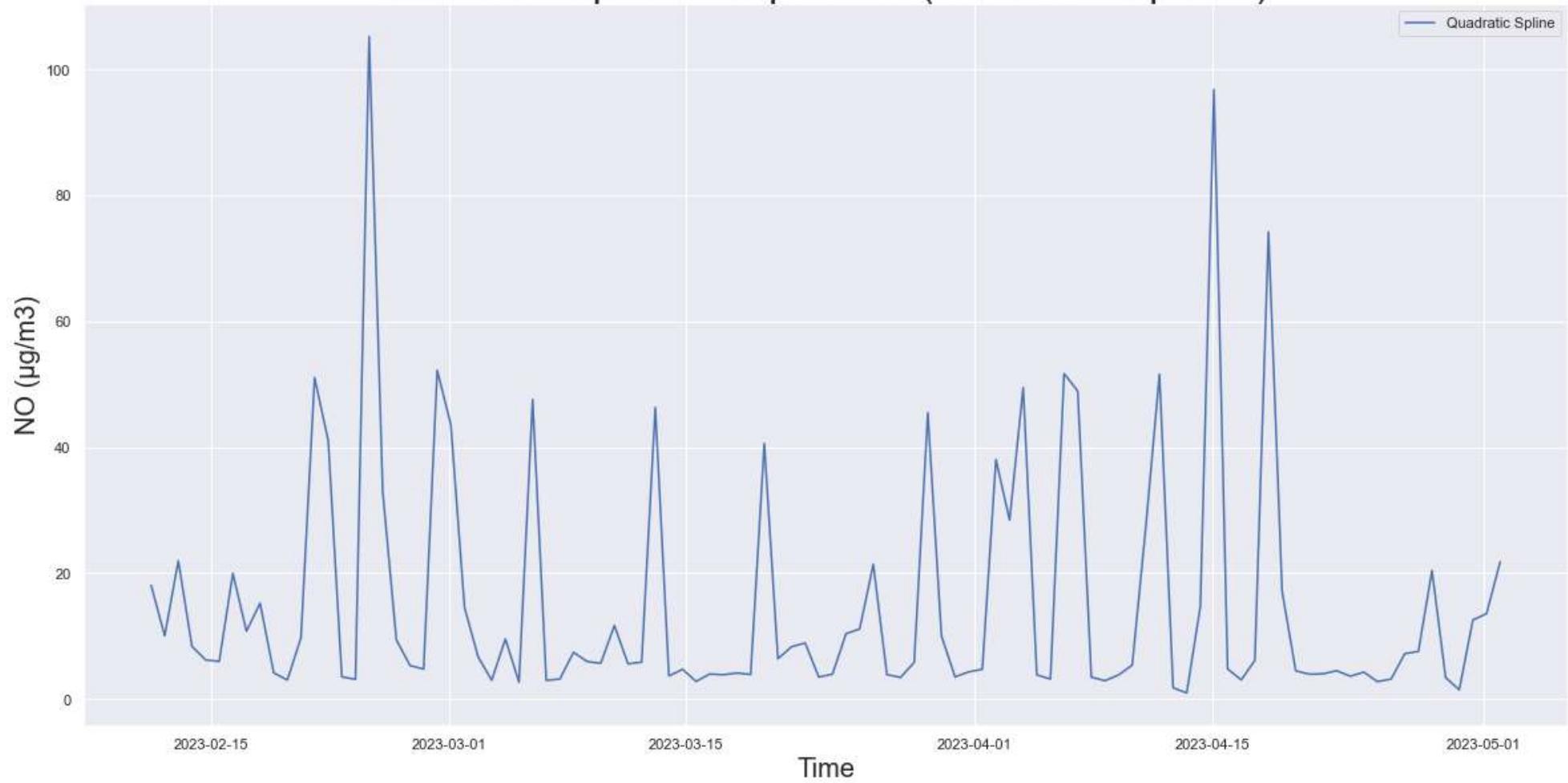
Cubic Interpolation (Data Points Excluded)



In [102]: `quadraticspline(df_n,pollutant)`

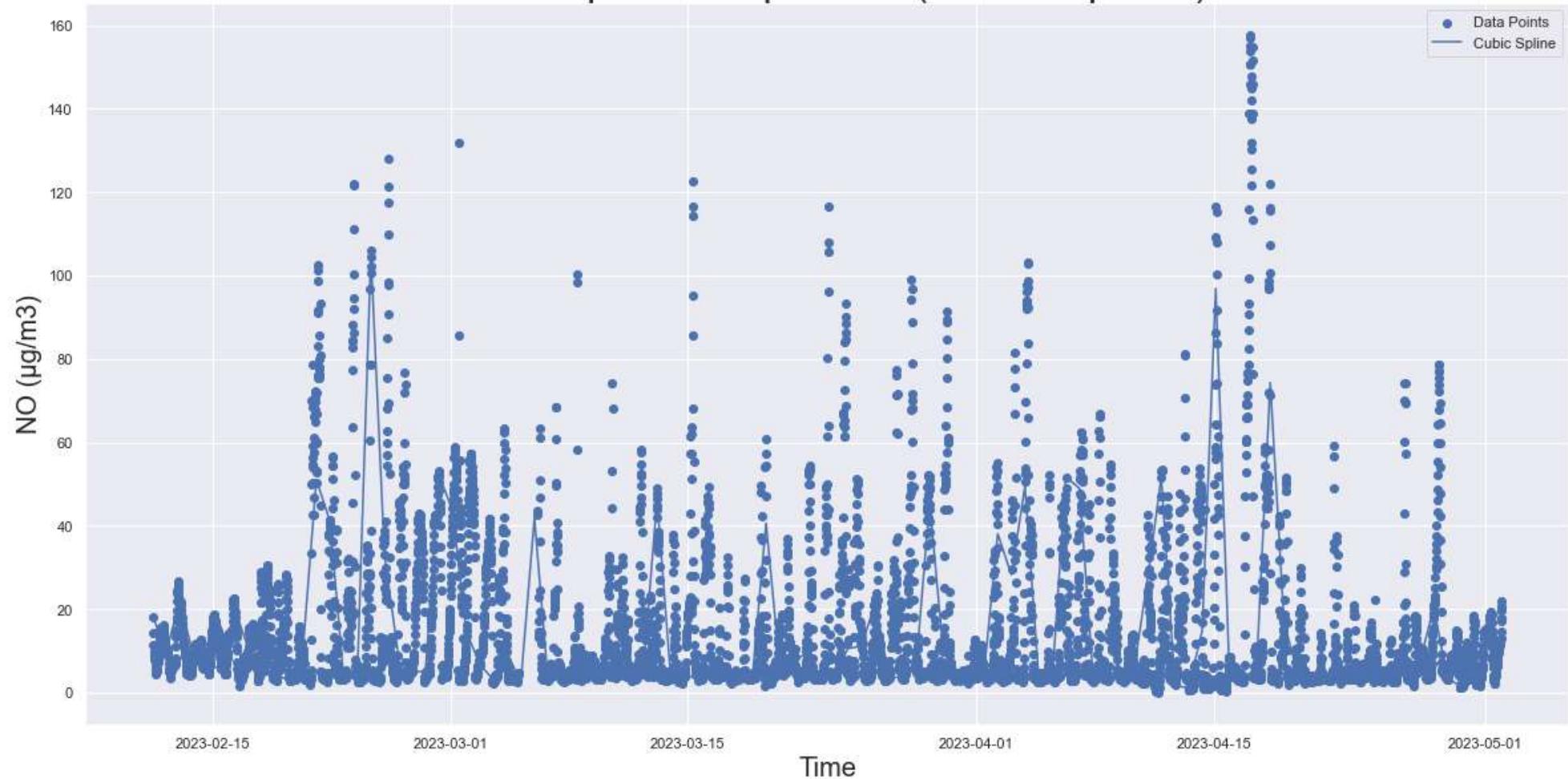


Quadratic Spline Interpolation (without datapoints)

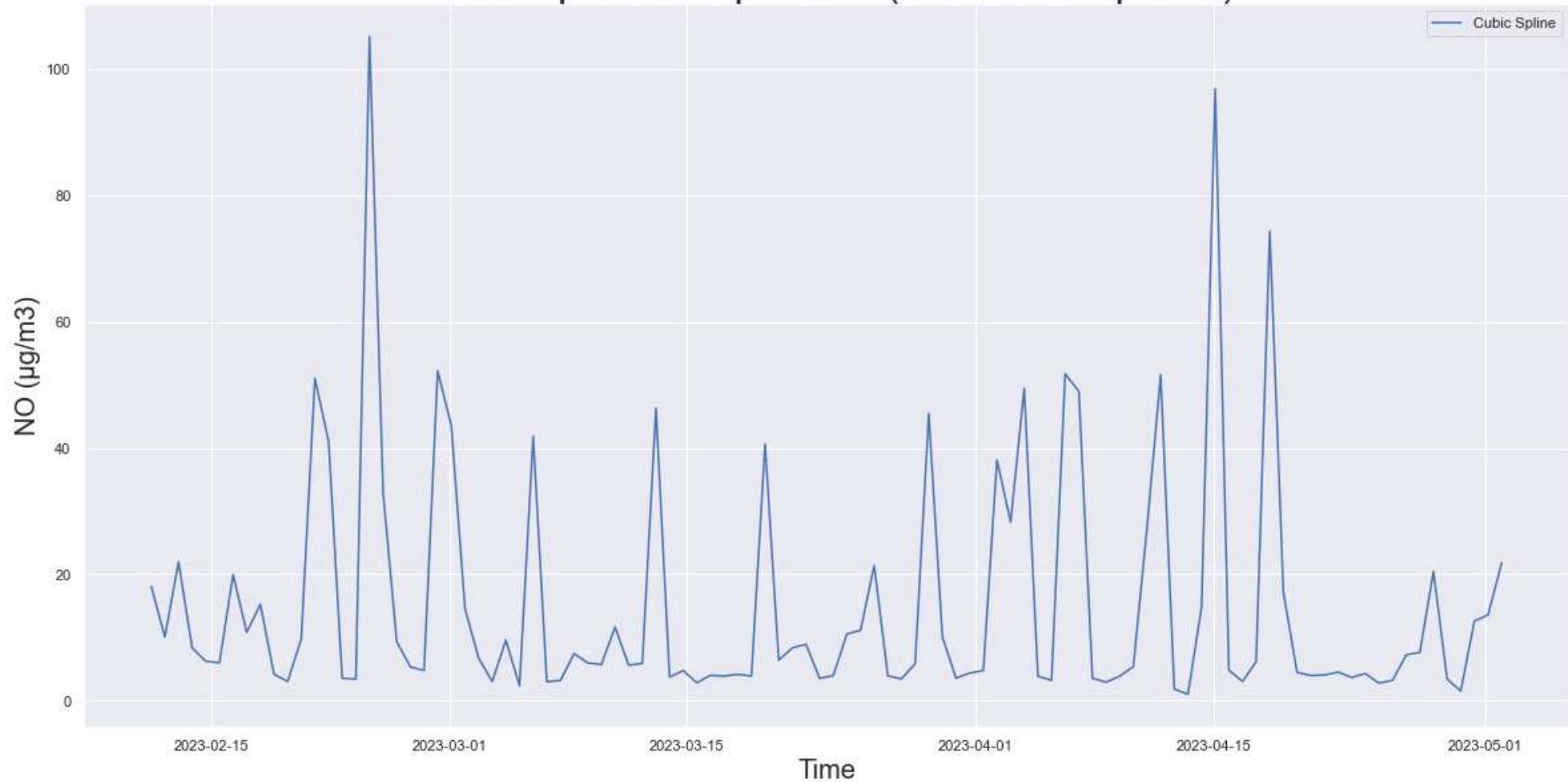


In [103]: `cubicspline(df_n,pollutant)`

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



Linear Interpolation seems to work best for rising and falling trends in data. Inspite of data having most of the data concentrated across the mean as evident from boxplot and histogram, the max-min range and shapes of plot suggest that data is oscillating around mean across its range.

In [104]: `df_quad[pollutant].isnull().sum()`

Out[104]: 1004

In [105]: `df_linear_f[pollutant].isnull().sum()`

Out[105]: 1004

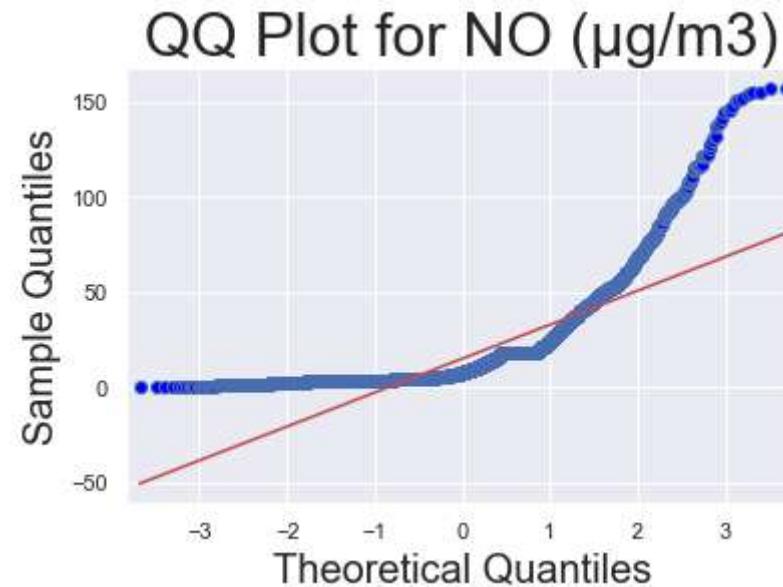
```
In [106]: df_linear_b[pollutant].isnull().sum()
```

```
Out[106]: 0
```

```
In [107]: df_new[pollutant] = df_linear_b[pollutant]
```

```
In [108]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>



```
In [109]: df_new.head()
```

```
Out[109]:
```

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)
--	-----------------------------------	------------------------------------	---------------------------------

Time	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)
2023-02-01 00:00:00	95.0	35.0	18.1
2023-02-01 00:15:00	95.0	35.0	18.1
2023-02-01 00:30:00	95.0	35.0	18.1
2023-02-01 00:45:00	122.0	34.0	18.1
2023-02-01 01:00:00	122.0	34.0	18.1

Time	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)
2023-02-01 00:00:00	95.0	35.0	18.1
2023-02-01 00:15:00	95.0	35.0	18.1
2023-02-01 00:30:00	95.0	35.0	18.1
2023-02-01 00:45:00	122.0	34.0	18.1
2023-02-01 01:00:00	122.0	34.0	18.1

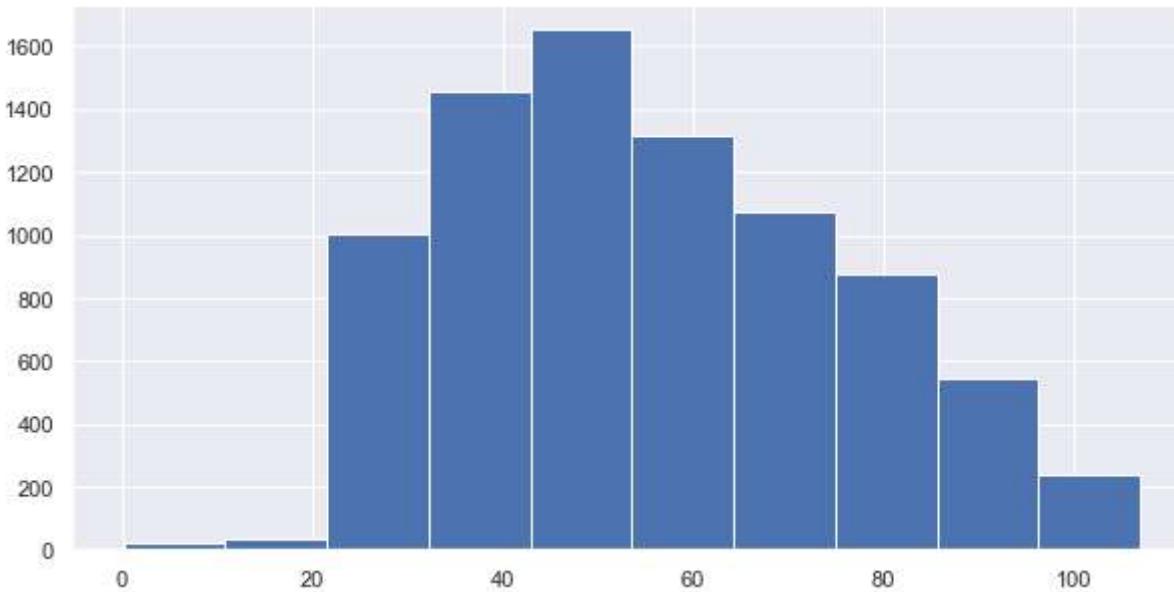
Analysis for "NO2 ($\mu\text{g}/\text{m}^3$)"

```
In [110]: pollutant = 'NO2 ( $\mu\text{g}/\text{m}^3$ )'
```

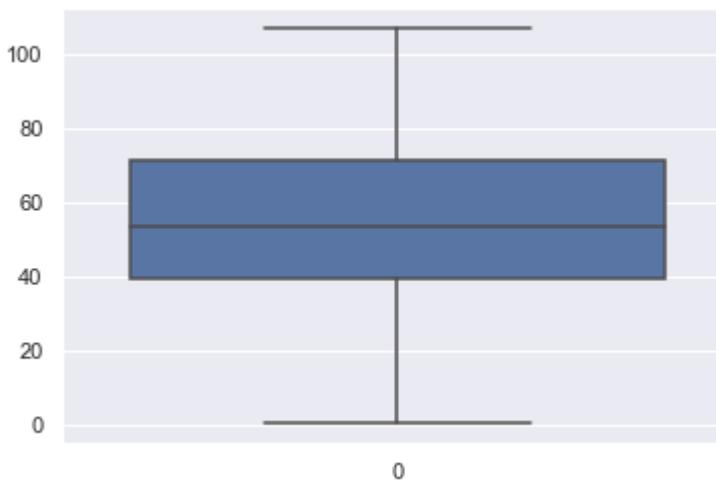
```
In [111]: df[pollutant].describe()
```

```
Out[111]: count    8224.000000
mean      55.757028
std       20.231407
min       0.200000
25%      39.400000
50%      53.200000
75%      71.025000
max     106.900000
Name: NO2 ( $\mu\text{g}/\text{m}^3$ ), dtype: float64
```

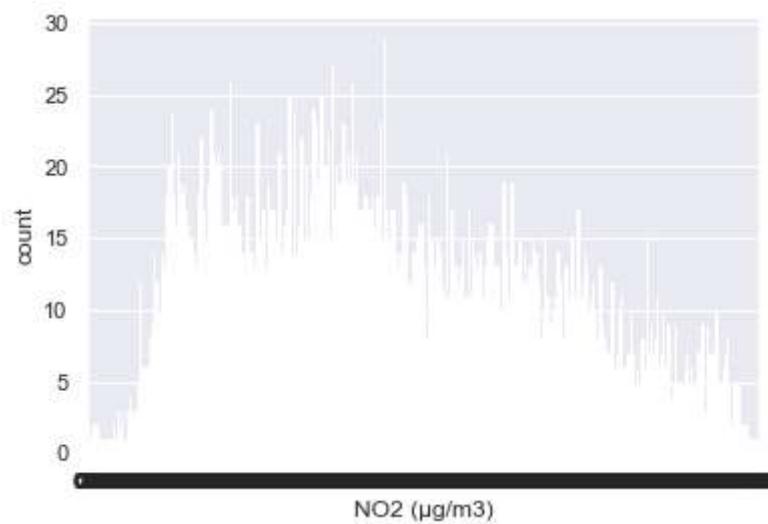
```
In [112]: histogram_plot(df_n,pollutant)
```



```
In [113]: boxplot_plot(df_n,pollutant)
```

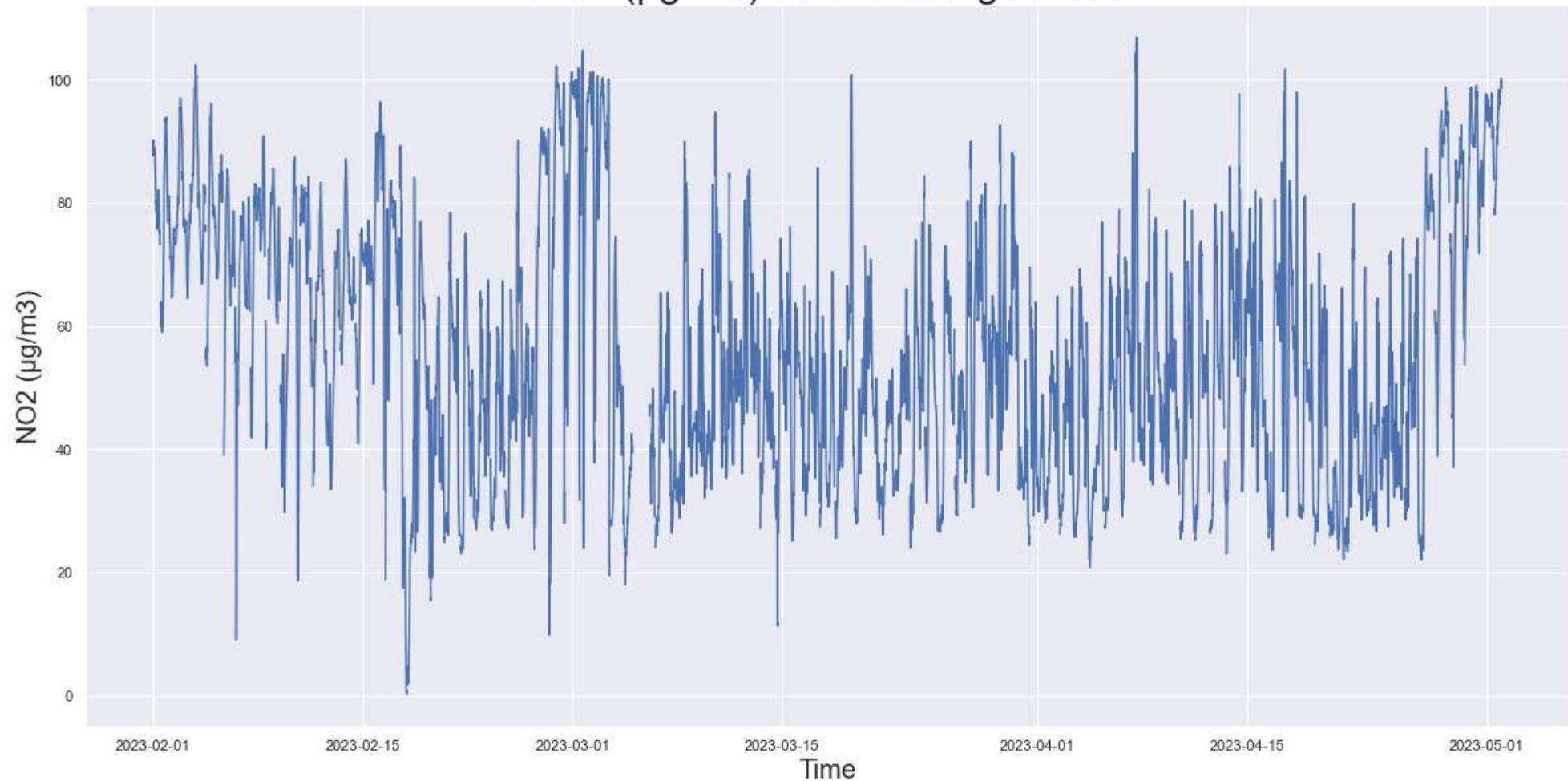


```
In [114]: countplot_plot(df_n,pollutant)
```



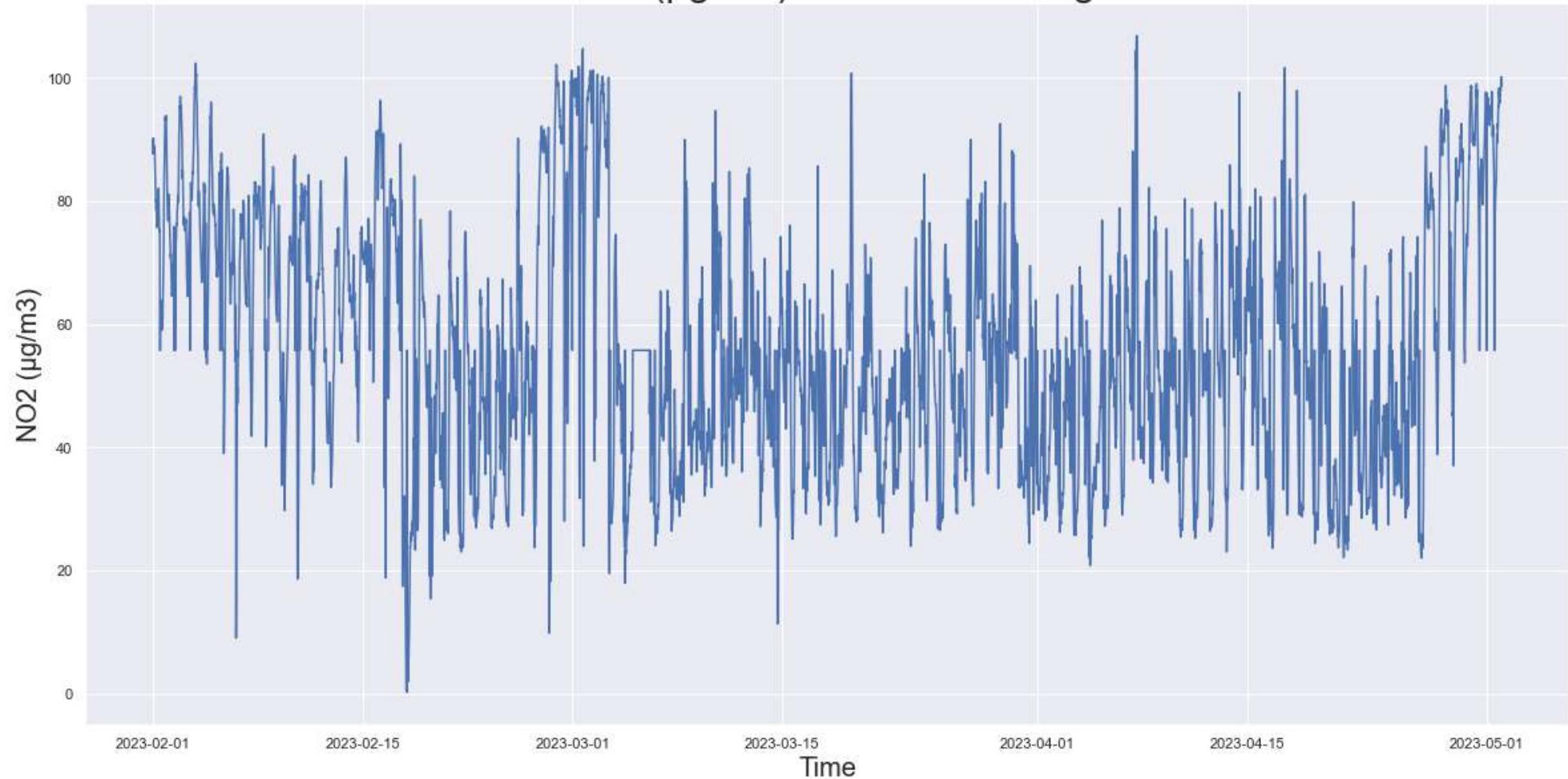
```
In [115]: simple_time_plot(df_n,pollutant,"With missing values")
```

NO2 ($\mu\text{g}/\text{m}^3$) With missing values



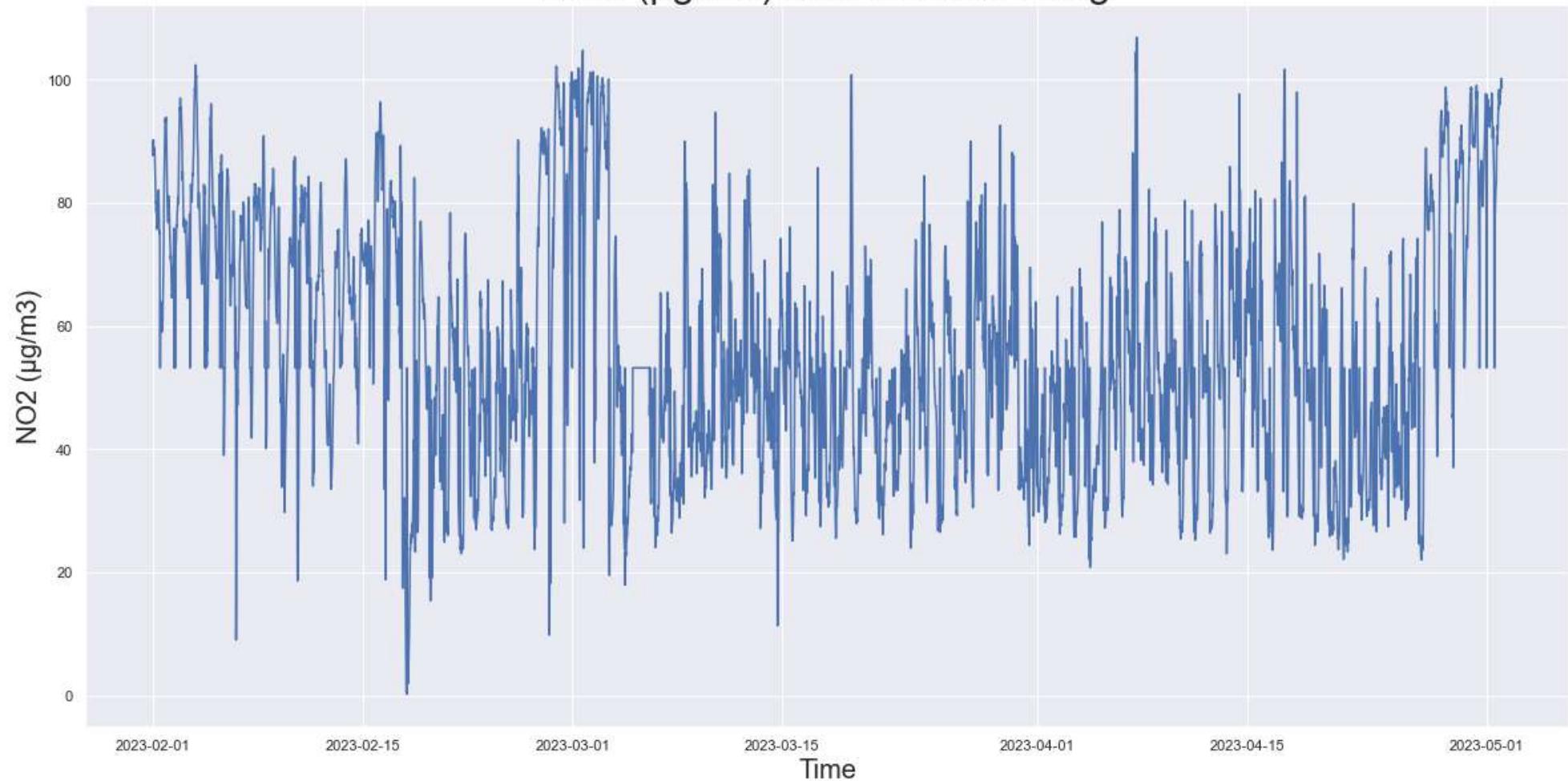
```
In [116]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

NO2 ($\mu\text{g}/\text{m}^3$) after mean filling



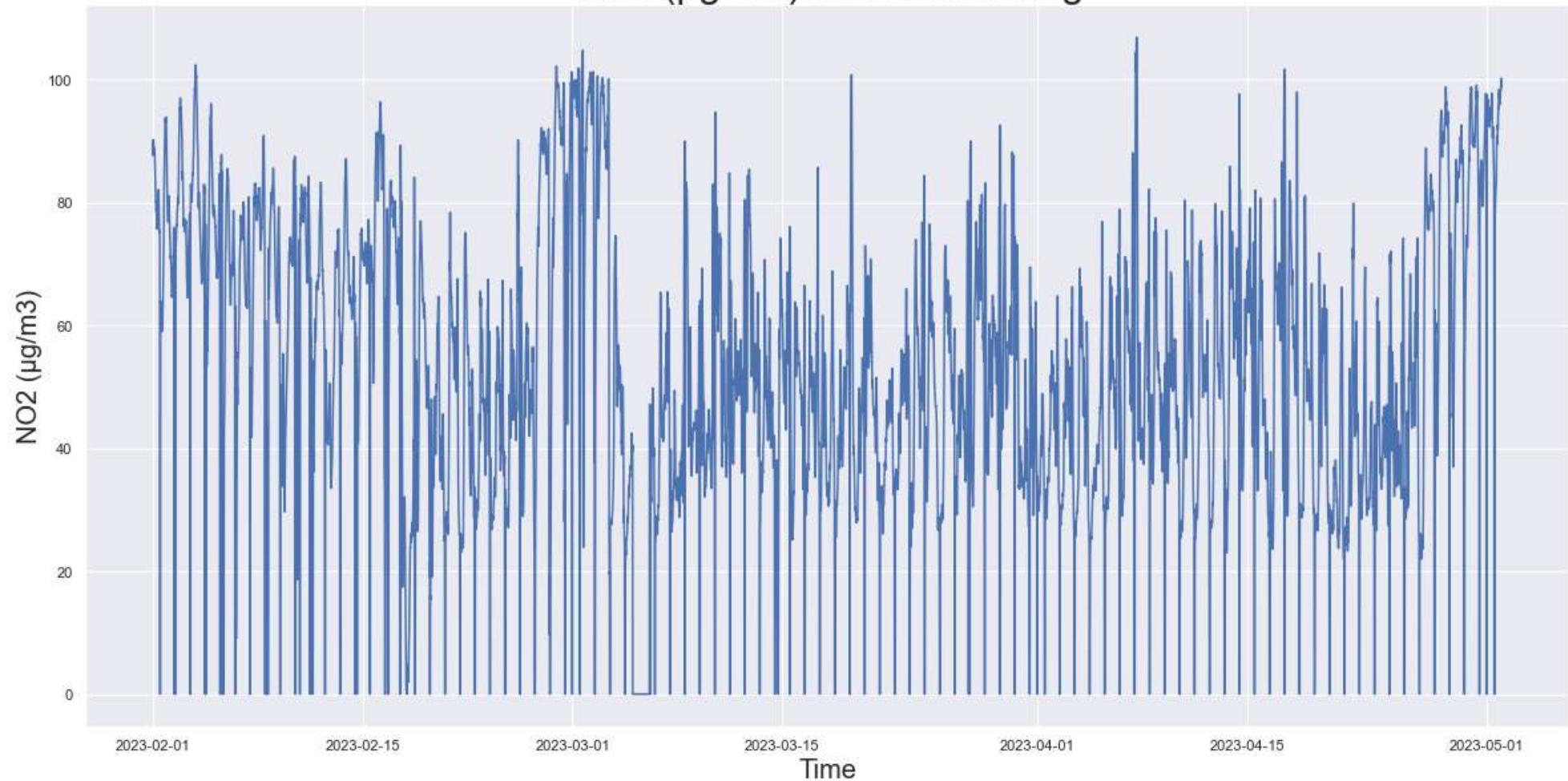
```
In [117]: simple_time_plot(df_median,pollutant,"after median filling")
```

NO2 ($\mu\text{g}/\text{m}^3$) after median filling

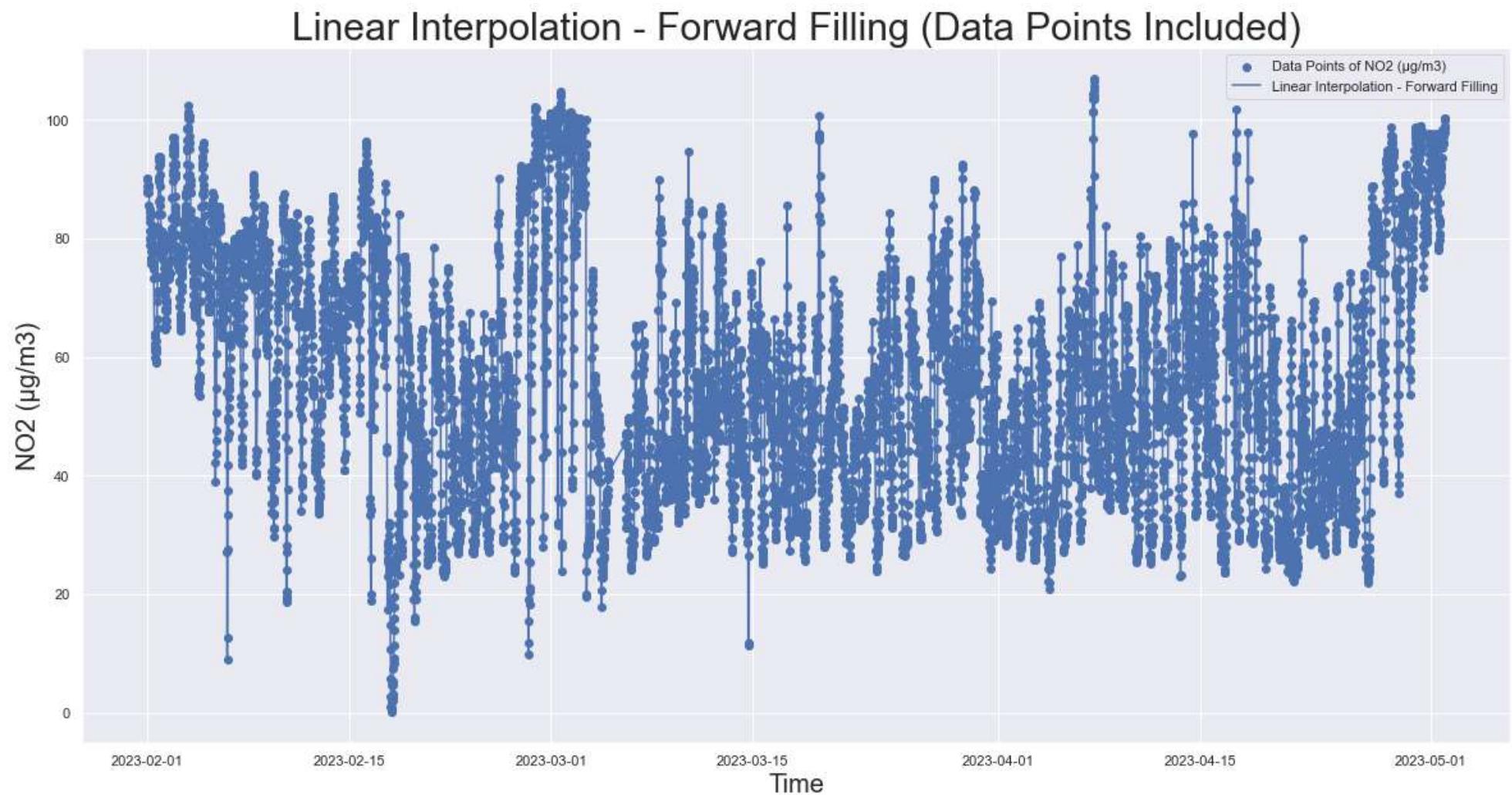


```
In [118]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

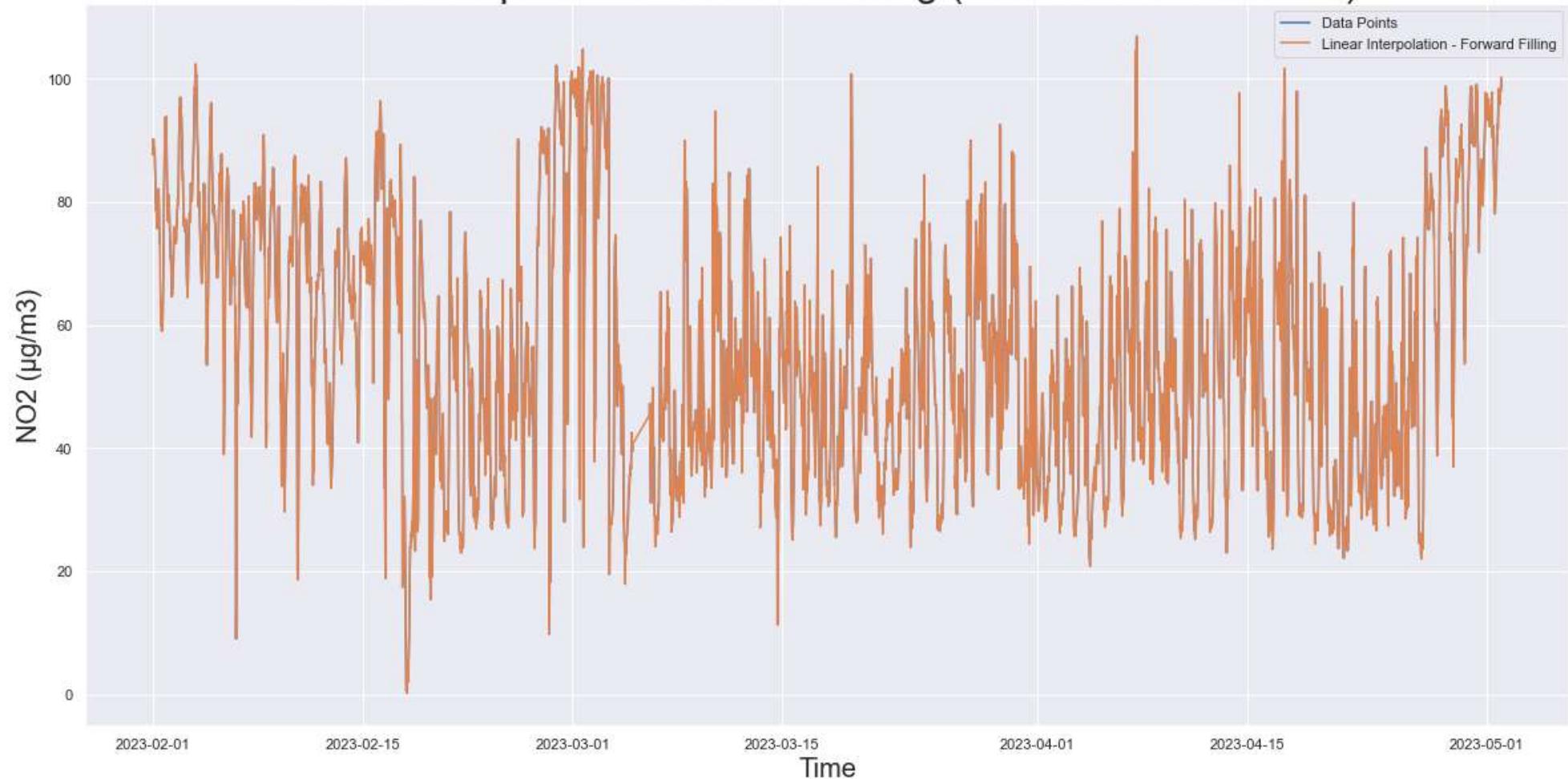
NO2 ($\mu\text{g}/\text{m}^3$) after zero filling



```
In [119]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```

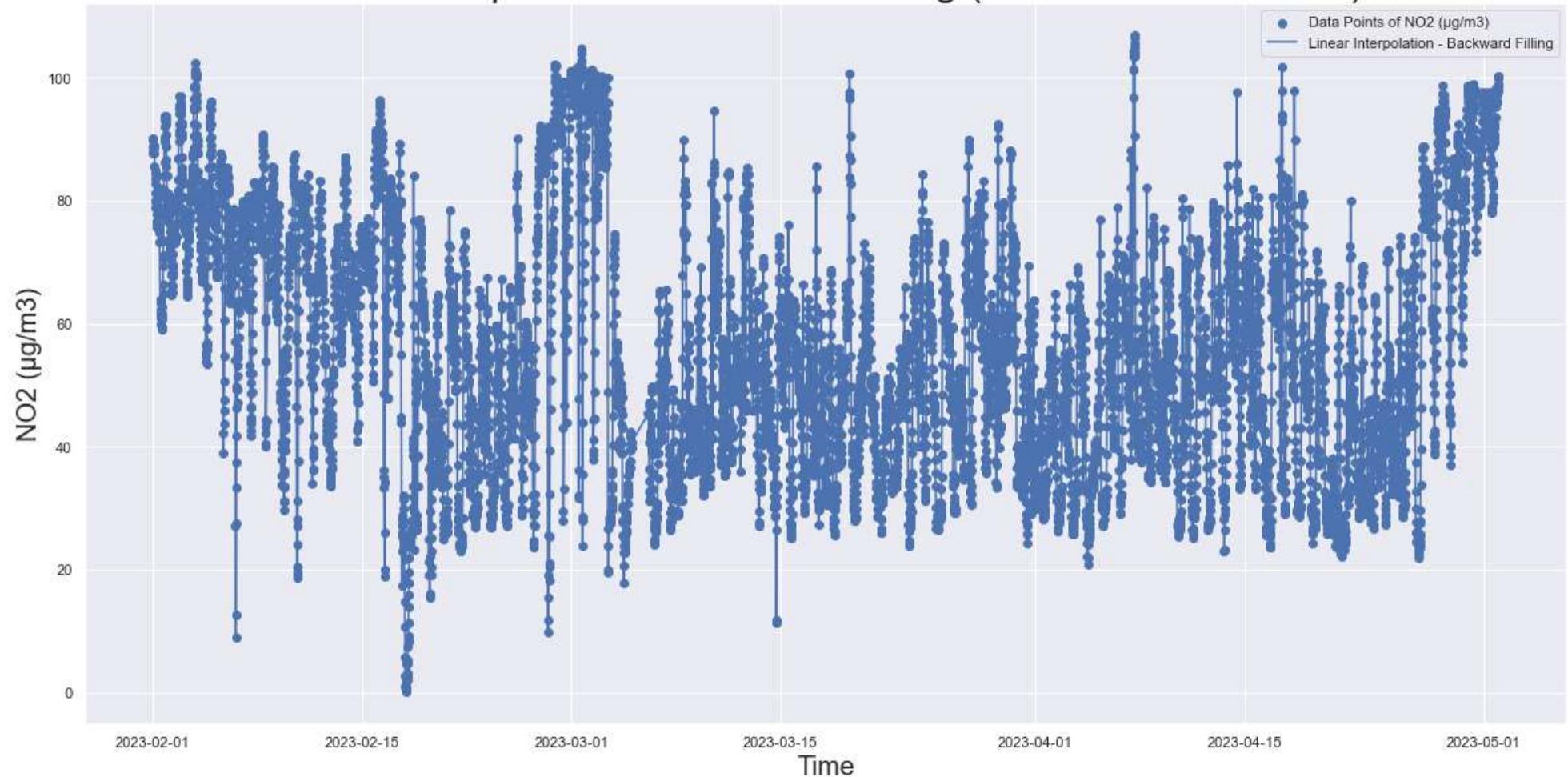


Linear Interpolation - Forward Filling (Data Points Excluded)

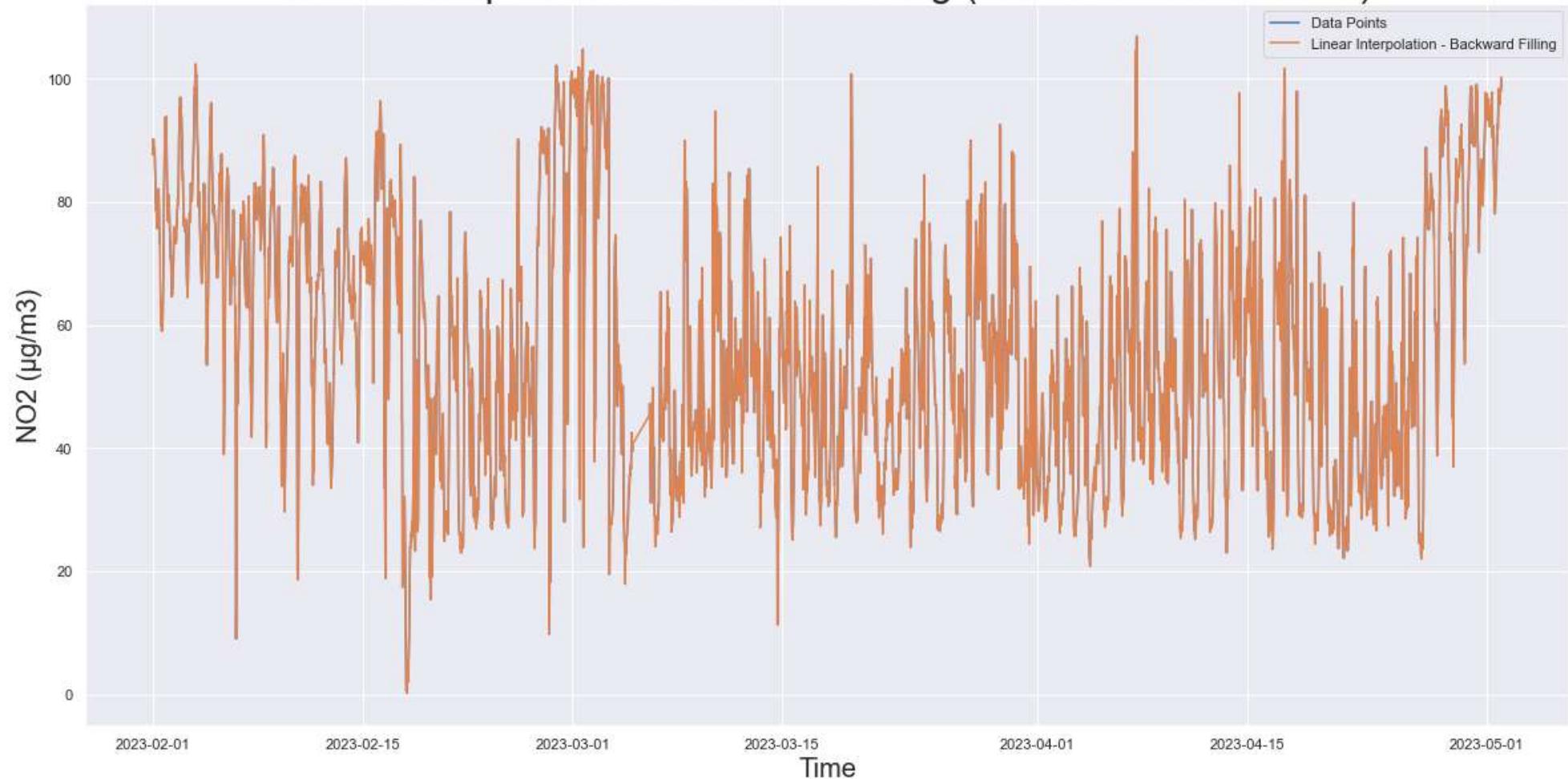


```
In [120]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

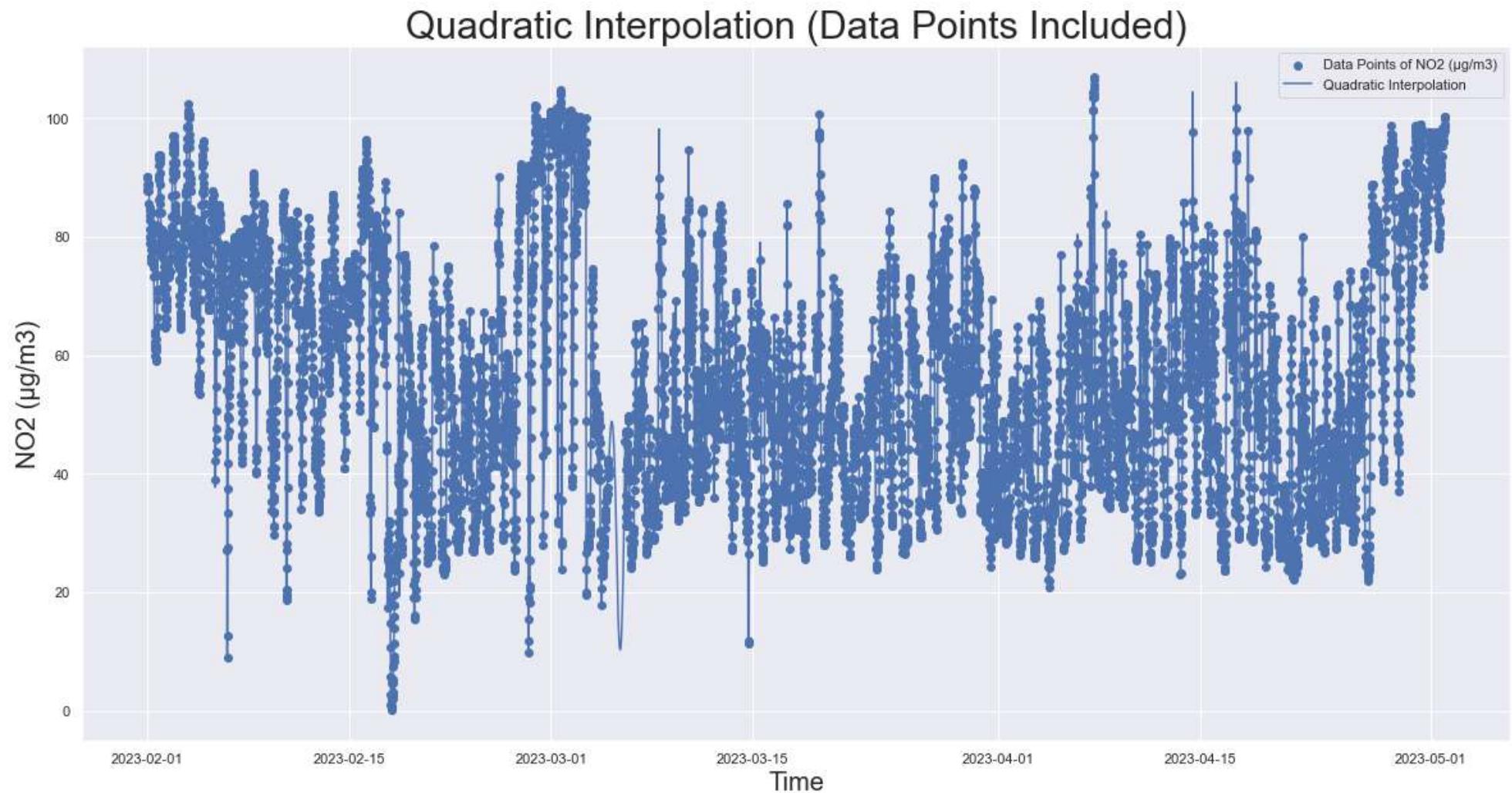
Linear Interpolation - Backward Filling (Data Points Included)



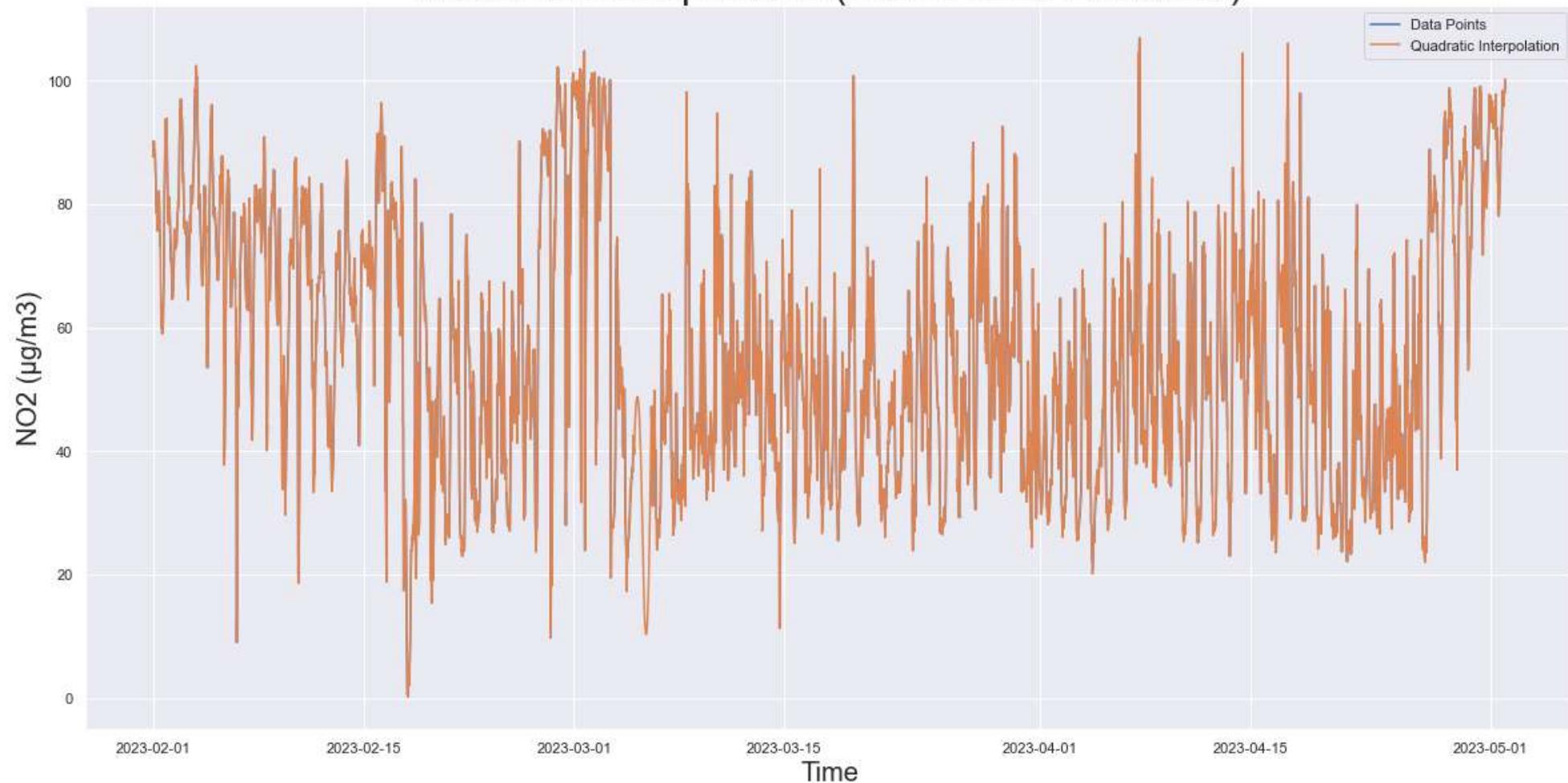
Linear Interpolation - Backward Filling (Data Points Excluded)



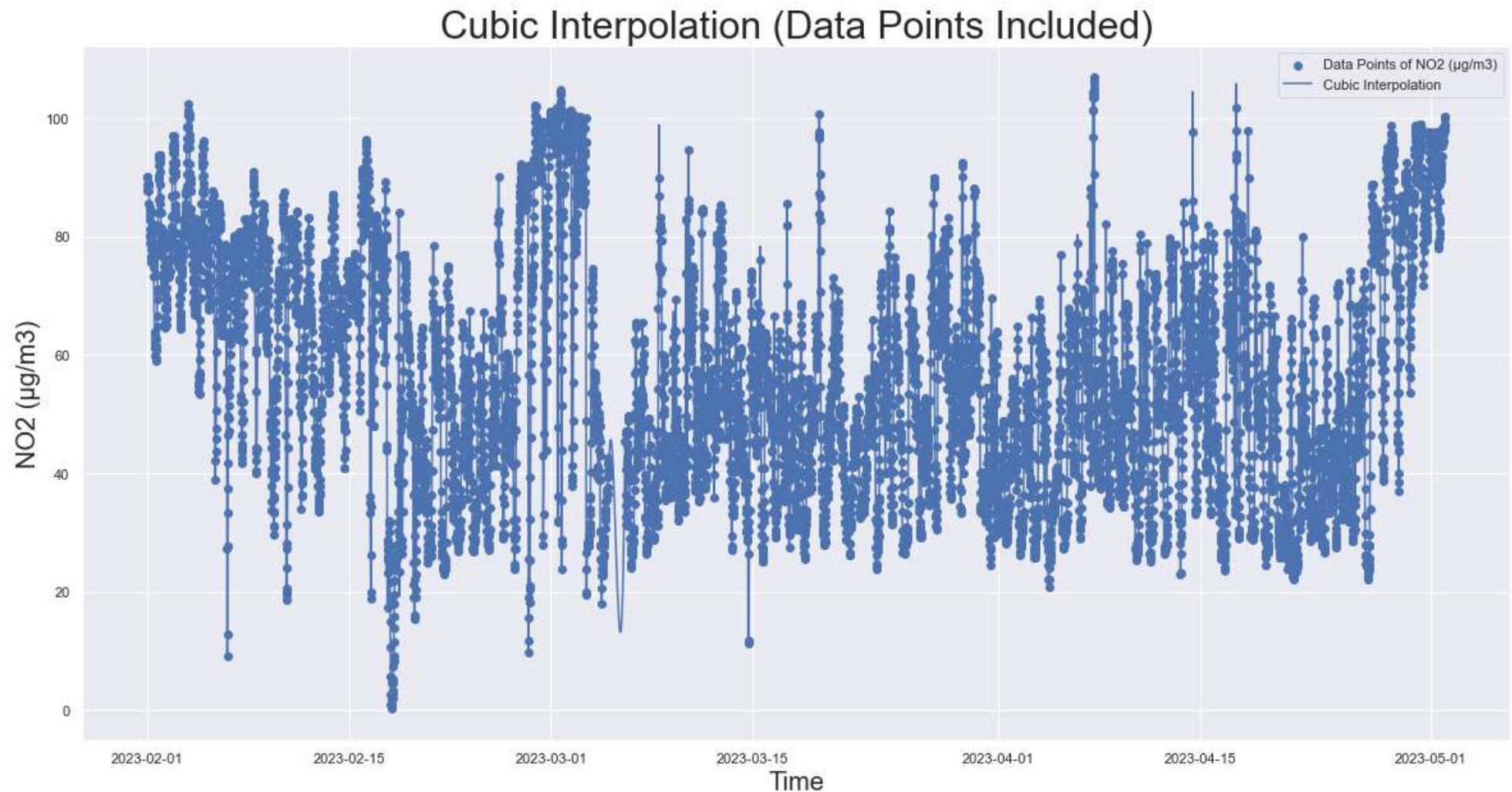
```
In [121]: interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")
```



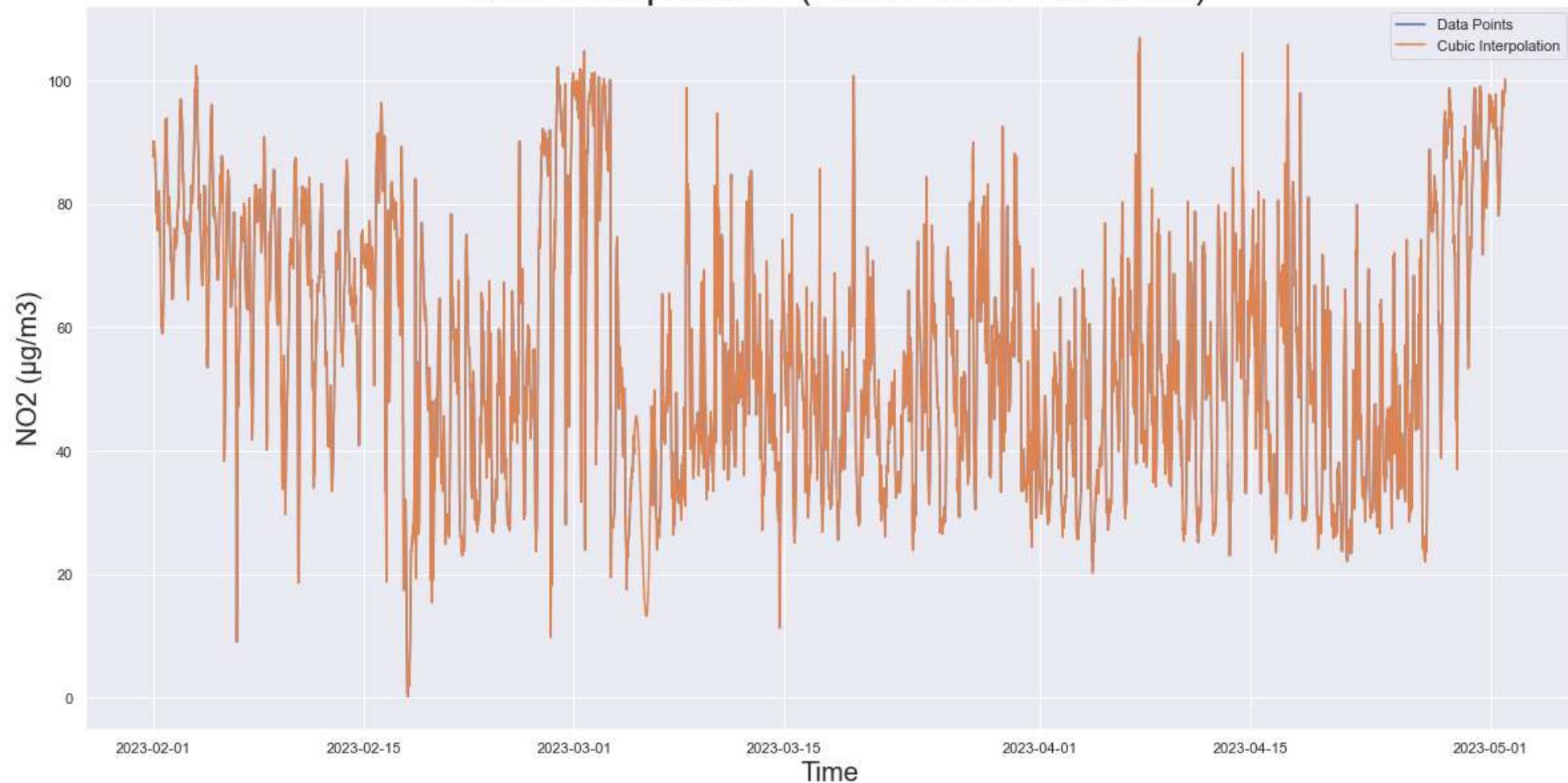
Quadratic Interpolation (Data Points Excluded)



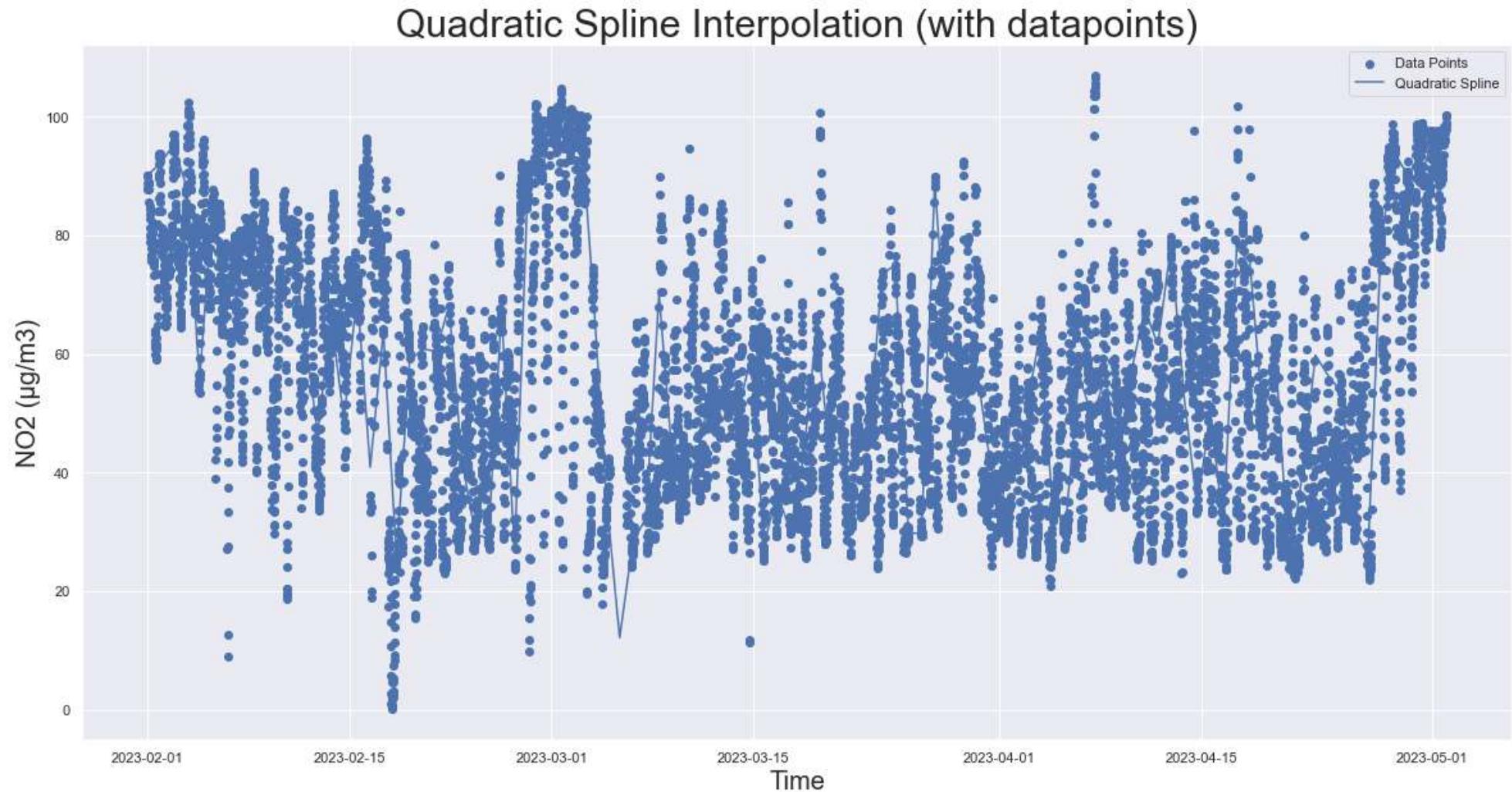
```
In [122]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```



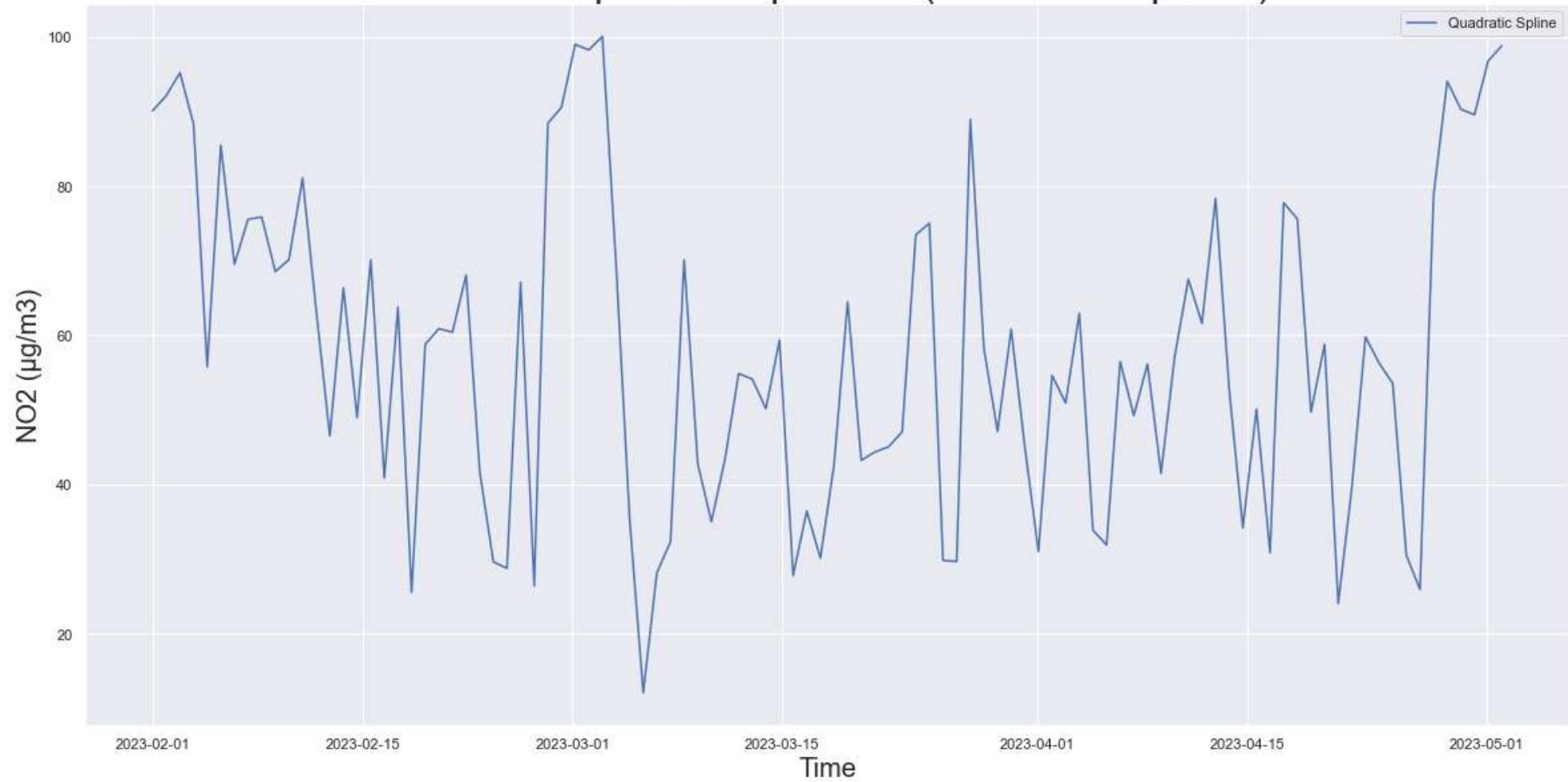
Cubic Interpolation (Data Points Excluded)



In [123]: `quadraticspline(df_n,pollutant)`

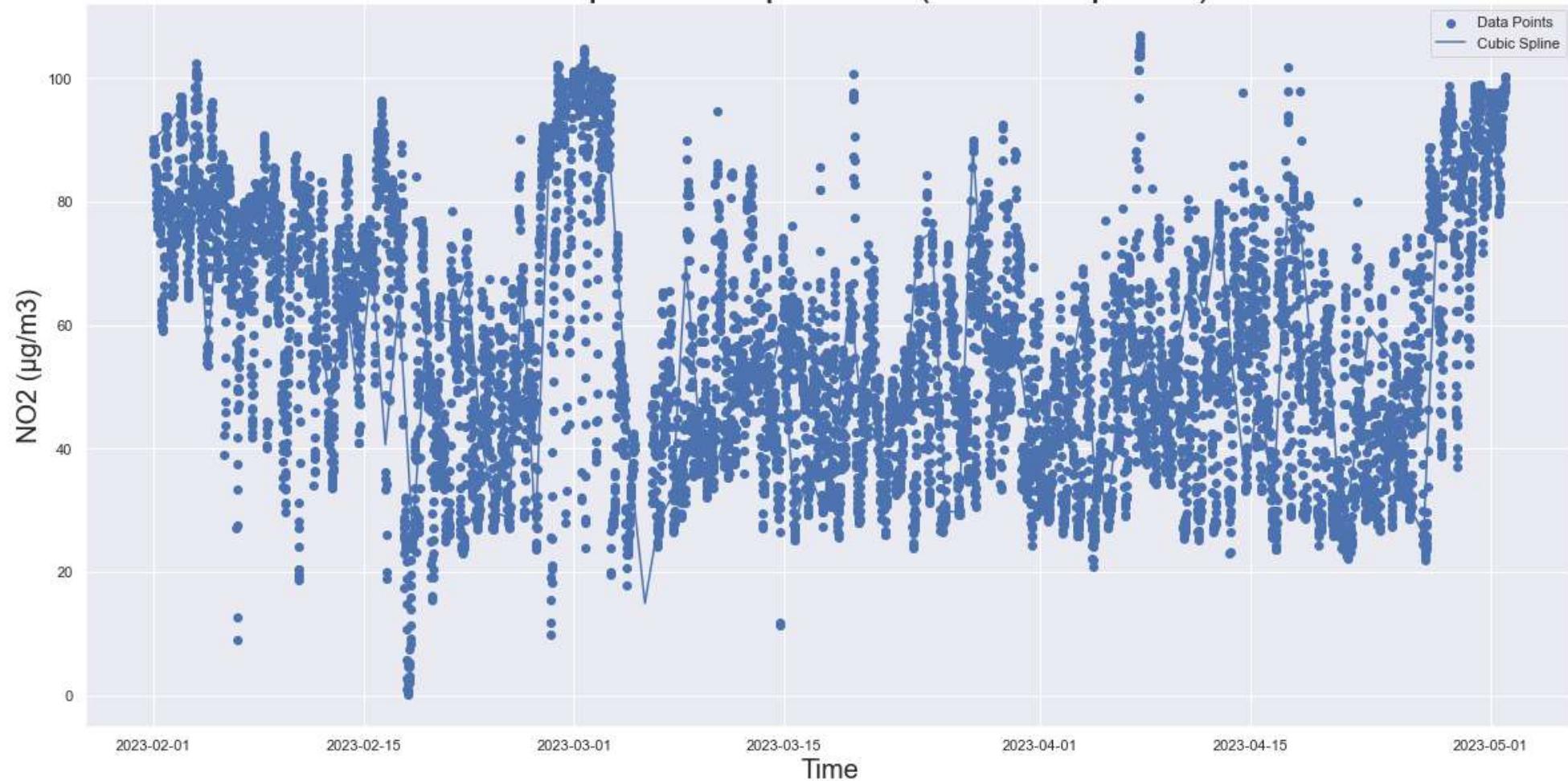


Quadratic Spline Interpolation (without datapoints)

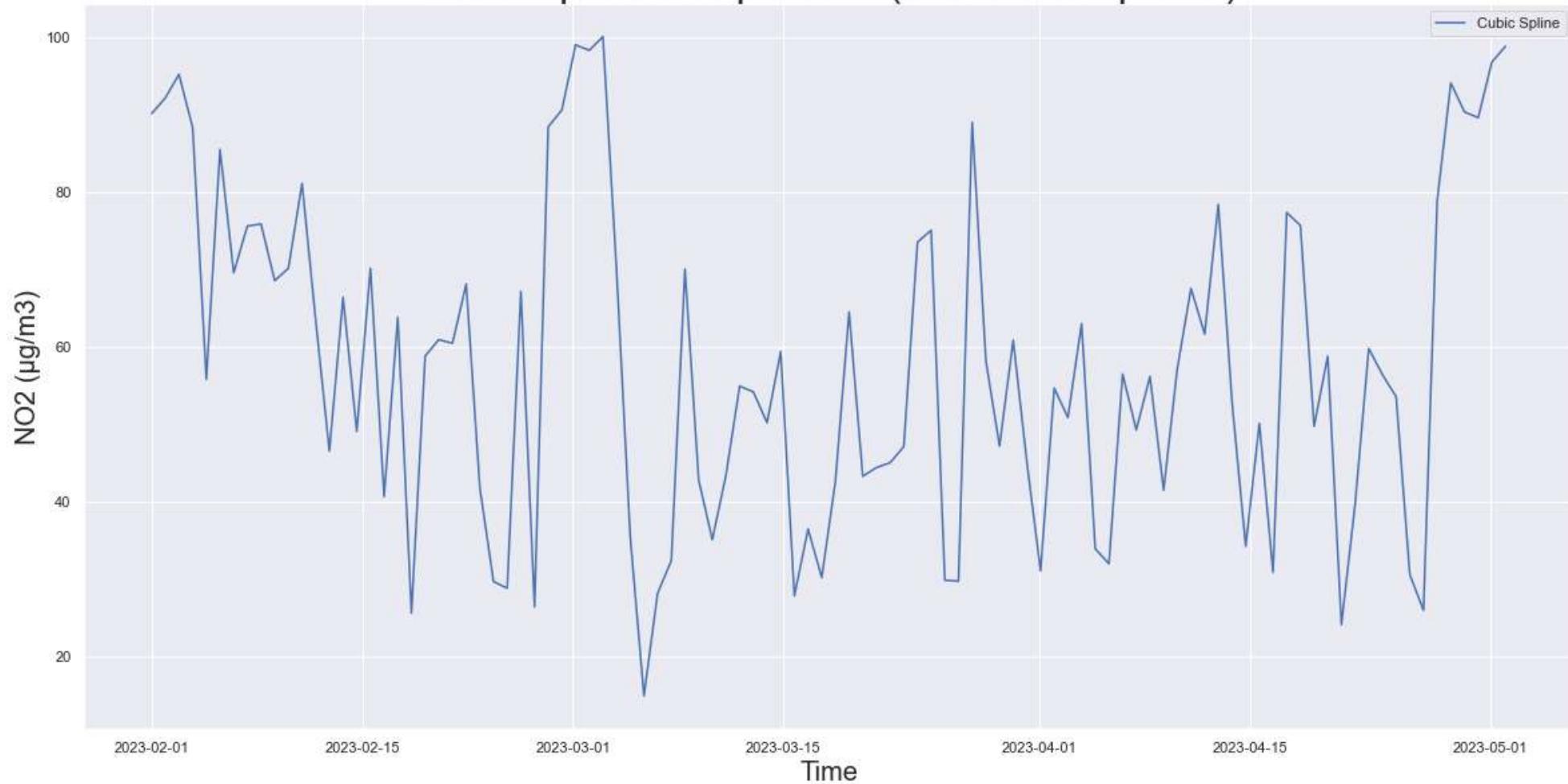


In [124]: `cubicspline(df_n,pollutant)`

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



Linear interpolation seems to work best. This is because it does not overestimates the data. Also this pollutant has less missing values, linear interpolation will conserve the trend of values as well. Since most of the data seems to be nicely distributed across mean, range of data is well around mean and no effect of outliers observed, mean filling can also be considered.

In [125]: `df_mean[pollutant].isnull().sum()`

Out[125]: 0

In [126]: `df_linear_f[pollutant].isnull().sum()`

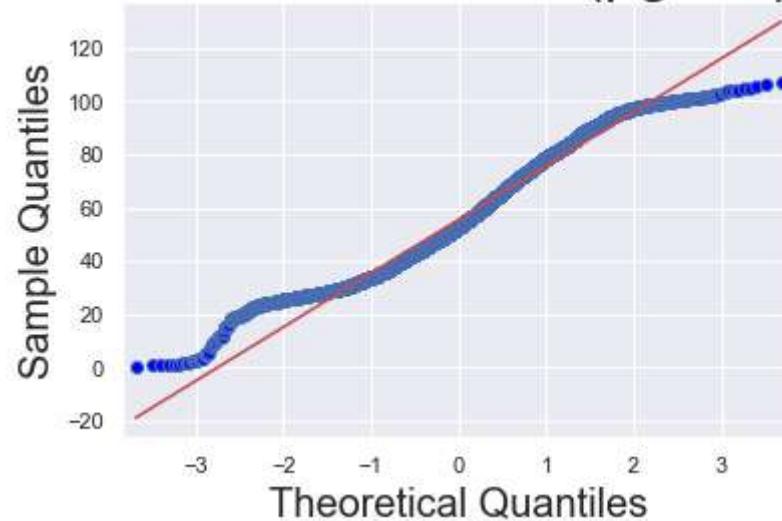
Out[126]: 0

```
In [127]: df_new[pollutant] = df_linear_f[pollutant]
```

```
In [128]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>

QQ Plot for NO2 ($\mu\text{g}/\text{m}^3$)



```
In [129]: df_new.head()
```

Out[129]:

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)
--	-----------------------------------	------------------------------------	---------------------------------	----------------------------------

Time	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)
2023-02-01 00:00:00	95.0	35.0	18.1	90.1
2023-02-01 00:15:00	95.0	35.0	18.1	88.0
2023-02-01 00:30:00	95.0	35.0	18.1	87.7
2023-02-01 00:45:00	122.0	34.0	18.1	88.9
2023-02-01 01:00:00	122.0	34.0	18.1	90.0

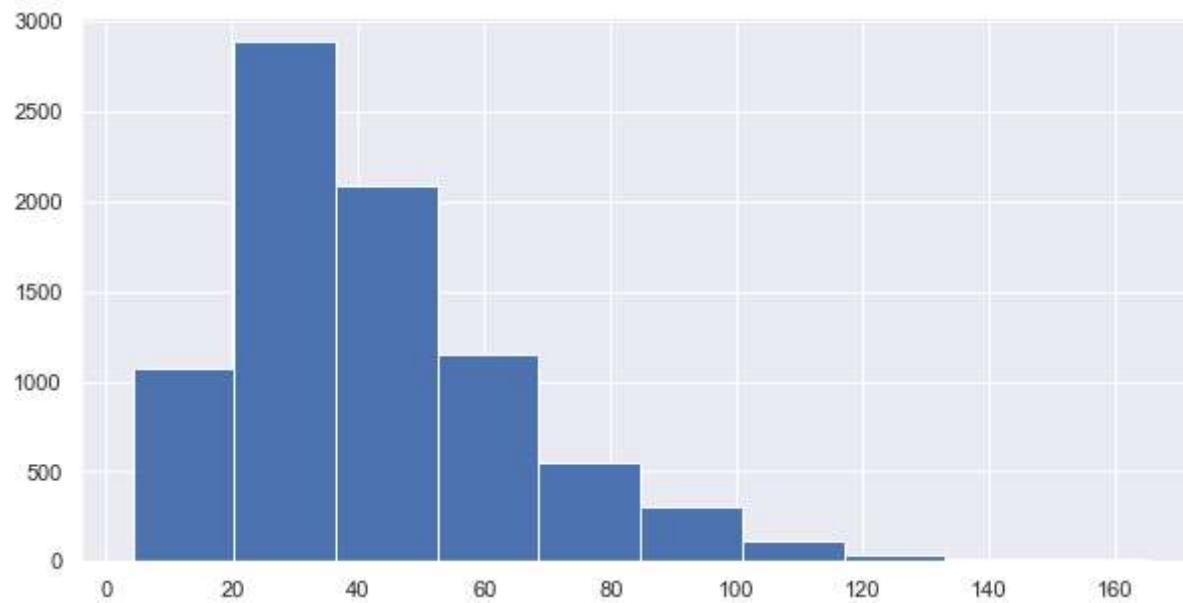
Analysis for "NOX (ppb)"

```
In [130]: pollutant = 'NOX (ppb)'
```

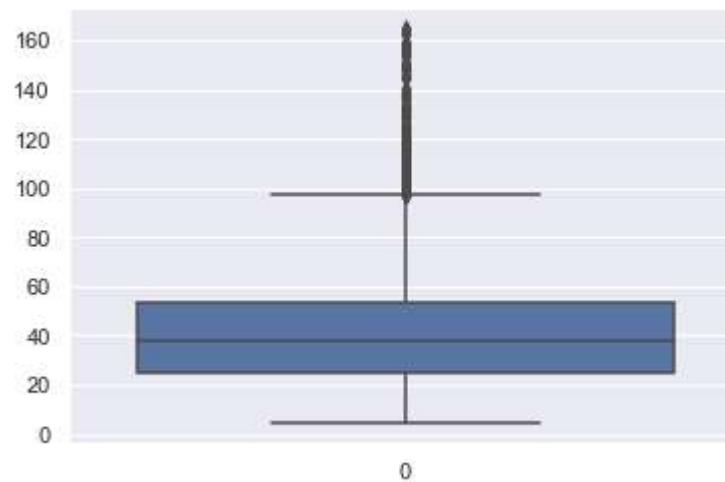
```
In [131]: df[pollutant].describe()
```

```
Out[131]: count    8225.000000
mean      42.672219
std       22.435262
min       4.200000
25%      25.000000
50%      37.700000
75%      53.800000
max     165.200000
Name: NOX (ppb), dtype: float64
```

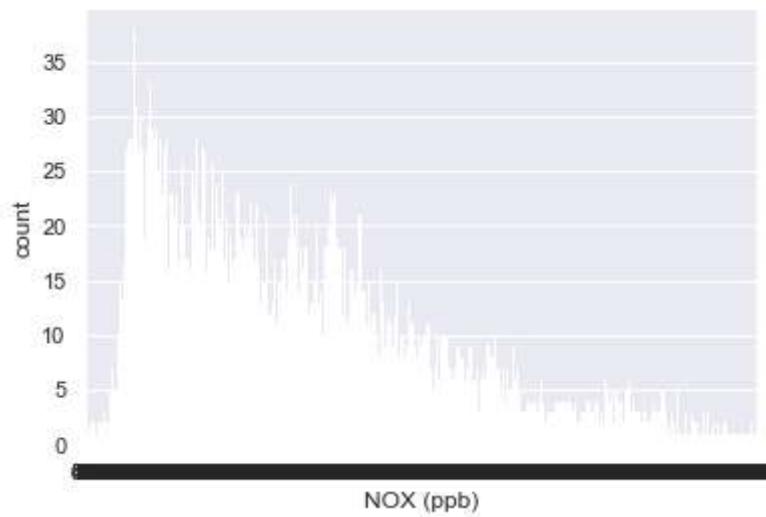
```
In [132]: histogram_plot(df_n,pollutant)
```



```
In [133]: boxplot_plot(df_n,pollutant)
```

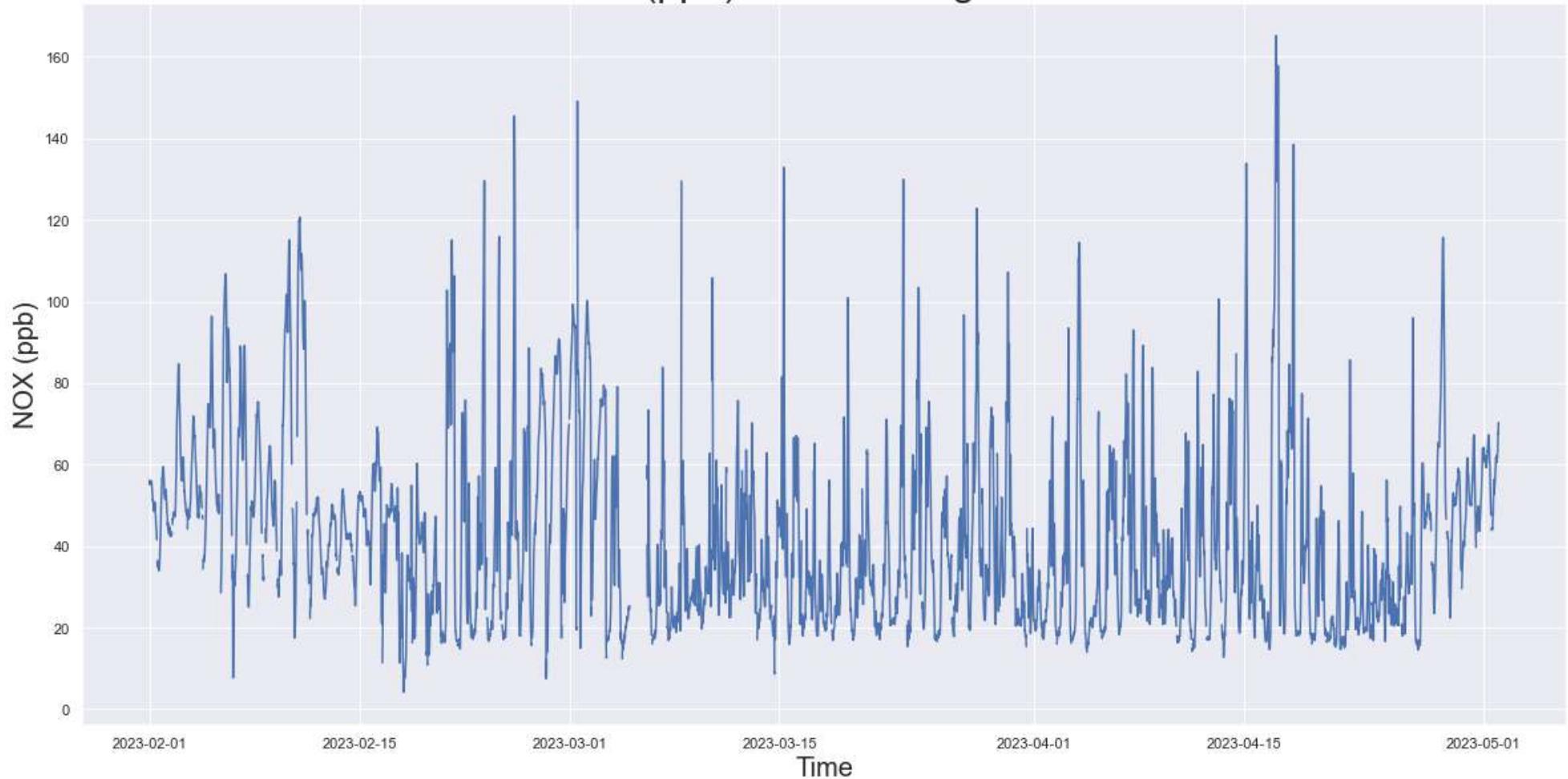


```
In [134]: countplot_plot(df_n,pollutant)
```



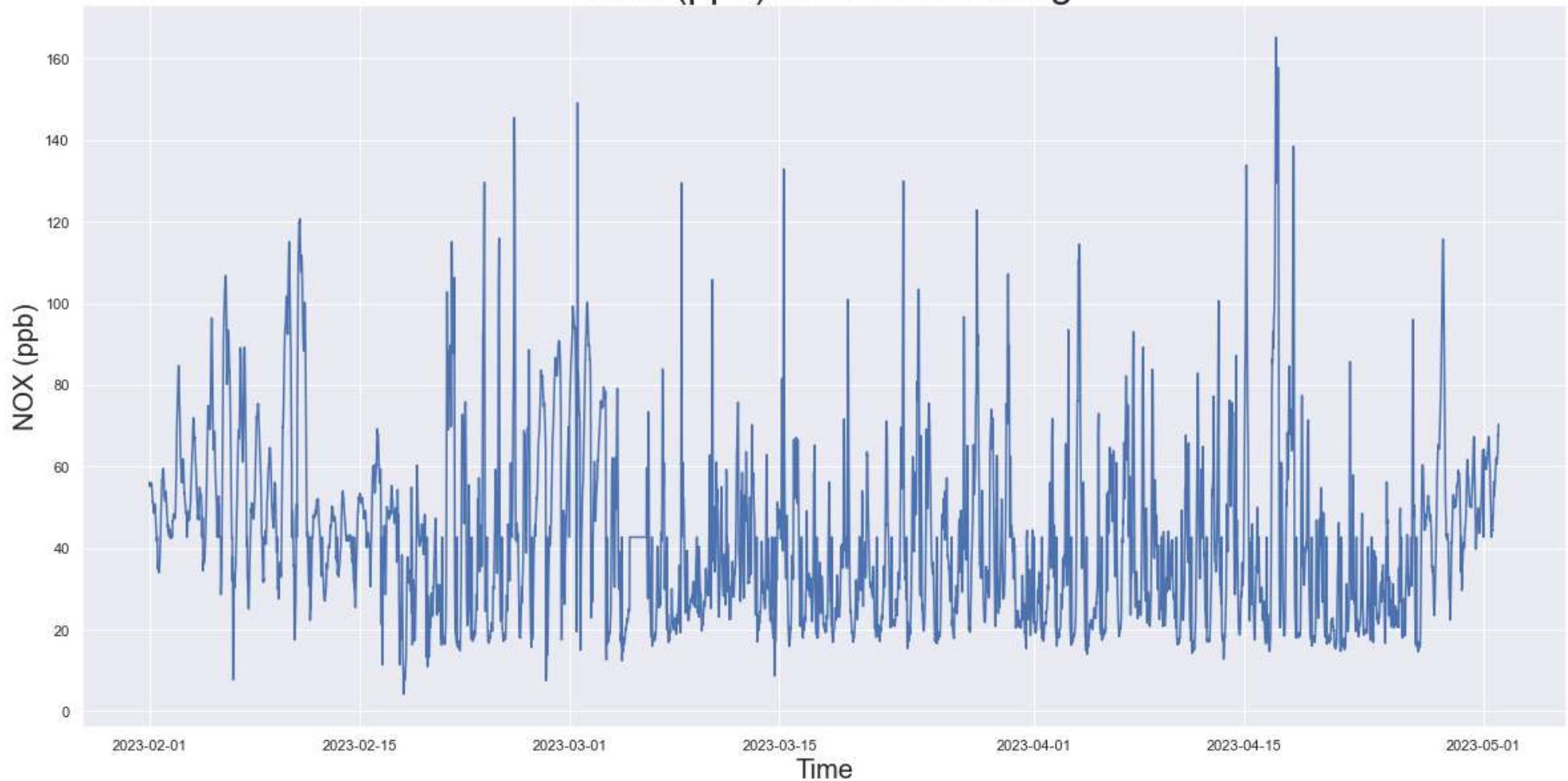
```
In [135]: simple_time_plot(df_n,pollutant,"With missing values")
```

NOX (ppb) With missing values



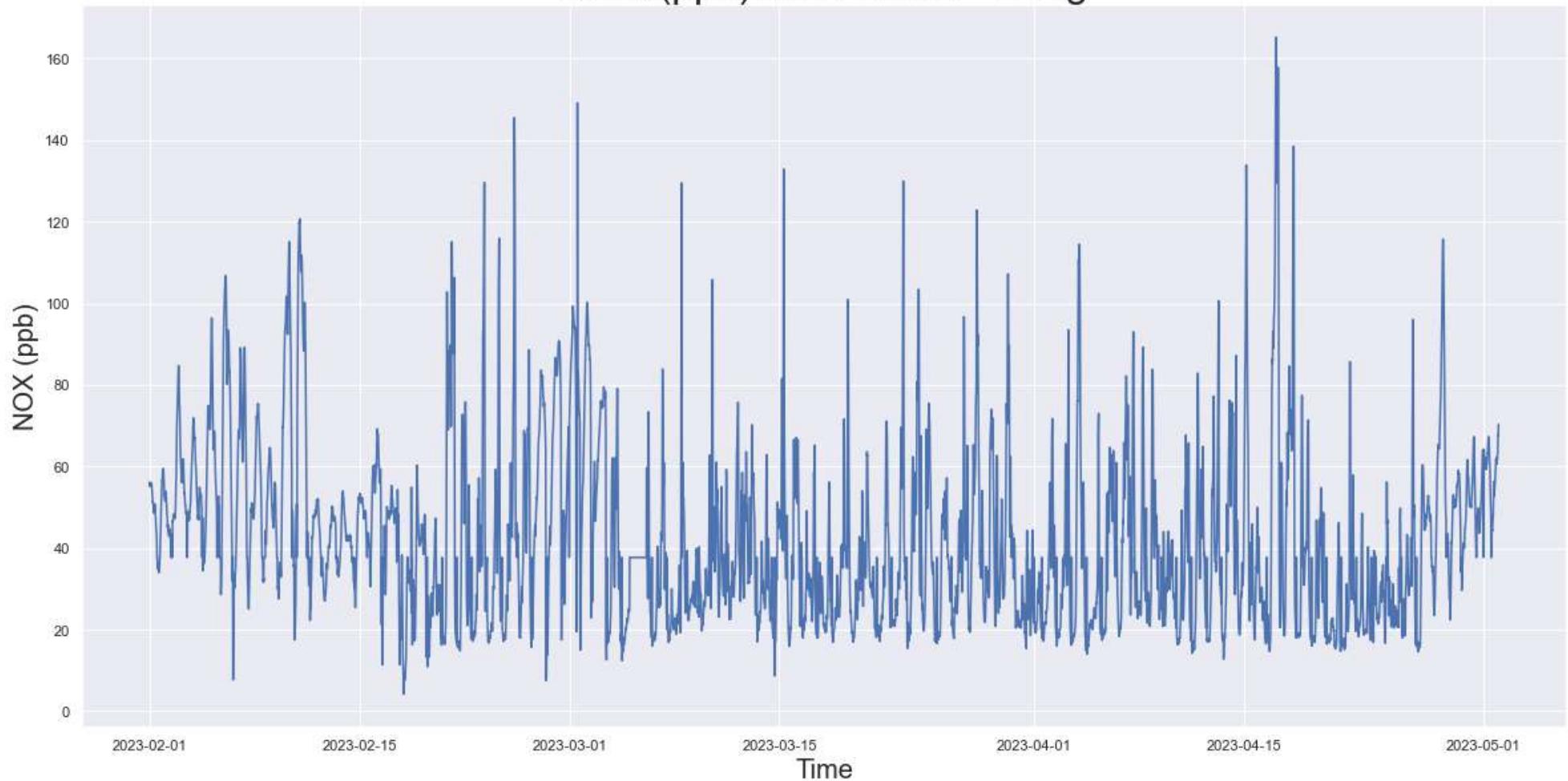
```
In [136]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

NOX (ppb) after mean filling



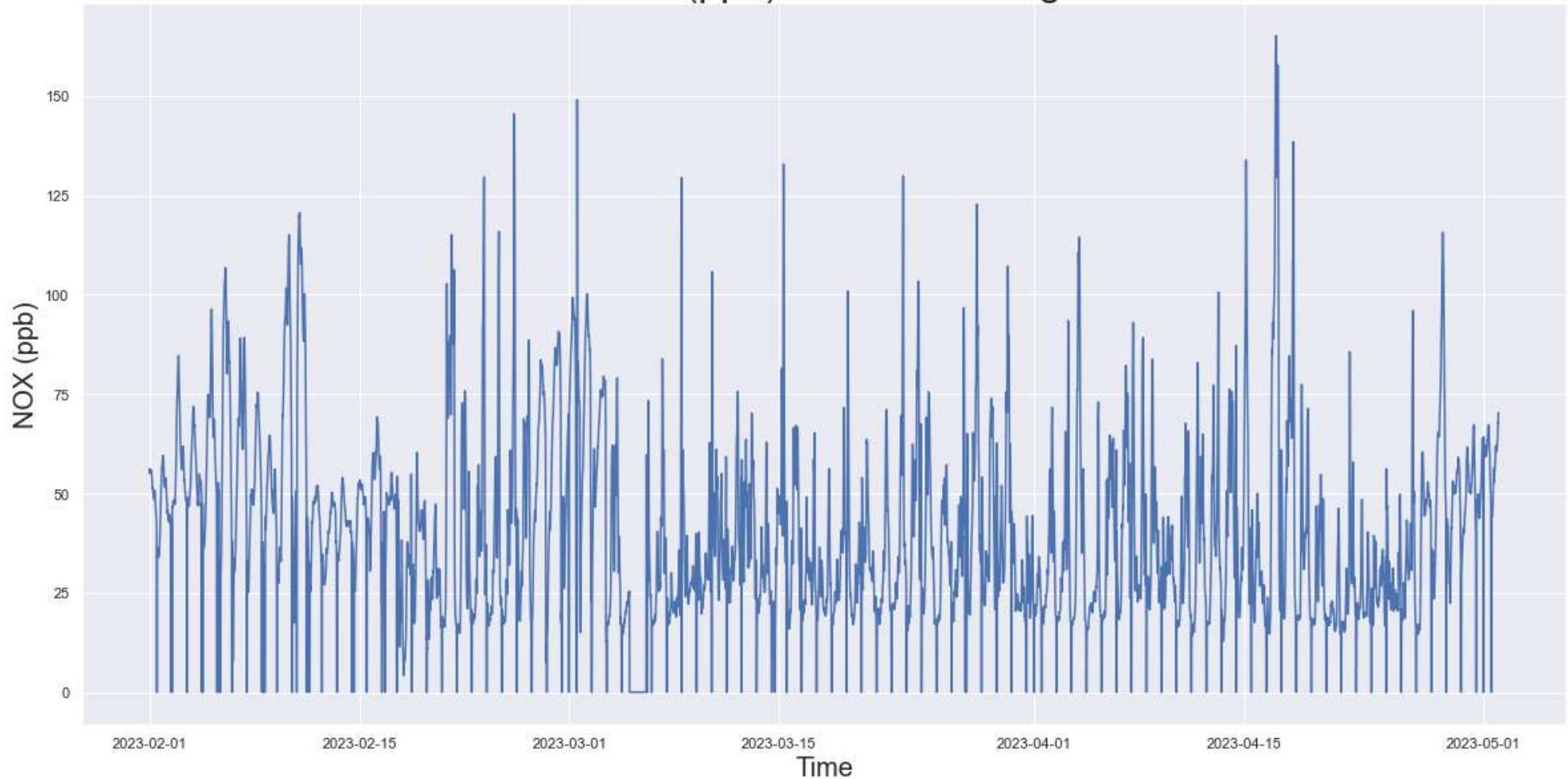
```
In [137]: simple_time_plot(df_median,pollutant,"after median filling")
```

NOX (ppb) after median filling



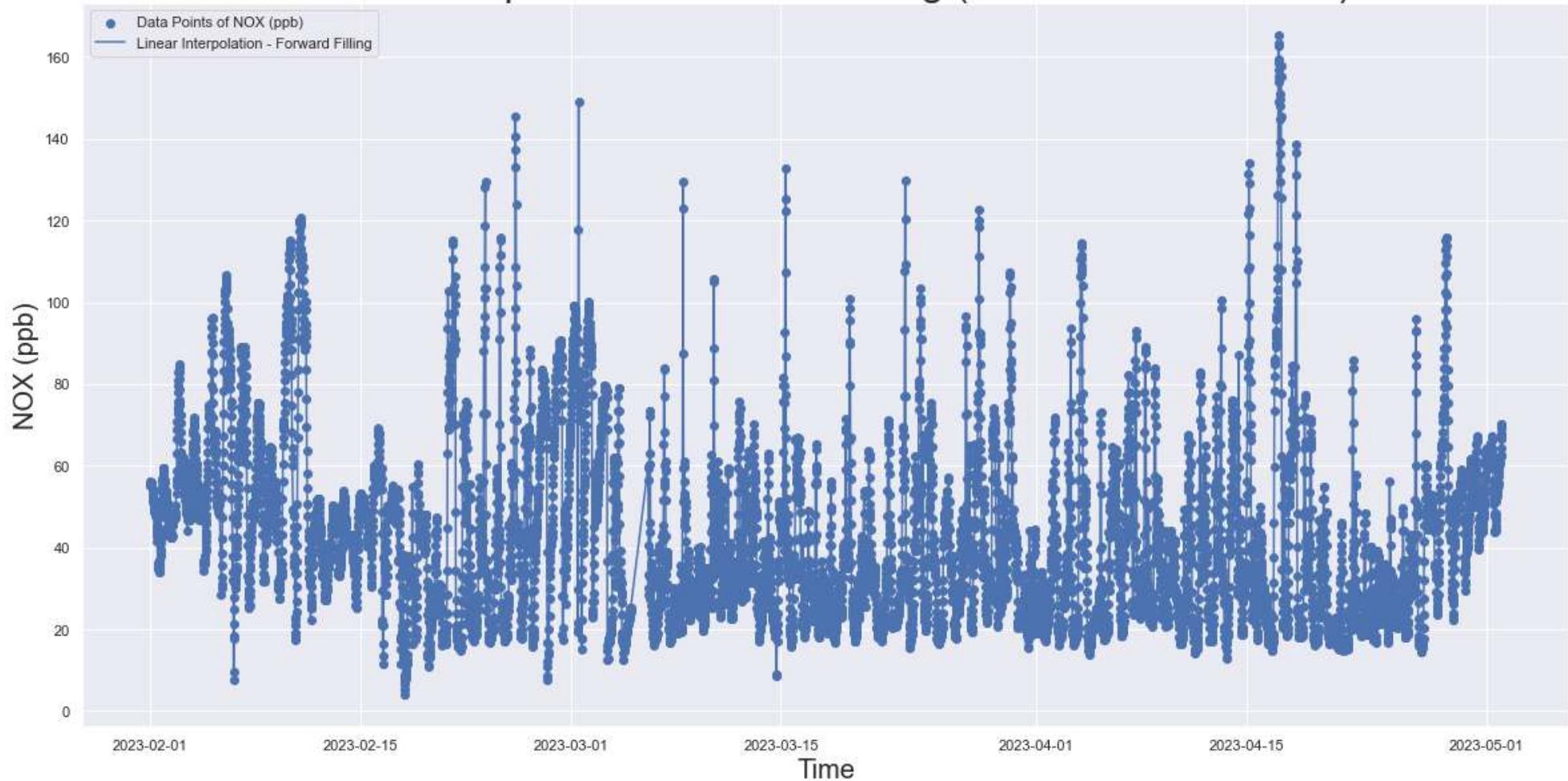
```
In [138]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

NOX (ppb) after zero filling

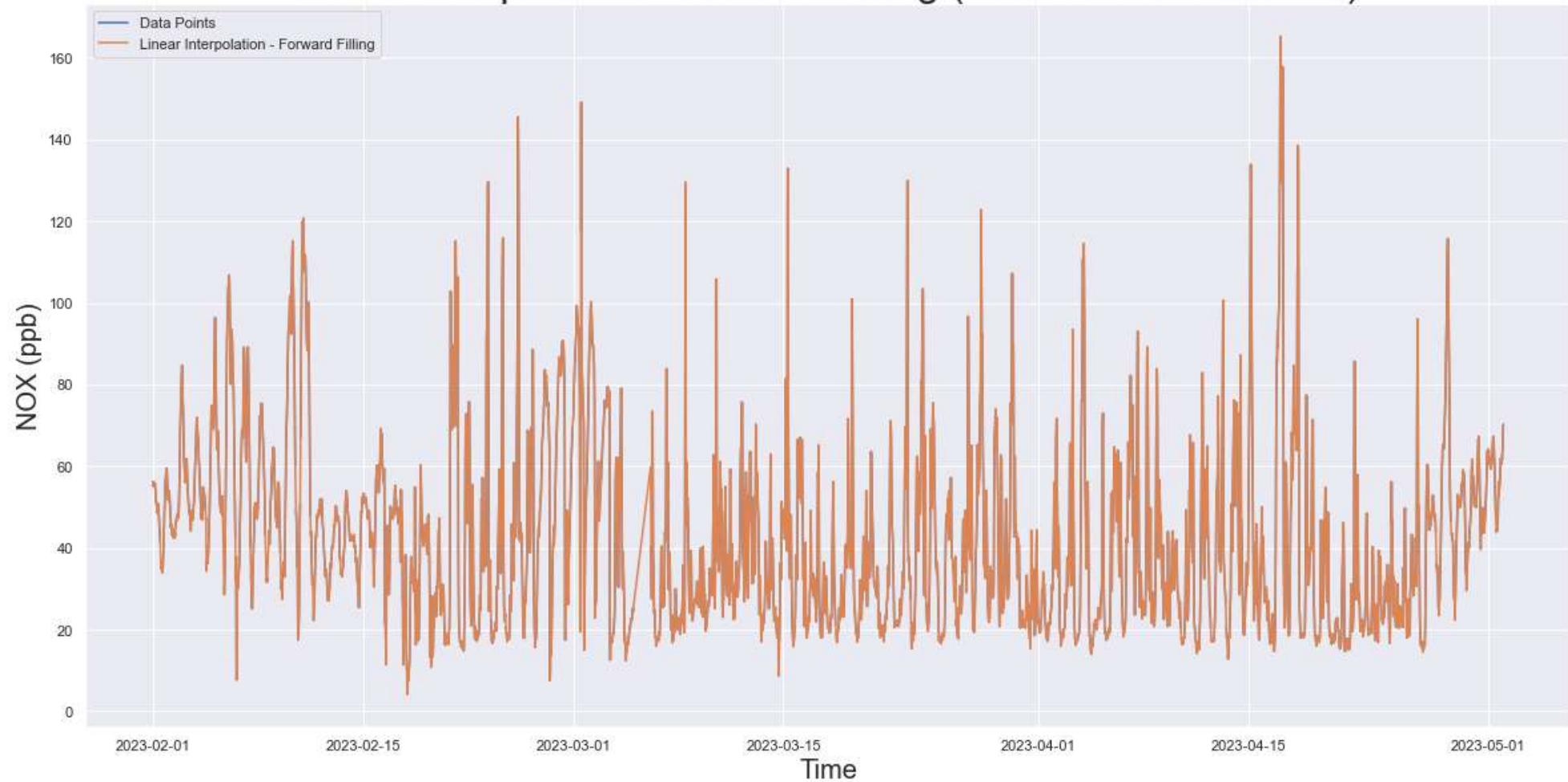


```
In [139]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```

Linear Interpolation - Forward Filling (Data Points Included)

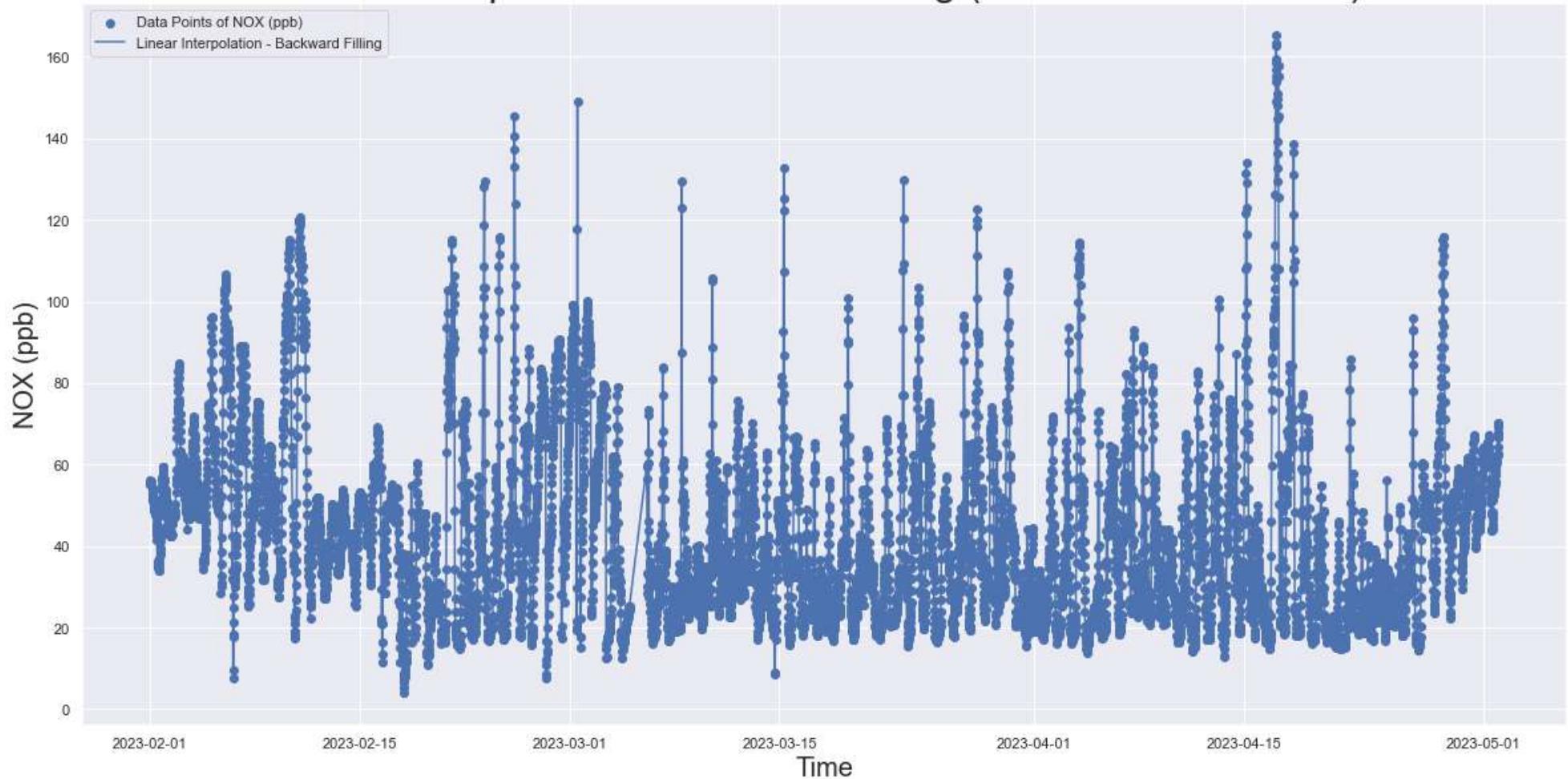


Linear Interpolation - Forward Filling (Data Points Excluded)

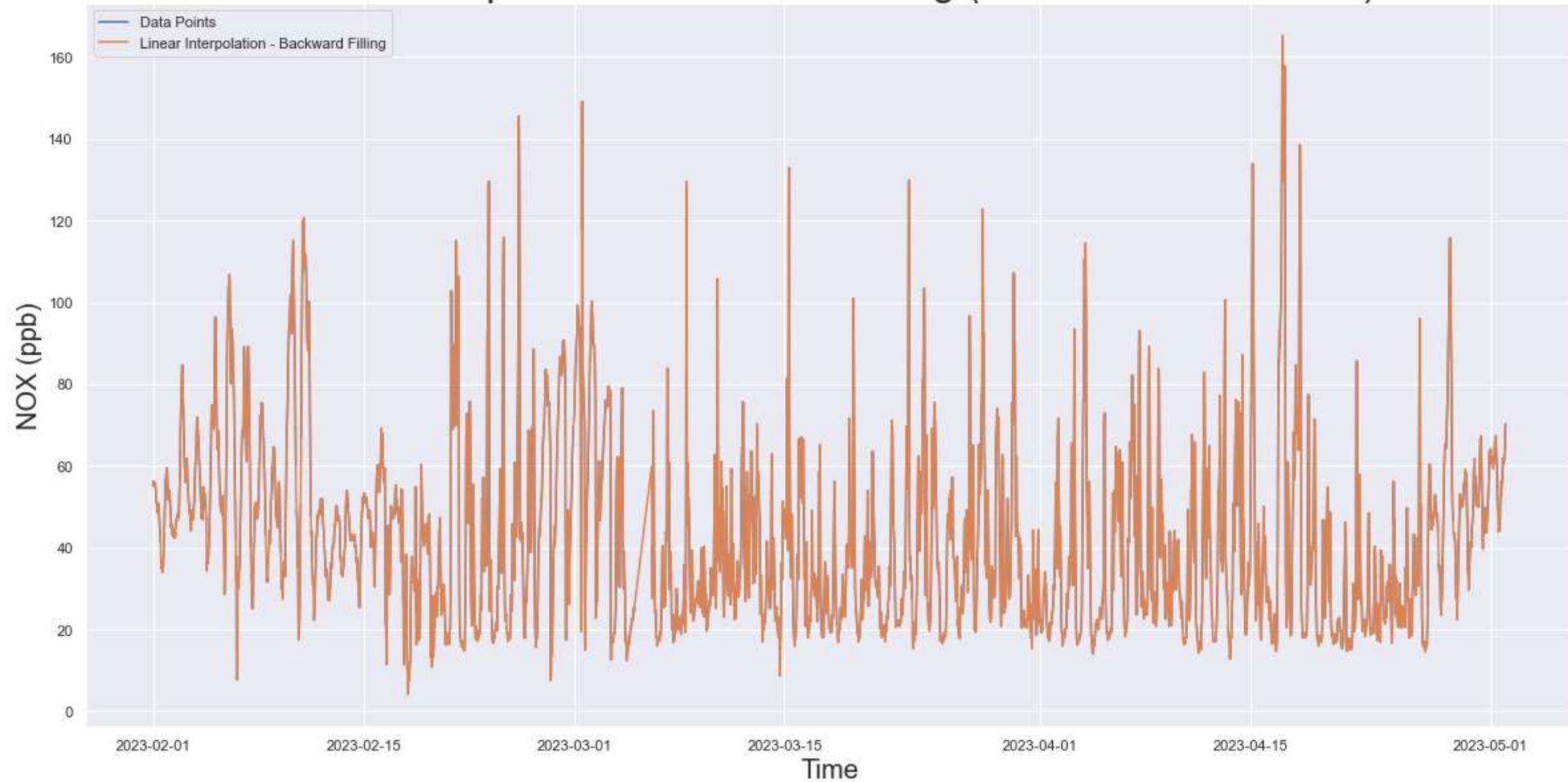


```
In [140]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

Linear Interpolation - Backward Filling (Data Points Included)

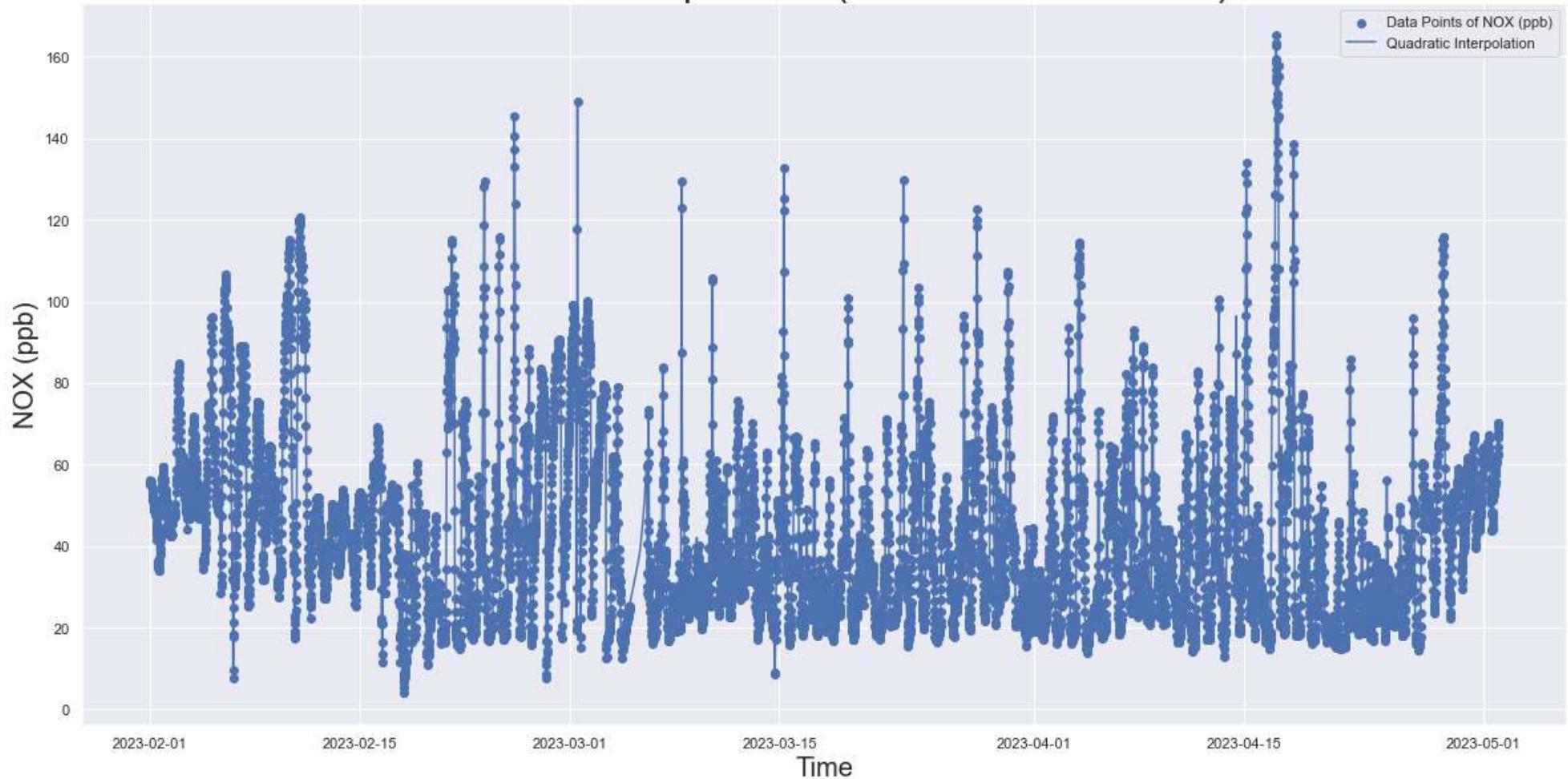


Linear Interpolation - Backward Filling (Data Points Excluded)

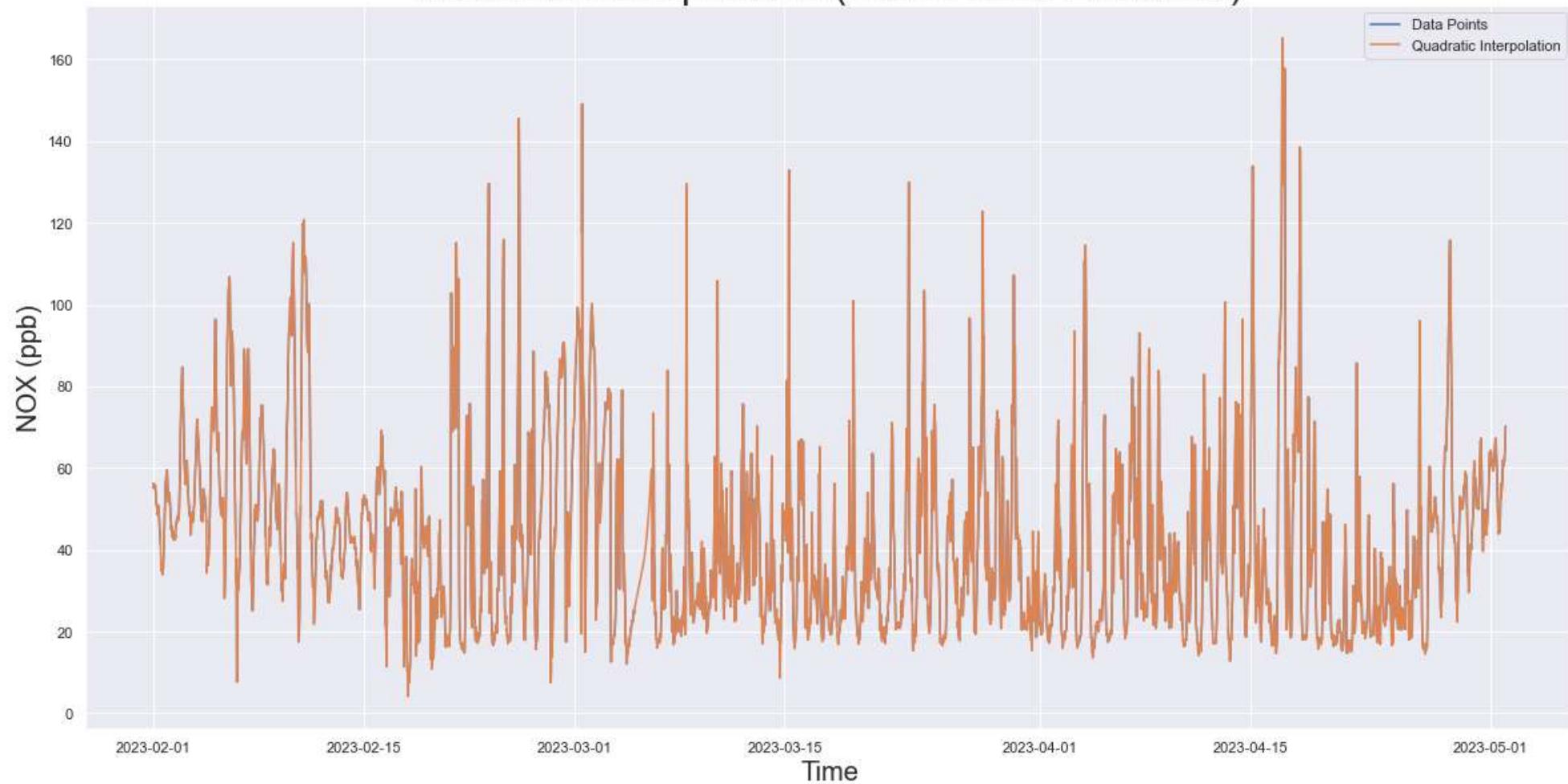


```
In [141]: interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")
```

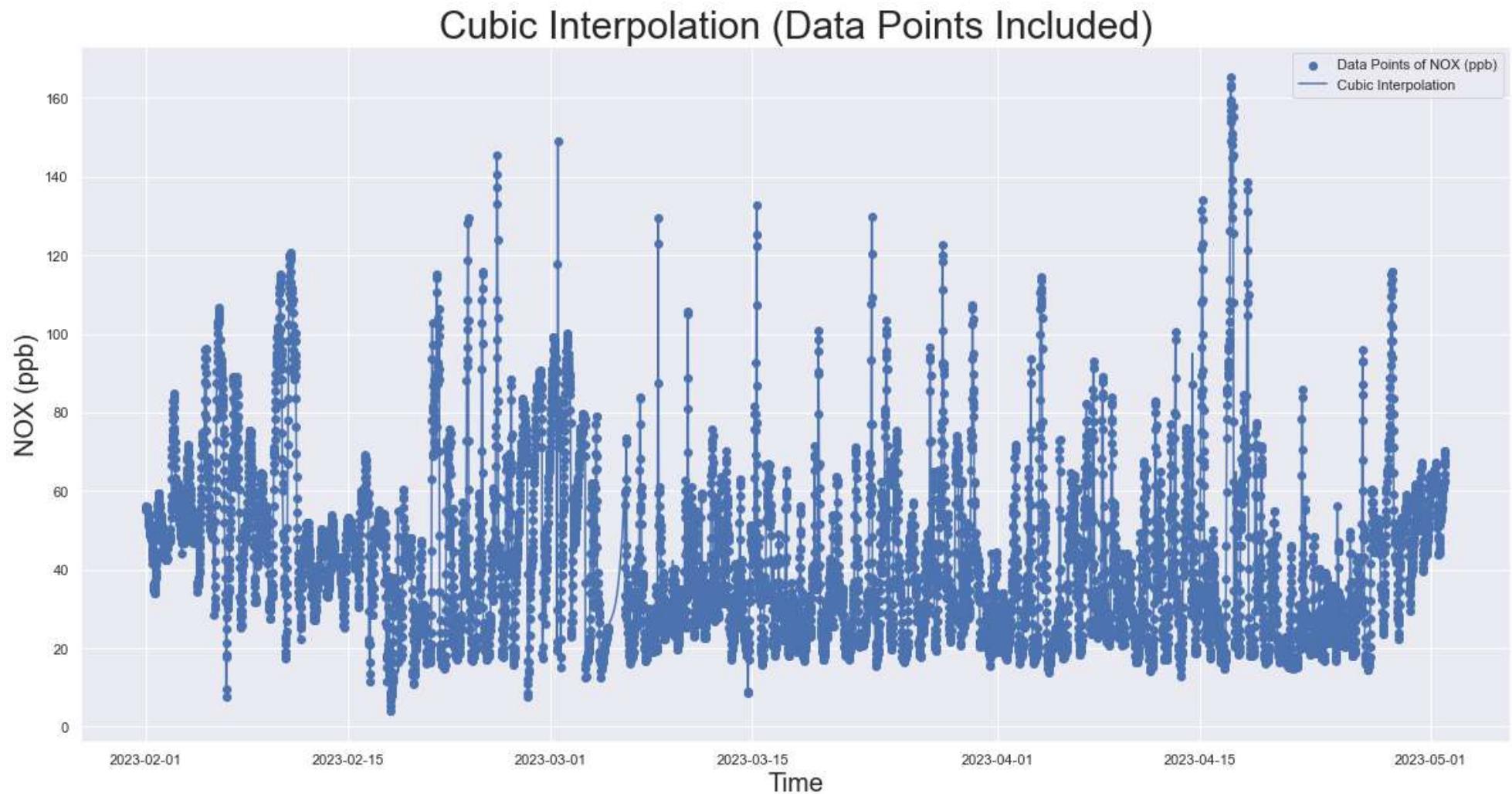
Quadratic Interpolation (Data Points Included)



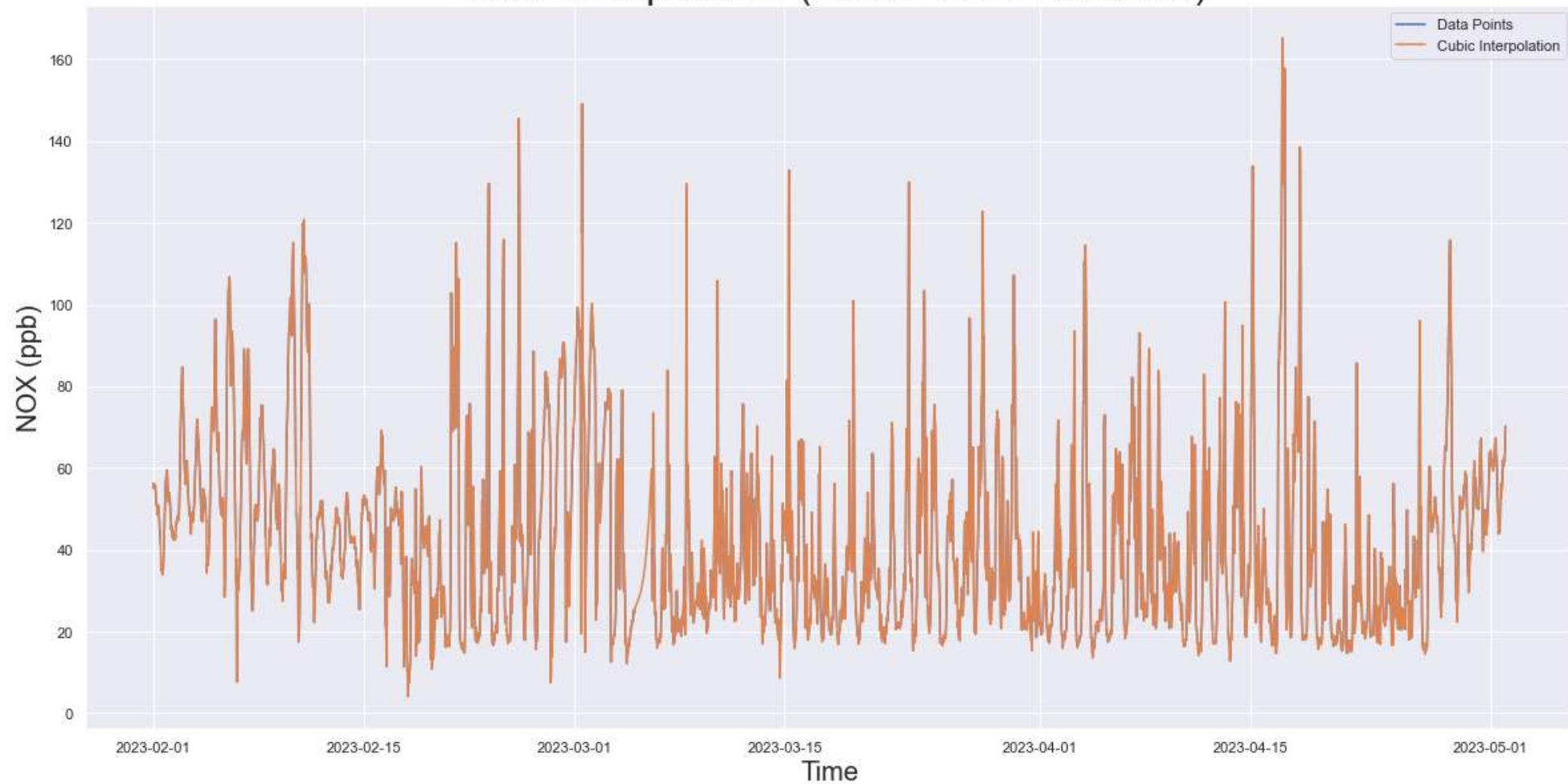
Quadratic Interpolation (Data Points Excluded)



```
In [142]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```

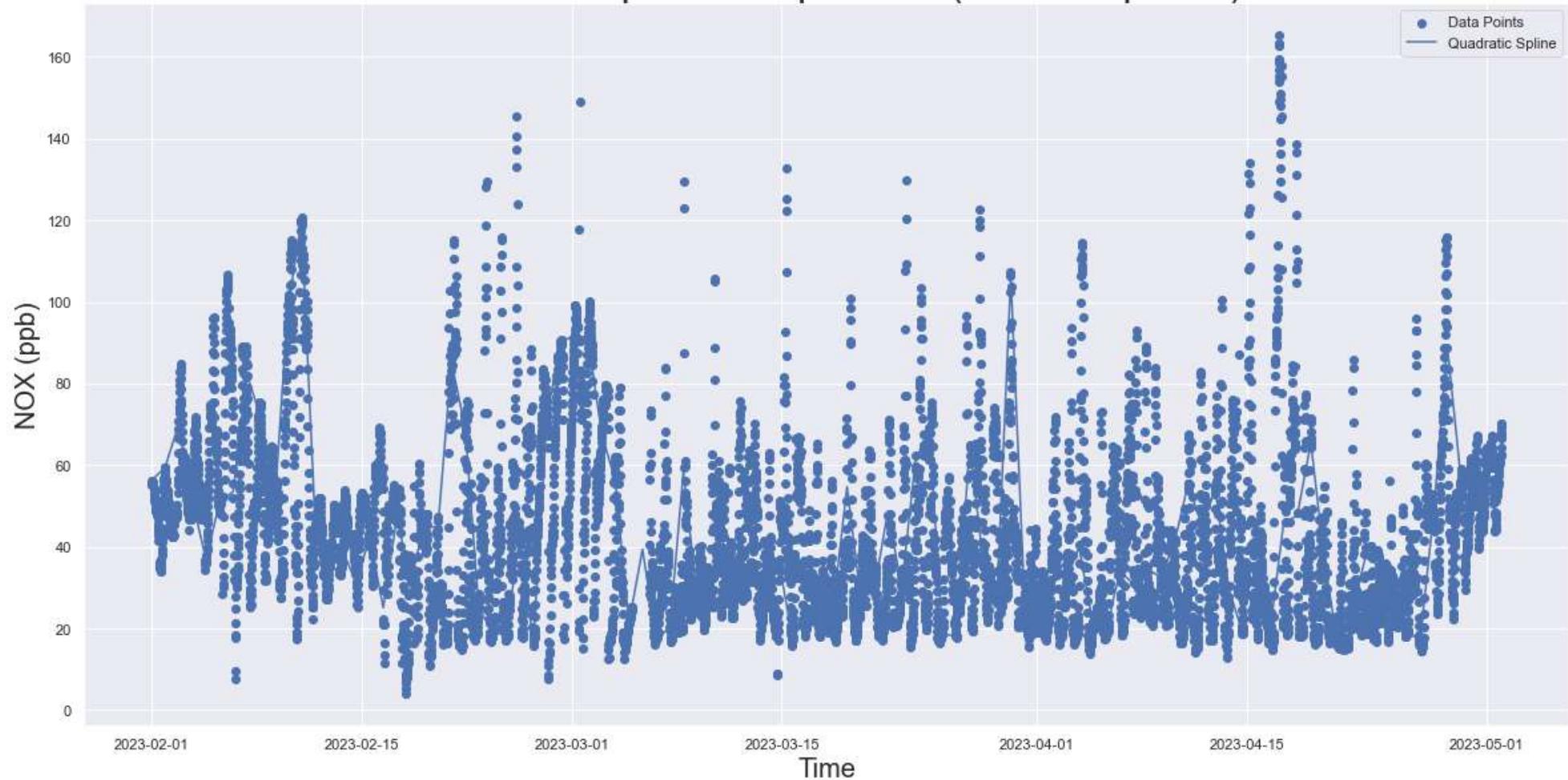


Cubic Interpolation (Data Points Excluded)

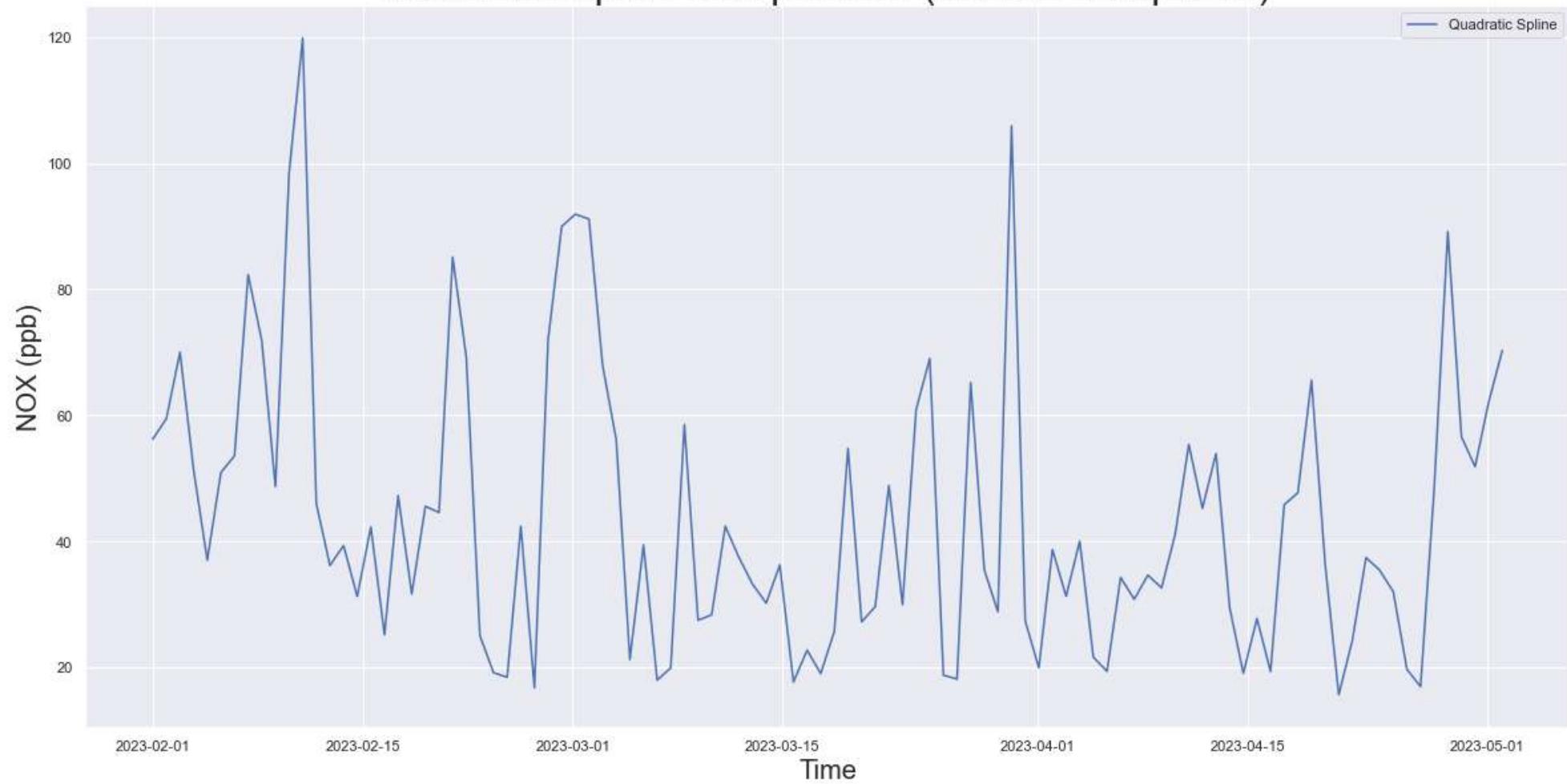


```
In [143]: quadraticspline(df_n,pollutant)
```

Quadratic Spline Interpolation (with datapoints)

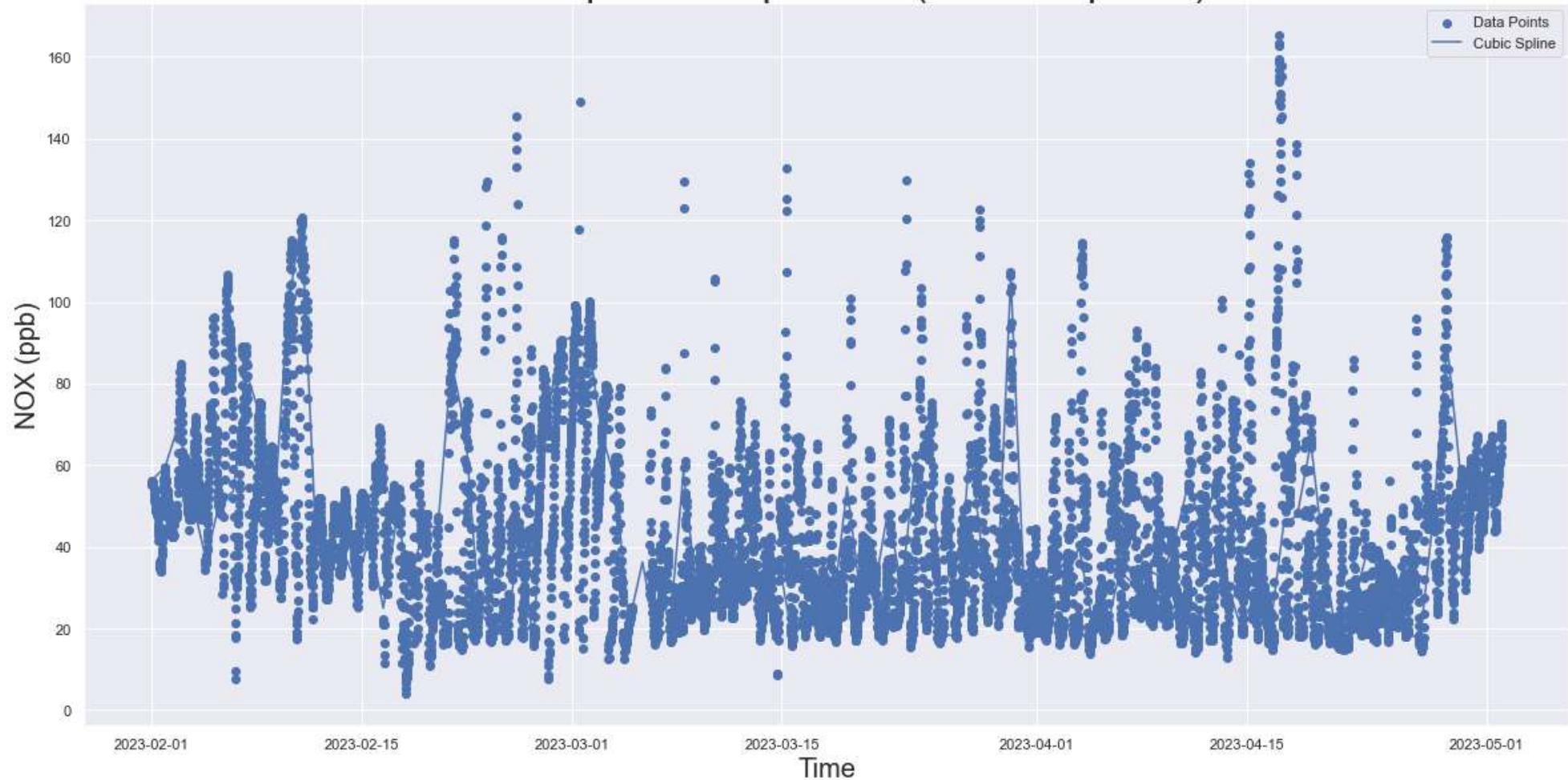


Quadratic Spline Interpolation (without datapoints)

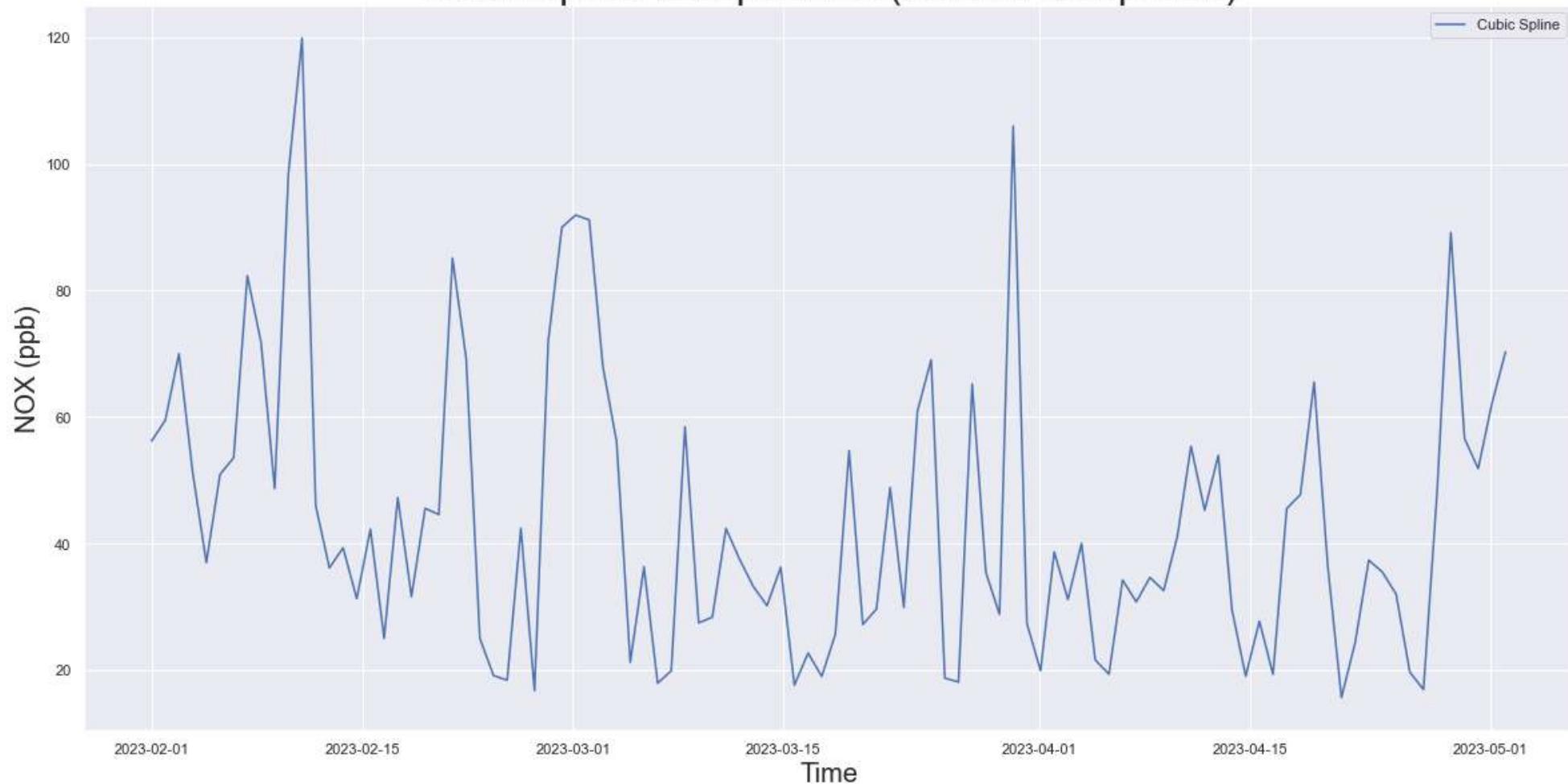


In [144]: `cubicspline(df_n,pollutant)`

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



data seems to be less concentrated across mean and values less than mean are more in number. Outliers present deteriorate the mean value. Since data oscillates and number of missing value sis less comparatively, any of linear, quadratic or cubic interpolation works accurately.

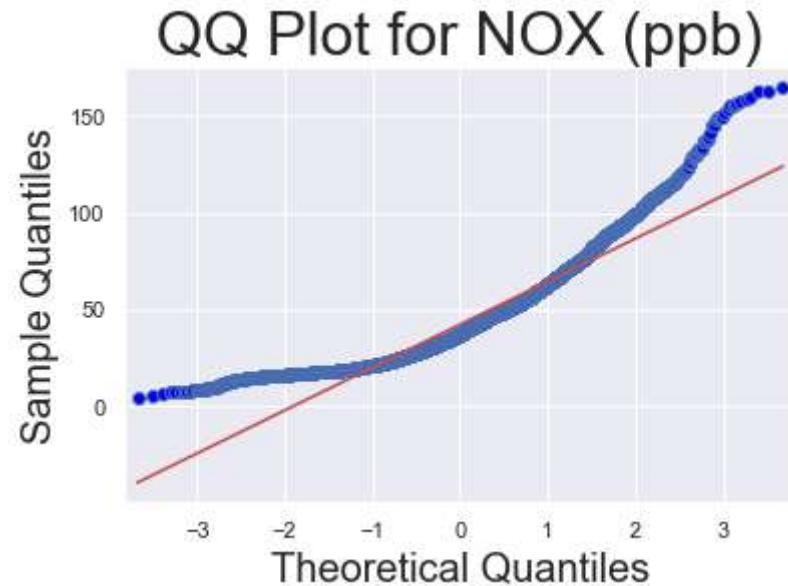
In [145]: `df_cubic[pollutant].isnull().sum()`

Out[145]: 0

In [146]: `df_new[pollutant] = df_cubic[pollutant]`

```
In [147]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>



```
In [148]: df_new.head()
```

Out[148]:

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)
--	-----------------------------------	------------------------------------	---------------------------------	----------------------------------	-----------

Time

2023-02-01 00:00:00	95.0	35.0	18.1	90.1	56.2
2023-02-01 00:15:00	95.0	35.0	18.1	88.0	55.1
2023-02-01 00:30:00	95.0	35.0	18.1	87.7	55.2
2023-02-01 00:45:00	122.0	34.0	18.1	88.9	55.7
2023-02-01 01:00:00	122.0	34.0	18.1	90.0	55.8

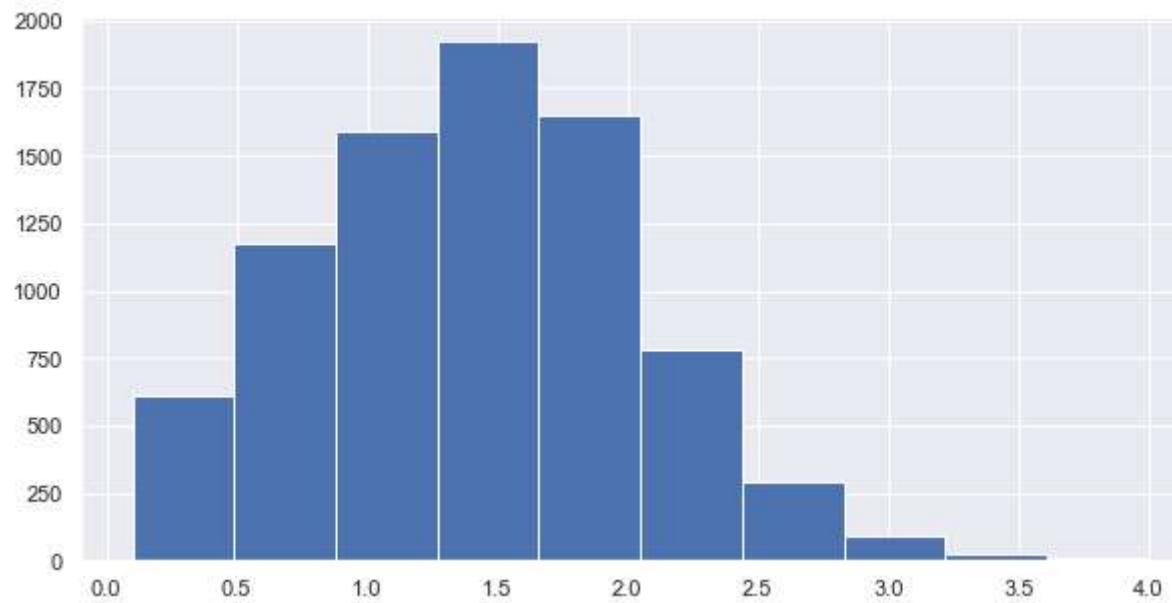
Analysis for "CO (mg/m3)"

```
In [149]: pollutant = 'CO (mg/m3)'
```

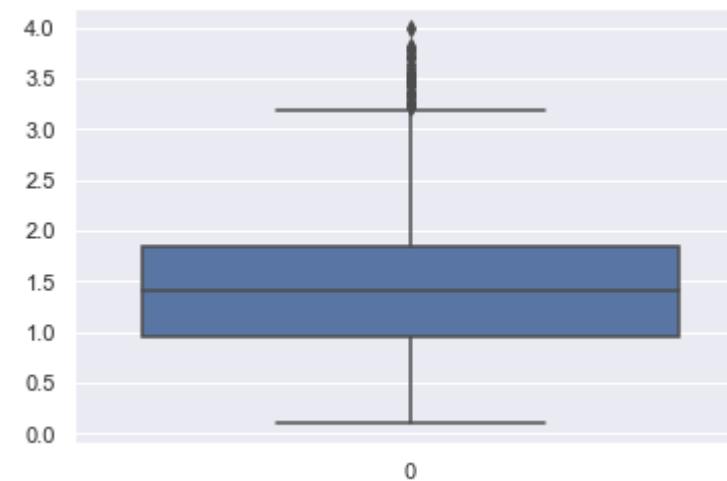
```
In [150]: df[pollutant].describe()
```

```
Out[150]: count    8144.000000
mean      1.408538
std       0.631056
min      0.100000
25%      0.950000
50%      1.420000
75%      1.850000
max      4.000000
Name: CO (mg/m3), dtype: float64
```

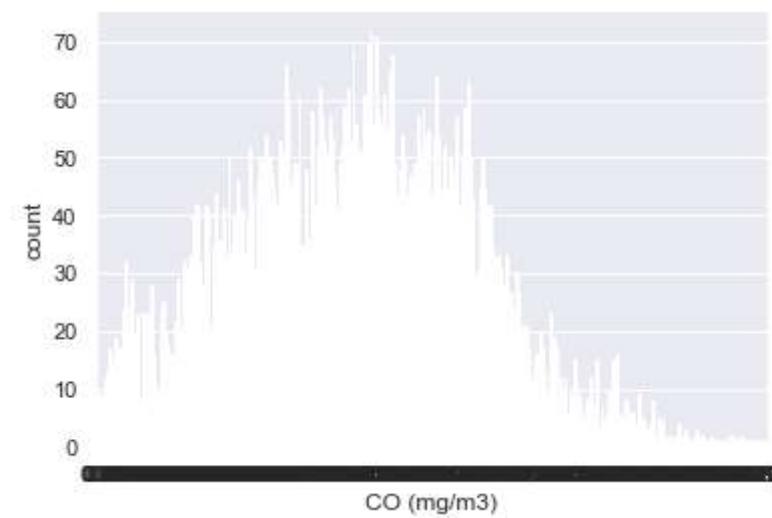
```
In [151]: histogram_plot(df_n,pollutant)
```



```
In [152]: boxplot_plot(df_n,pollutant)
```

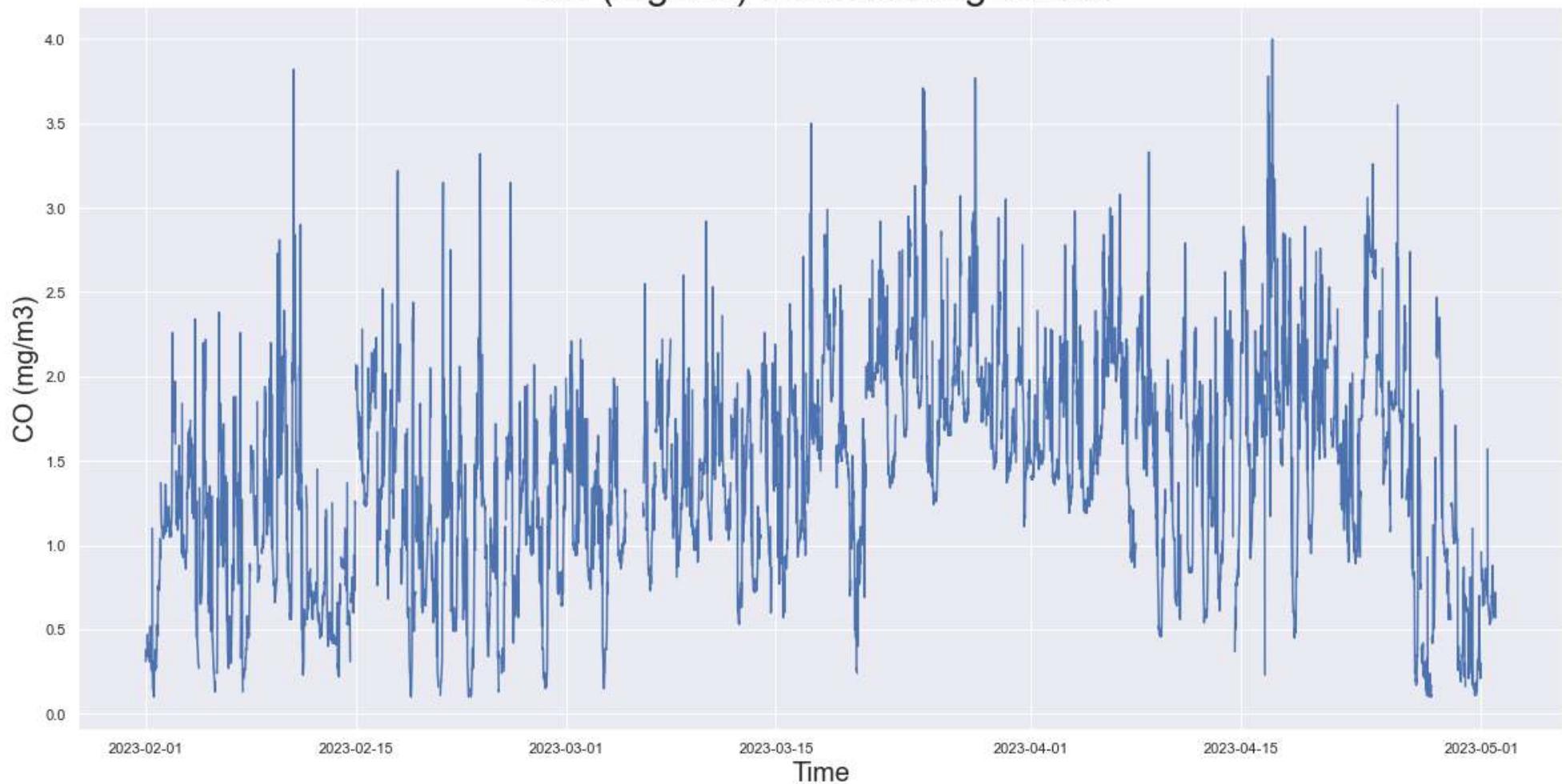


```
In [153]: countplot_plot(df_n,pollutant)
```



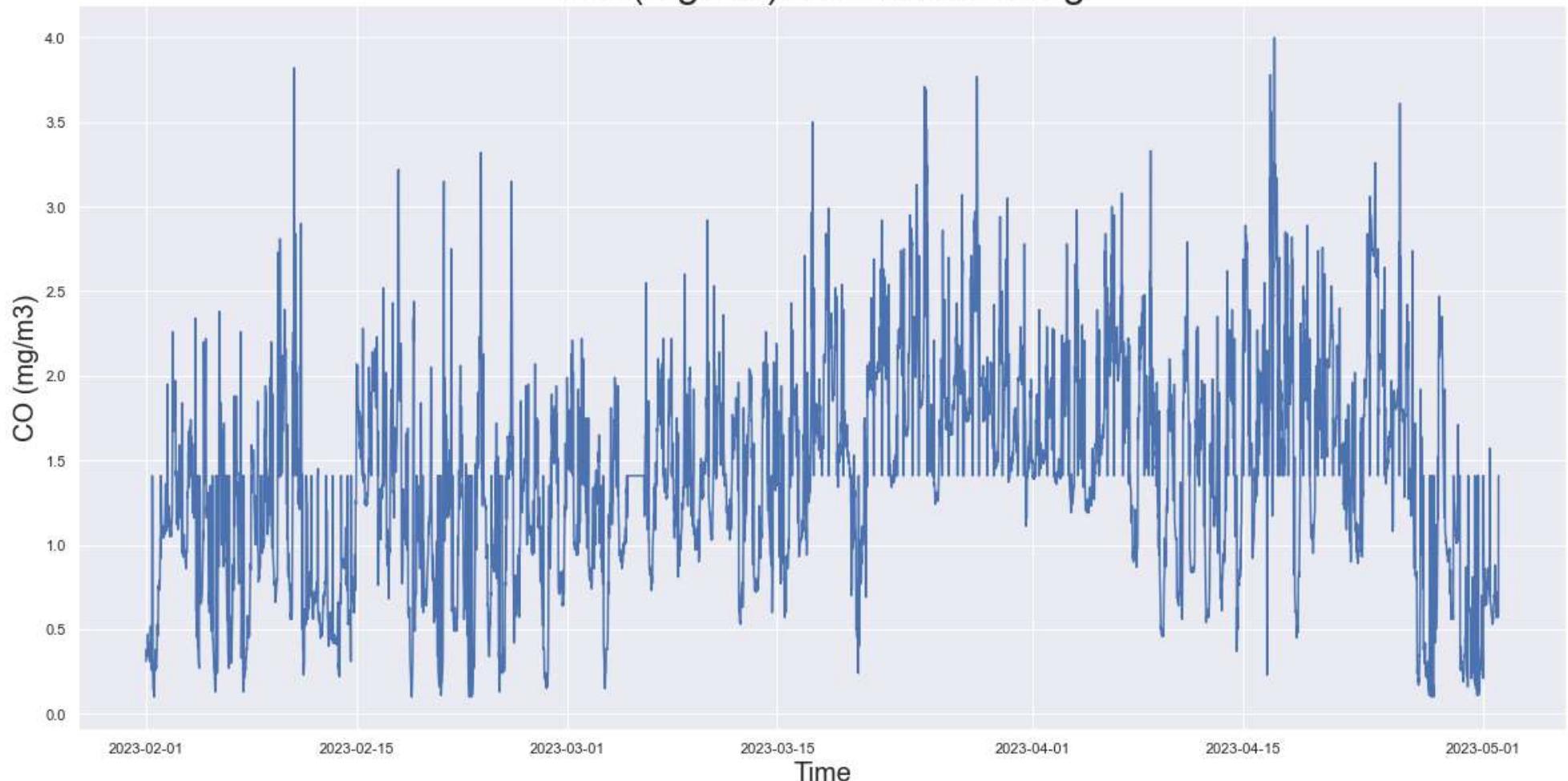
```
In [154]: simple_time_plot(df_n,pollutant,"With missing values")
```

CO (mg/m3) With missing values



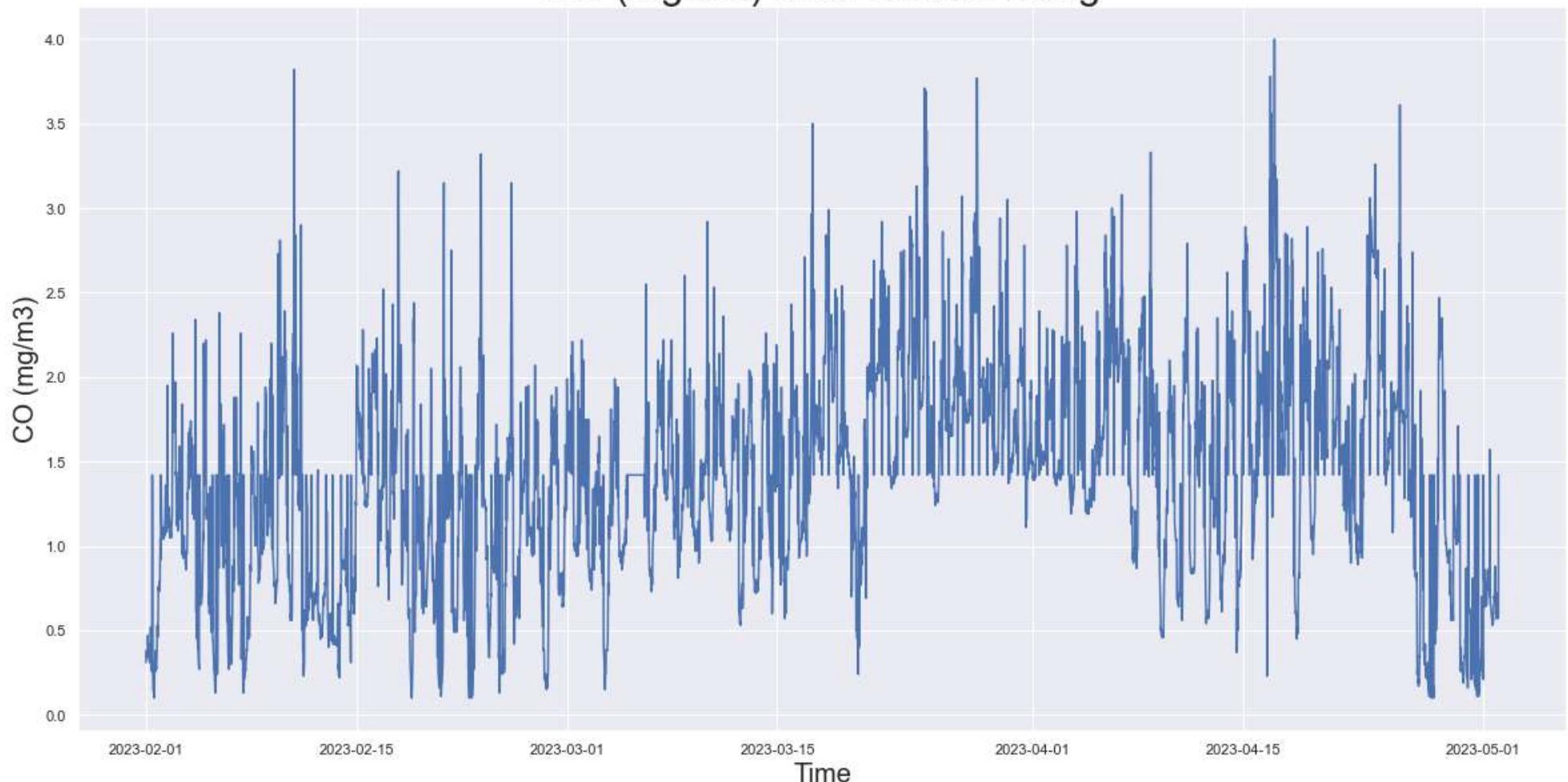
```
In [155]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

CO (mg/m3) after mean filling



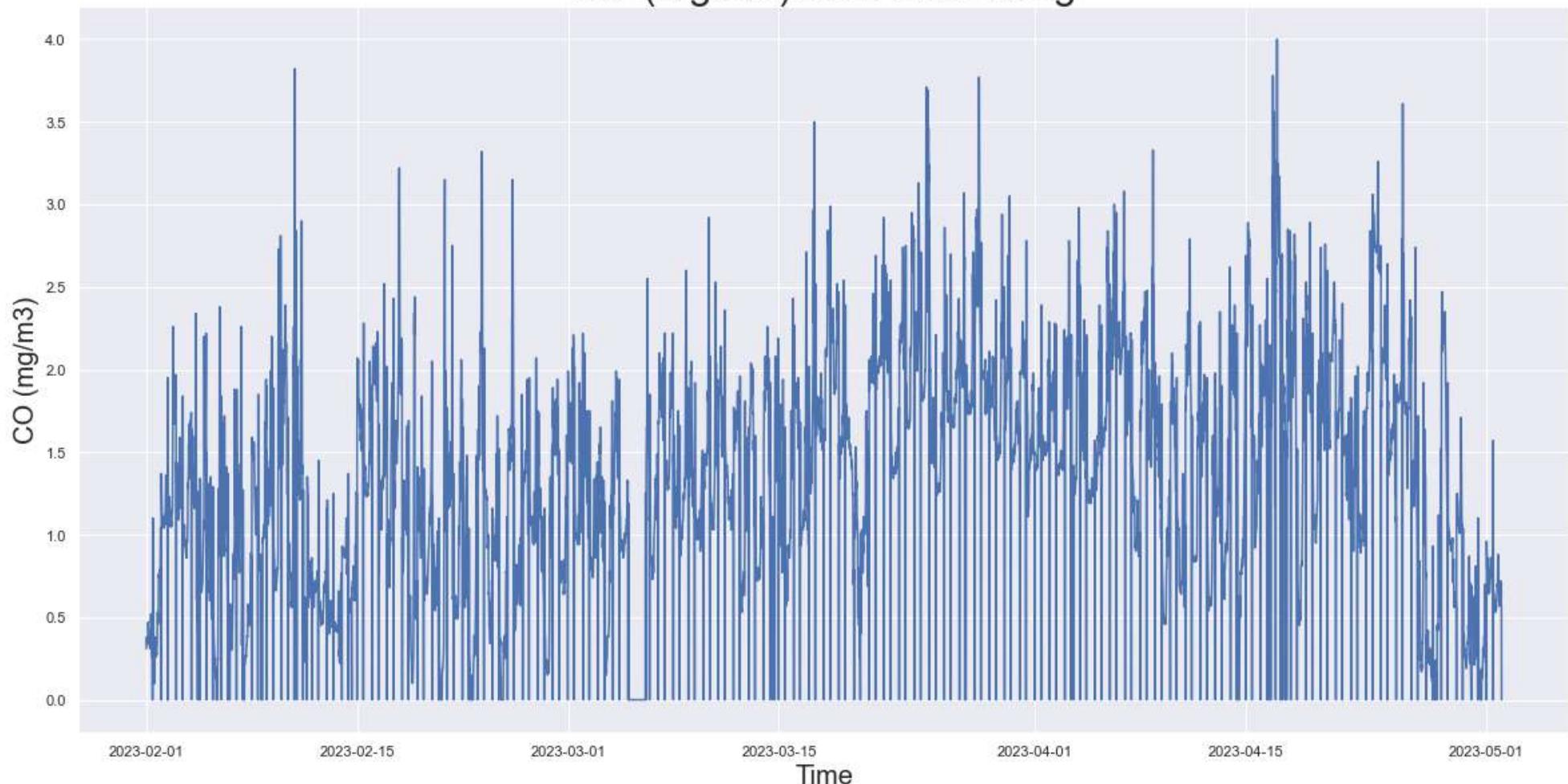
```
In [156]: simple_time_plot(df_median,pollutant,"after median filling")
```

CO (mg/m³) after median filling



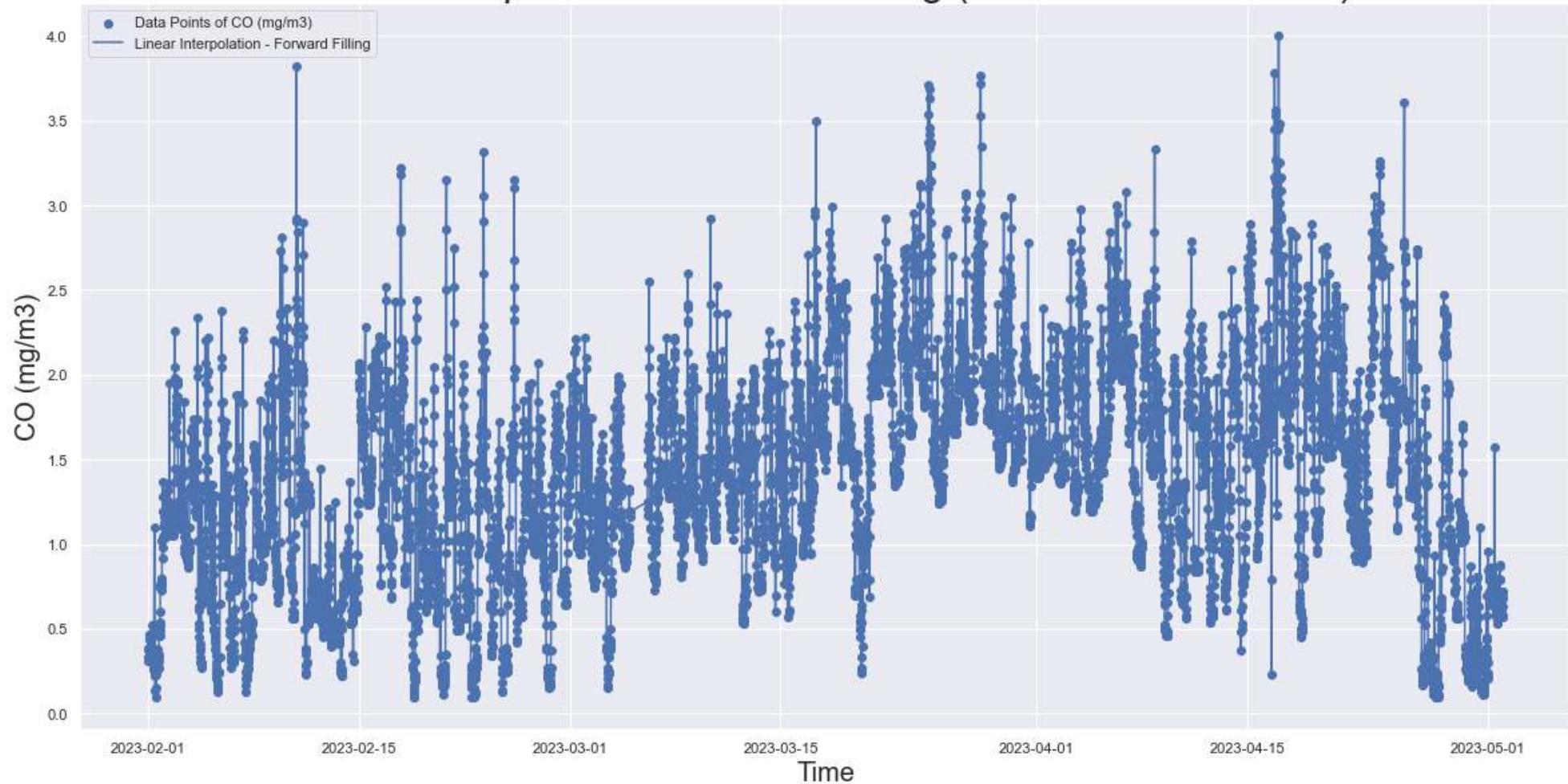
```
In [157]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

CO (mg/m³) after zero filling

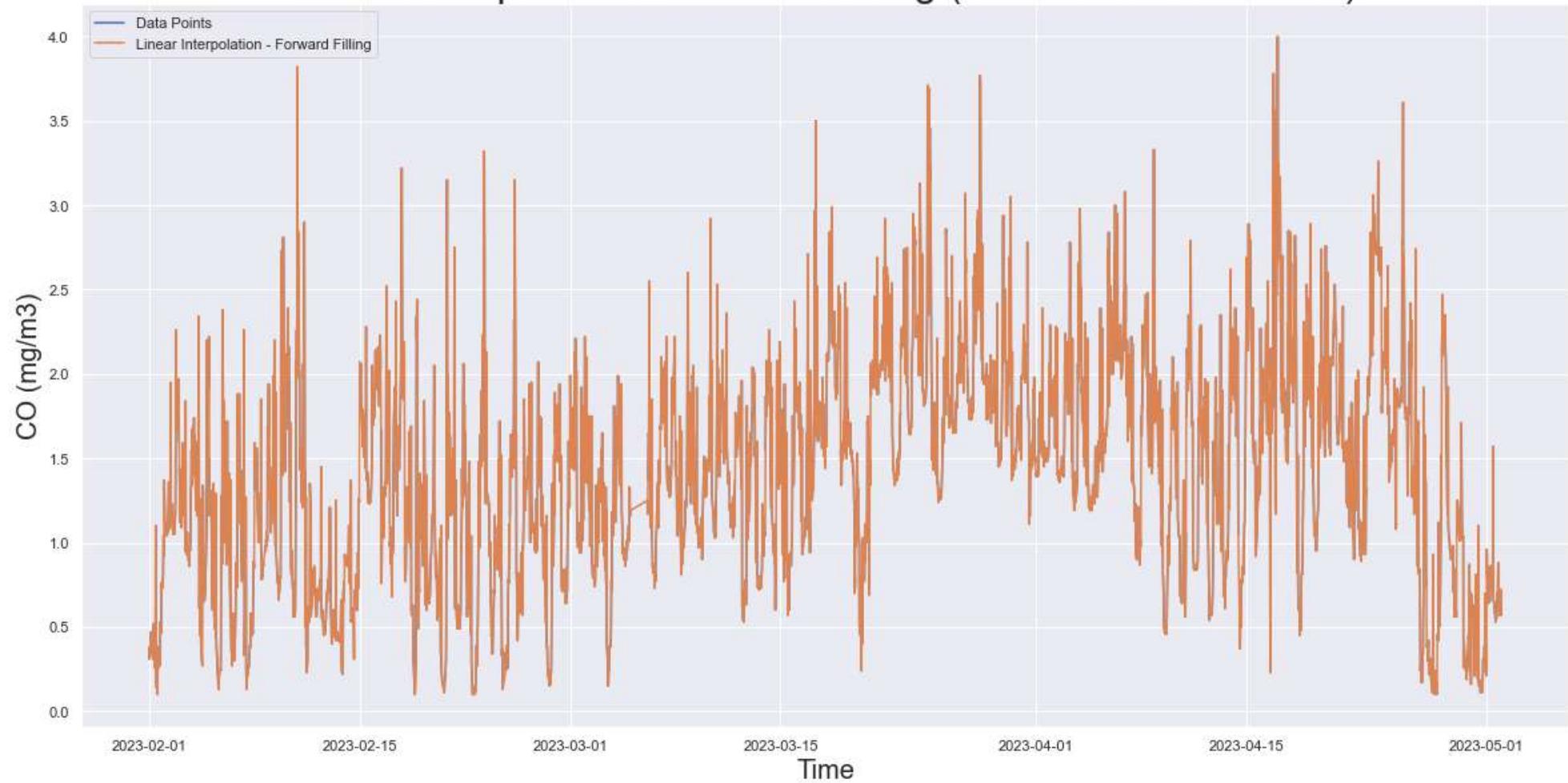


```
In [158]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```

Linear Interpolation - Forward Filling (Data Points Included)

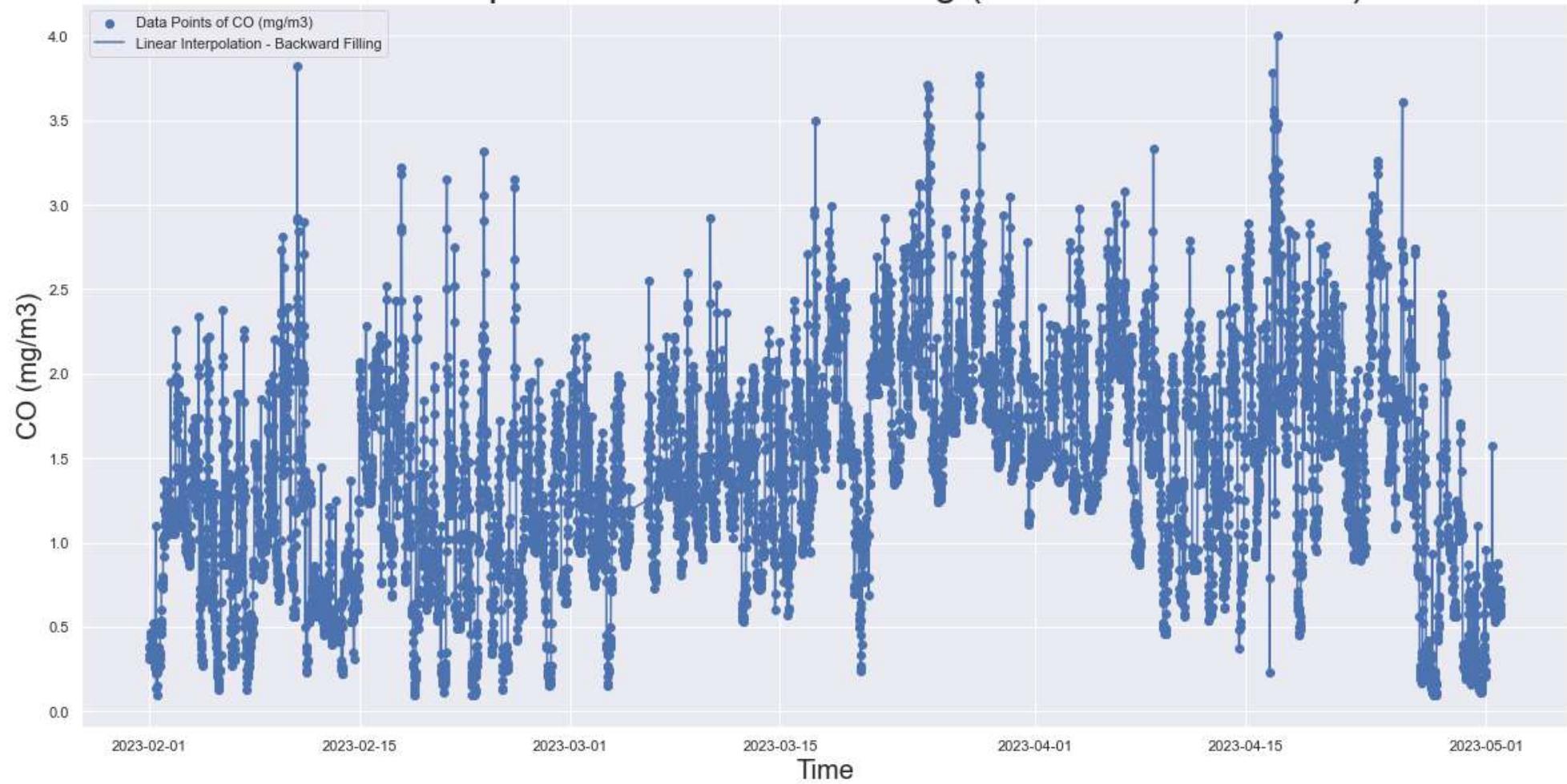


Linear Interpolation - Forward Filling (Data Points Excluded)

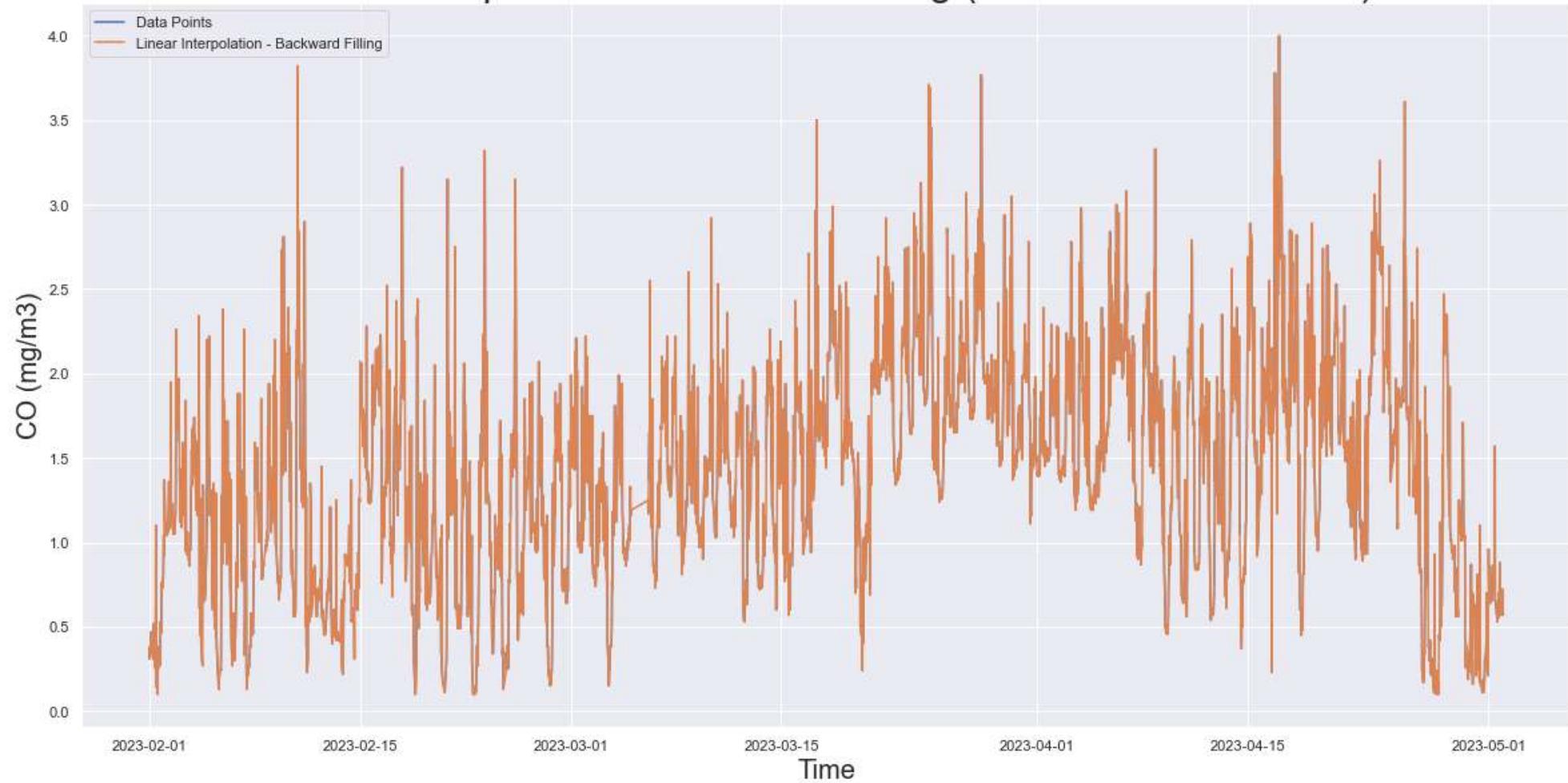


```
In [159]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

Linear Interpolation - Backward Filling (Data Points Included)

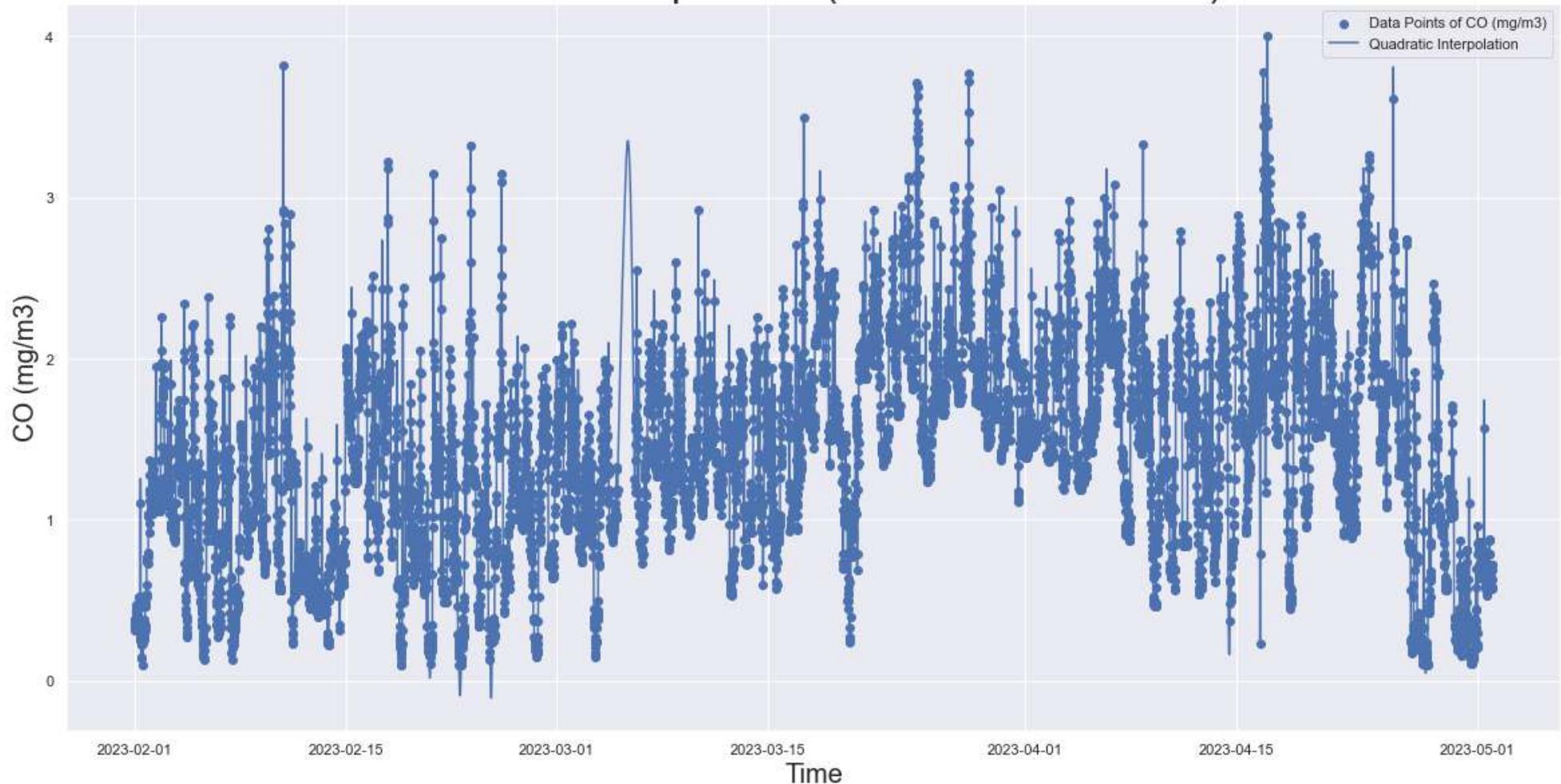


Linear Interpolation - Backward Filling (Data Points Excluded)

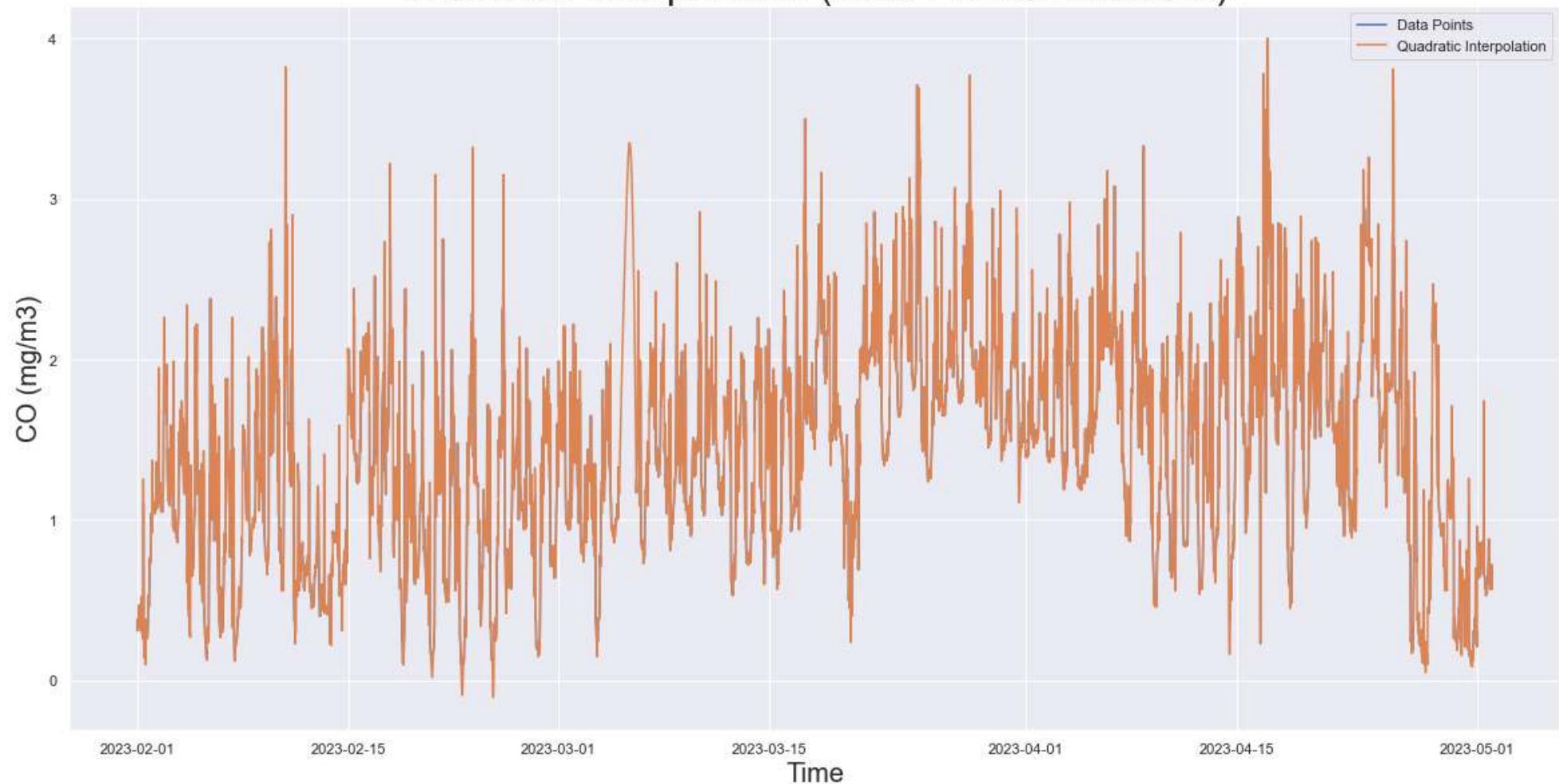


```
In [160]: interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")
```

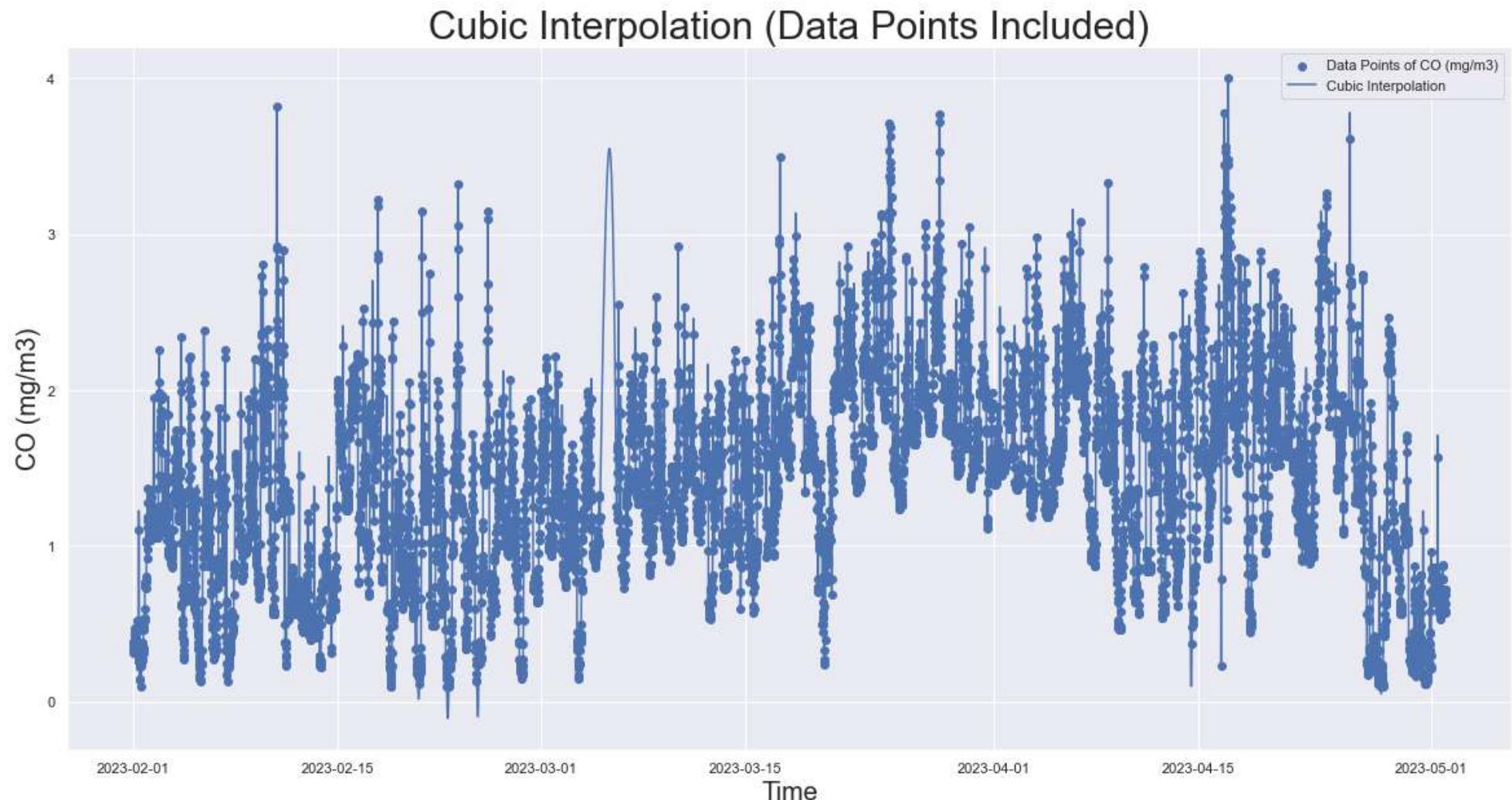
Quadratic Interpolation (Data Points Included)



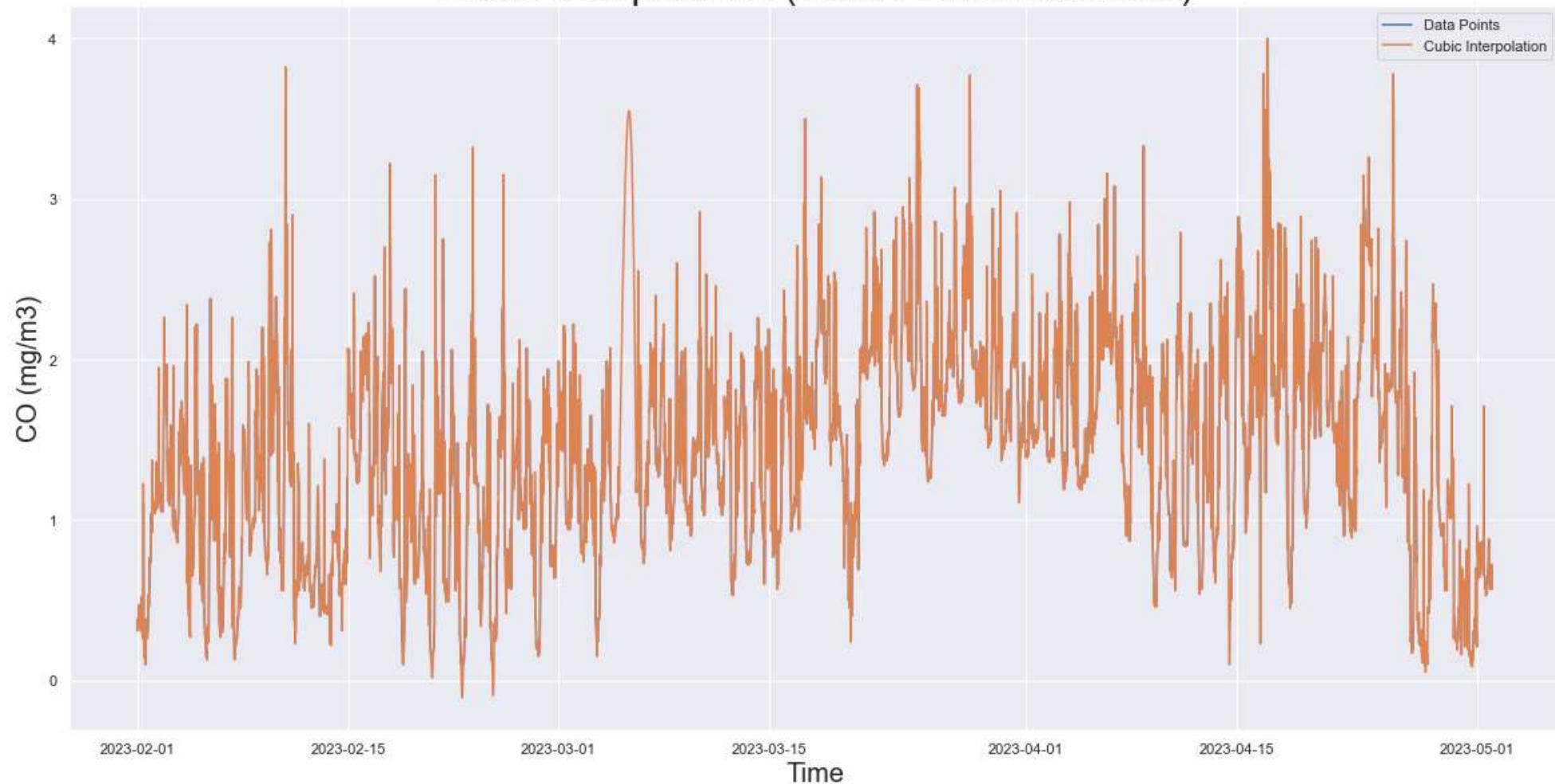
Quadratic Interpolation (Data Points Excluded)



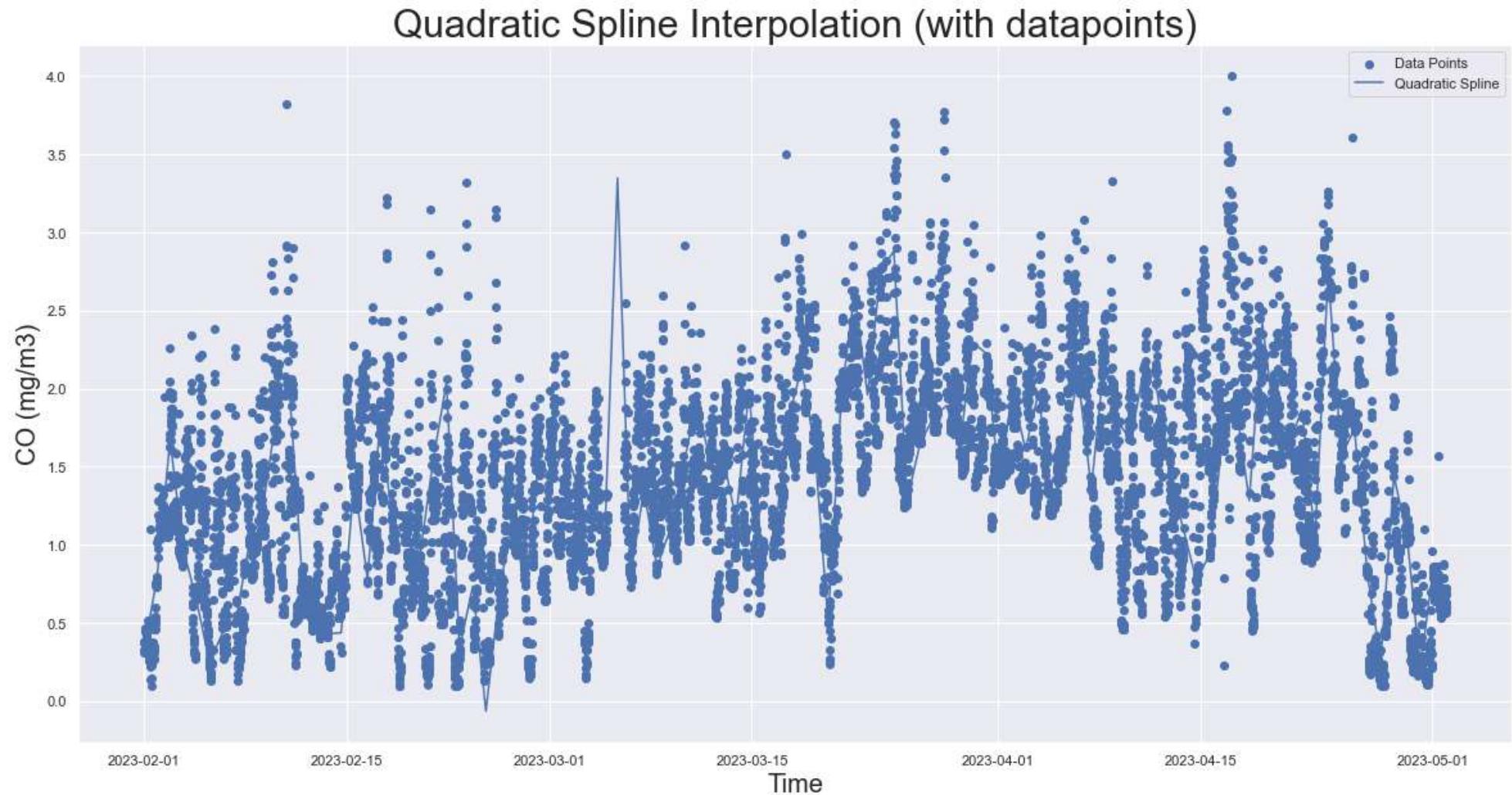
```
In [161]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```



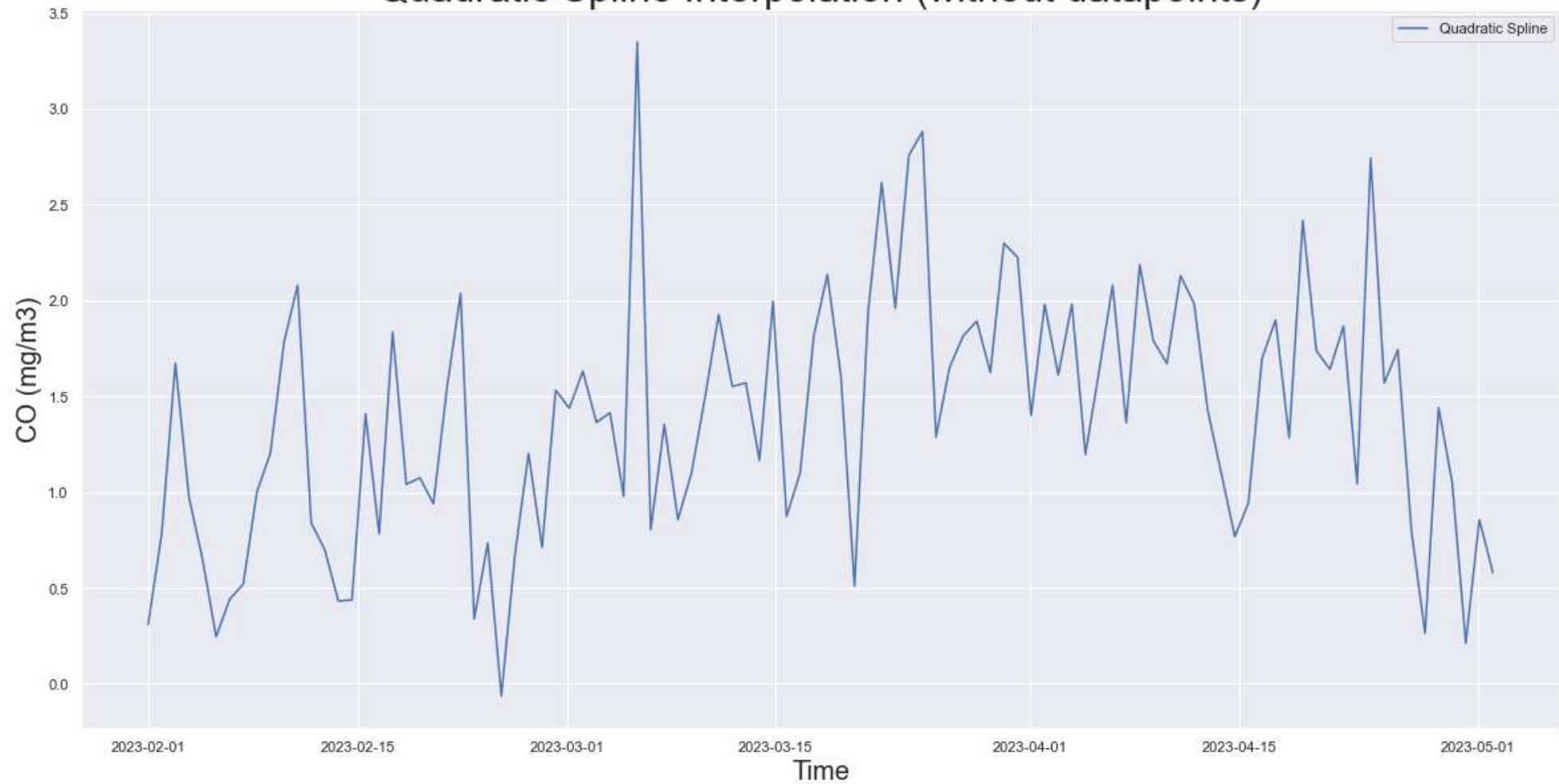
Cubic Interpolation (Data Points Excluded)



In [162]: `quadraticspline(df_n,pollutant)`

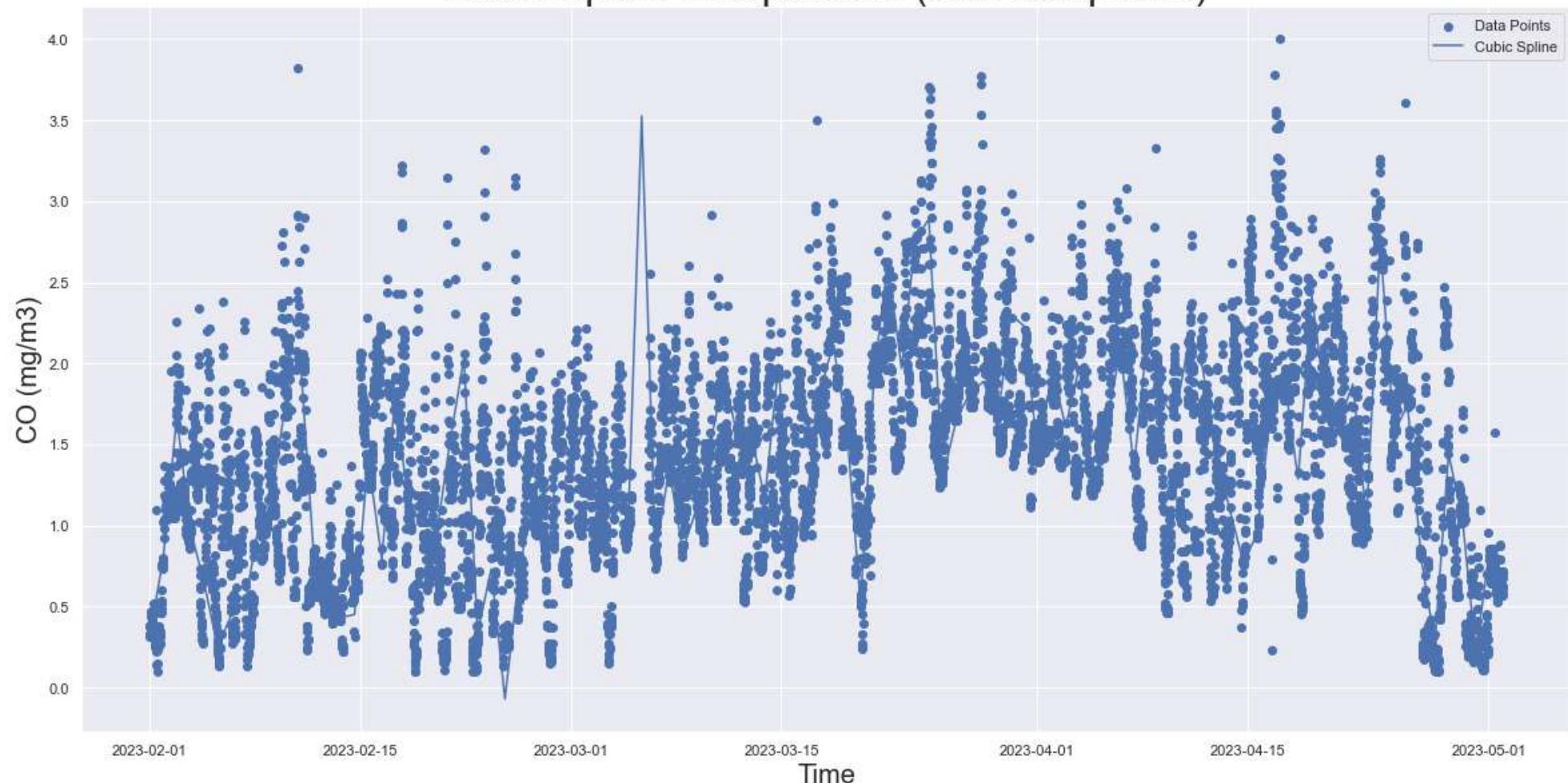


Quadratic Spline Interpolation (without datapoints)

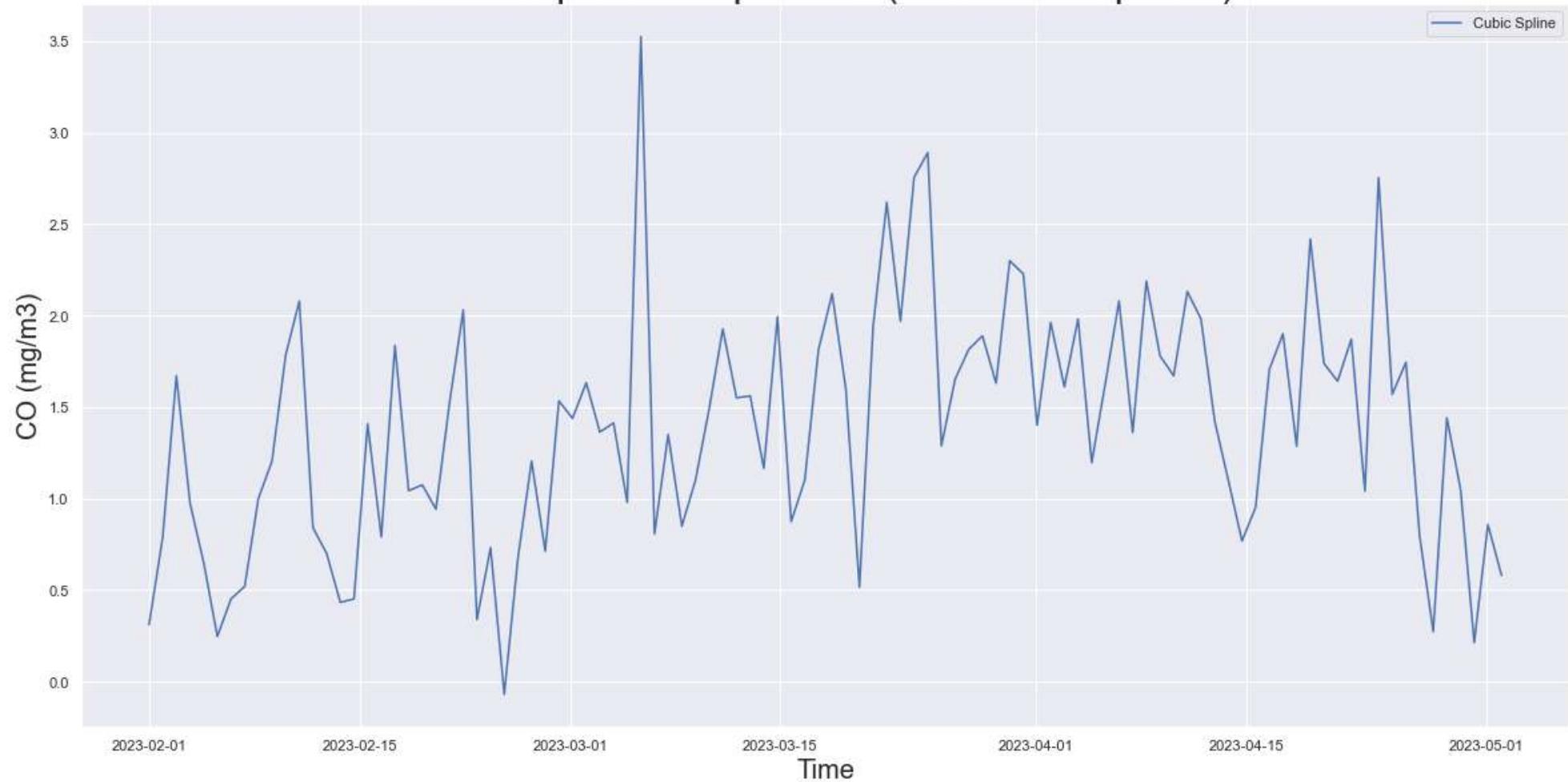


```
In [163]: cubicspline(df_n,pollutant)
```

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



Missing values are less and data is more or less across the mean and does not seem to be effected much by the outliers, mean filling seems to work fine. Cubic and quadratic interpolations and splines introduce negative values which seems to be impractical. Any of the linear interpolation or mean filling will serve the purpose.

In [164]: `df_linear_f[pollutant].isnull().sum()`

Out[164]: 0

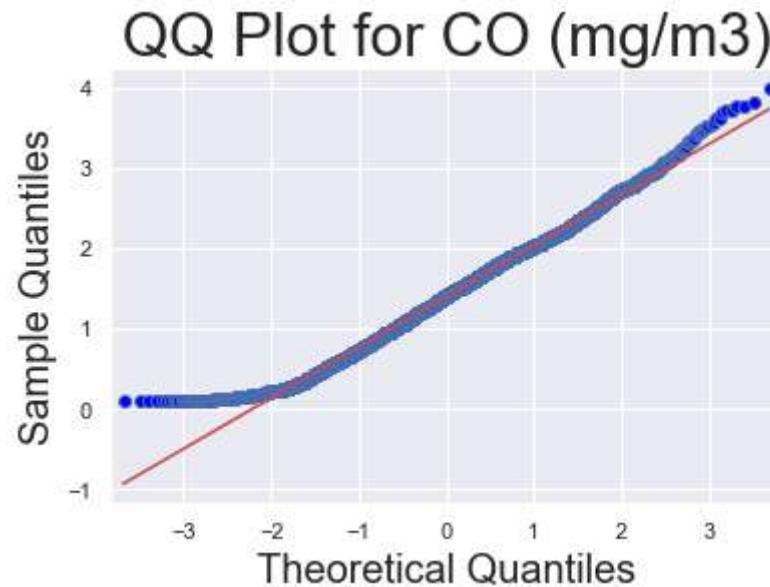
In [165]: `df_mean[pollutant].isnull().sum()`

Out[165]: 0

```
In [166]: df_new[pollutant] = df_linear_f[pollutant]
```

```
In [167]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>



```
In [168]: df_new.head()
```

Out[168]:

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)	CO (mg/m3)
--	-----------------------------------	------------------------------------	---------------------------------	----------------------------------	-----------	------------

Time

2023-02-01 00:00:00	95.0	35.0	18.1	90.1	56.2	0.31
2023-02-01 00:15:00	95.0	35.0	18.1	88.0	55.1	0.33
2023-02-01 00:30:00	95.0	35.0	18.1	87.7	55.2	0.38
2023-02-01 00:45:00	122.0	34.0	18.1	88.9	55.7	0.38
2023-02-01 01:00:00	122.0	34.0	18.1	90.0	55.8	0.38

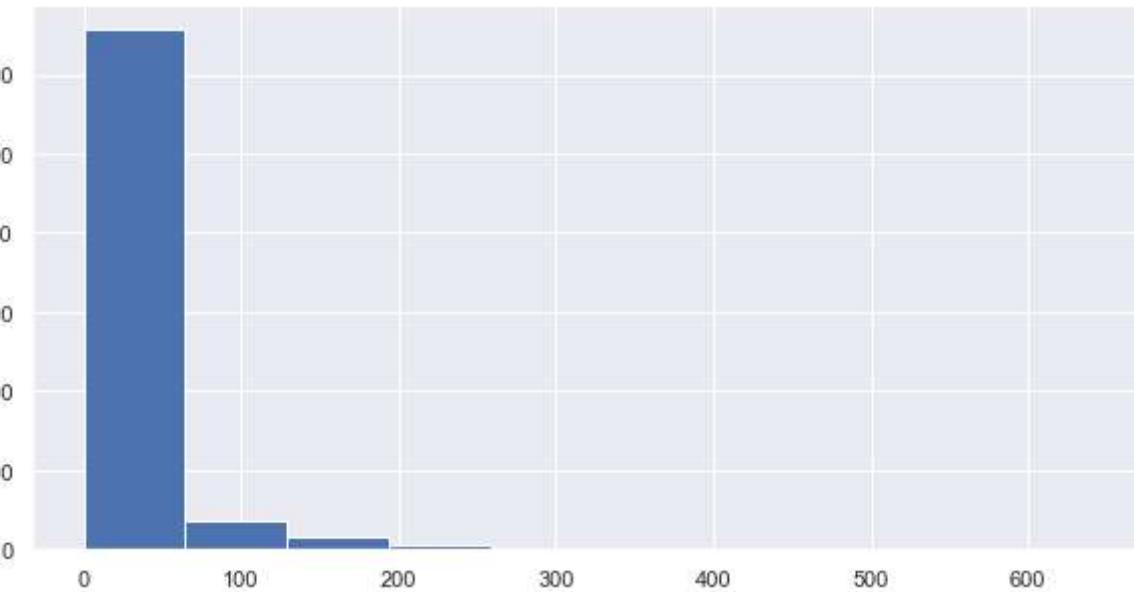
Analysis for "SO2 ($\mu\text{g}/\text{m}^3$)"

```
In [169]: pollutant = 'SO2 ( $\mu\text{g}/\text{m}^3$ )'
```

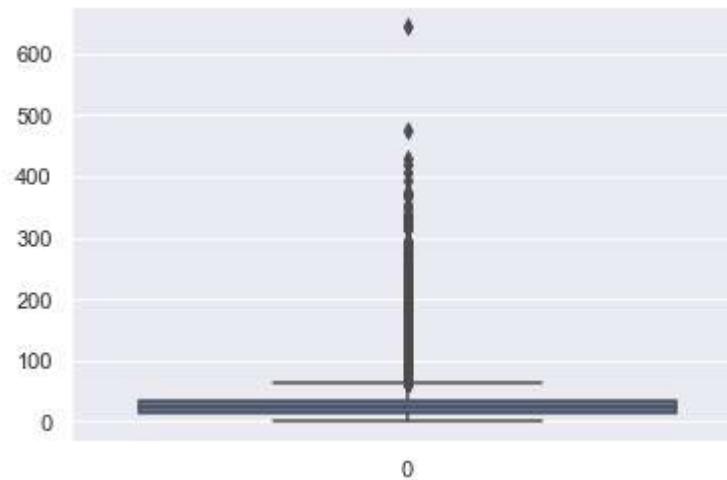
```
In [170]: df[pollutant].describe()
```

```
Out[170]: count    7189.000000
mean      34.232731
std       39.452131
min       0.100000
25%      16.100000
50%      25.300000
75%      35.200000
max     645.600000
Name: SO2 ( $\mu\text{g}/\text{m}^3$ ), dtype: float64
```

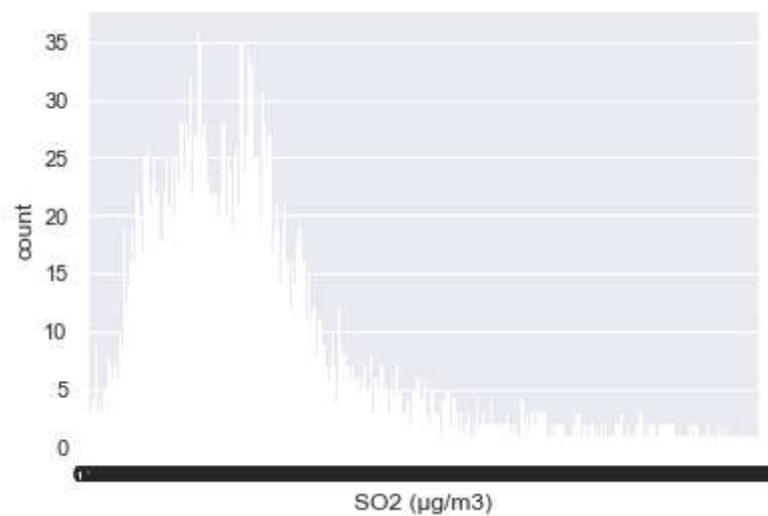
```
In [171]: histogram_plot(df_n,pollutant)
```



```
In [172]: boxplot_plot(df_n,pollutant)
```

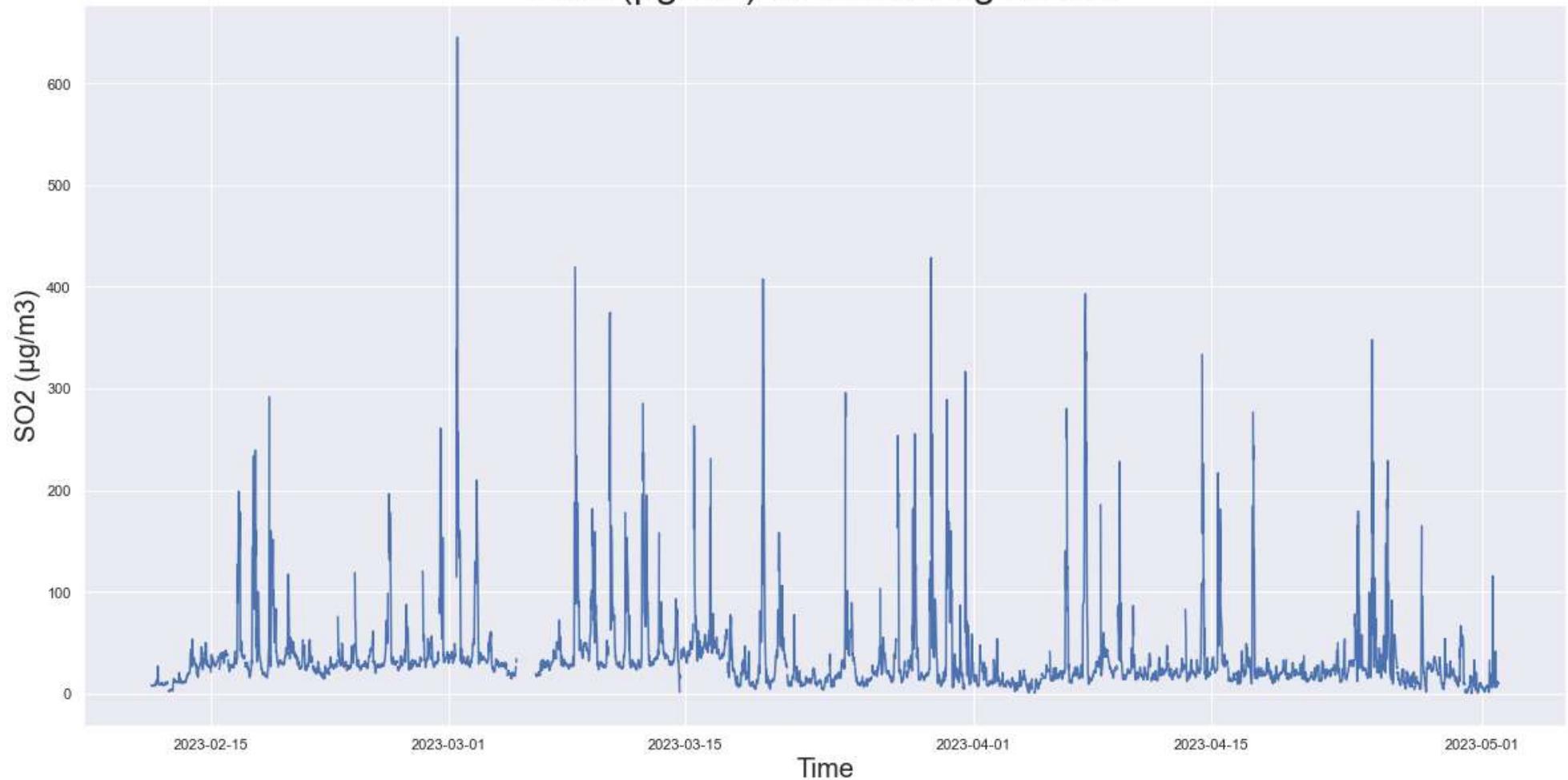


```
In [173]: countplot_plot(df_n,pollutant)
```



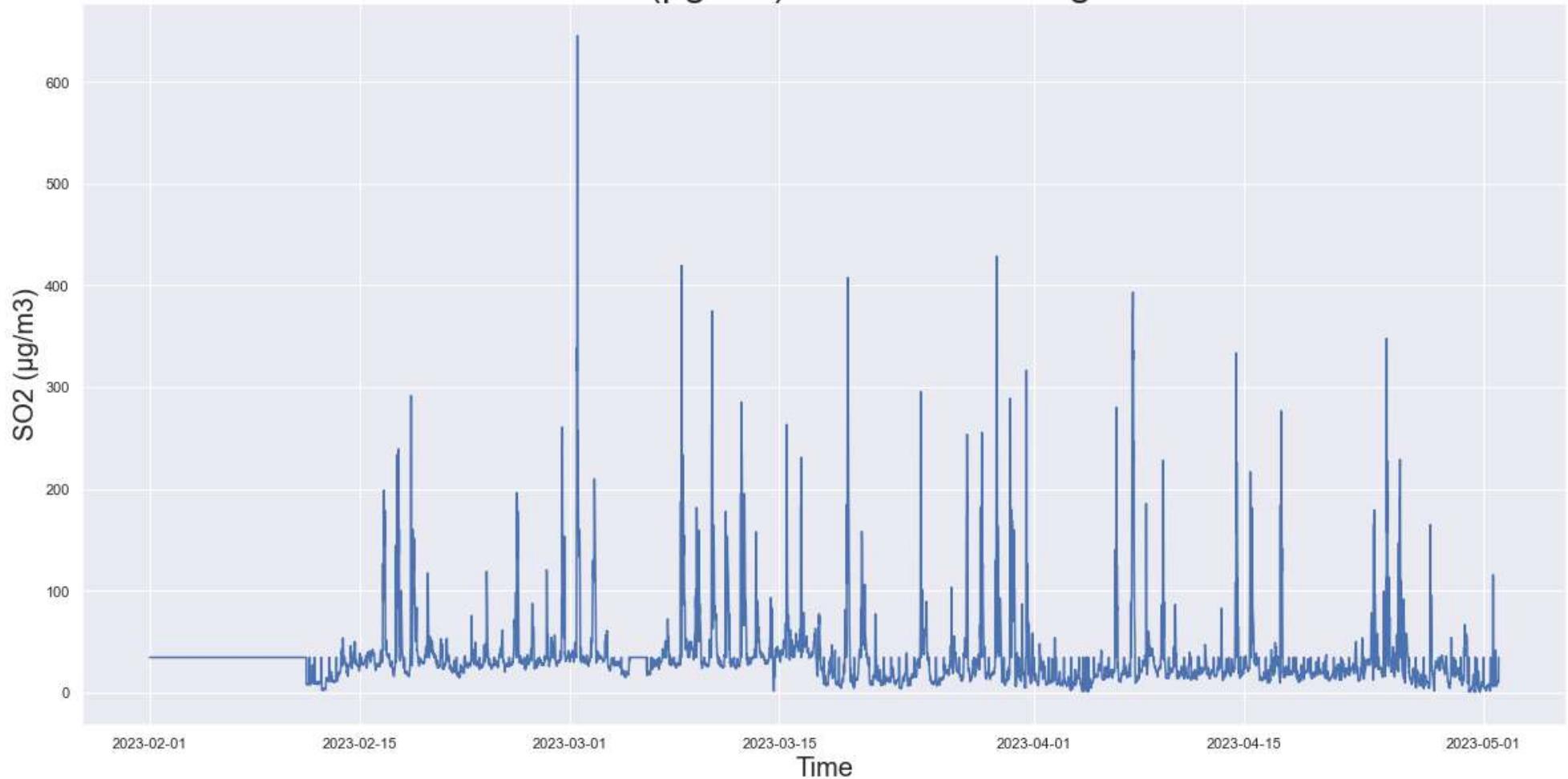
```
In [174]: simple_time_plot(df_n,pollutant,"With missing values")
```

SO2 ($\mu\text{g}/\text{m}^3$) With missing values



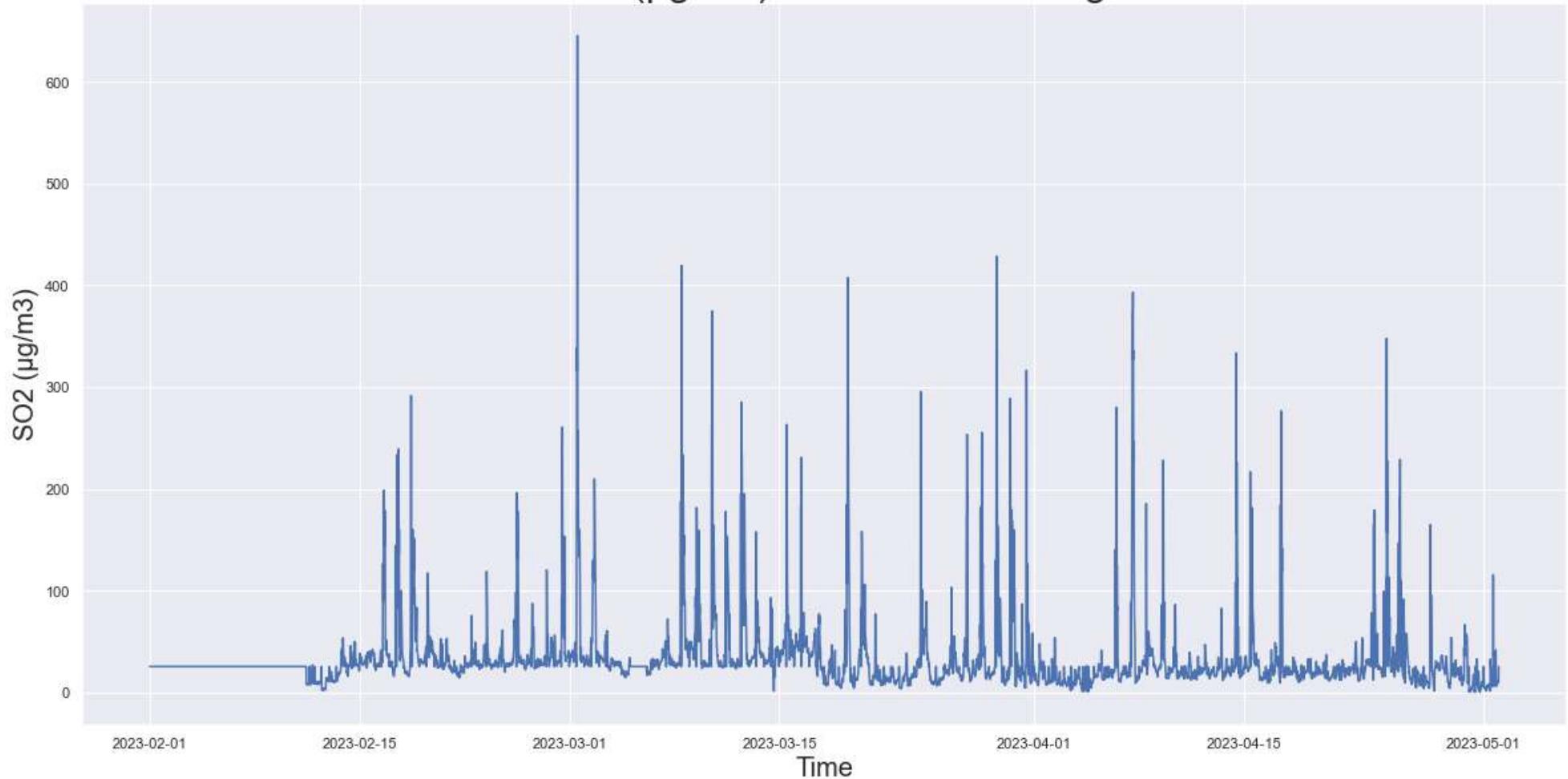
```
In [175]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

SO2 ($\mu\text{g}/\text{m}^3$) after mean filling



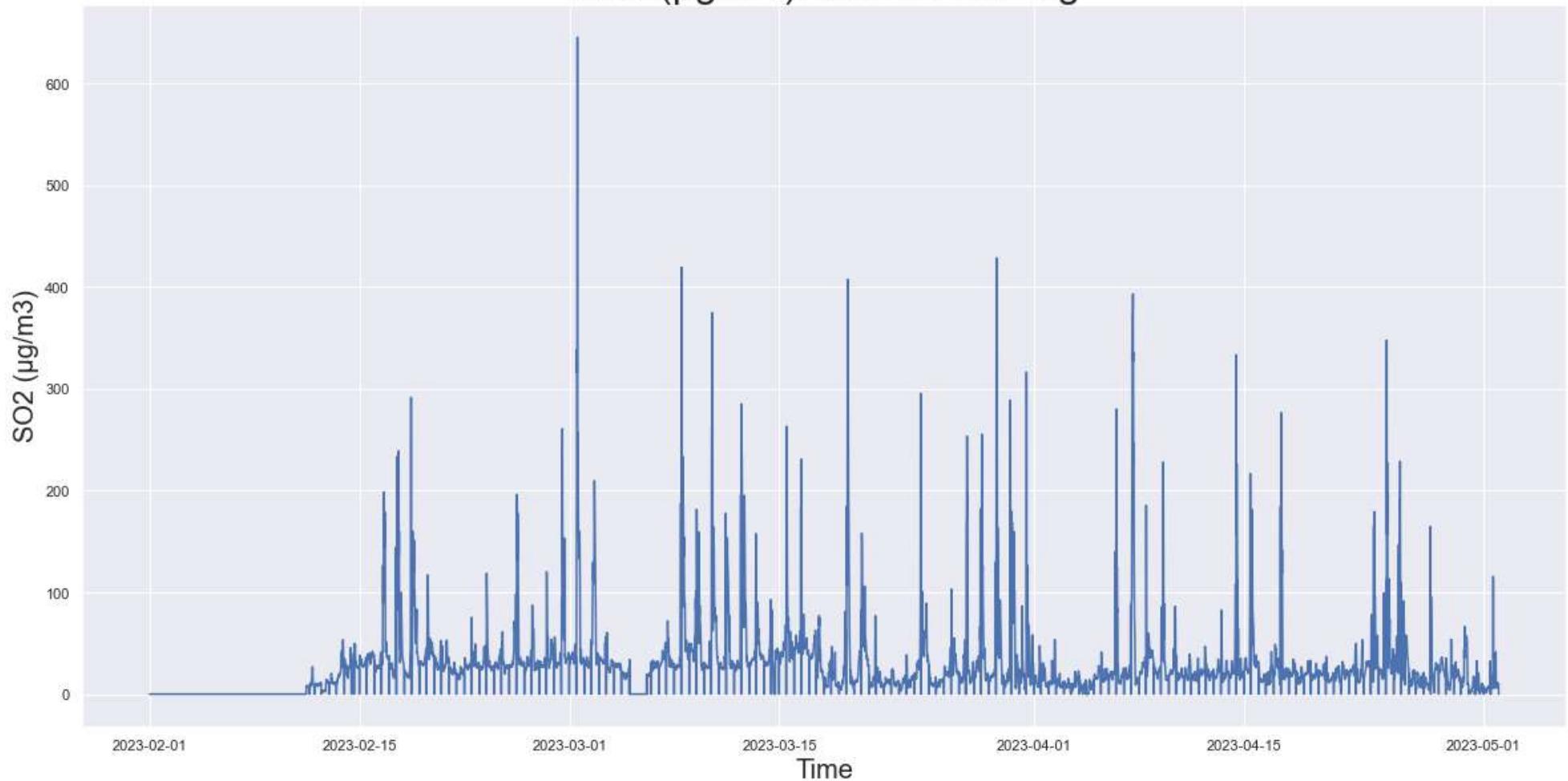
```
In [176]: simple_time_plot(df_median,pollutant,"after median filling")
```

SO2 ($\mu\text{g}/\text{m}^3$) after median filling



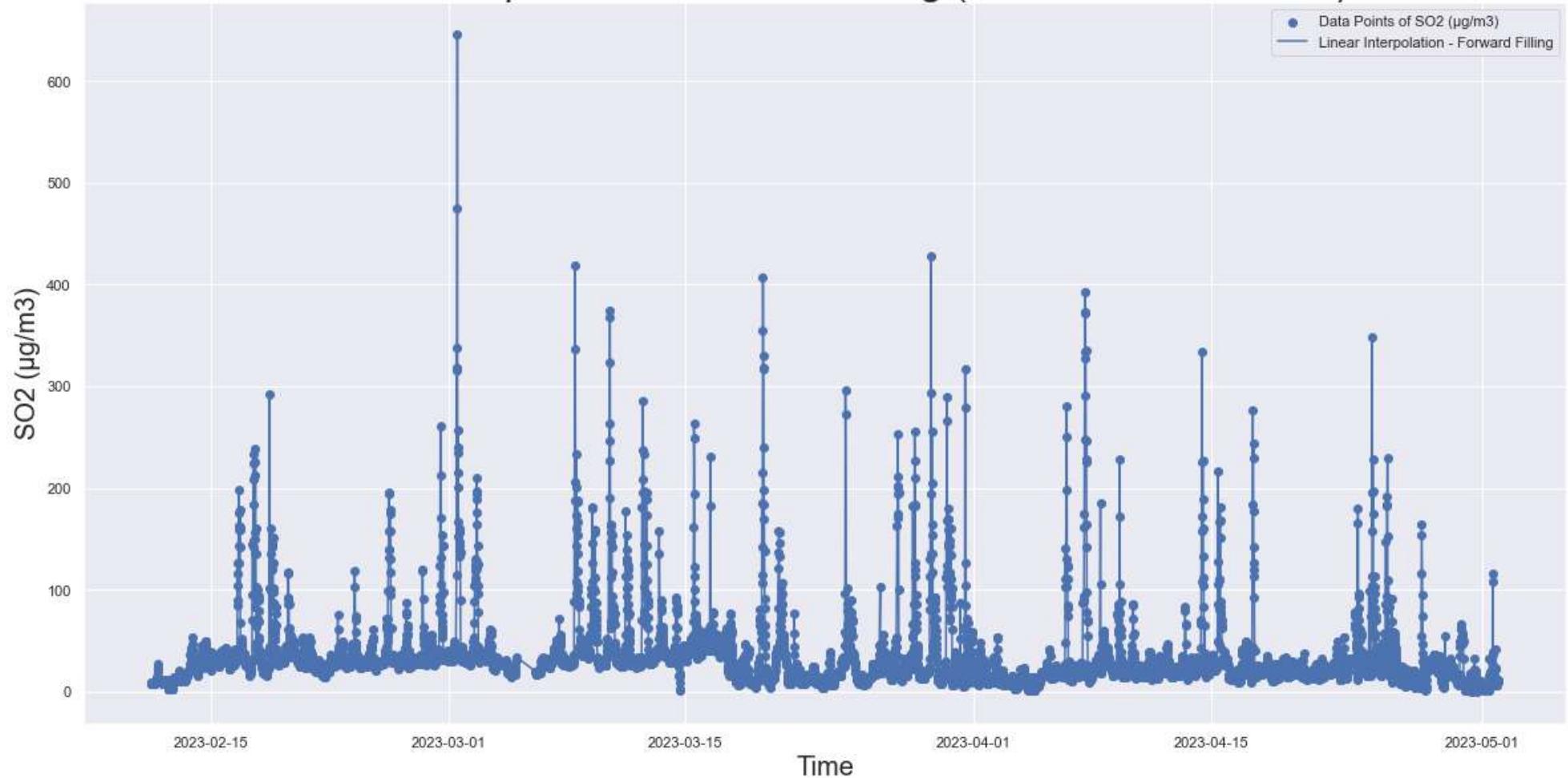
```
In [177]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

SO2 ($\mu\text{g}/\text{m}^3$) after zero filling

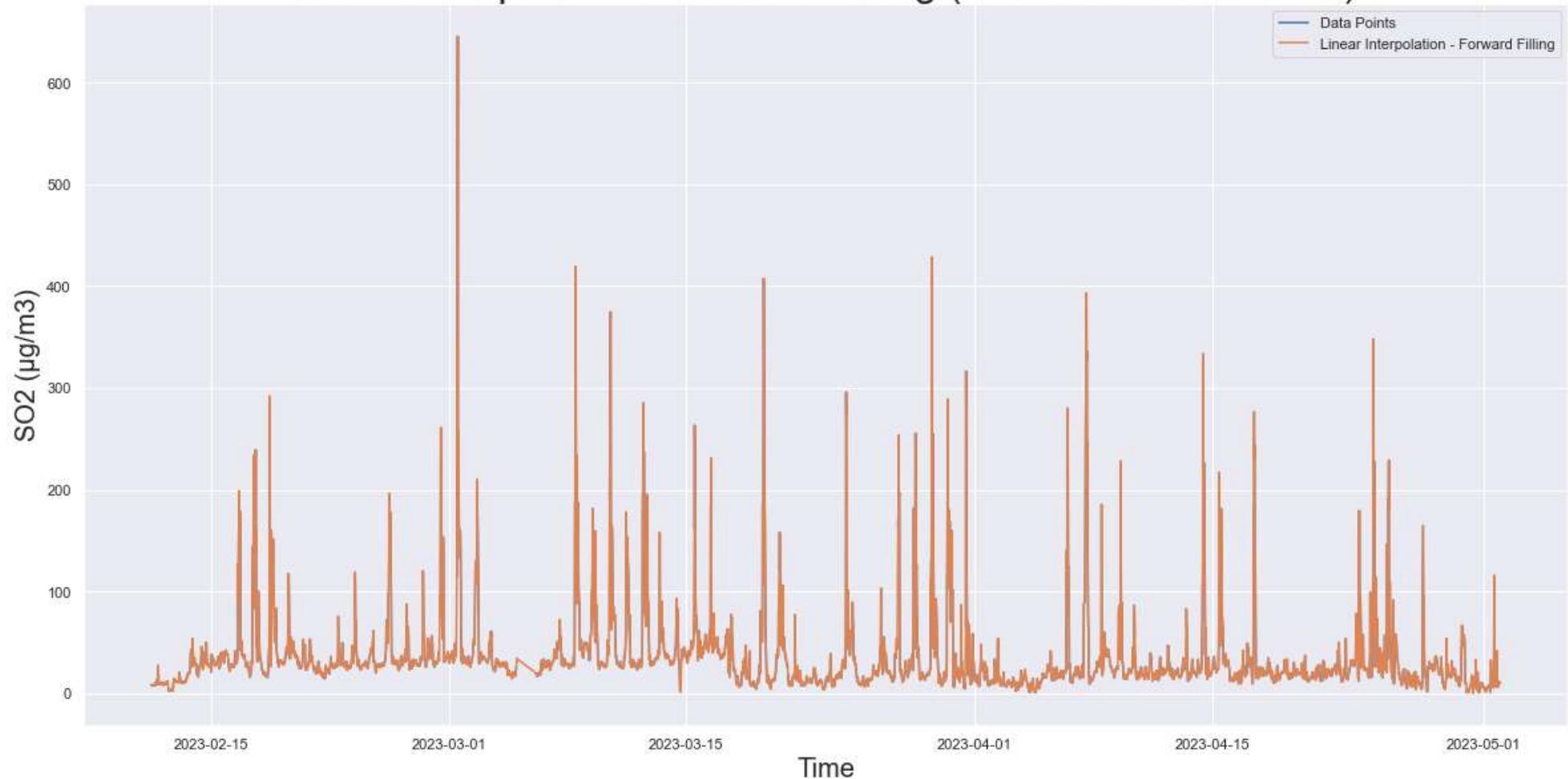


```
In [178]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```

Linear Interpolation - Forward Filling (Data Points Included)

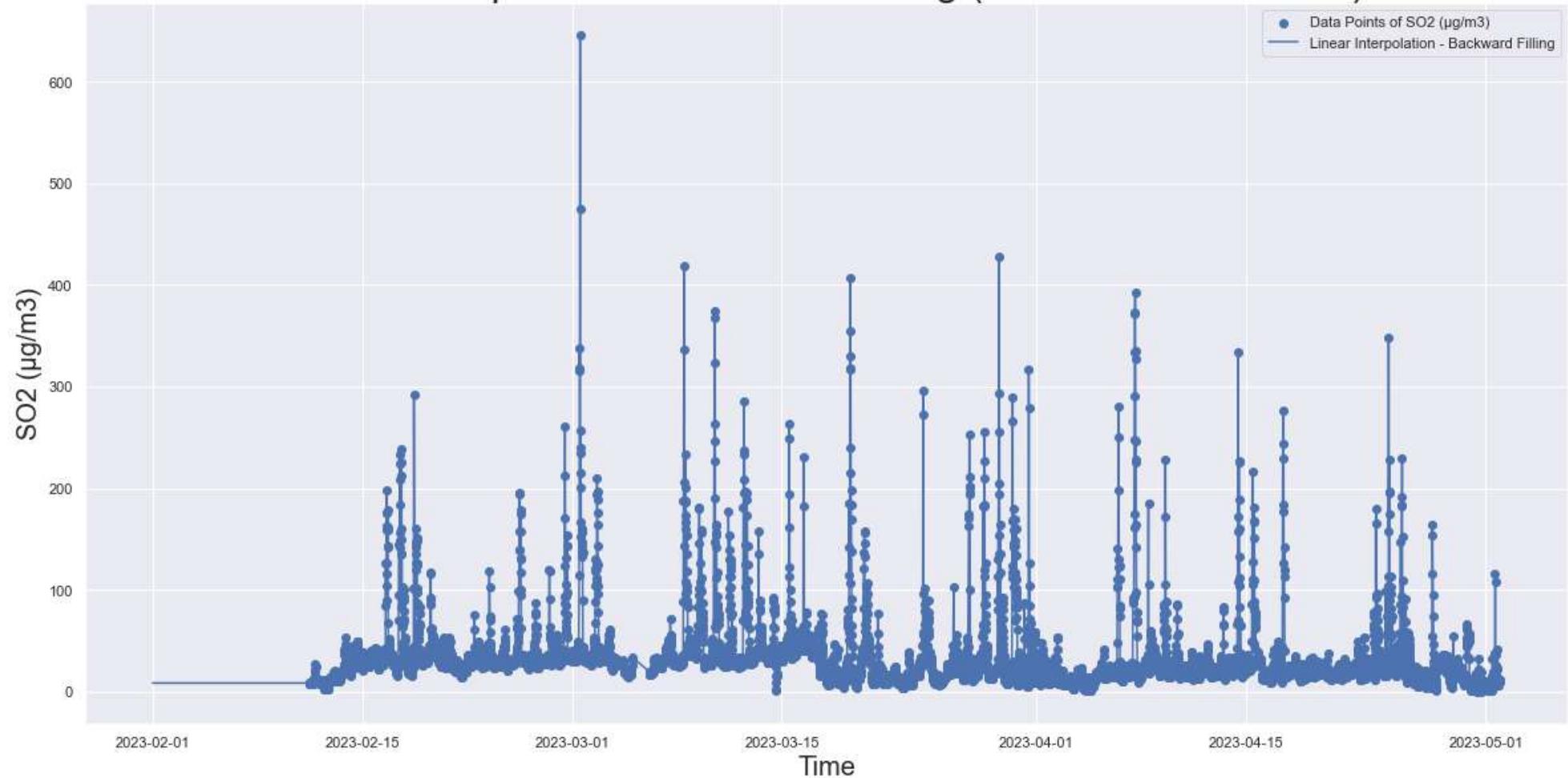


Linear Interpolation - Forward Filling (Data Points Excluded)

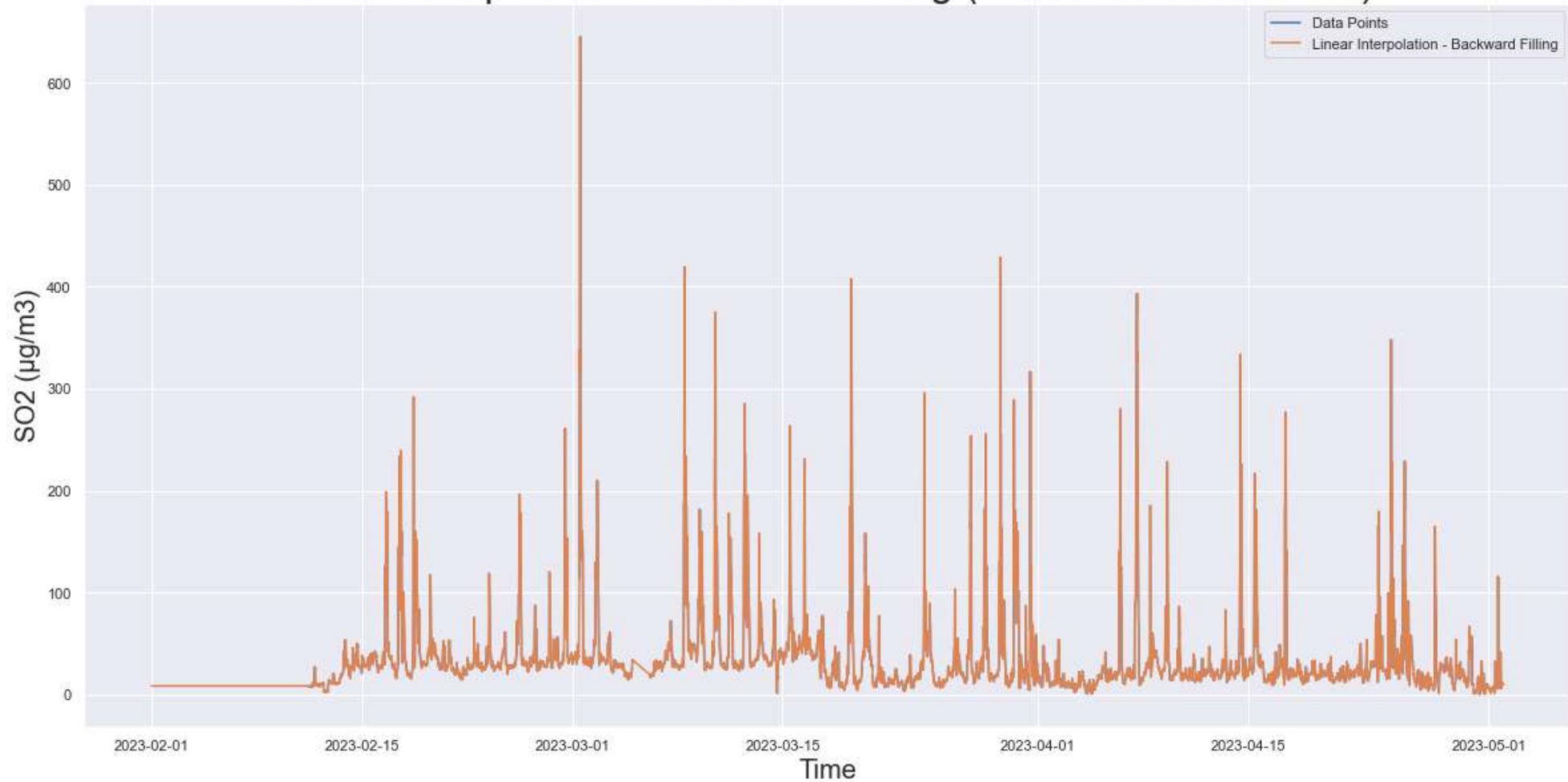


```
In [179]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

Linear Interpolation - Backward Filling (Data Points Included)

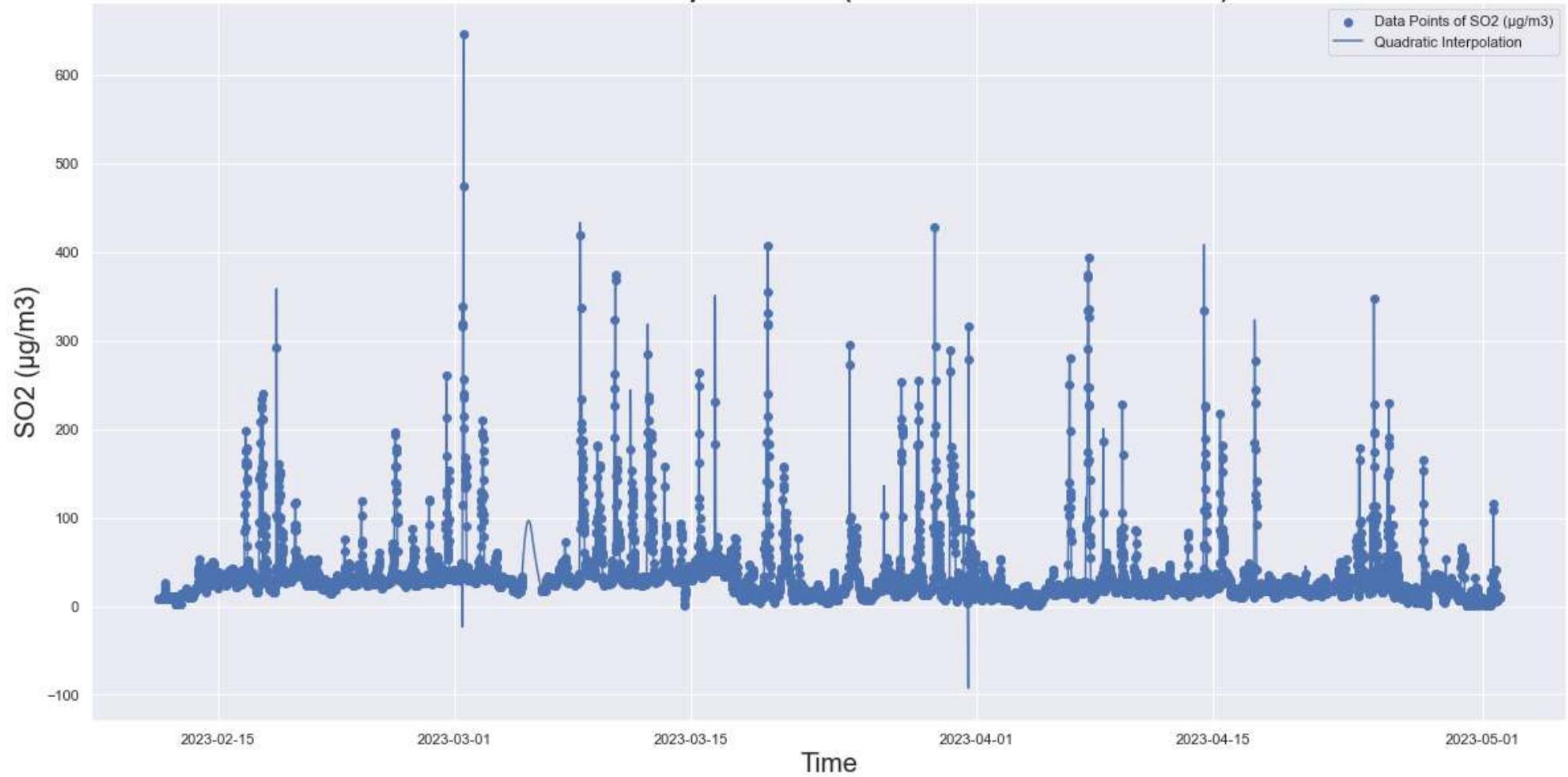


Linear Interpolation - Backward Filling (Data Points Excluded)

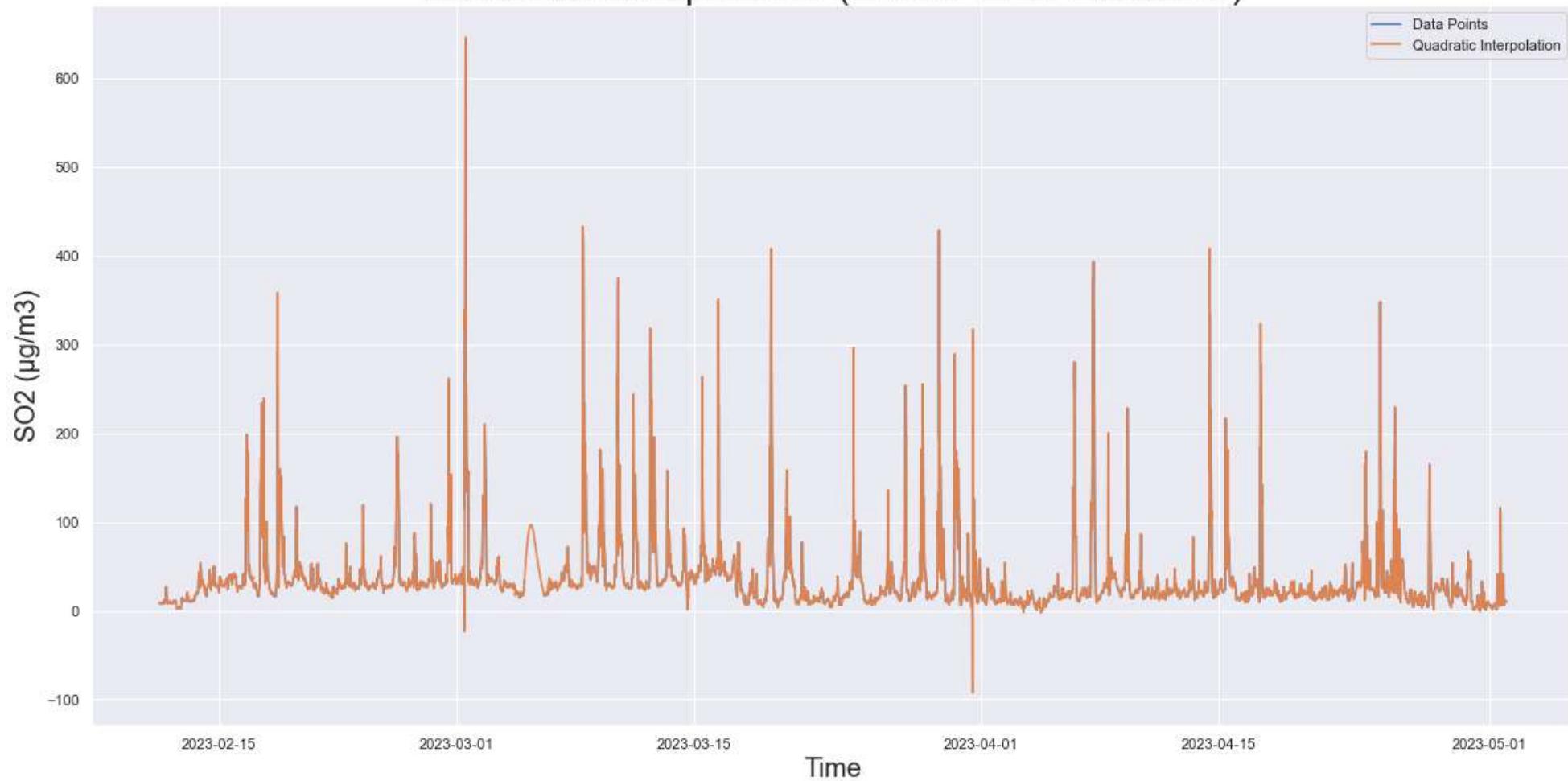


```
In [180]: interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")
```

Quadratic Interpolation (Data Points Included)

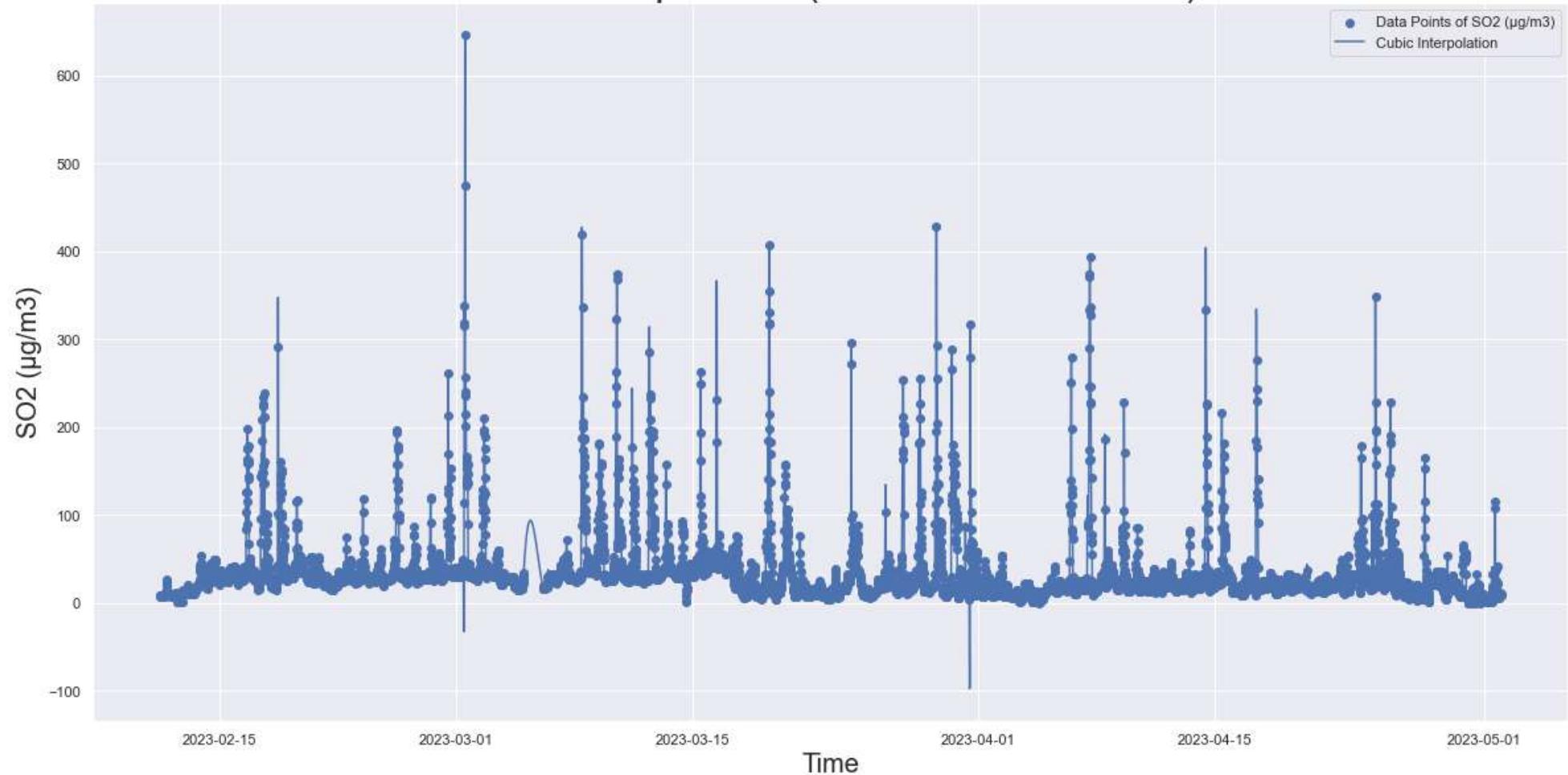


Quadratic Interpolation (Data Points Excluded)

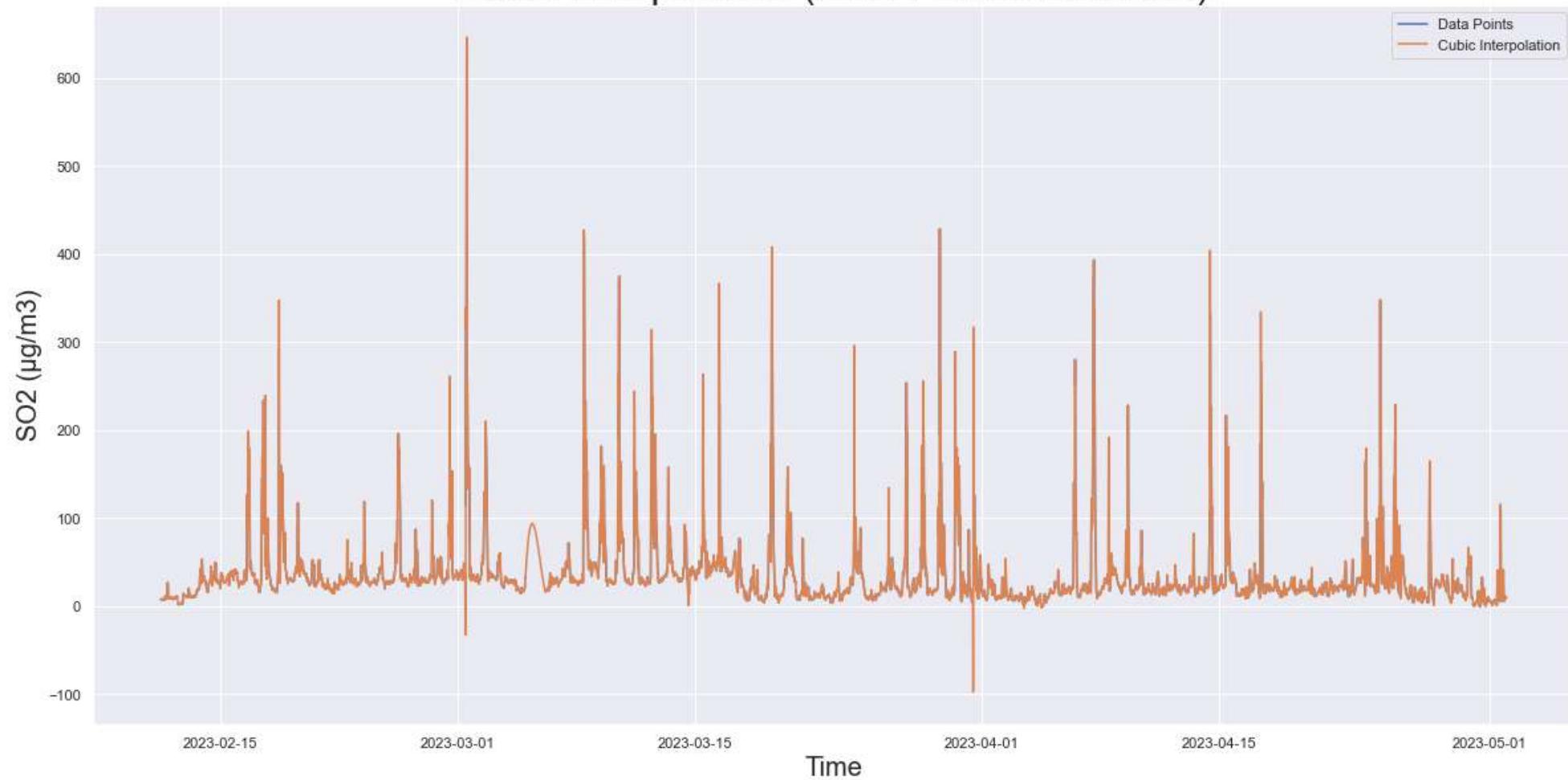


```
In [181]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```

Cubic Interpolation (Data Points Included)

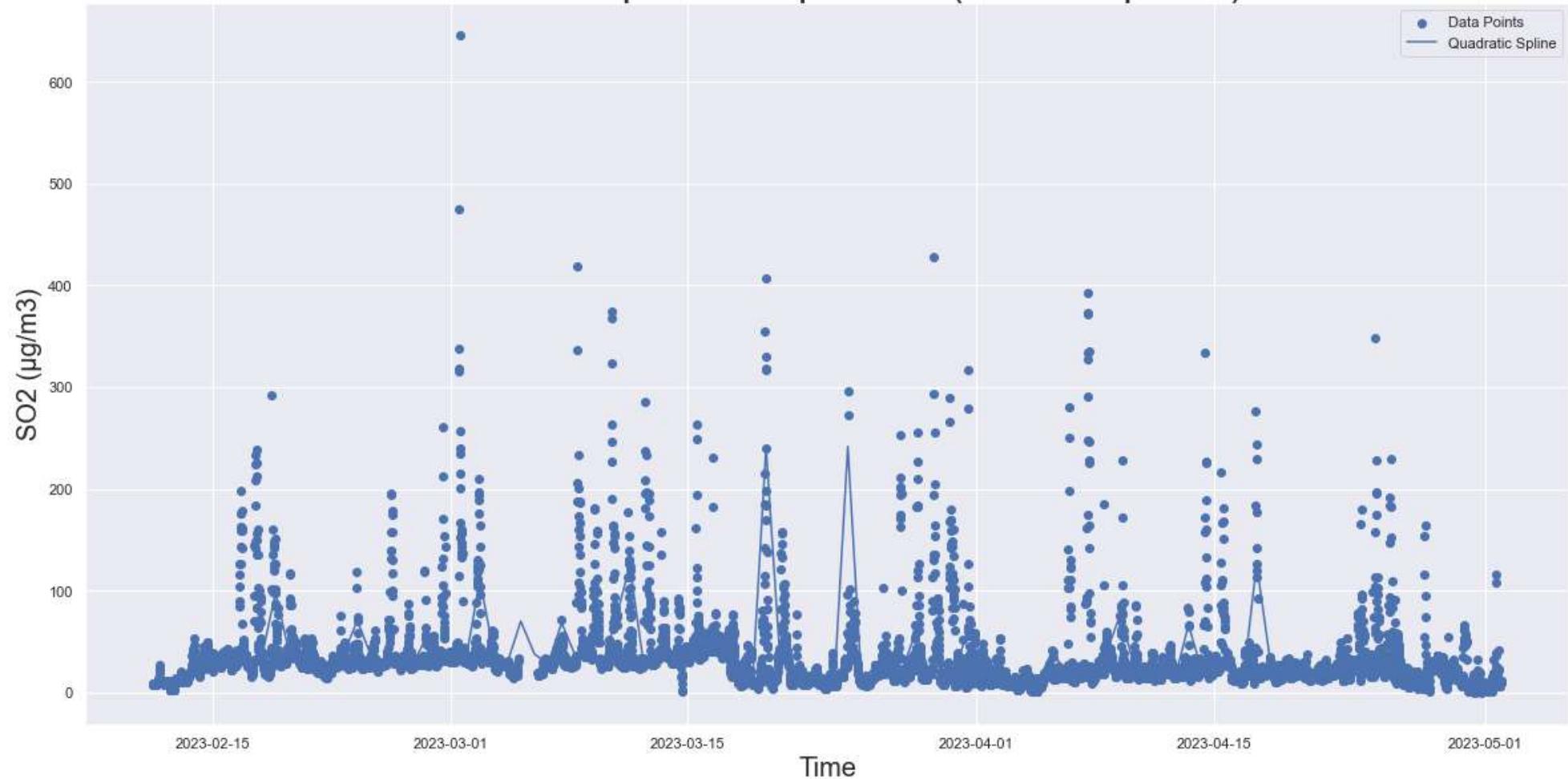


Cubic Interpolation (Data Points Excluded)

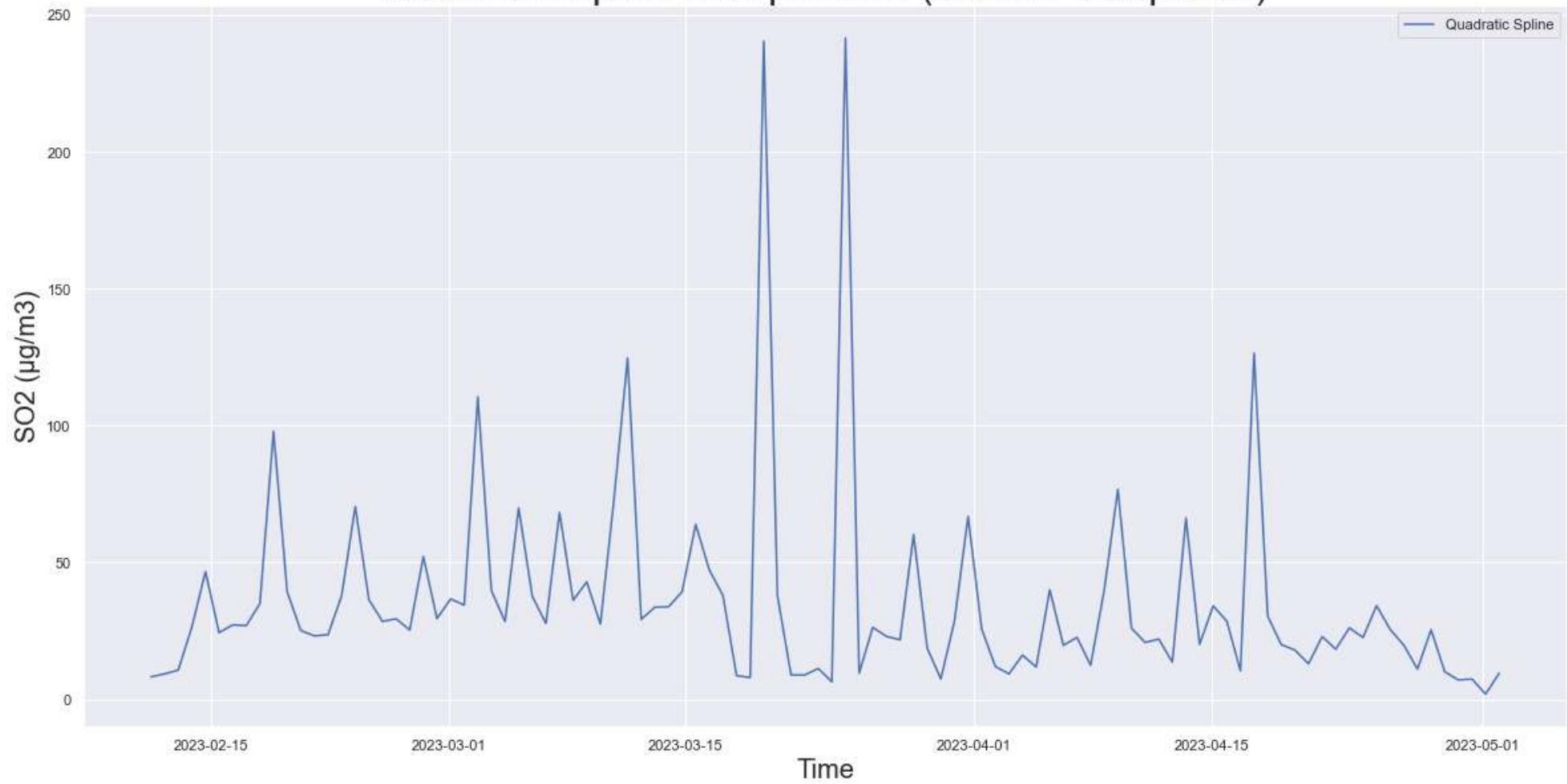


```
In [182]: quadraticspline(df_n,pollutant)
```

Quadratic Spline Interpolation (with datapoints)

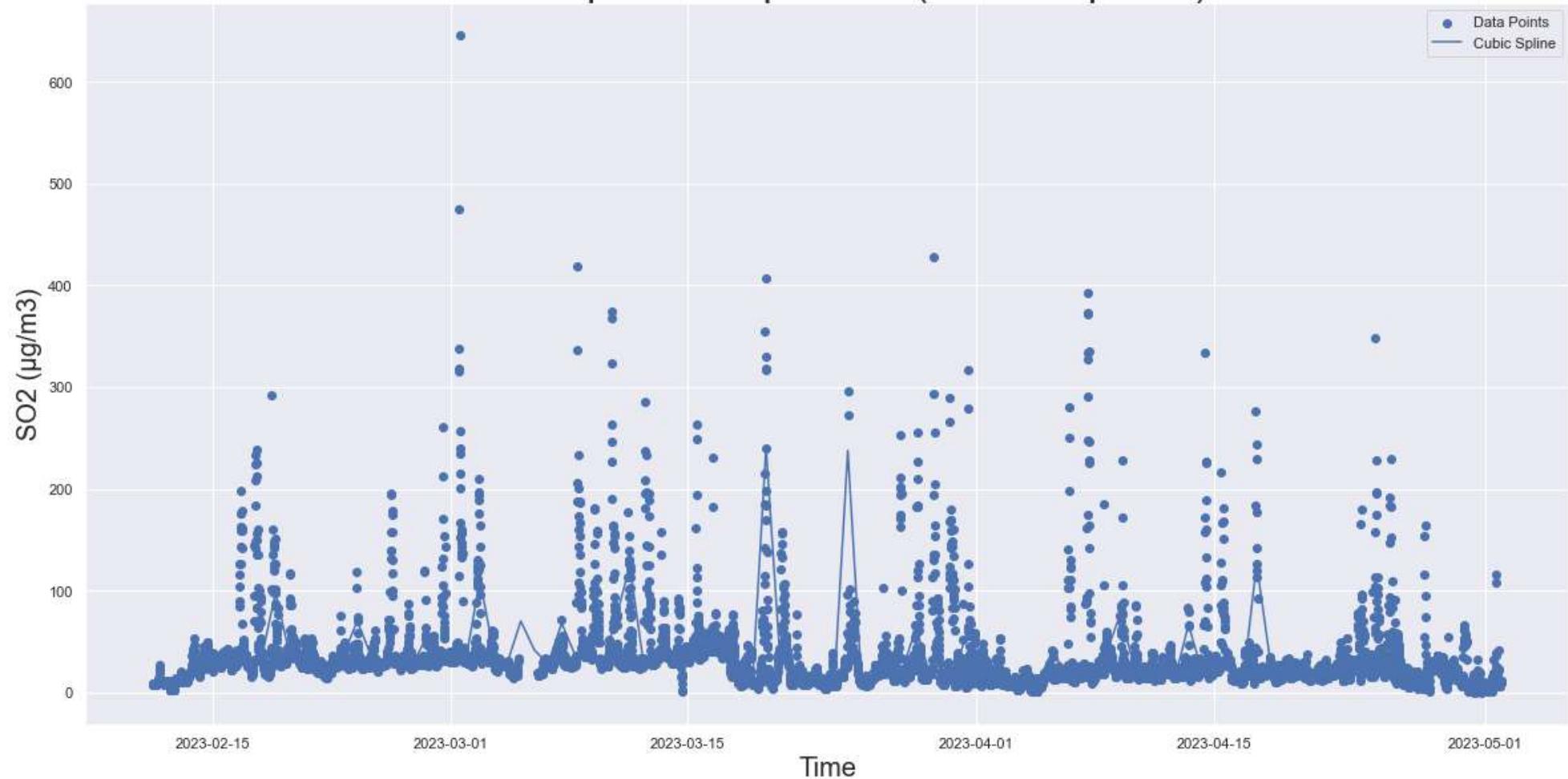


Quadratic Spline Interpolation (without datapoints)

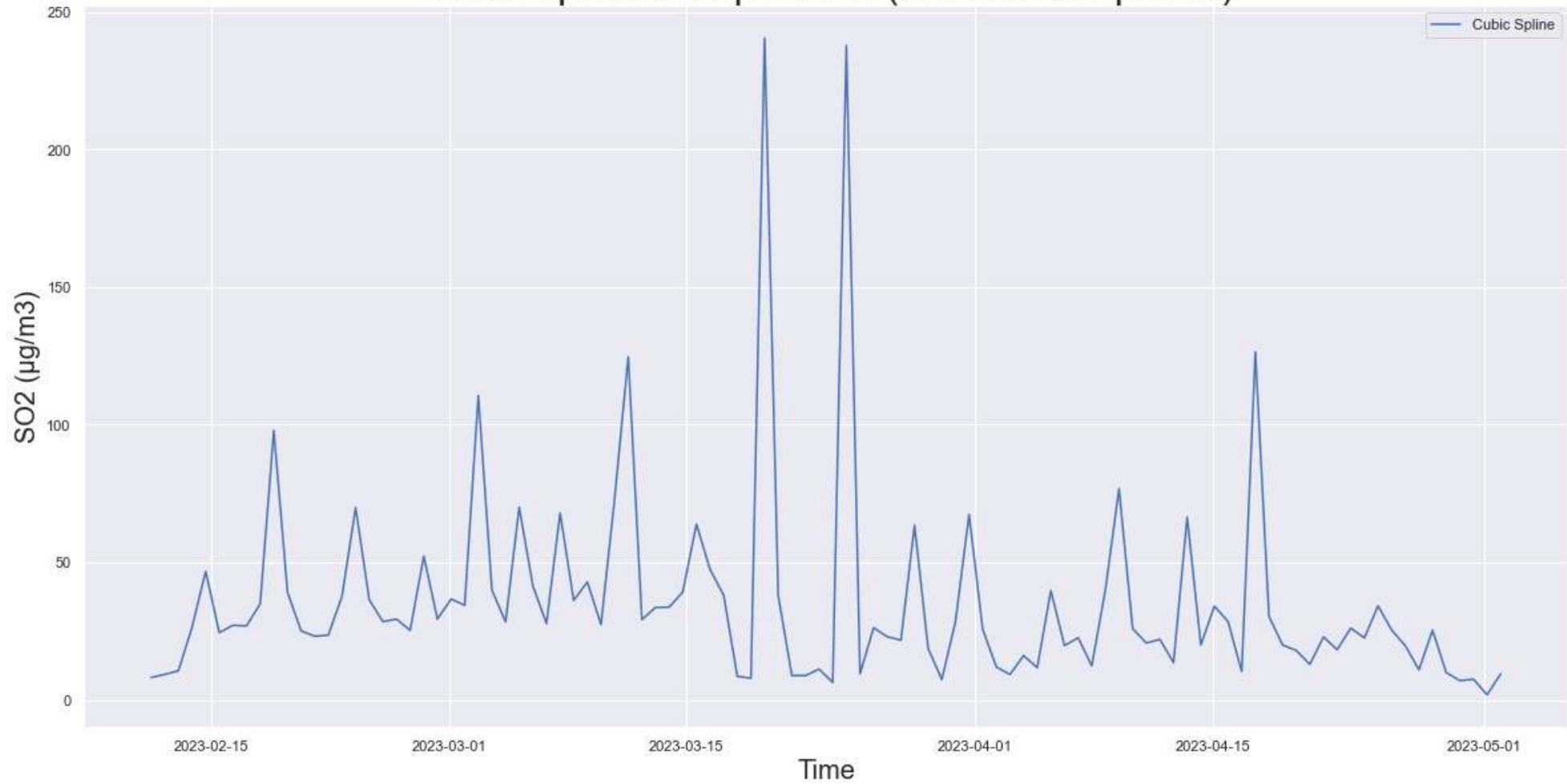


```
In [183]: cubicspline(df_n,pollutant)
```

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



Cubic and Quadratic interpolations and spline introduce negative values which is practically not feasible. Therefore linear interpolation works best. Mean is affected by the outliers as is evident from histogram and boxplots.

In [184]: `df_linear_f[pollutant].isnull().sum()`

Out[184]: 1004

In [185]: `df_linear_b[pollutant].isnull().sum()`

Out[185]: 1

```
In [186]: df_linear_b[pollutant].fillna(df_linear_f[pollutant].mean(), inplace=True)
```

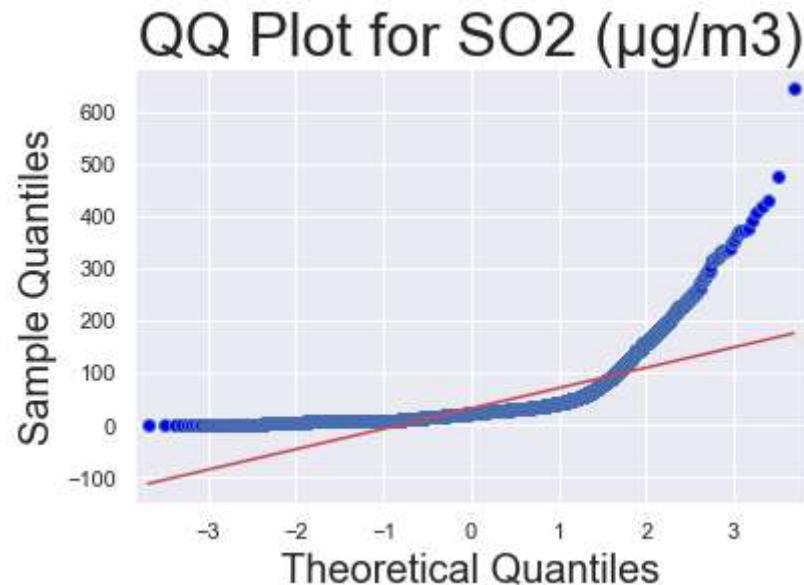
```
In [187]: df_linear_b[pollutant].isnull().sum()
```

```
Out[187]: 0
```

```
In [188]: df_new[pollutant] = df_linear_b[pollutant]
```

```
In [189]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>



```
In [190]: df_new.head()
```

```
Out[190]:
```

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)	CO (mg/m^3)	SO2 ($\mu\text{g}/\text{m}^3$)
--	-----------------------------------	------------------------------------	---------------------------------	----------------------------------	-----------	-------------------------------	----------------------------------

Time

2023-02-01 00:00:00	95.0	35.0	18.1	90.1	56.2	0.31	8.2
2023-02-01 00:15:00	95.0	35.0	18.1	88.0	55.1	0.33	8.2
2023-02-01 00:30:00	95.0	35.0	18.1	87.7	55.2	0.38	8.2
2023-02-01 00:45:00	122.0	34.0	18.1	88.9	55.7	0.38	8.2
2023-02-01 01:00:00	122.0	34.0	18.1	90.0	55.8	0.38	8.2

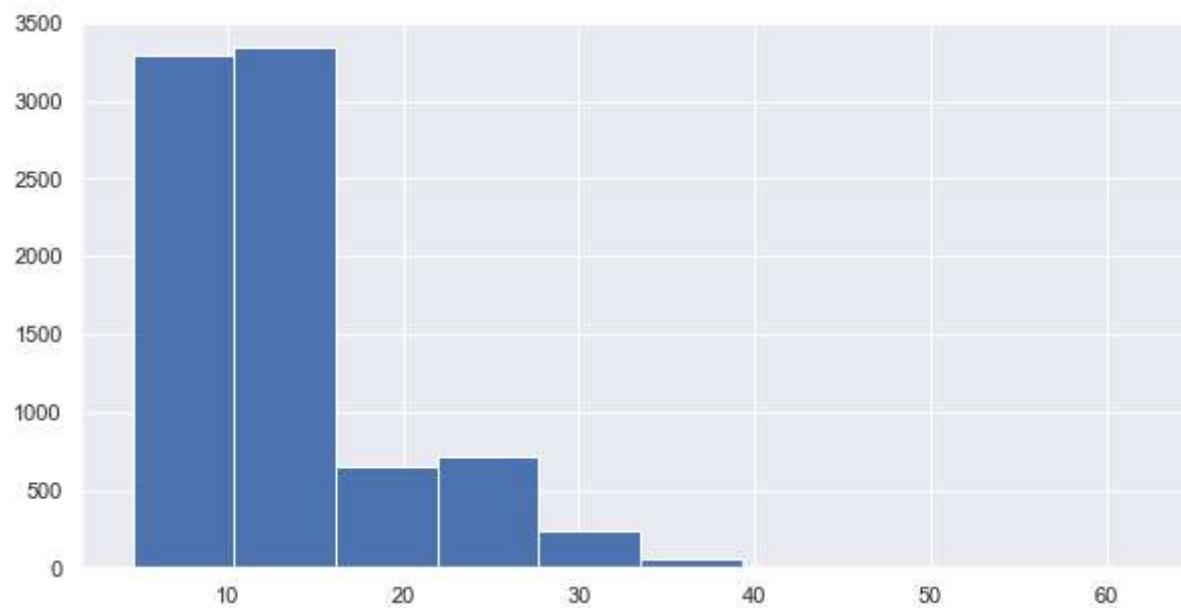
Analysis for "NH3 ($\mu\text{g}/\text{m}^3$)"

```
In [191]: pollutant = 'NH3 ( $\mu\text{g}/\text{m}^3$ )'
```

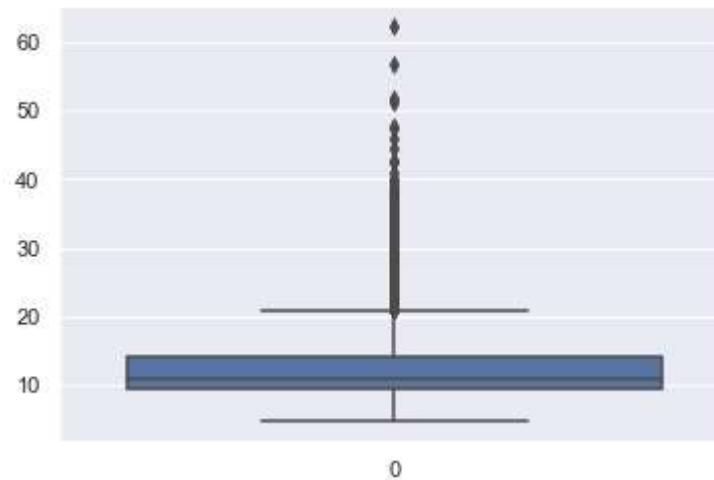
```
In [192]: df[pollutant].describe()
```

```
Out[192]: count    8314.000000
mean      13.242663
std       6.151034
min      4.600000
25%     9.400000
50%    11.000000
75%    14.000000
max    62.400000
Name: NH3 ( $\mu\text{g}/\text{m}^3$ ), dtype: float64
```

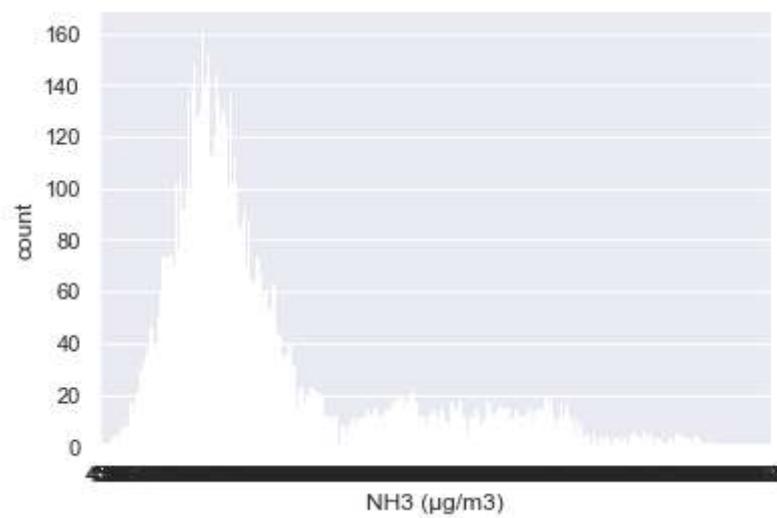
```
In [193]: histogram_plot(df_n,pollutant)
```



```
In [194]: boxplot_plot(df_n,pollutant)
```

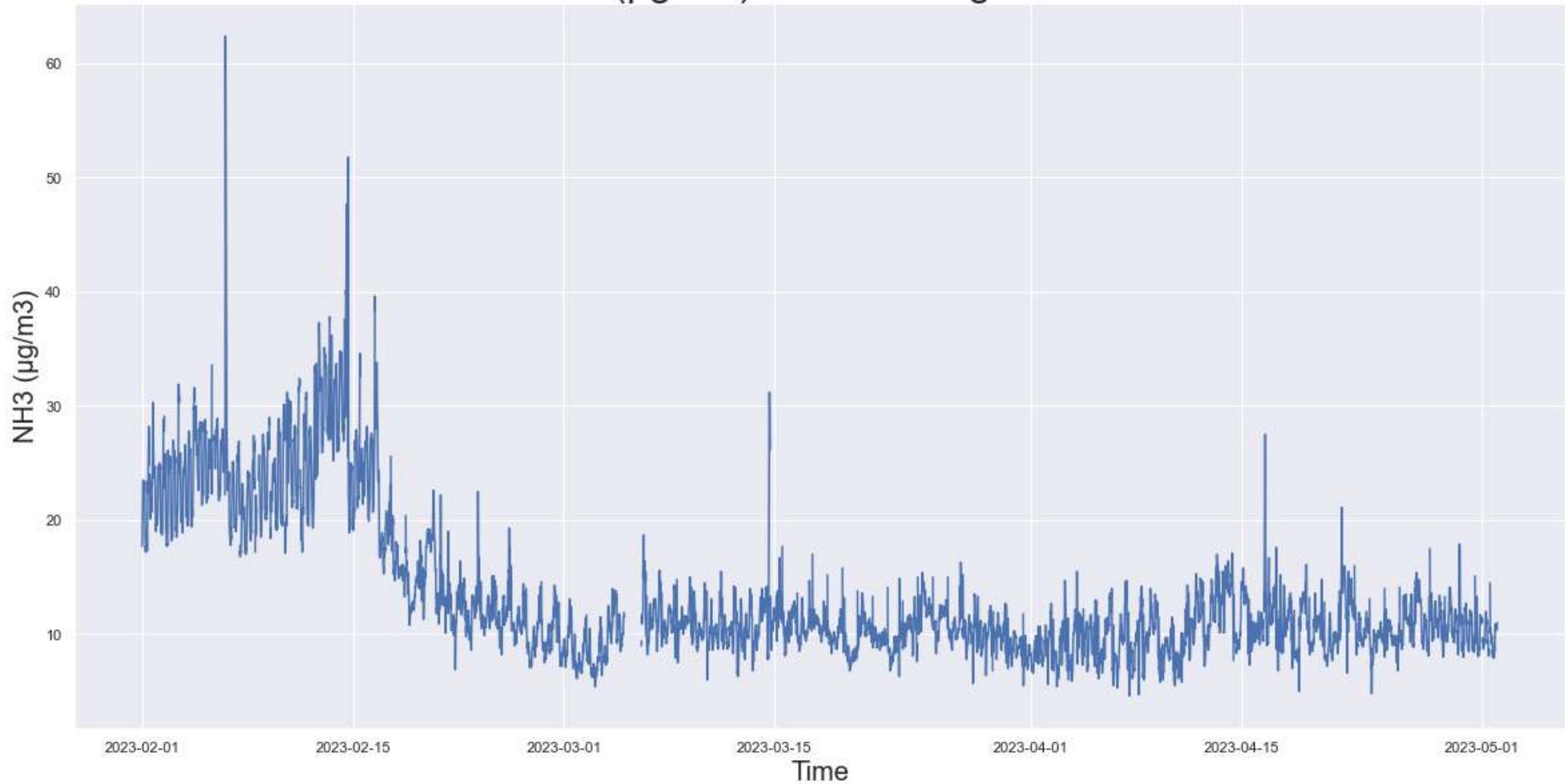


```
In [195]: countplot_plot(df_n,pollutant)
```



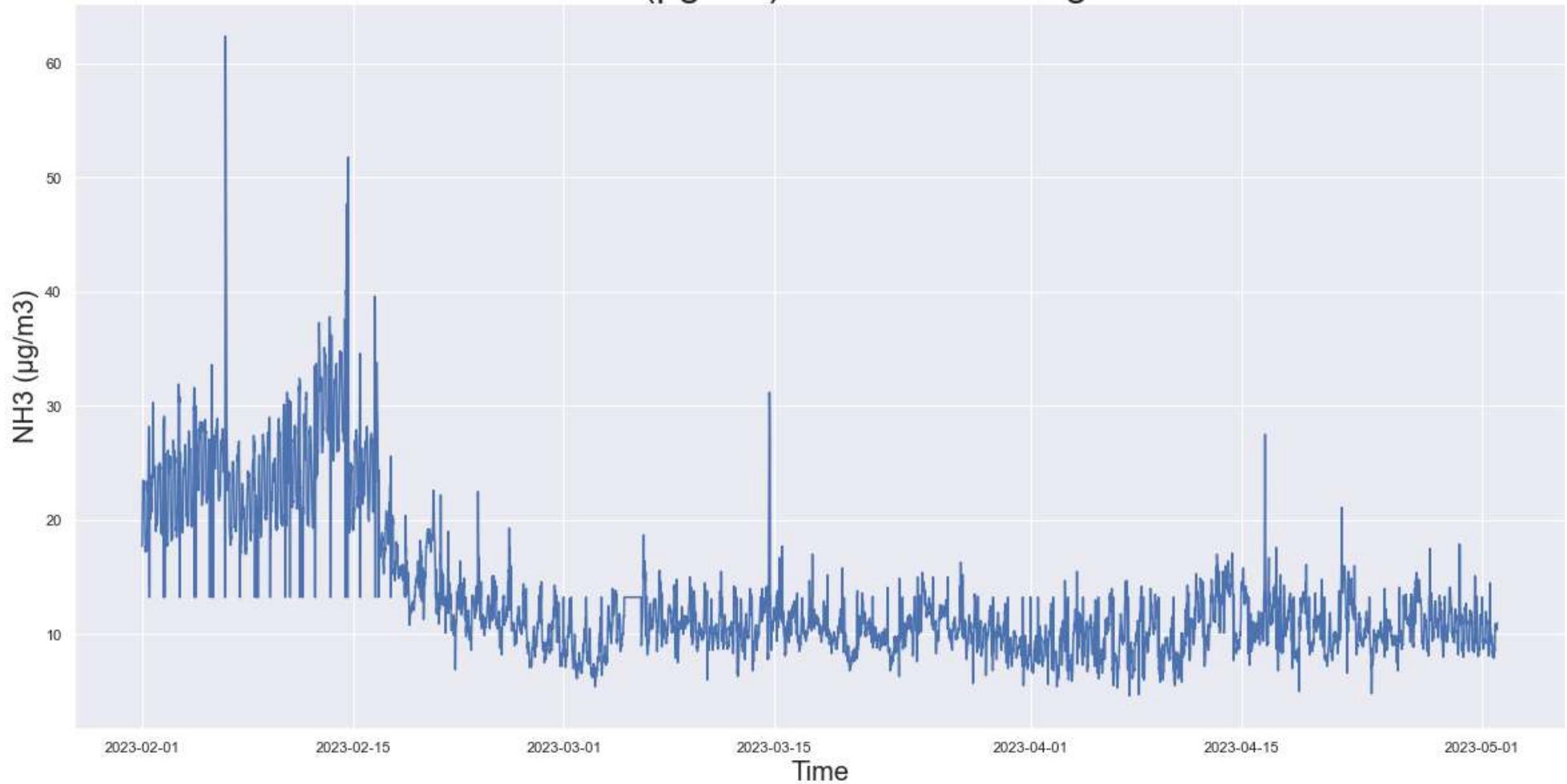
```
In [196]: simple_time_plot(df_n,pollutant,"With missing values")
```

NH3 ($\mu\text{g}/\text{m}^3$) With missing values



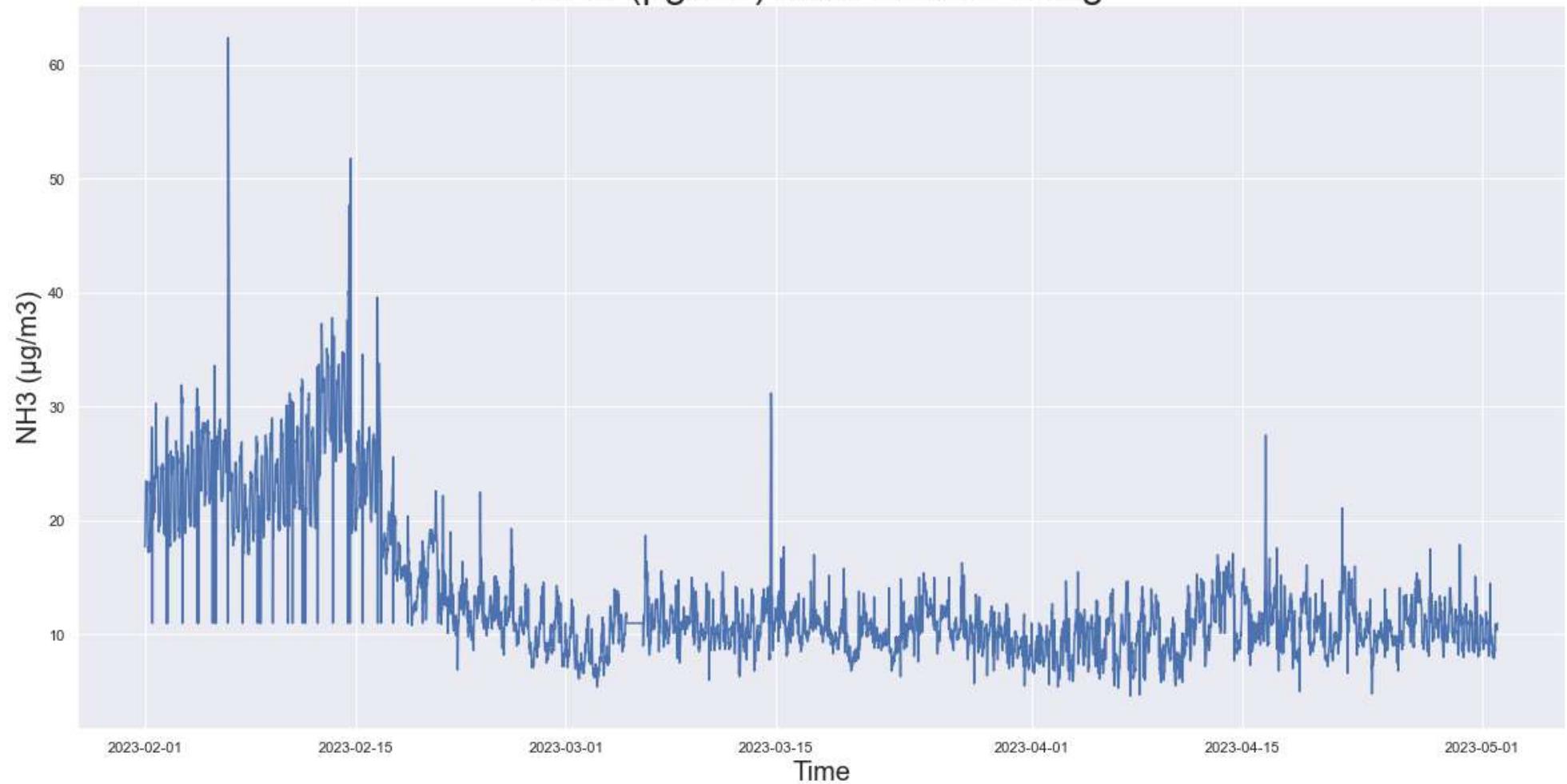
```
In [197]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

NH3 ($\mu\text{g}/\text{m}^3$) after mean filling



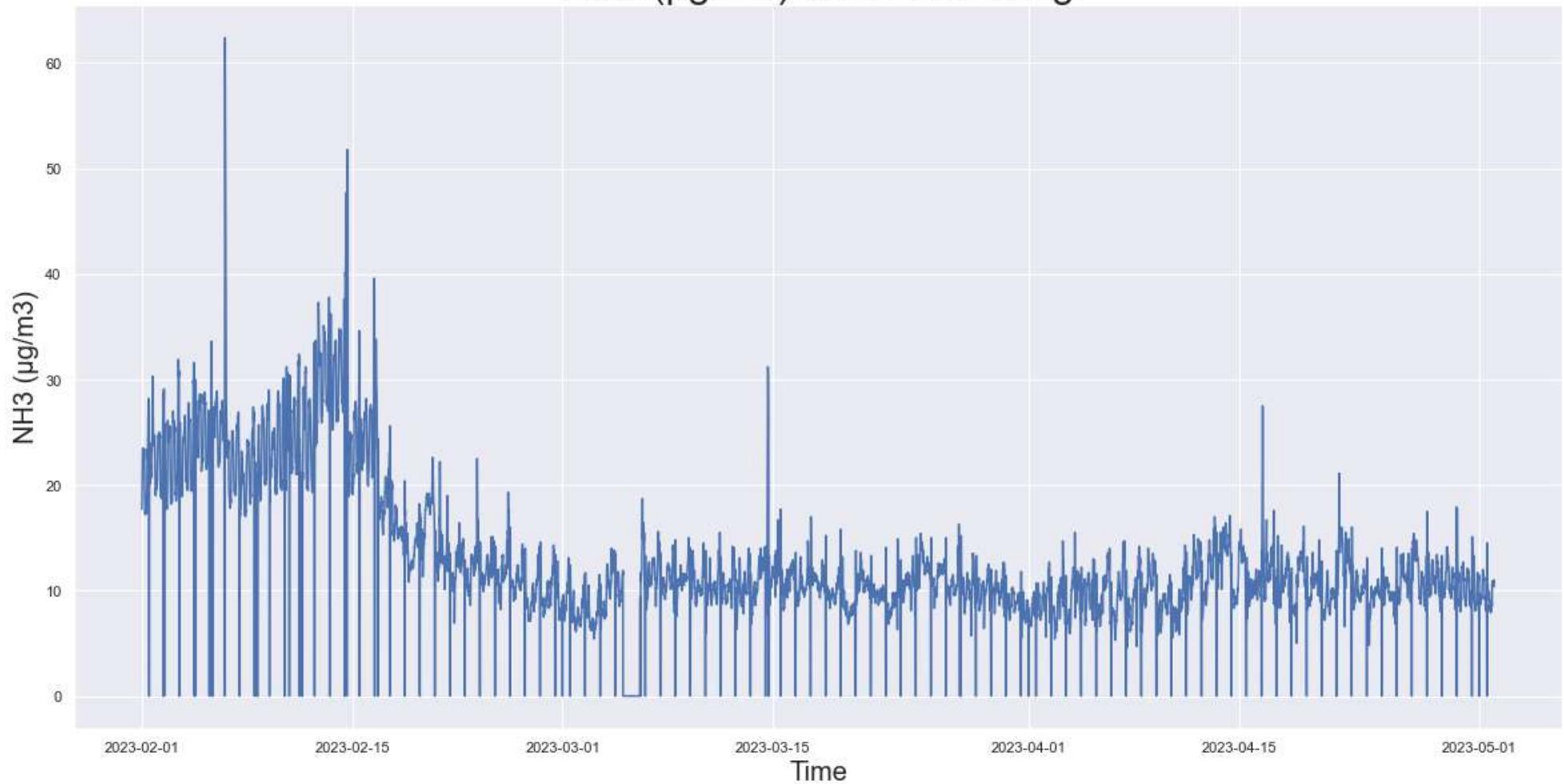
```
In [198]: simple_time_plot(df_median,pollutant,"after median filling")
```

NH3 ($\mu\text{g}/\text{m}^3$) after median filling

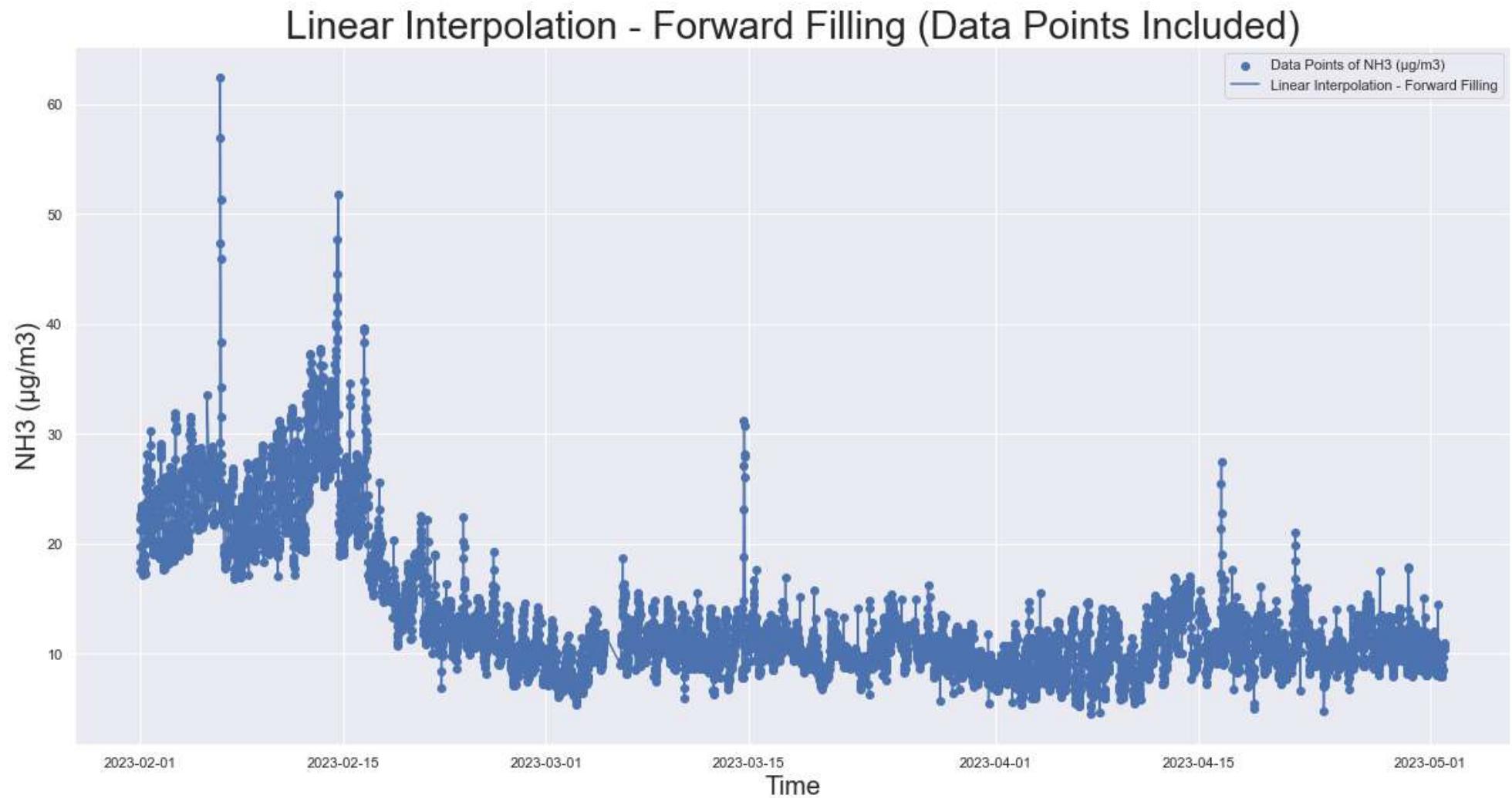


```
In [199]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

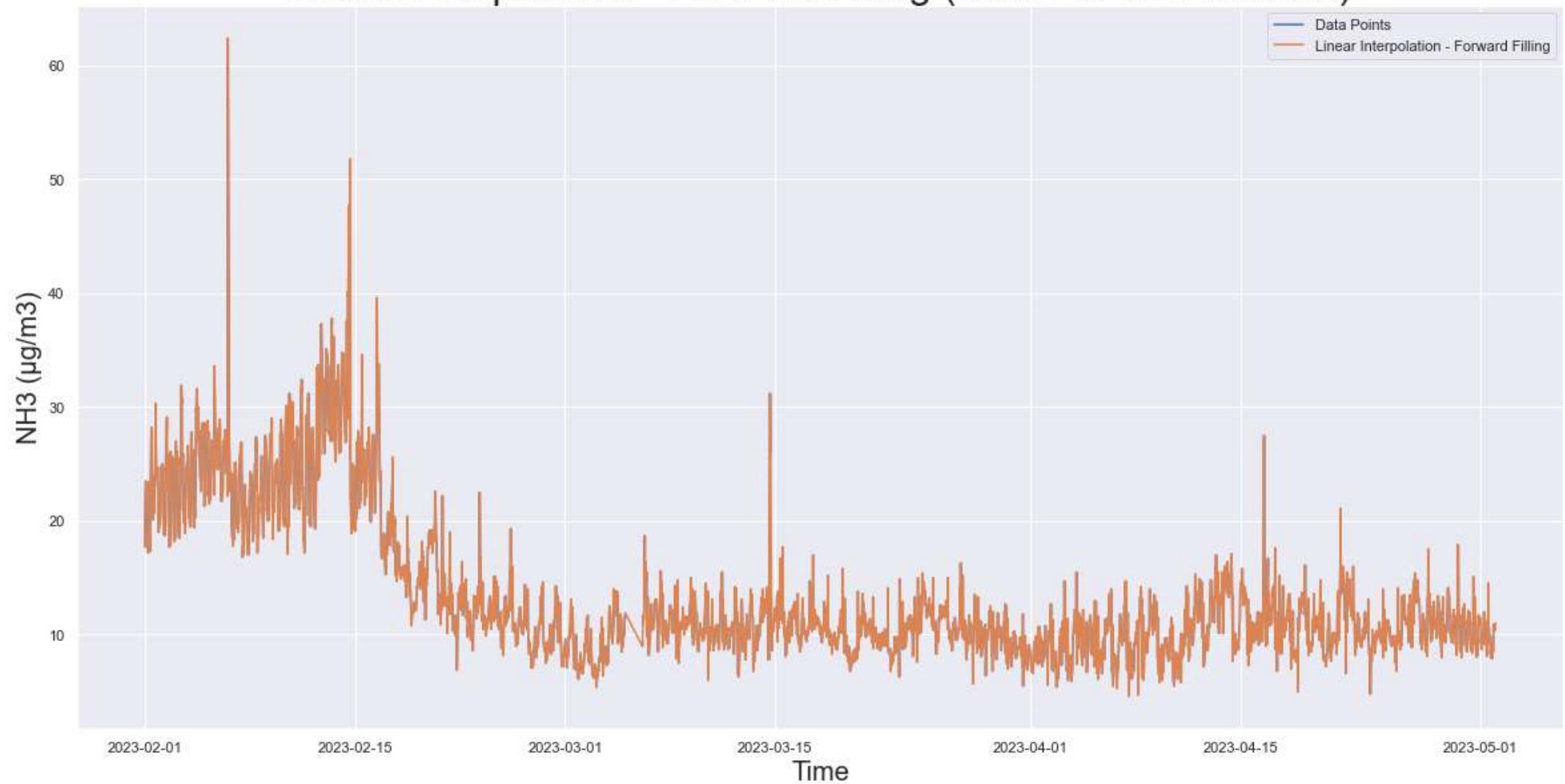
NH3 ($\mu\text{g}/\text{m}^3$) after zero filling



```
In [200]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```

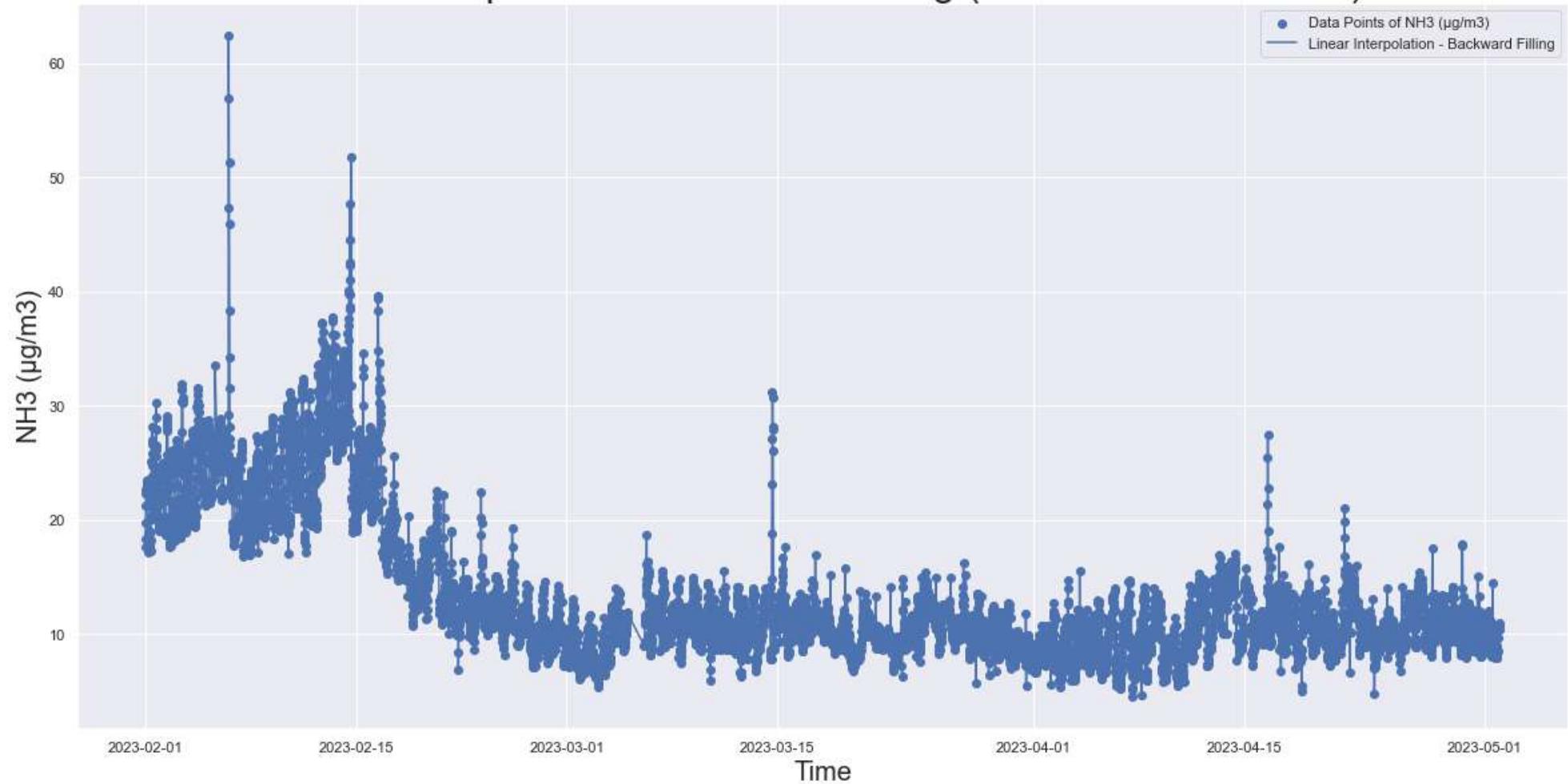


Linear Interpolation - Forward Filling (Data Points Excluded)

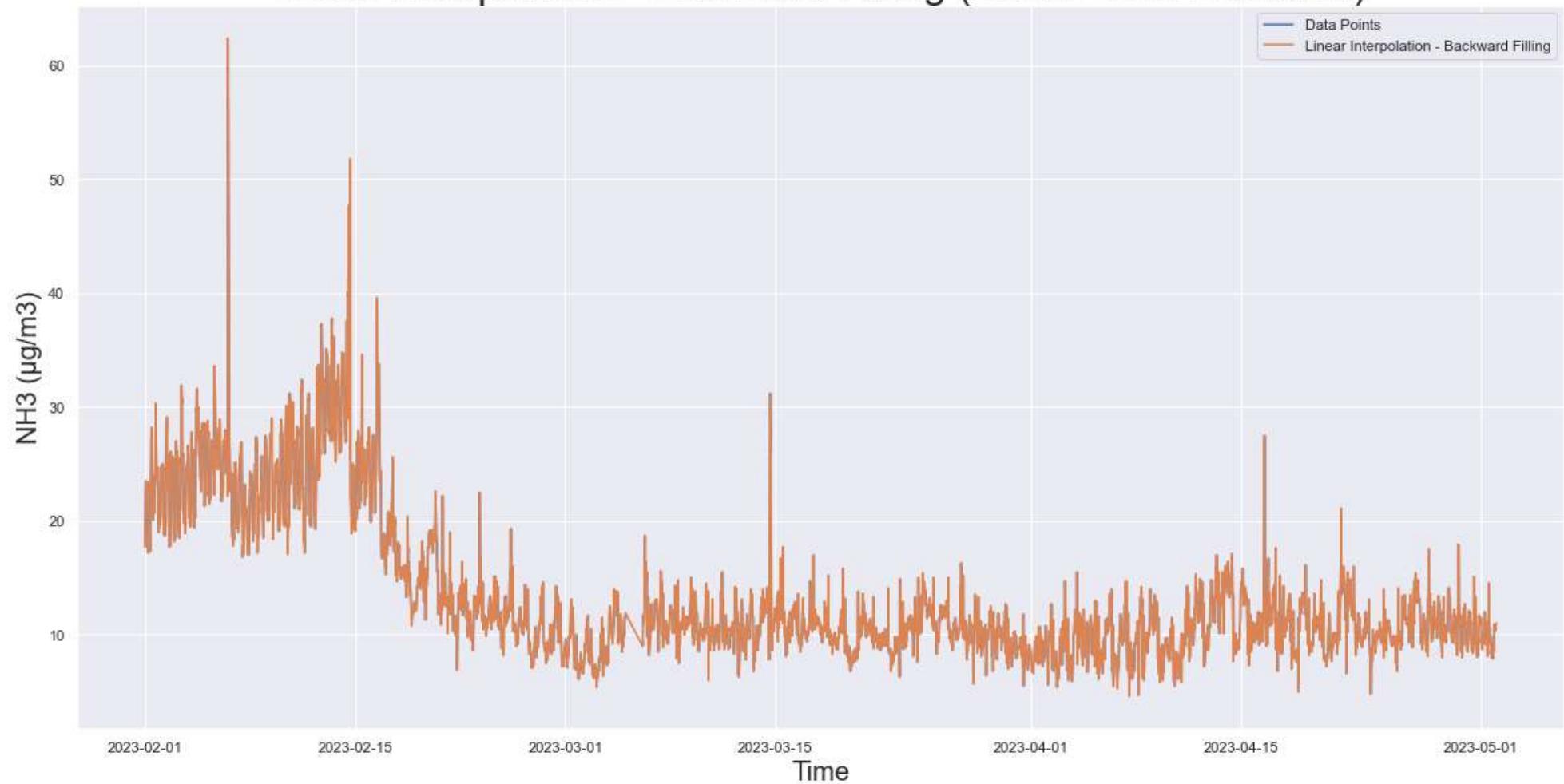


```
In [201]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

Linear Interpolation - Backward Filling (Data Points Included)

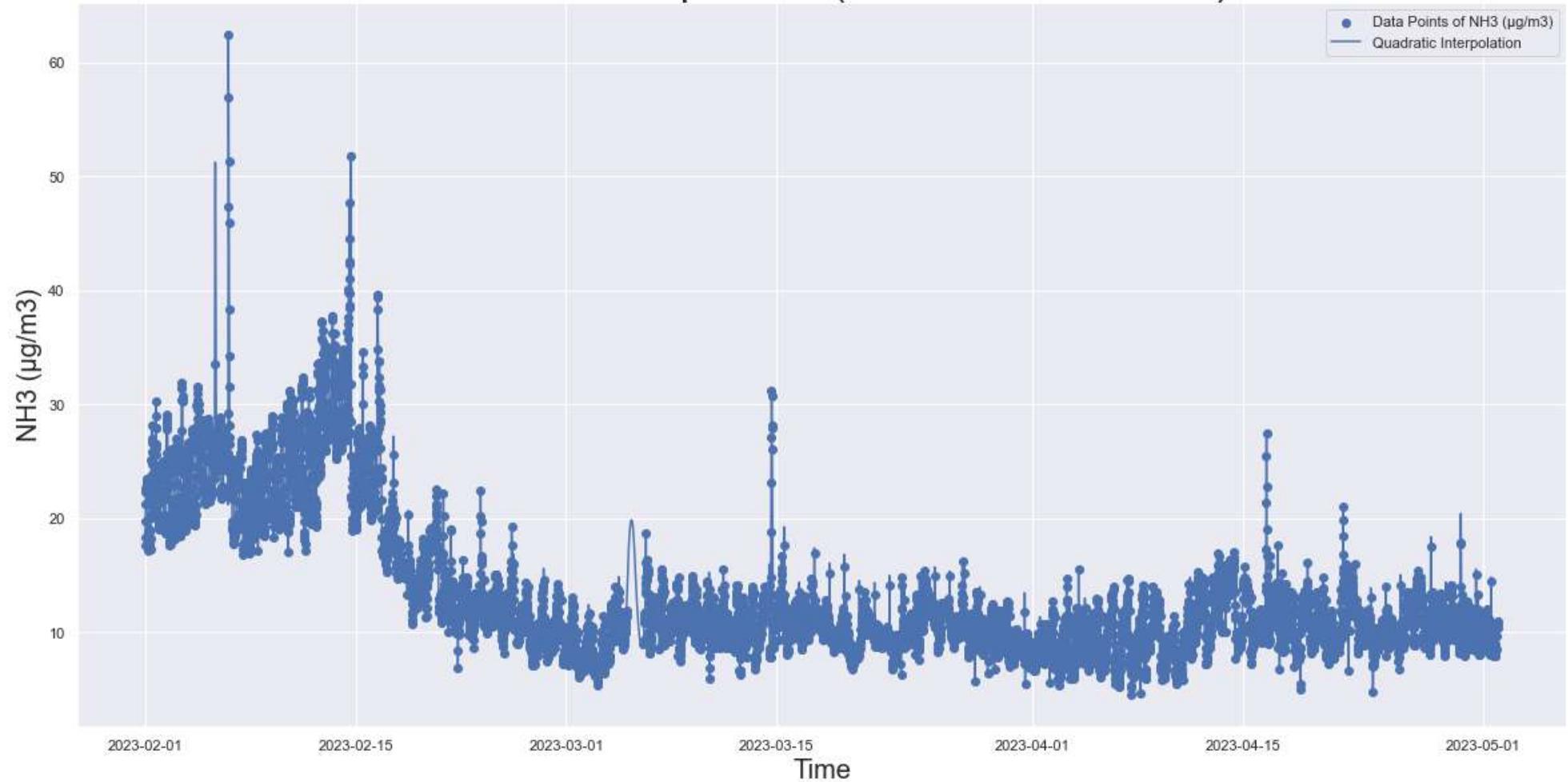


Linear Interpolation - Backward Filling (Data Points Excluded)

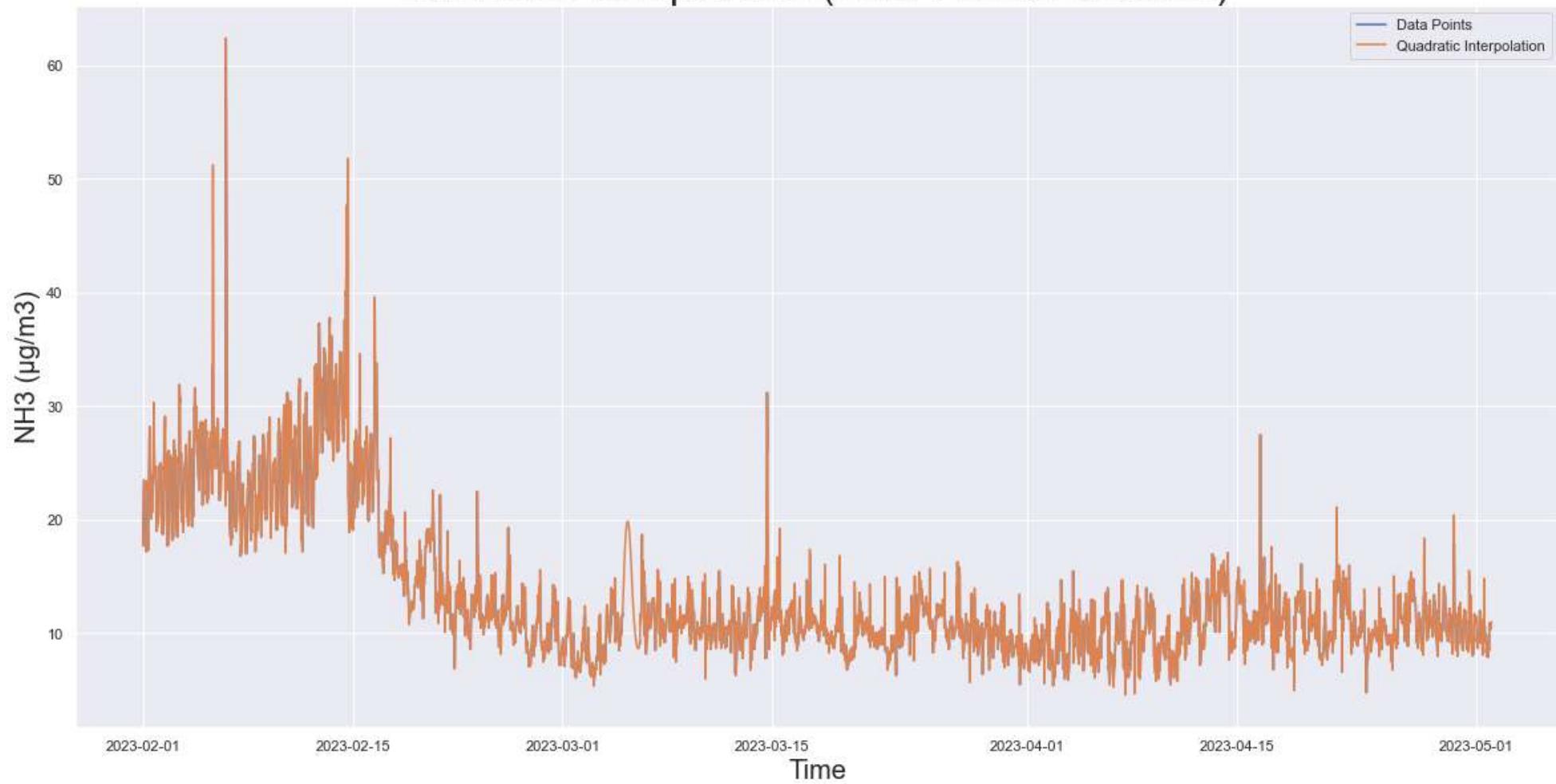


In [202]: `interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")`

Quadratic Interpolation (Data Points Included)

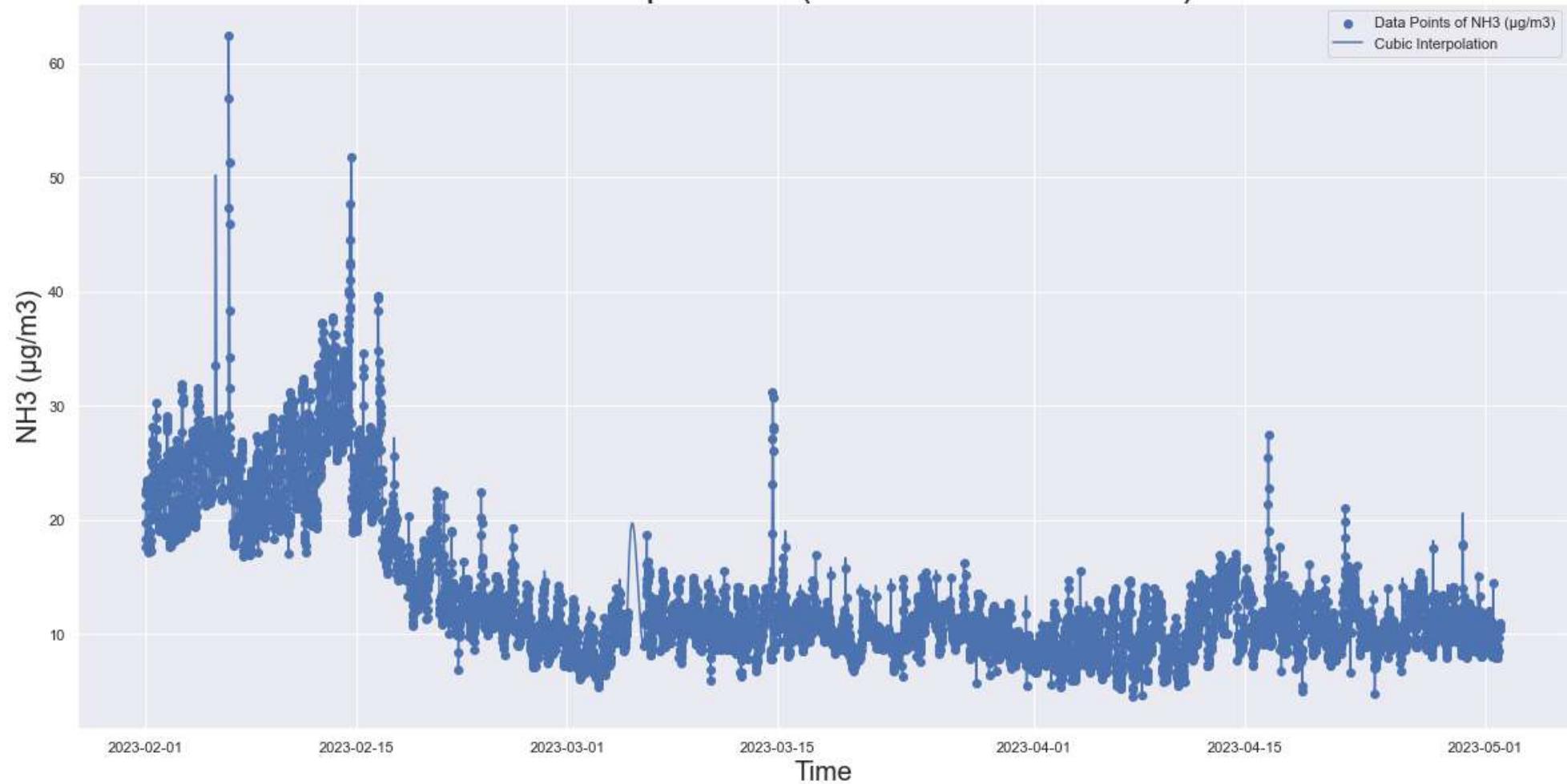


Quadratic Interpolation (Data Points Excluded)

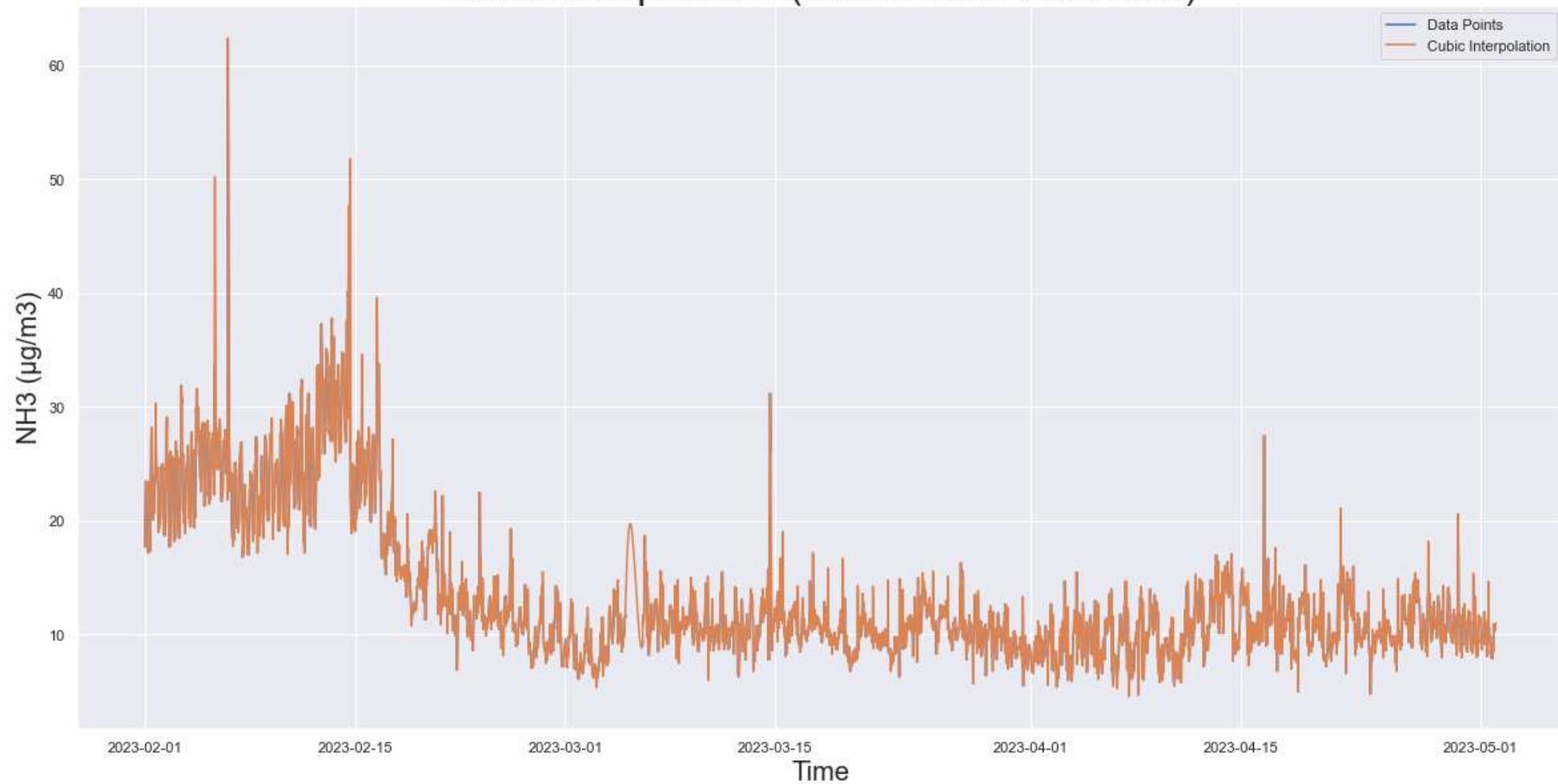


```
In [203]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```

Cubic Interpolation (Data Points Included)

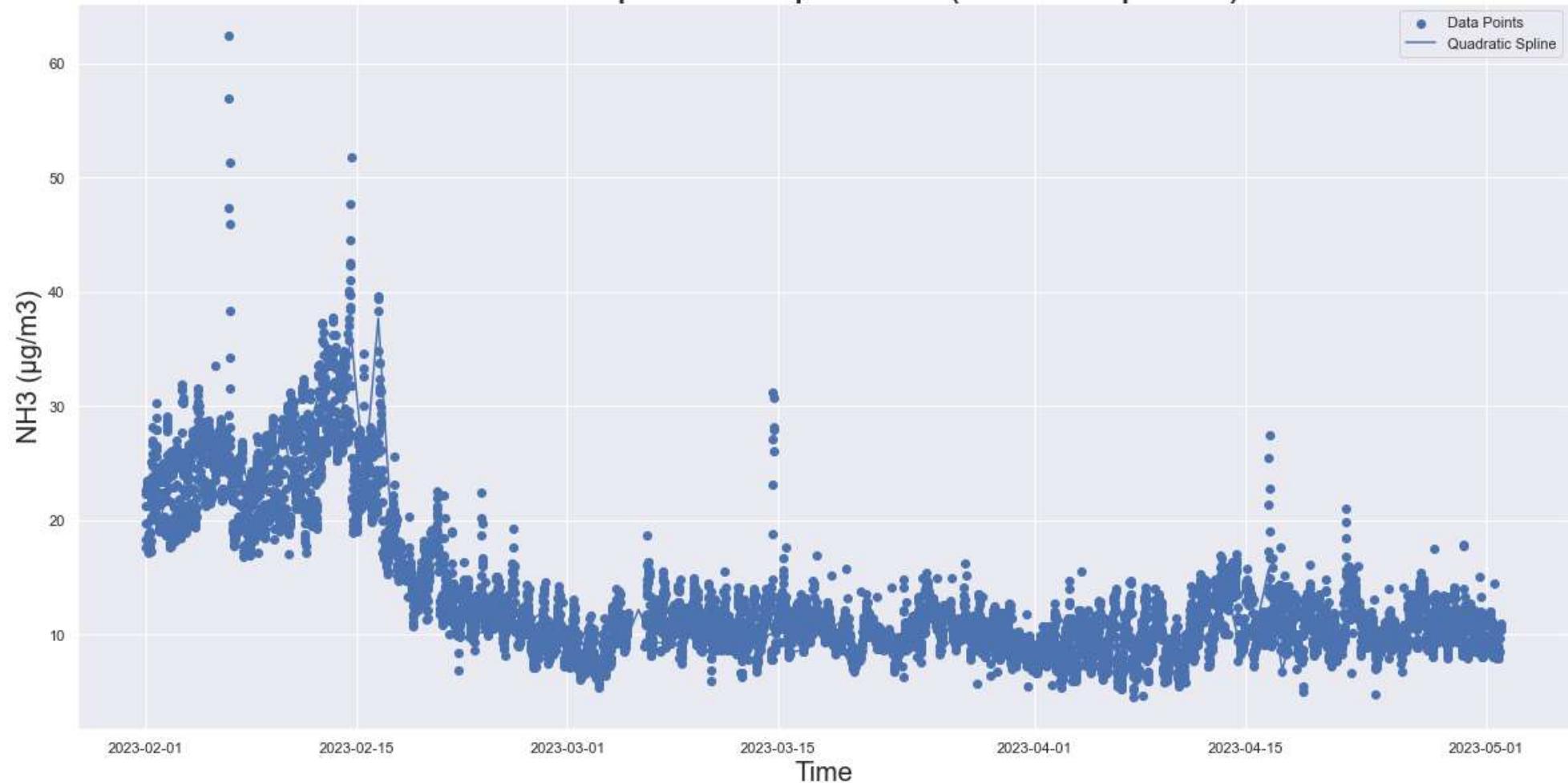


Cubic Interpolation (Data Points Excluded)

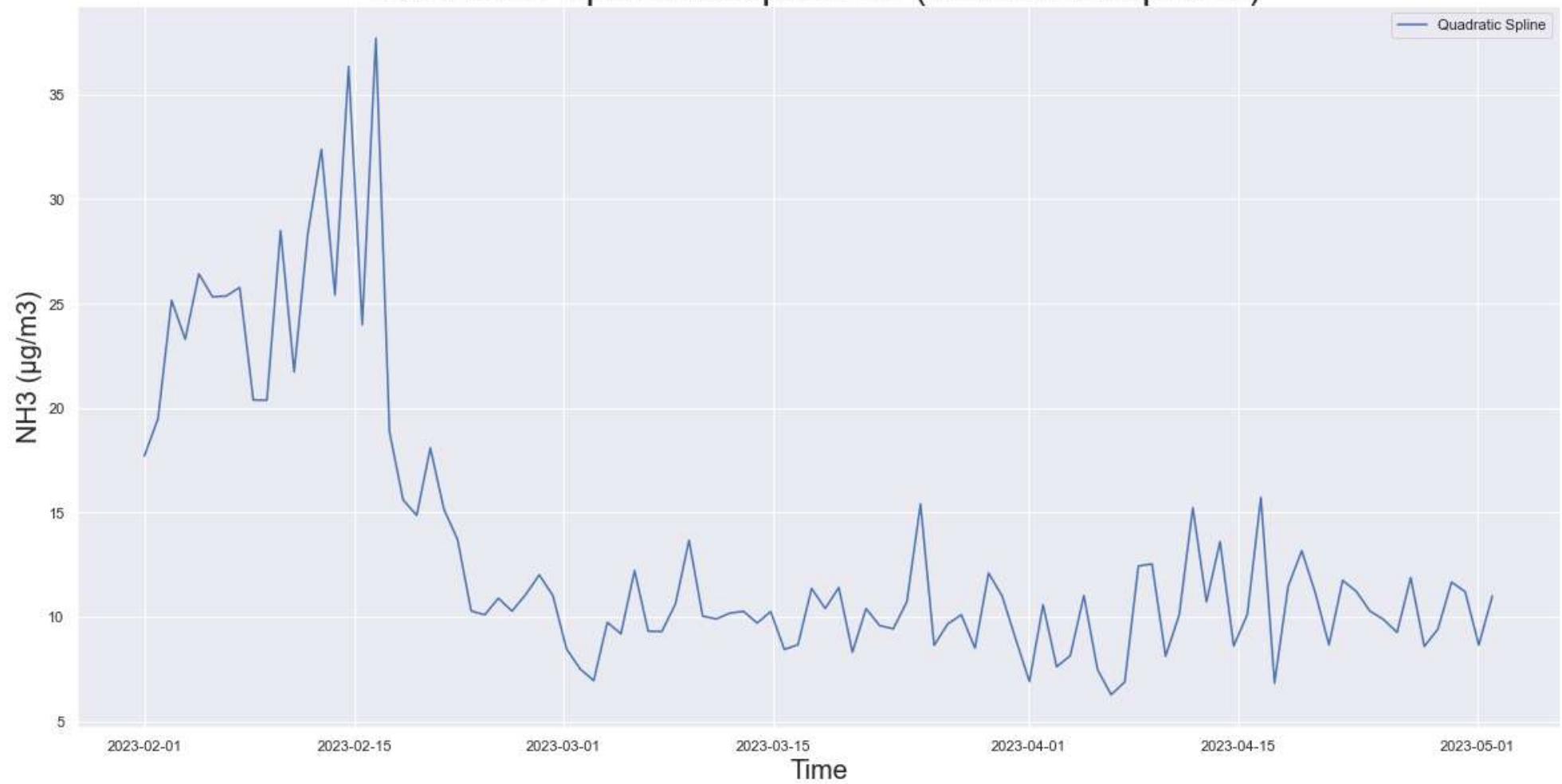


In [204]: `quadraticspline(df_n,pollutant)`

Quadratic Spline Interpolation (with datapoints)

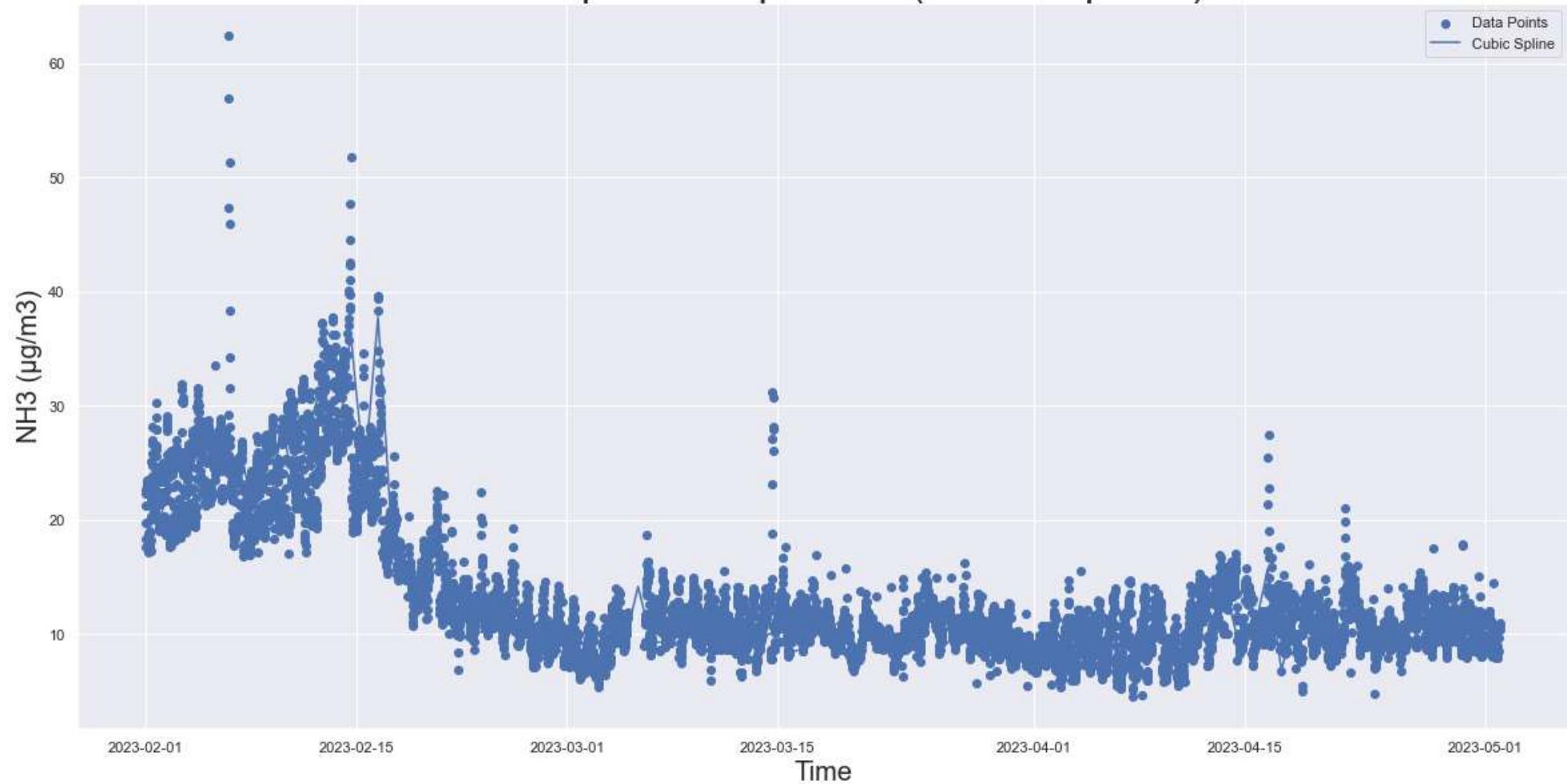


Quadratic Spline Interpolation (without datapoints)

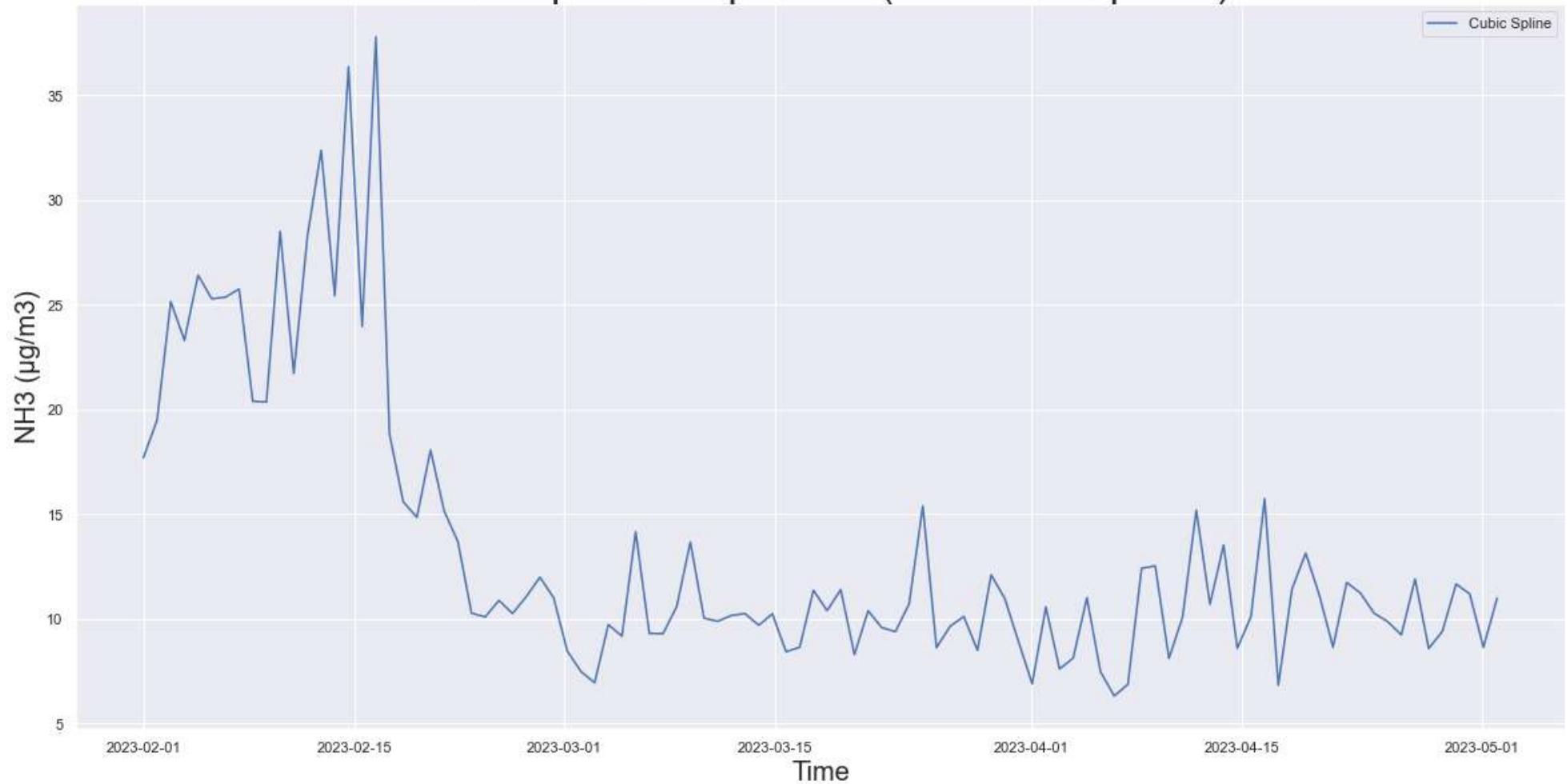


```
In [205]: cubicspline(df_n,pollutant)
```

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



As is evident from the plots, linear interpolation seems to work best.

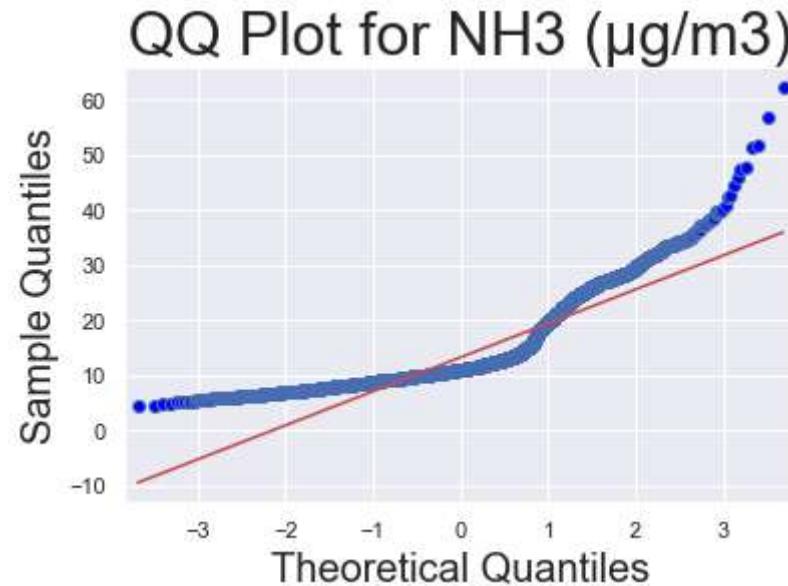
In [206]: `df_linear_b[pollutant].isnull().sum()`

Out[206]: 0

In [207]: `df_new[pollutant] = df_linear_b[pollutant]`

```
In [208]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>



```
In [209]: df_new.head()
```

Out[209]:

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)	CO (mg/m^3)	SO2 ($\mu\text{g}/\text{m}^3$)	NH3 ($\mu\text{g}/\text{m}^3$)
--	-----------------------------------	------------------------------------	---------------------------------	----------------------------------	-----------	-------------------------------	----------------------------------	----------------------------------

Time

2023-02-01 00:00:00	95.0	35.0	18.1	90.1	56.2	0.31	8.2	17.7
2023-02-01 00:15:00	95.0	35.0	18.1	88.0	55.1	0.33	8.2	18.3
2023-02-01 00:30:00	95.0	35.0	18.1	87.7	55.2	0.38	8.2	19.7
2023-02-01 00:45:00	122.0	34.0	18.1	88.9	55.7	0.38	8.2	21.3
2023-02-01 01:00:00	122.0	34.0	18.1	90.0	55.8	0.38	8.2	22.3

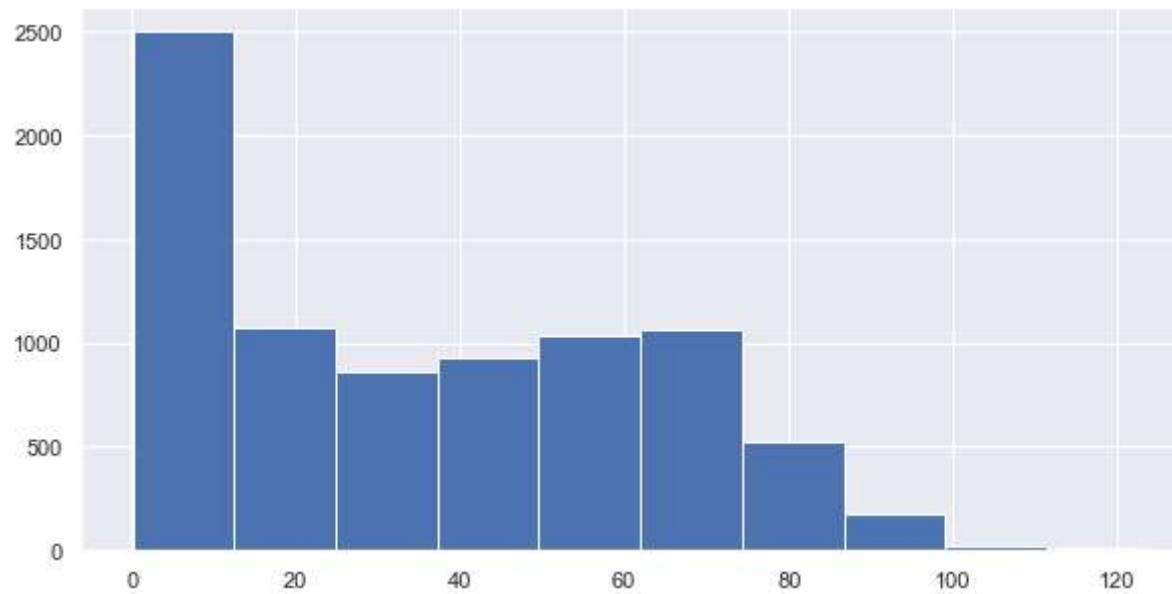
Analysis for "Ozone ($\mu\text{g}/\text{m}^3$)"

```
In [210]: pollutant = 'Ozone ( $\mu\text{g}/\text{m}^3$ )'
```

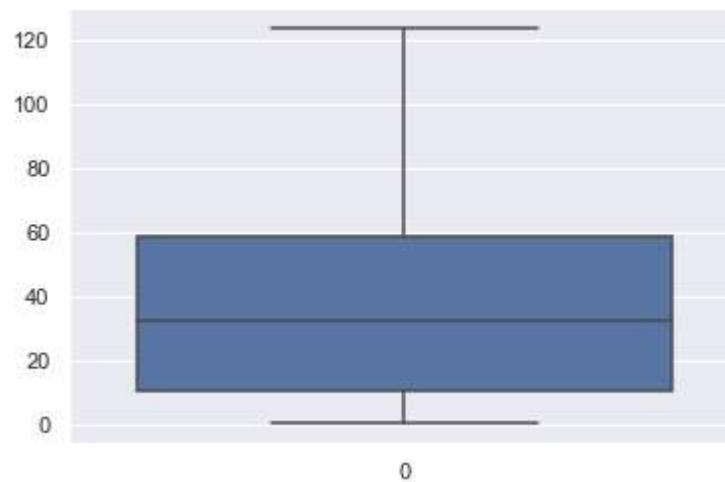
```
In [211]: df[pollutant].describe()
```

```
Out[211]: count    8187.000000
mean      35.626530
std       27.018693
min       0.100000
25%      10.500000
50%      32.400000
75%      58.800000
max     123.800000
Name: Ozone (\mu g/m3), dtype: float64
```

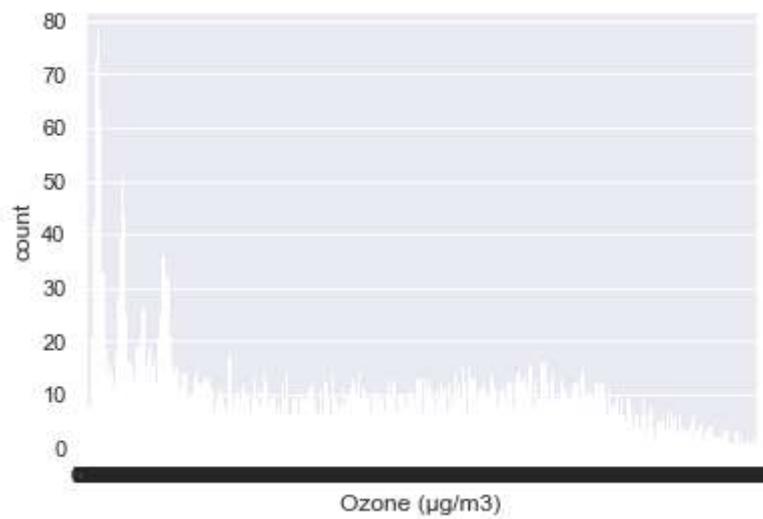
```
In [212]: histogram_plot(df_n,pollutant)
```



```
In [213]: boxplot_plot(df_n,pollutant)
```

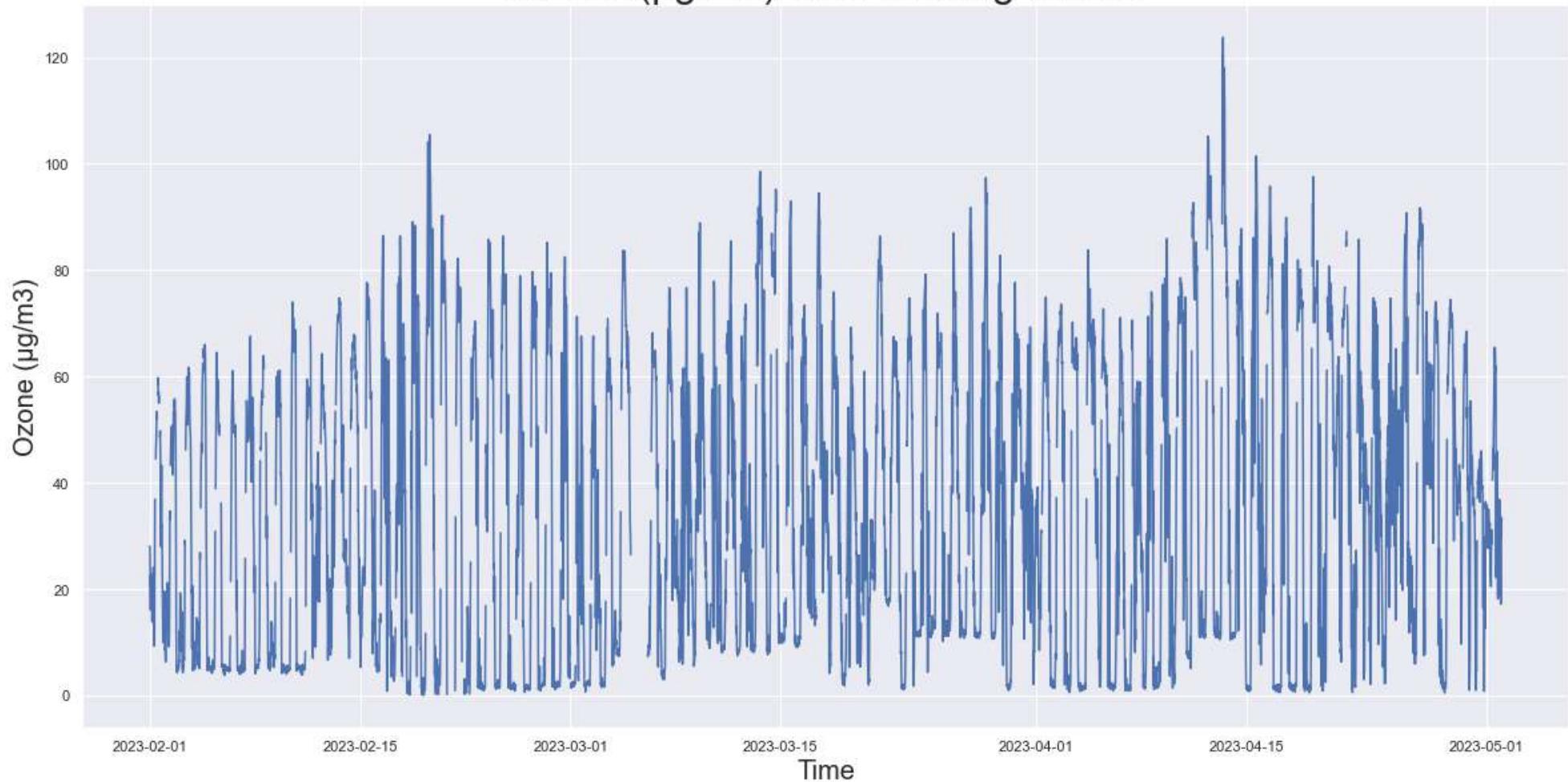


```
In [214]: countplot_plot(df_n,pollutant)
```



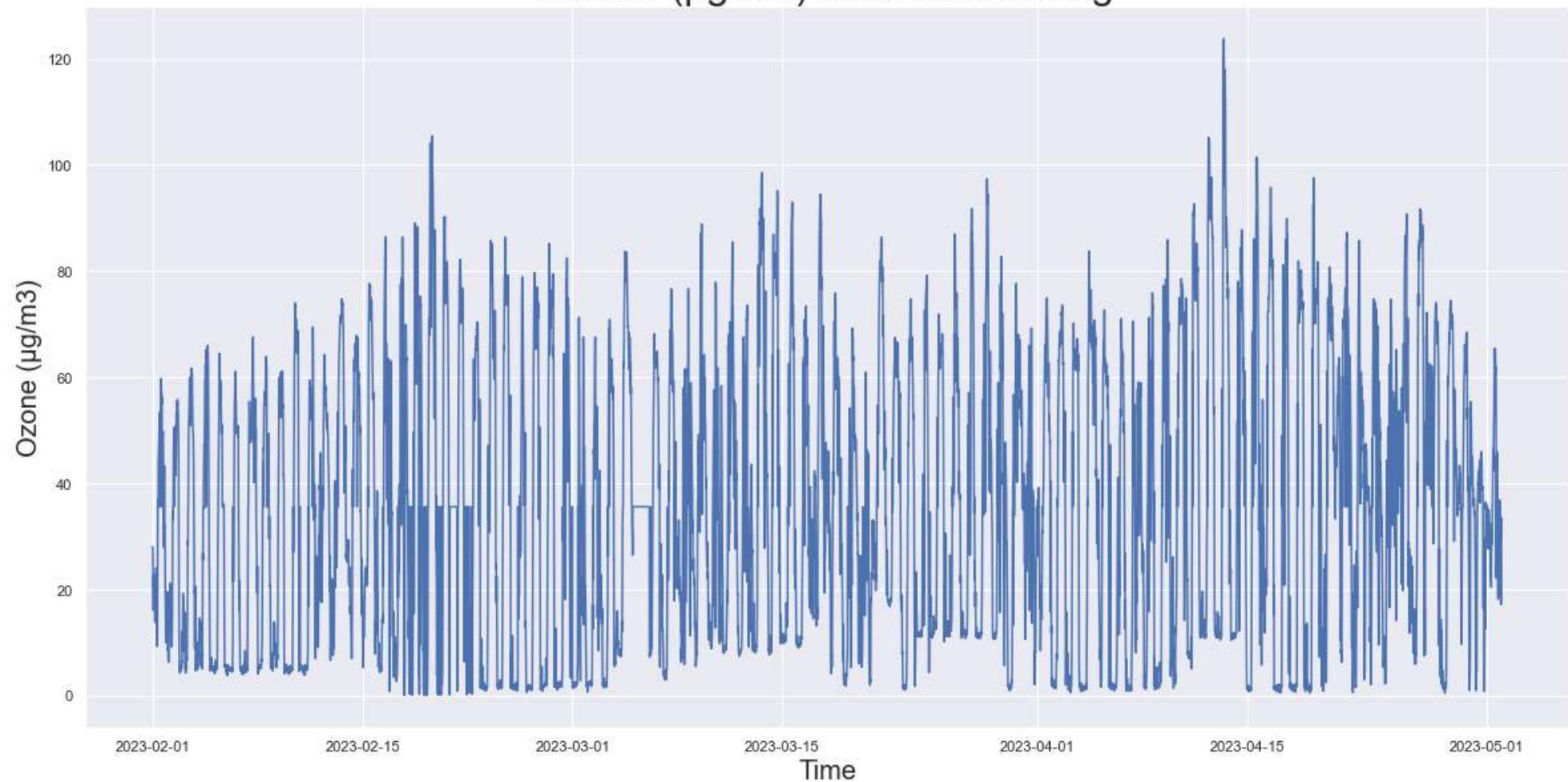
```
In [215]: simple_time_plot(df_n,pollutant,"With missing values")
```

Ozone ($\mu\text{g}/\text{m}^3$) With missing values



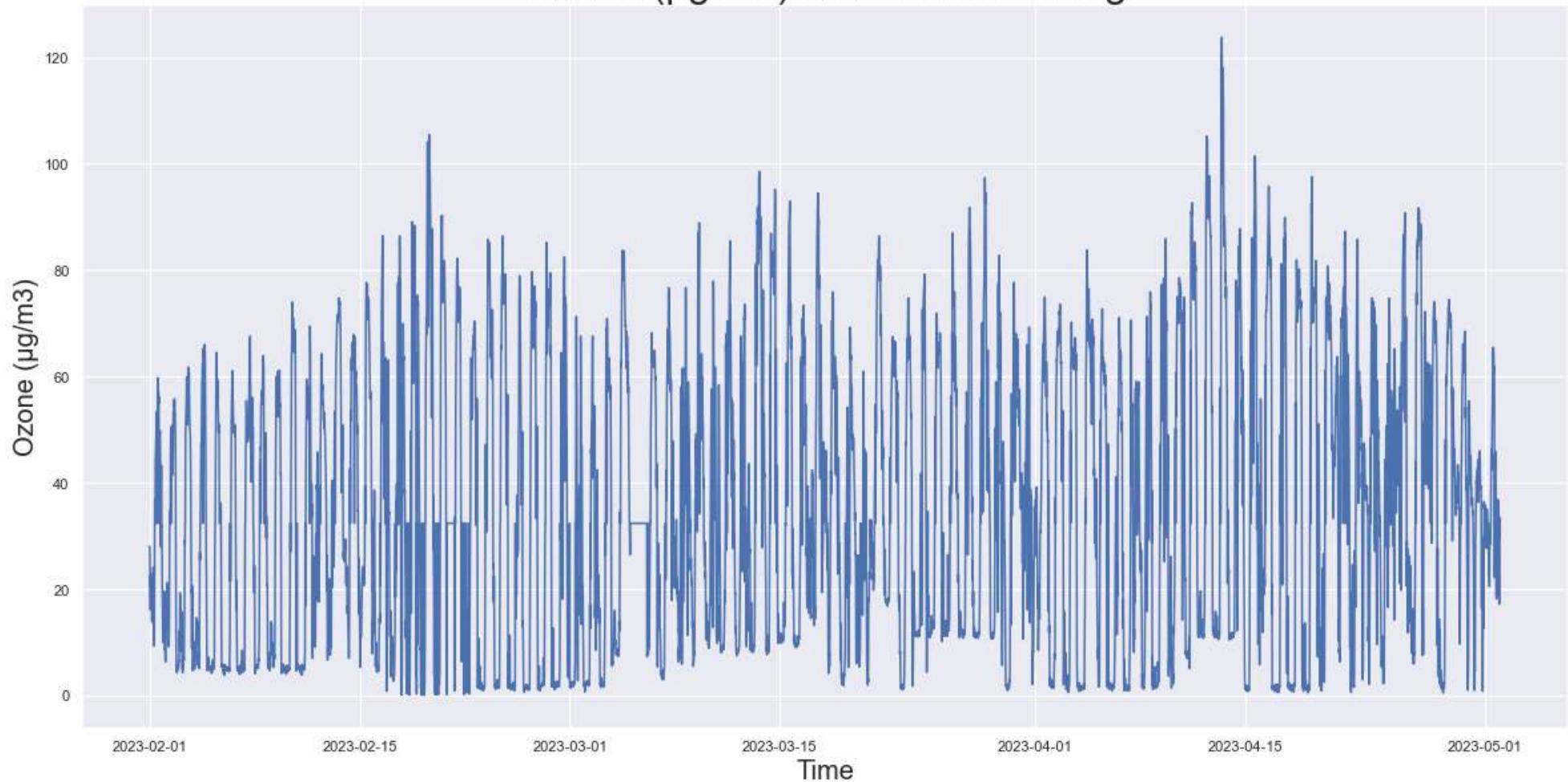
```
In [216]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

Ozone ($\mu\text{g}/\text{m}^3$) after mean filling



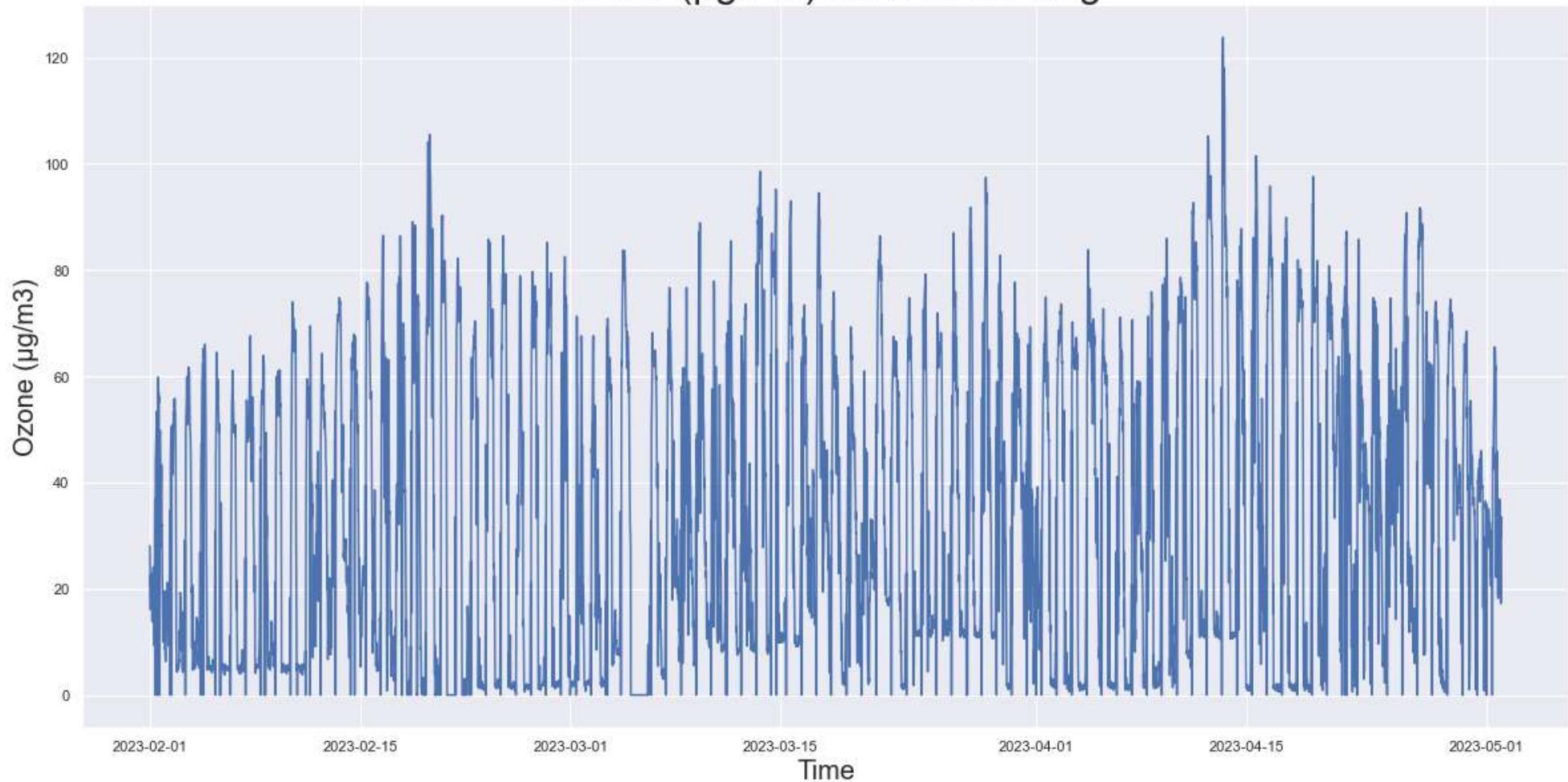
```
In [217]: simple_time_plot(df_median,pollutant,"after median filling")
```

Ozone ($\mu\text{g}/\text{m}^3$) after median filling



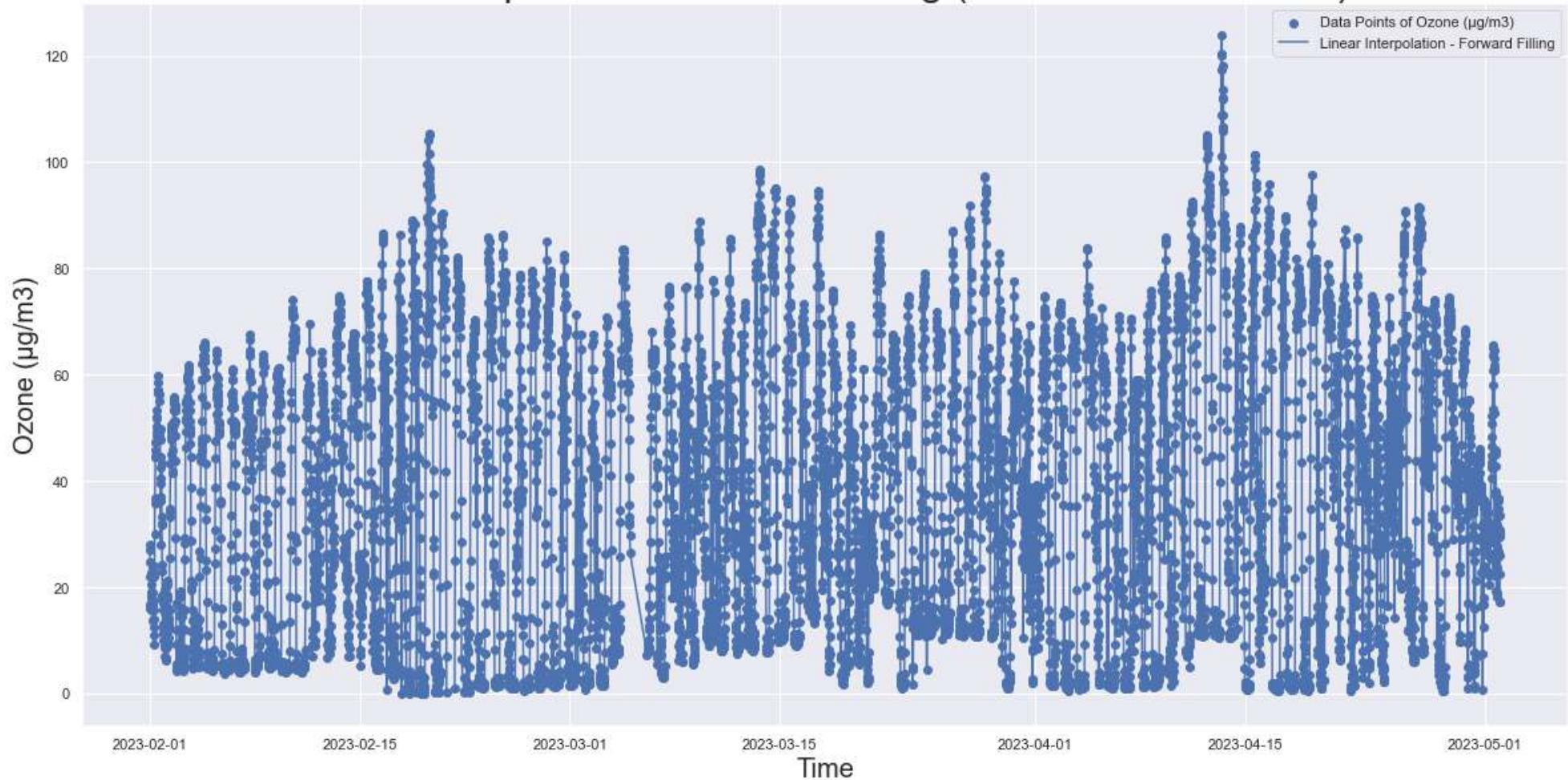
```
In [218]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

Ozone ($\mu\text{g}/\text{m}^3$) after zero filling

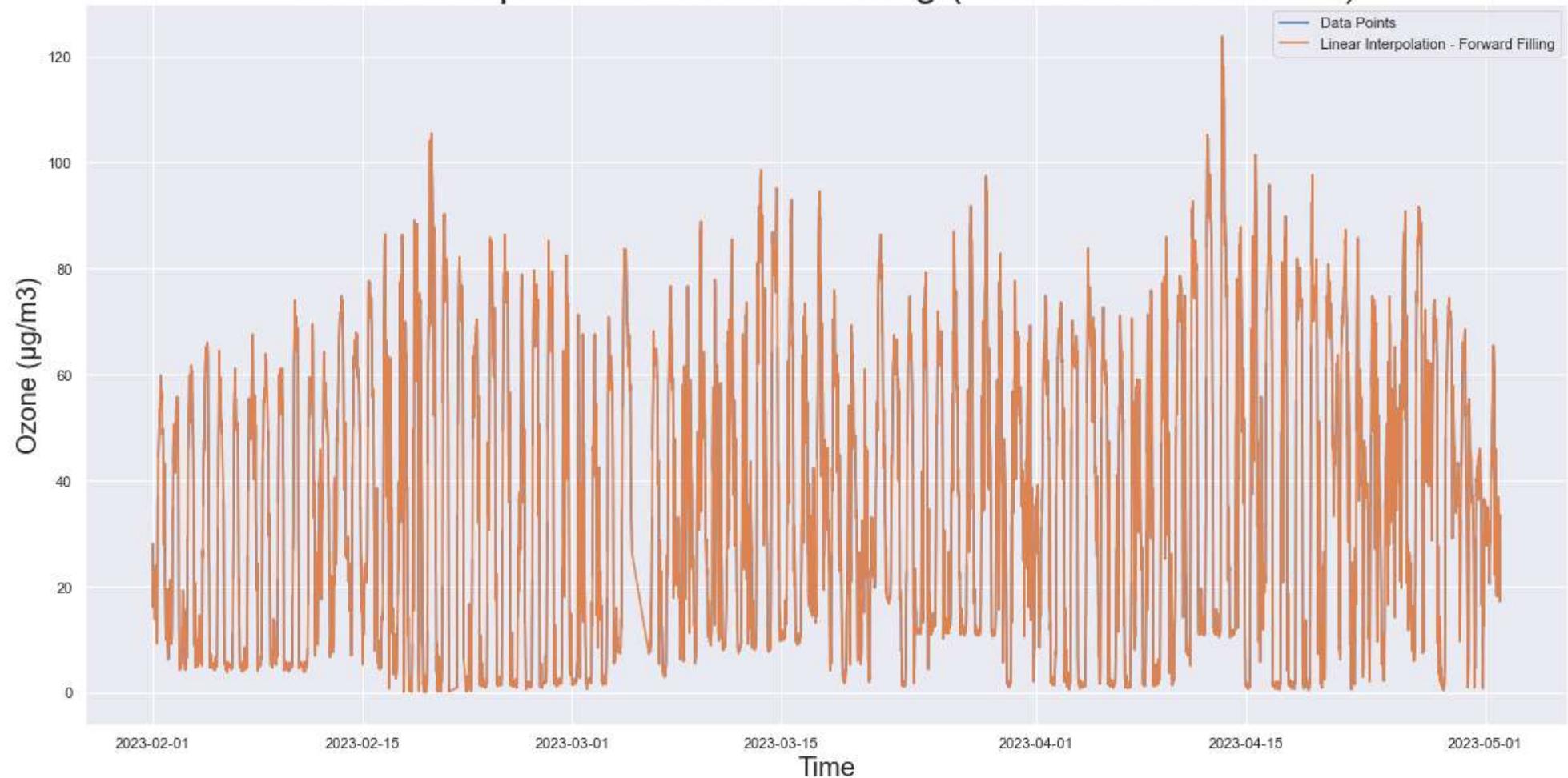


```
In [219]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```

Linear Interpolation - Forward Filling (Data Points Included)

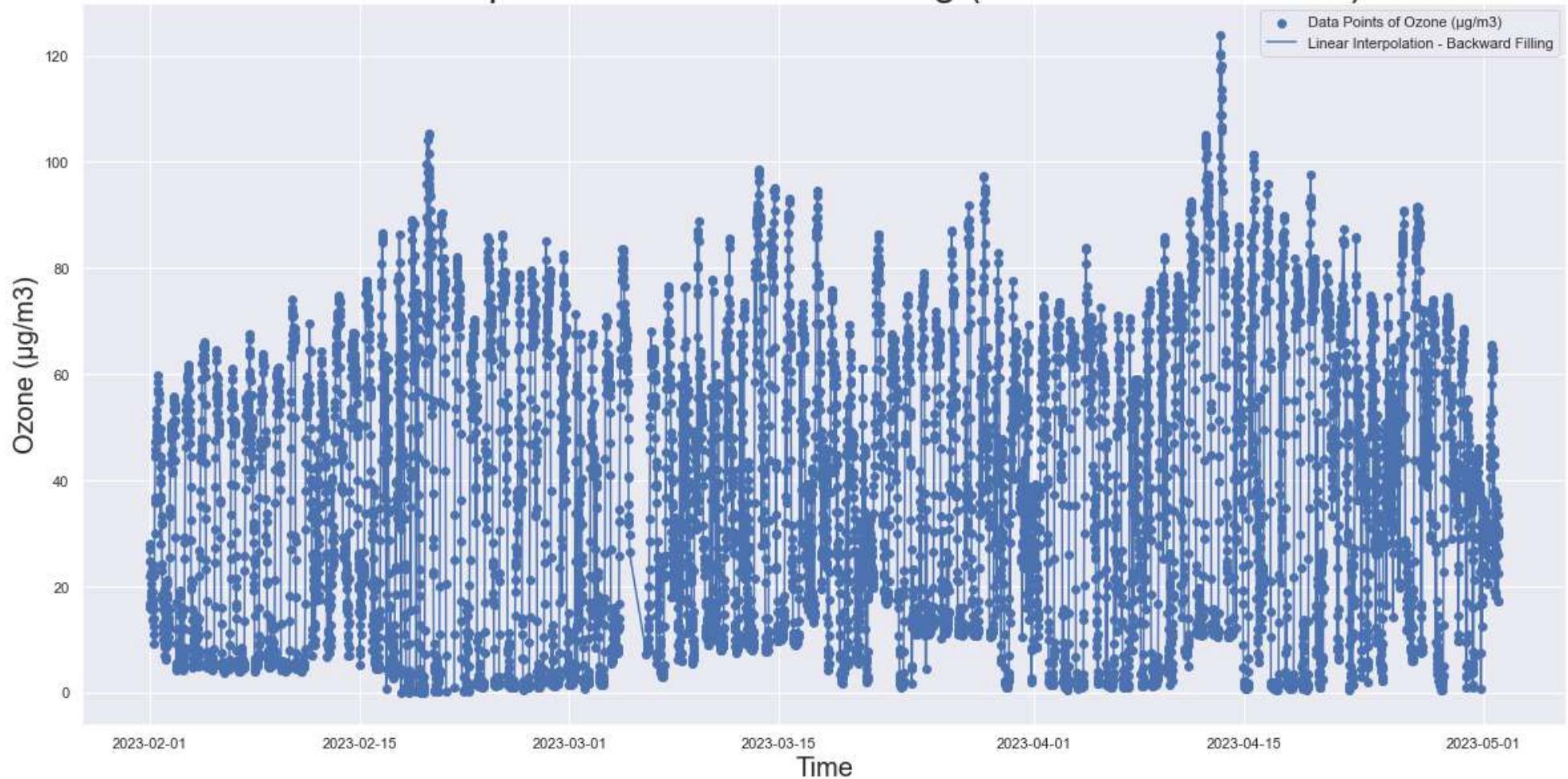


Linear Interpolation - Forward Filling (Data Points Excluded)

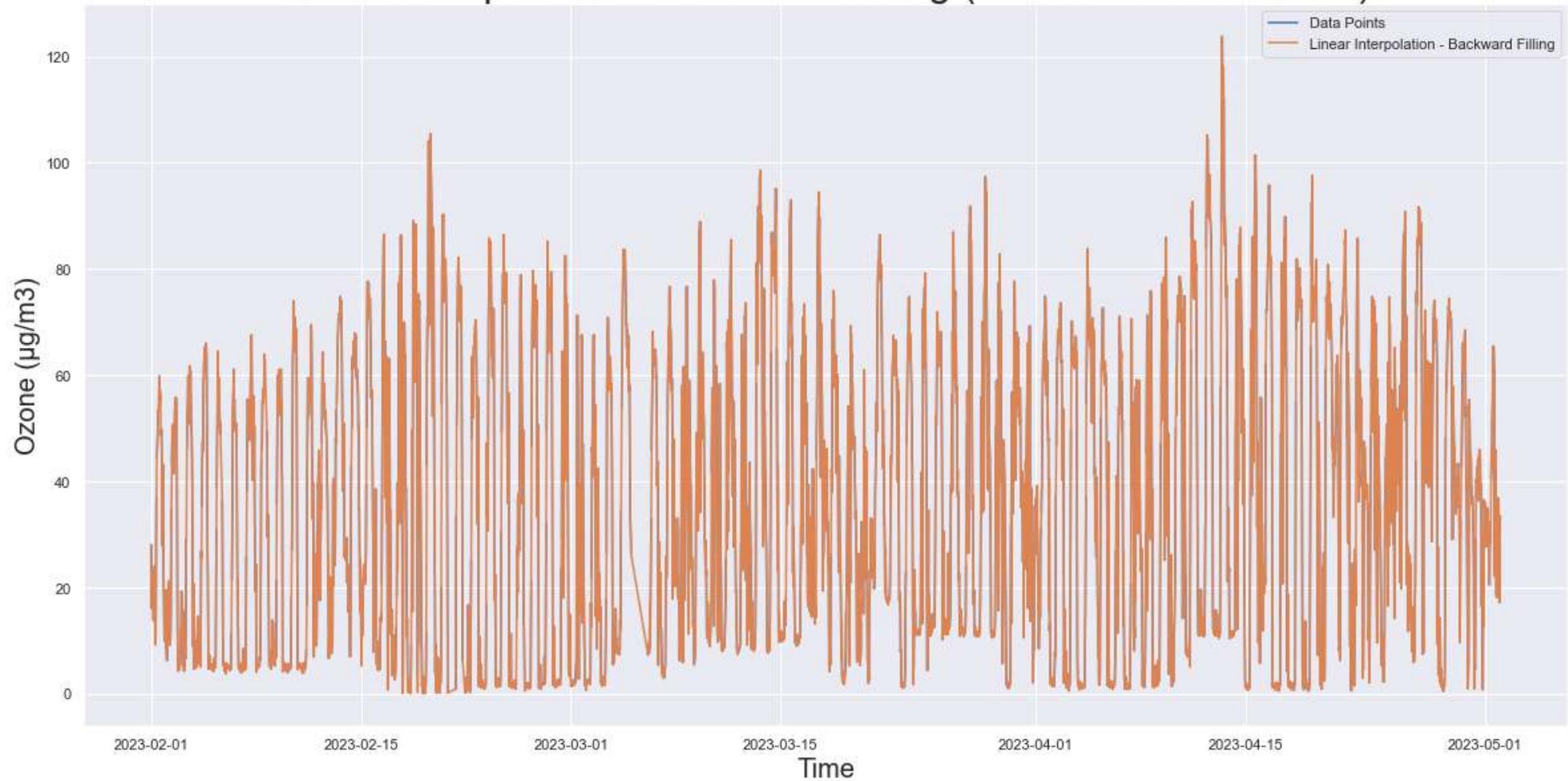


```
In [220]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

Linear Interpolation - Backward Filling (Data Points Included)

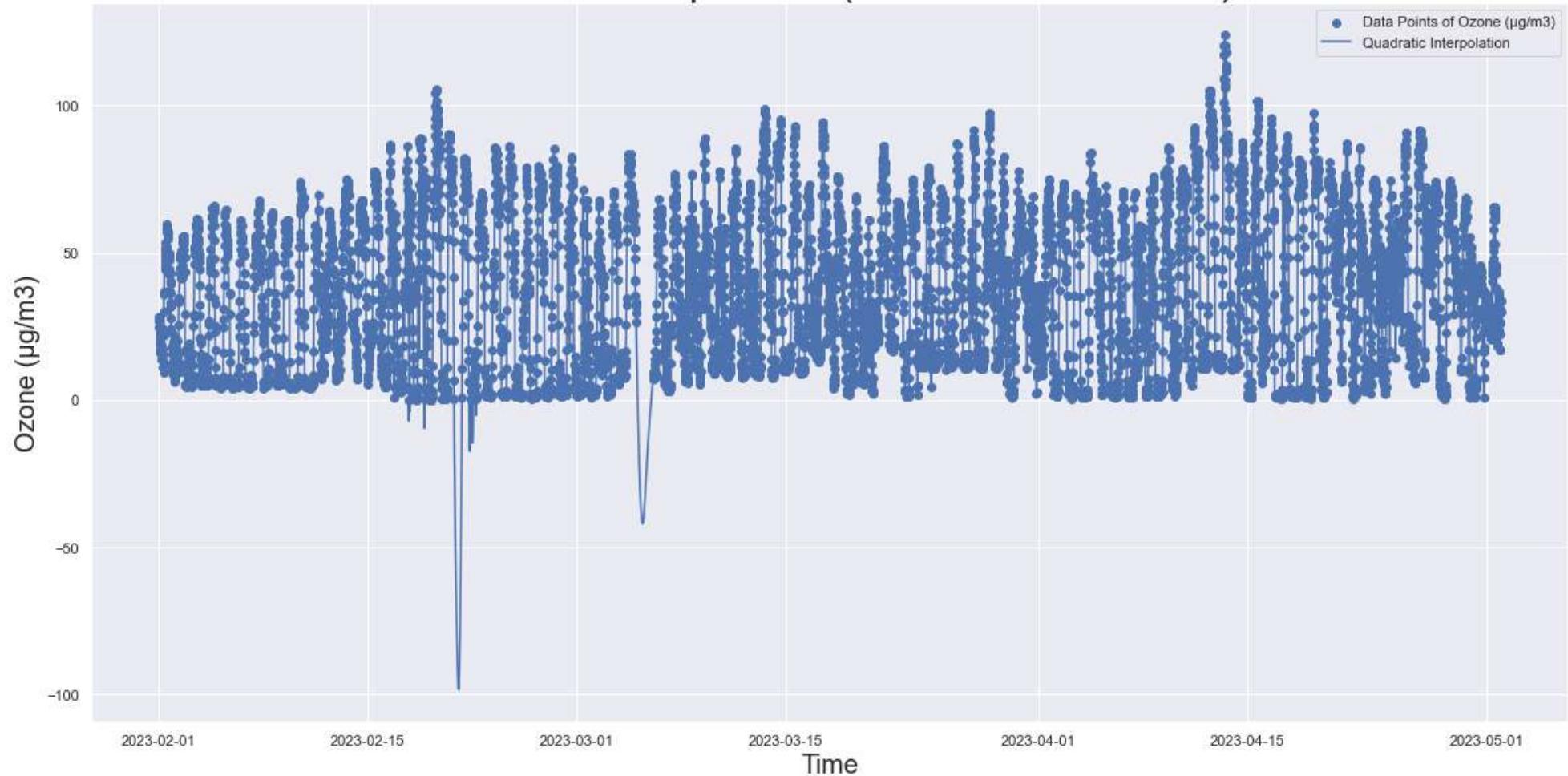


Linear Interpolation - Backward Filling (Data Points Excluded)

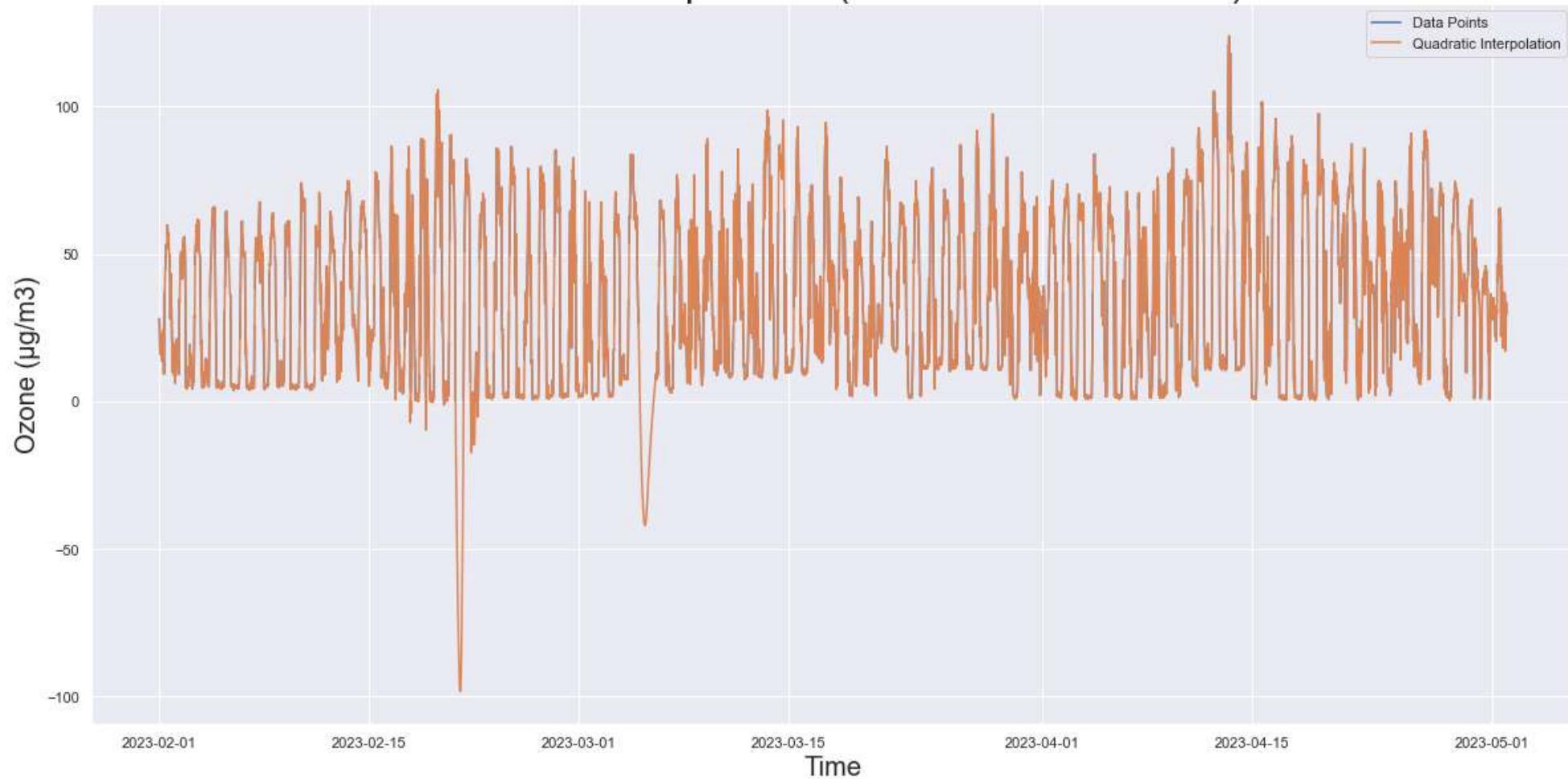


```
In [221]: interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")
```

Quadratic Interpolation (Data Points Included)

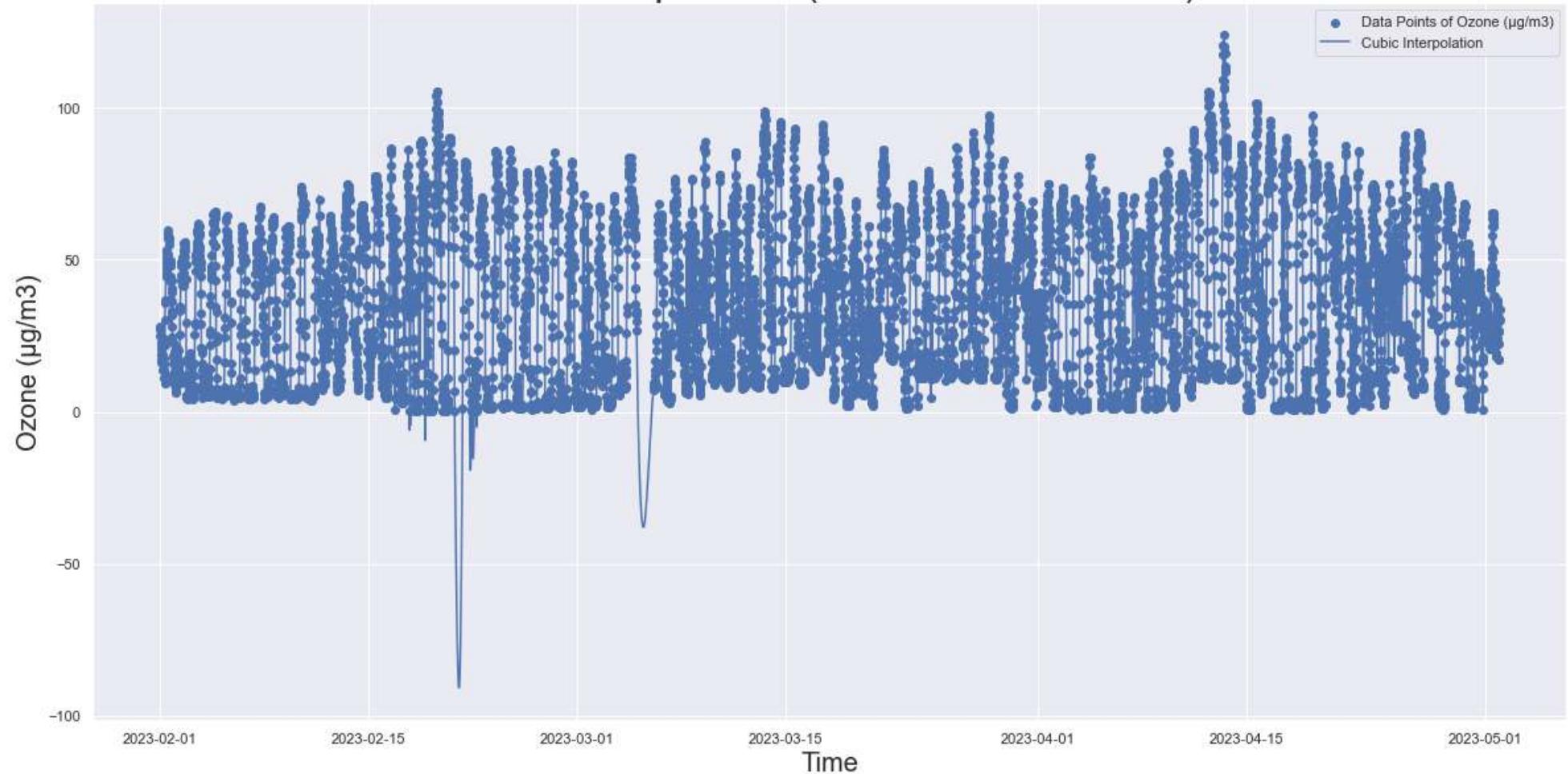


Quadratic Interpolation (Data Points Excluded)

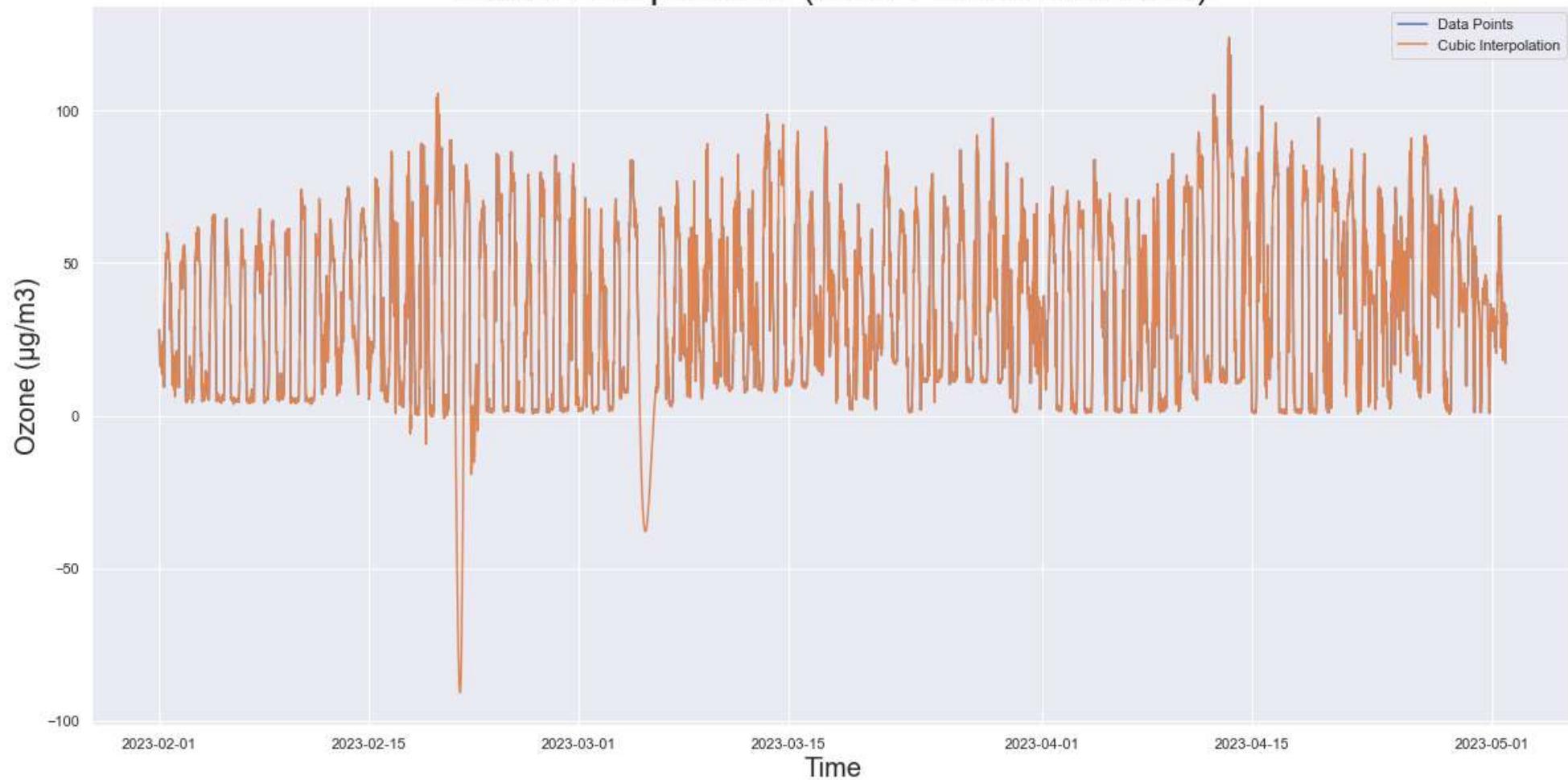


```
In [222]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```

Cubic Interpolation (Data Points Included)

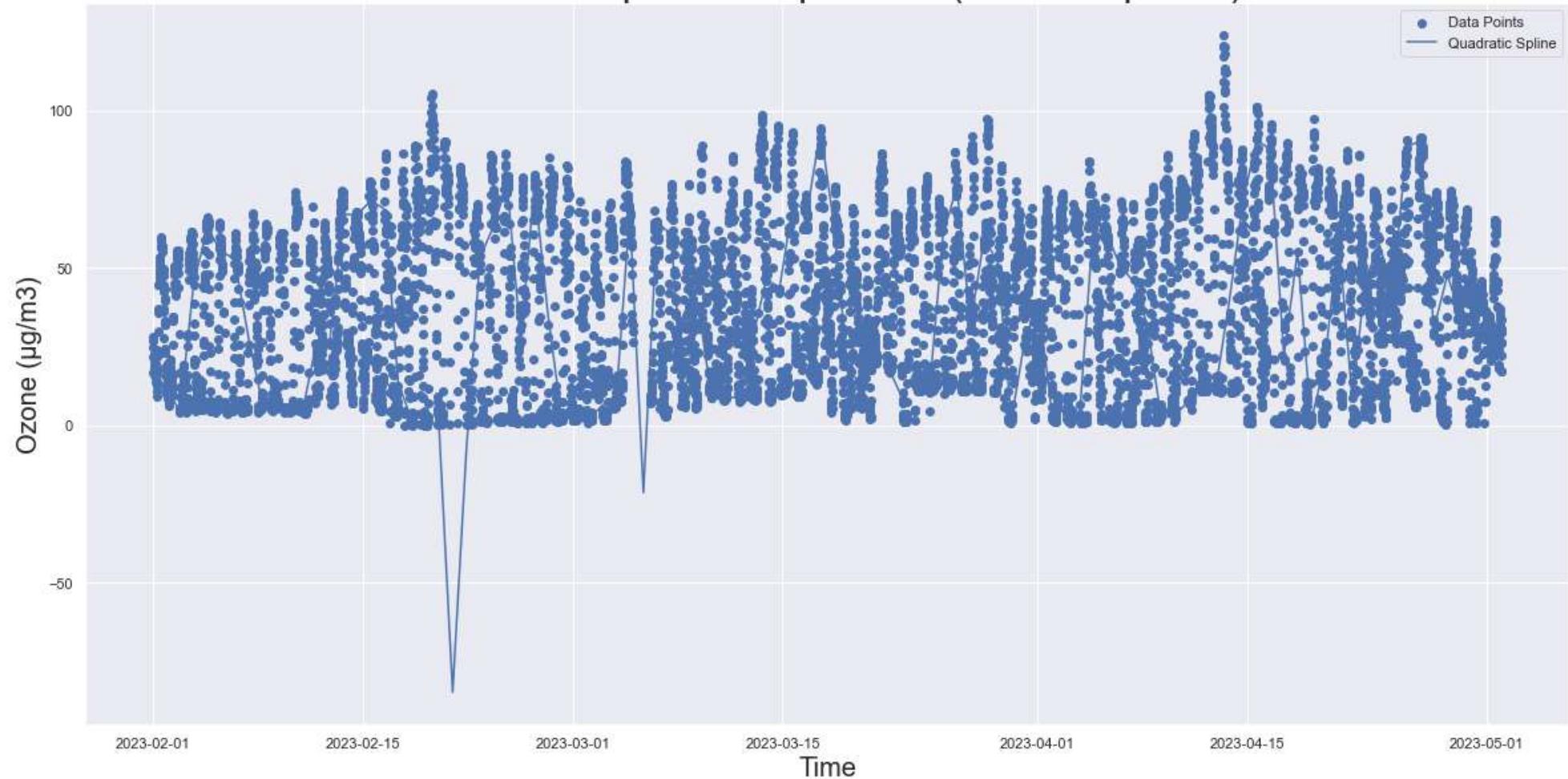


Cubic Interpolation (Data Points Excluded)

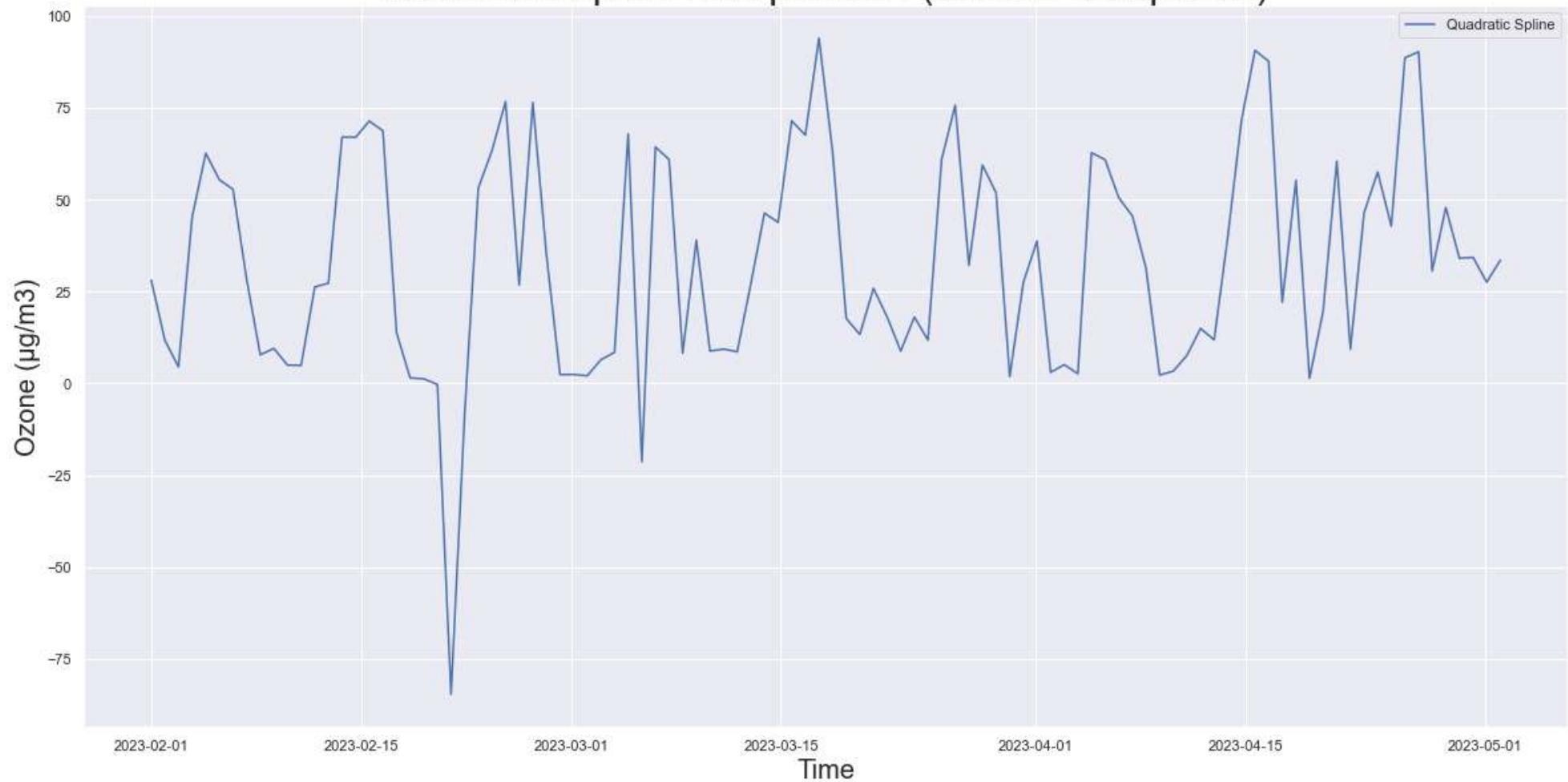


In [223]: `quadraticspline(df_n,pollutant)`

Quadratic Spline Interpolation (with datapoints)

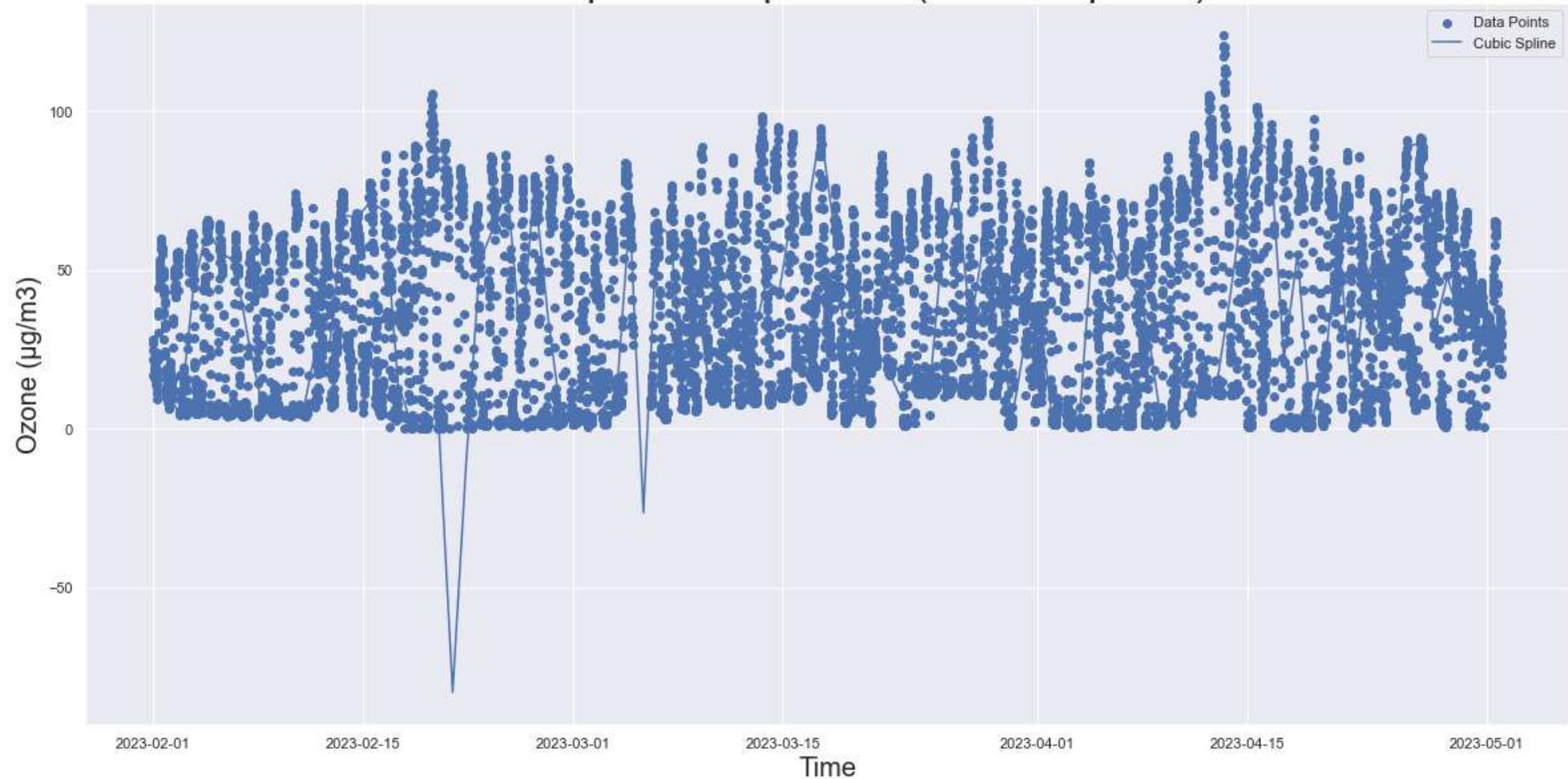


Quadratic Spline Interpolation (without datapoints)

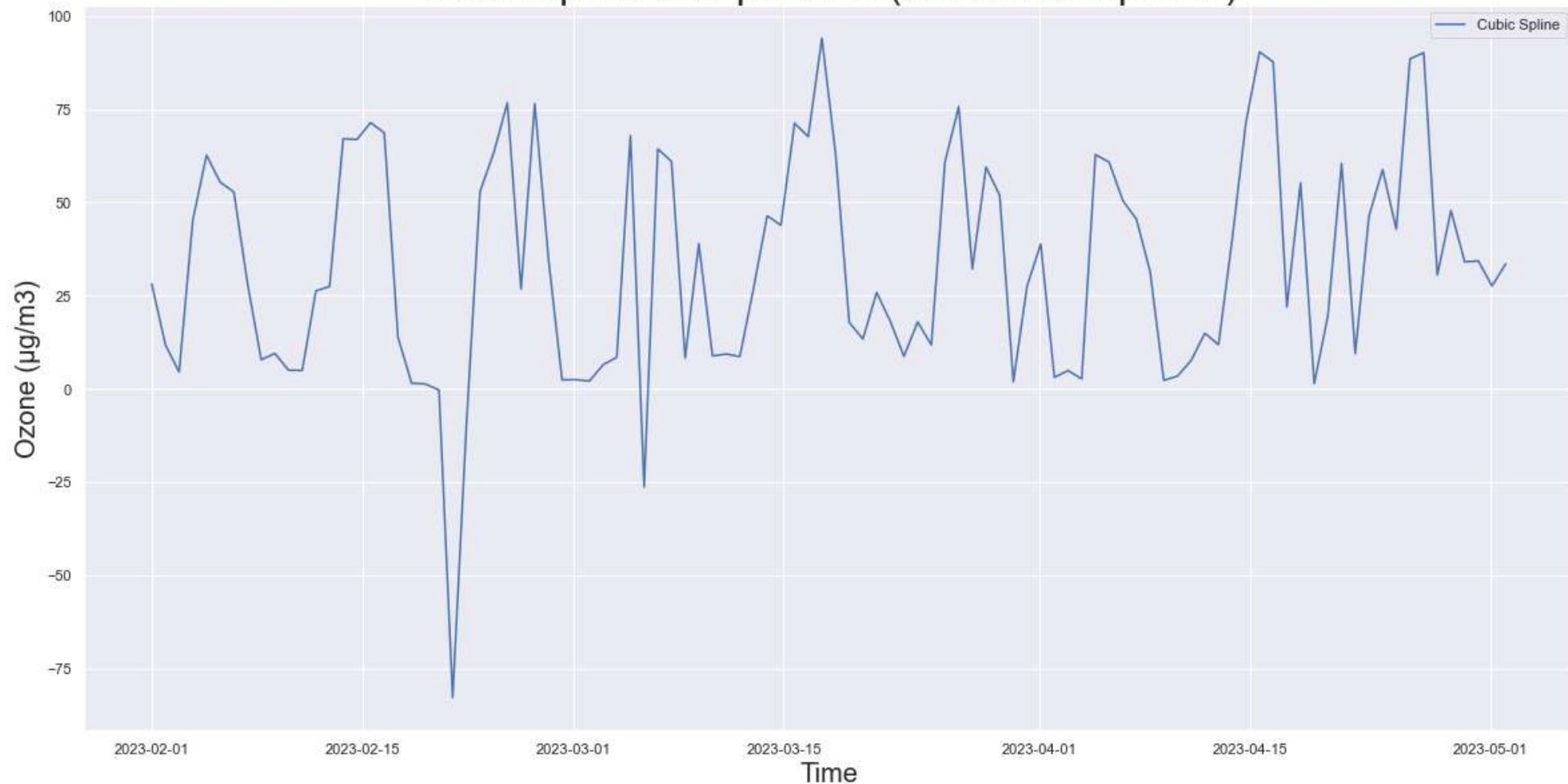


In [224]: `cubicspline(df_n,pollutant)`

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



As is evident from the plots, Cubic and Quadratic interpolations and spline introduce negative values which is practically not feasible. Therefore linear interpolation works best. Mean is affected by the outliers as is evident from histogram and boxplots.

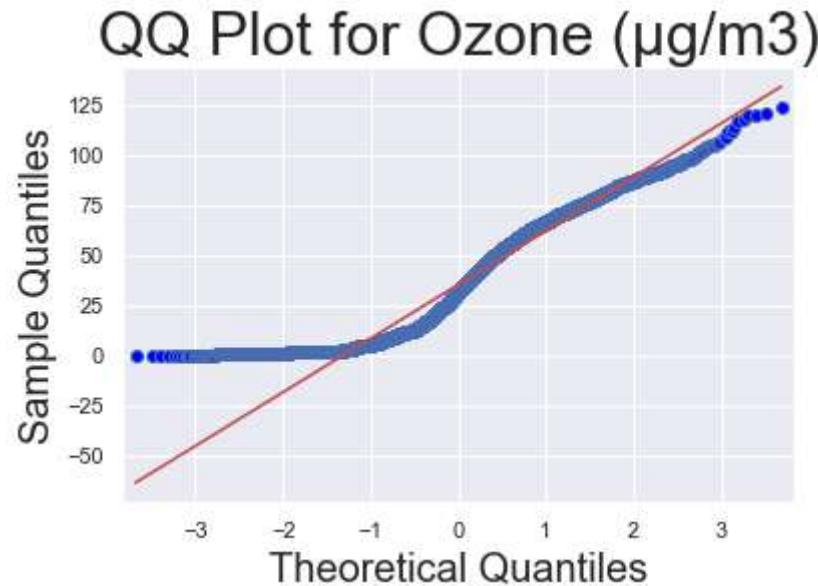
In [225]: `df_linear_f[pollutant].isnull().sum()`

Out[225]: 0

In [226]: `df_new[pollutant] = df_linear_f[pollutant]`

```
In [227]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>



```
In [228]: df_new.head()
```

Out[228]:

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)	CO (mg/ m^3)	SO2 ($\mu\text{g}/\text{m}^3$)	NH3 ($\mu\text{g}/\text{m}^3$)	Ozone ($\mu\text{g}/\text{m}^3$)
--	-----------------------------------	------------------------------------	---------------------------------	----------------------------------	-----------	------------------------	----------------------------------	----------------------------------	------------------------------------

Time

2023-02-01 00:00:00	95.0	35.0	18.1	90.1	56.2	0.31	8.2	17.7	28.1
2023-02-01 00:15:00	95.0	35.0	18.1	88.0	55.1	0.33	8.2	18.3	27.1
2023-02-01 00:30:00	95.0	35.0	18.1	87.7	55.2	0.38	8.2	19.7	24.9
2023-02-01 00:45:00	122.0	34.0	18.1	88.9	55.7	0.38	8.2	21.3	21.9
2023-02-01 01:00:00	122.0	34.0	18.1	90.0	55.8	0.38	8.2	22.3	16.7

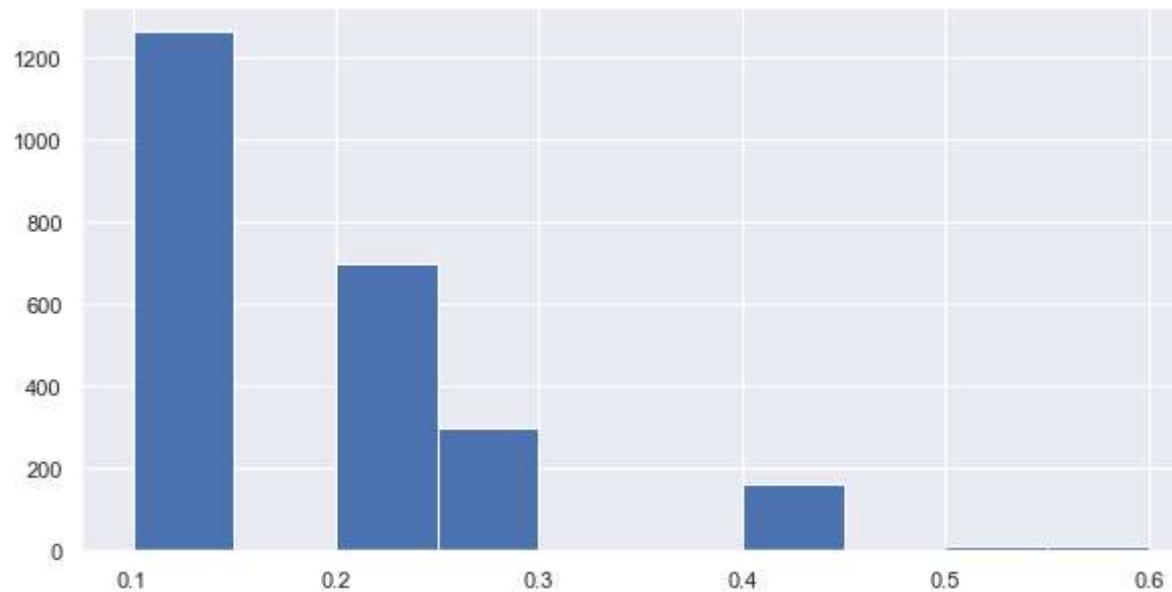
Analysis for "Benzene ($\mu\text{g}/\text{m}^3$)"

```
In [229]: pollutant = 'Benzene (\u03bcg/m\u00b3)'
```

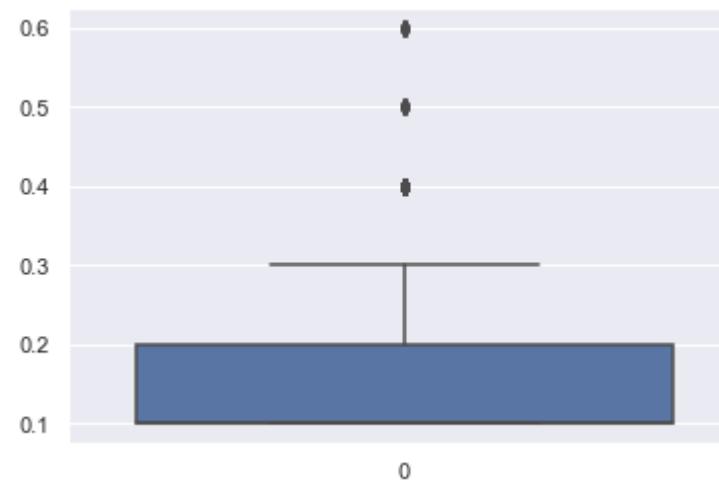
```
In [230]: df[pollutant].describe()
```

```
Out[230]: count    2445.000000
mean      0.177505
std       0.098895
min      0.100000
25%      0.100000
50%      0.100000
75%      0.200000
max      0.600000
Name: Benzene (\u00b5g/m\u00b3), dtype: float64
```

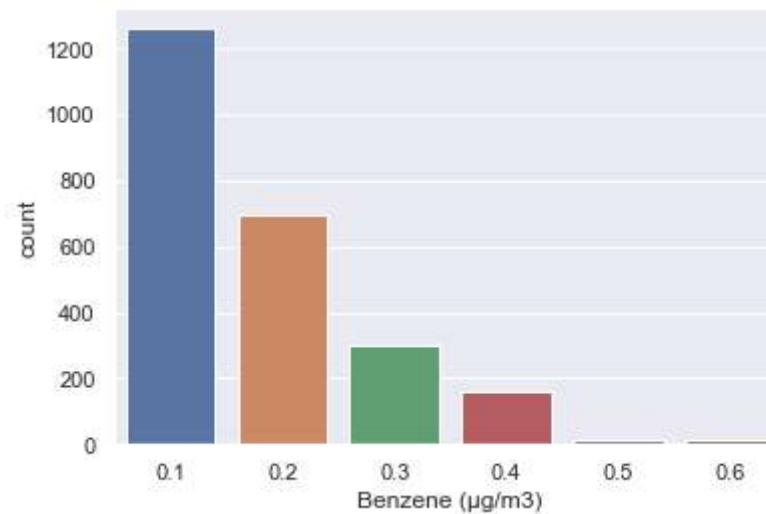
```
In [231]: histogram_plot(df_n,pollutant)
```



```
In [232]: boxplot_plot(df_n,pollutant)
```

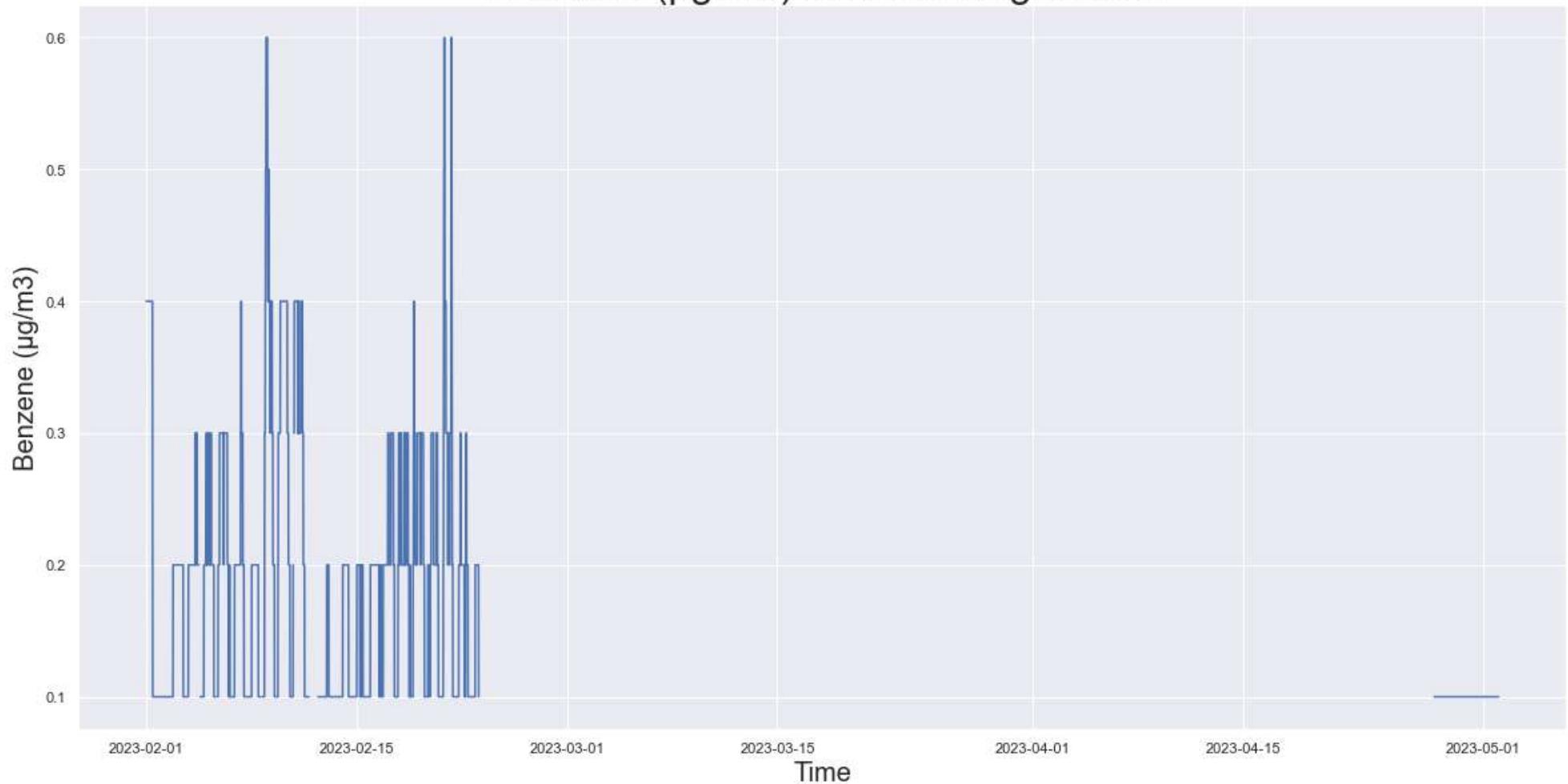


```
In [233]: countplot_plot(df_n,pollutant)
```



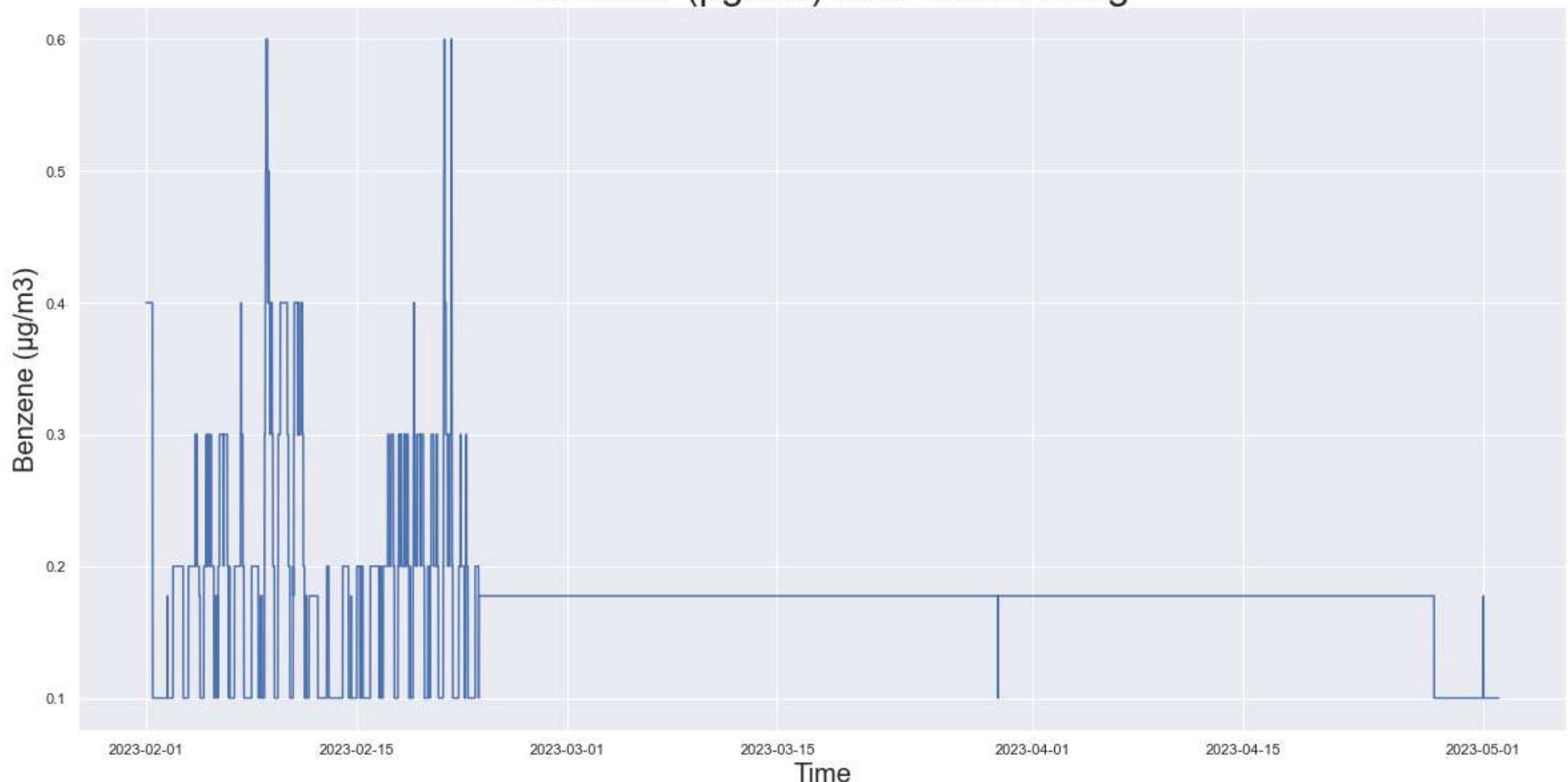
```
In [234]: simple_time_plot(df_n,pollutant,"With missing values")
```

Benzene ($\mu\text{g}/\text{m}^3$) With missing values



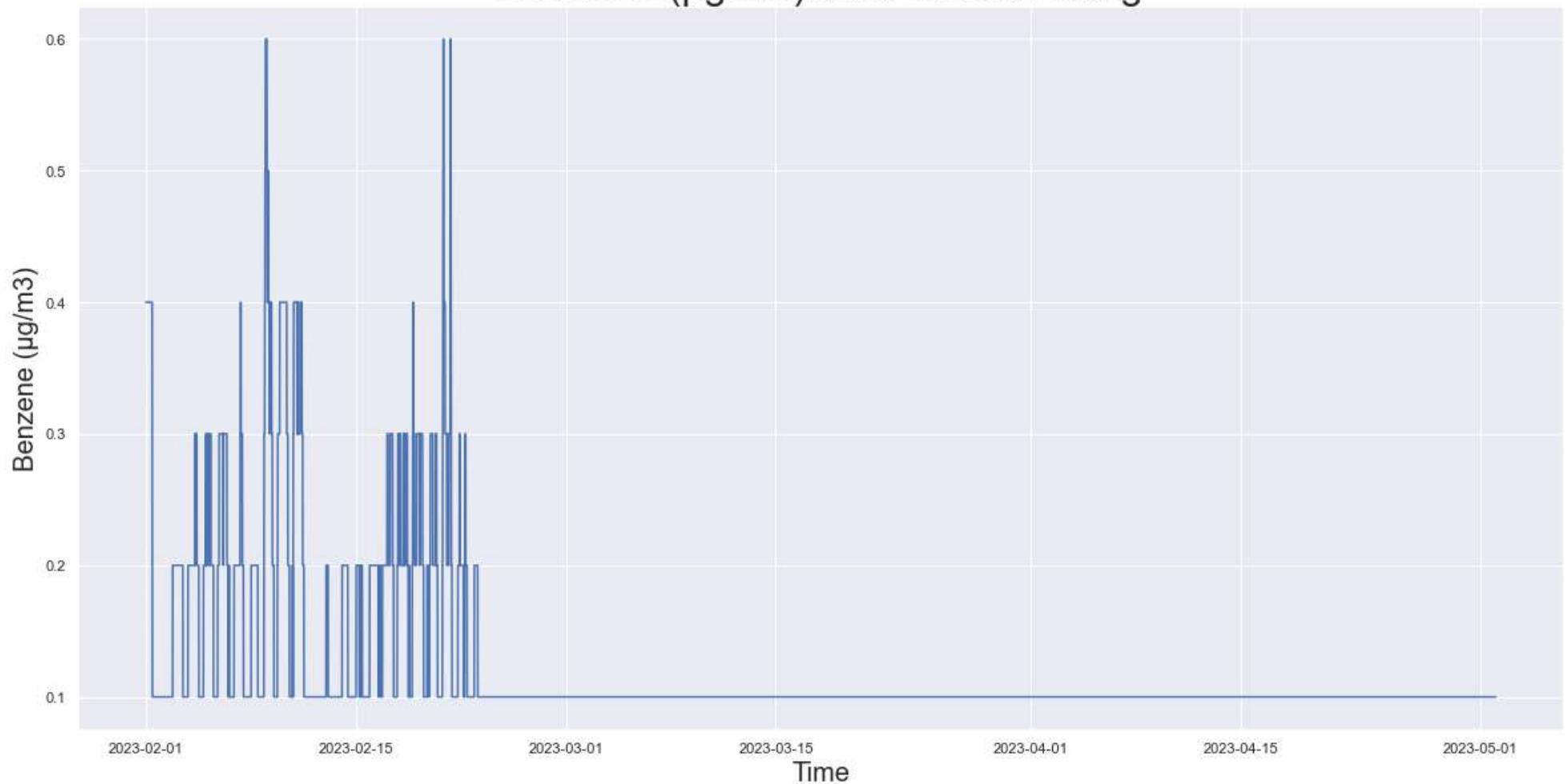
```
In [235]: simple_time_plot(df_mean,pollutant,"after mean filling")
```

Benzene ($\mu\text{g}/\text{m}^3$) after mean filling



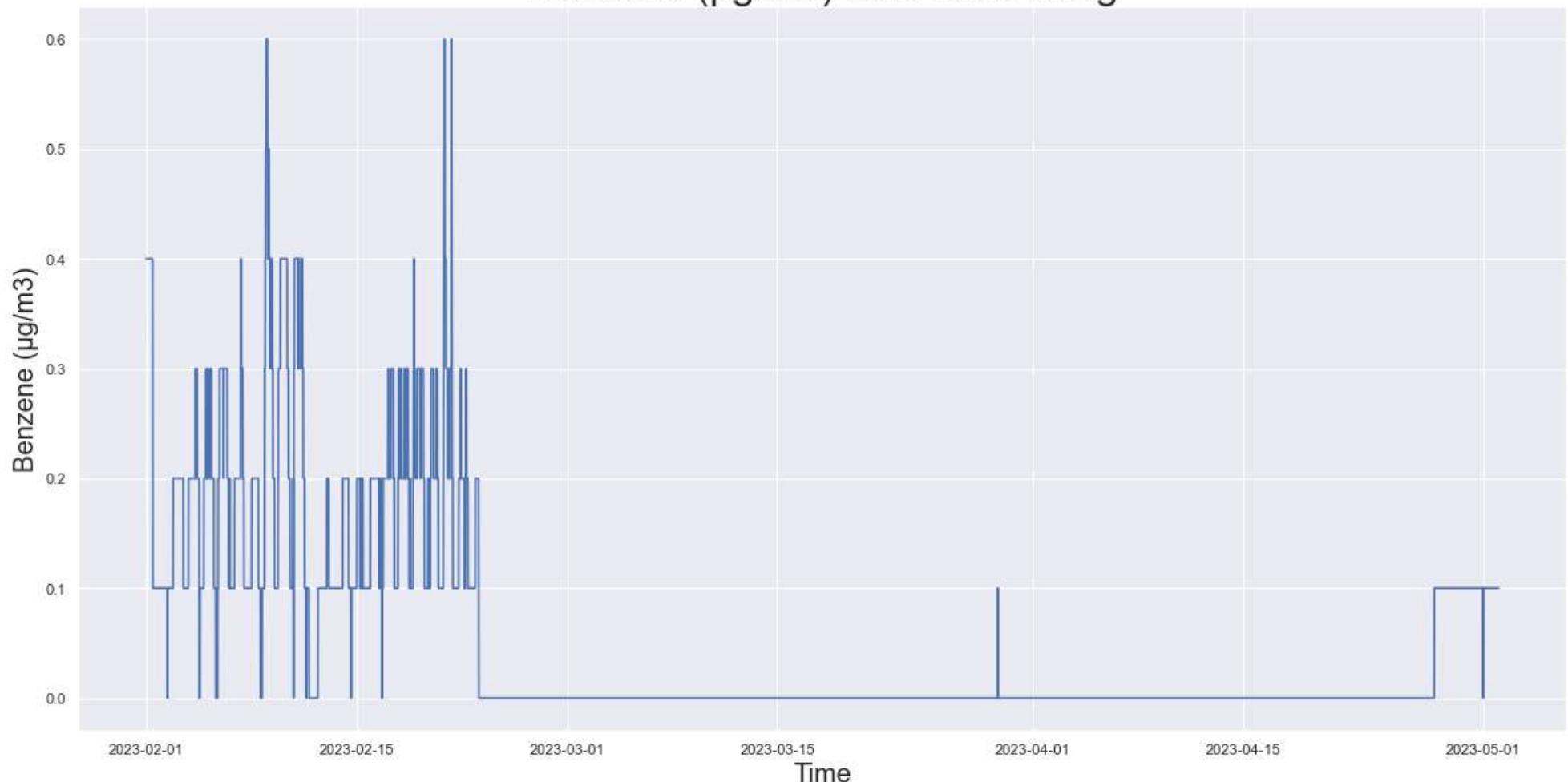
```
In [236]: simple_time_plot(df_median,pollutant,"after median filling")
```

Benzene ($\mu\text{g}/\text{m}^3$) after median filling

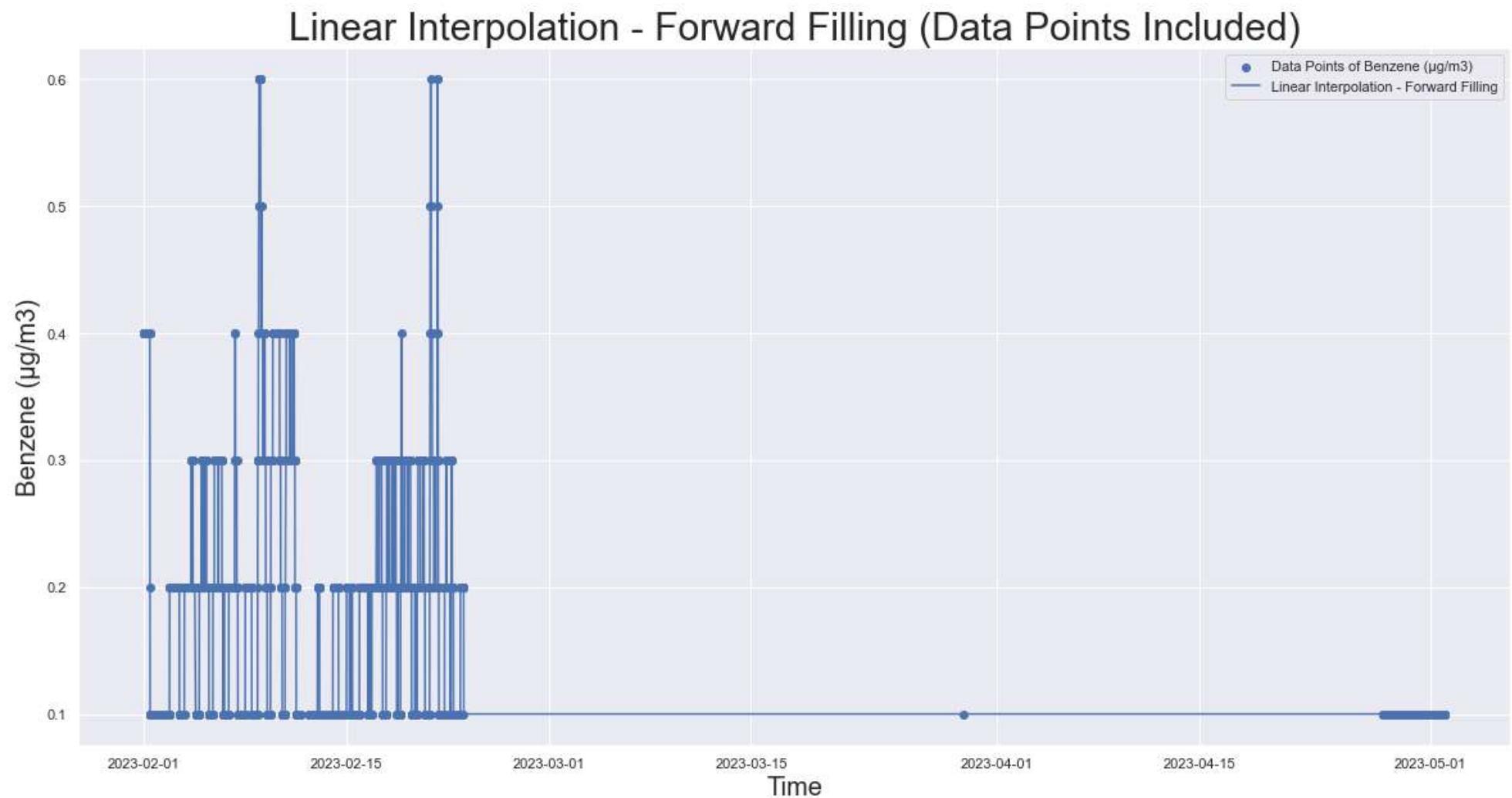


```
In [237]: simple_time_plot(df_zero,pollutant,"after zero filling")
```

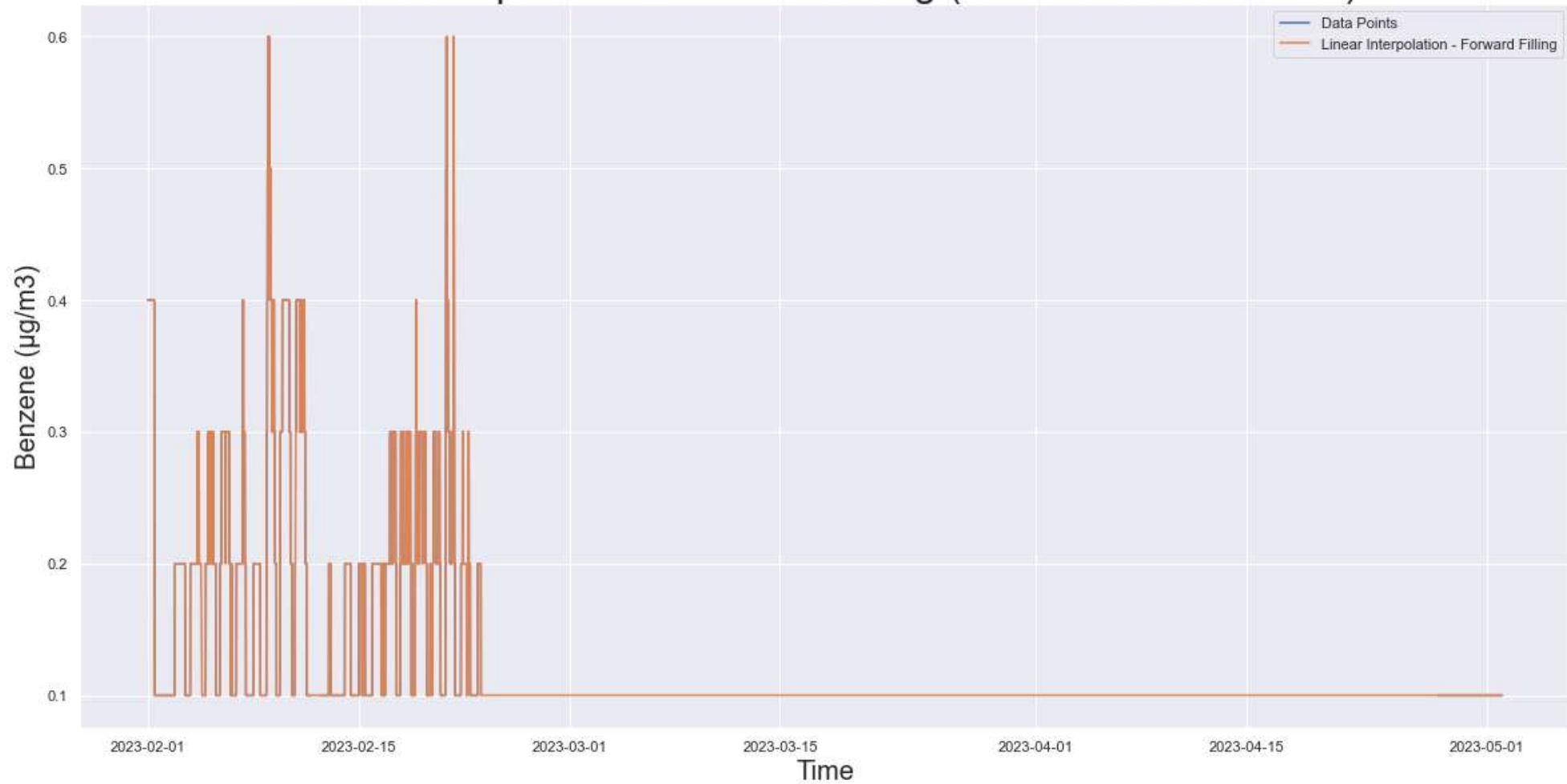
Benzene ($\mu\text{g}/\text{m}^3$) after zero filling



```
In [238]: interpolation_plot(df_n,df_linear_f,pollutant,"Linear Interpolation - Forward Filling")
```

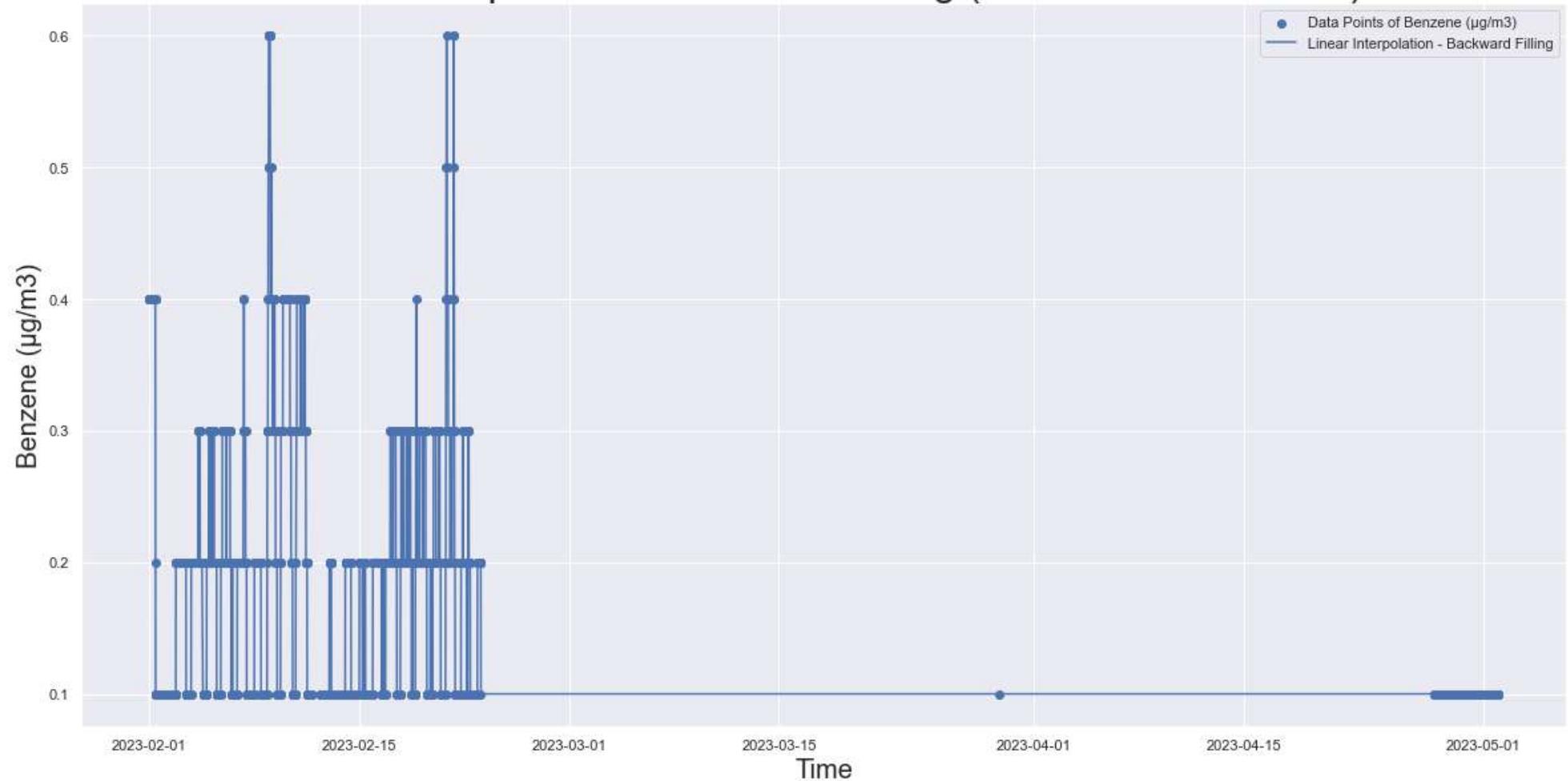


Linear Interpolation - Forward Filling (Data Points Excluded)

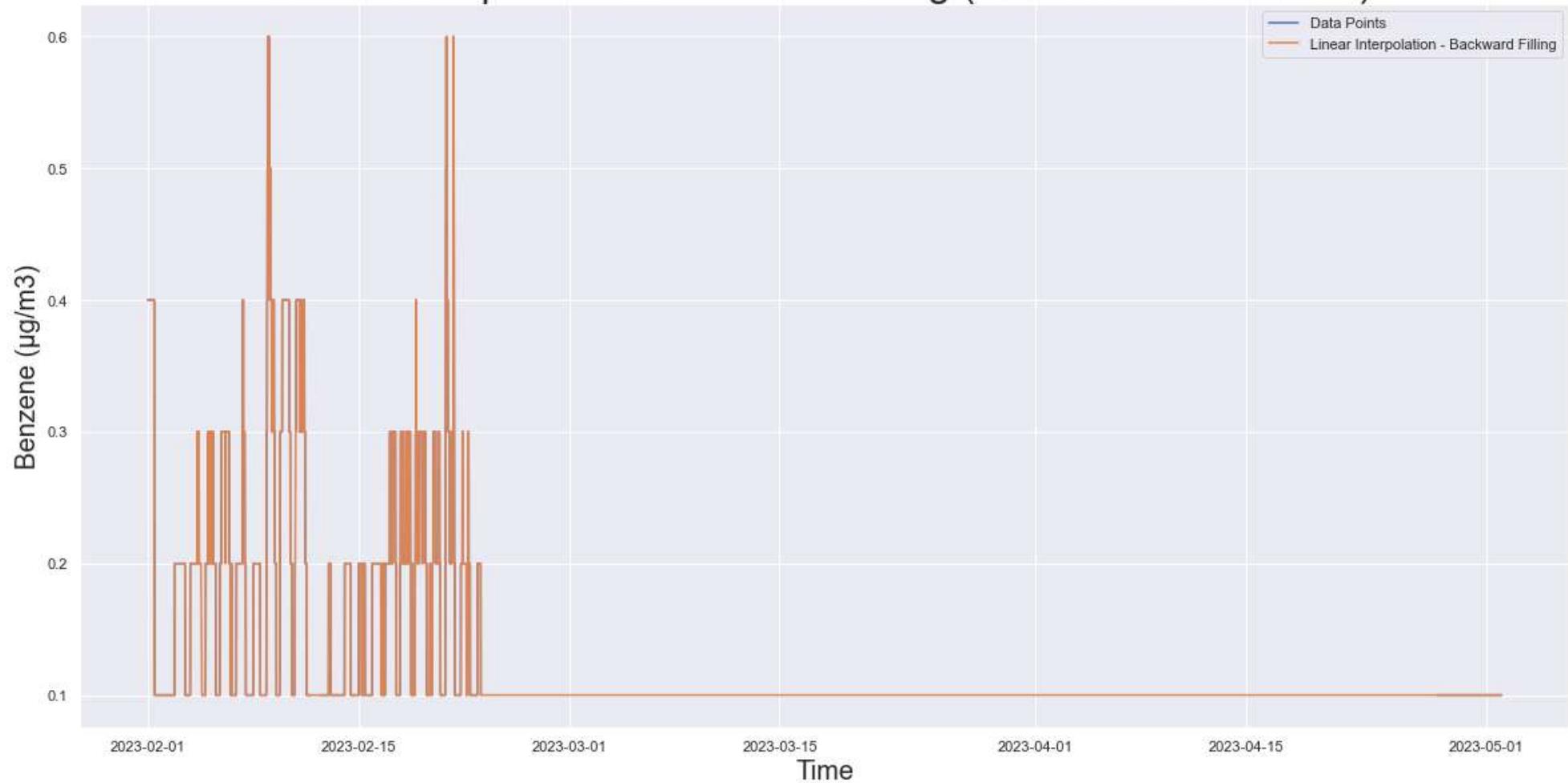


```
In [239]: interpolation_plot(df_n,df_linear_b,pollutant,"Linear Interpolation - Backward Filling")
```

Linear Interpolation - Backward Filling (Data Points Included)

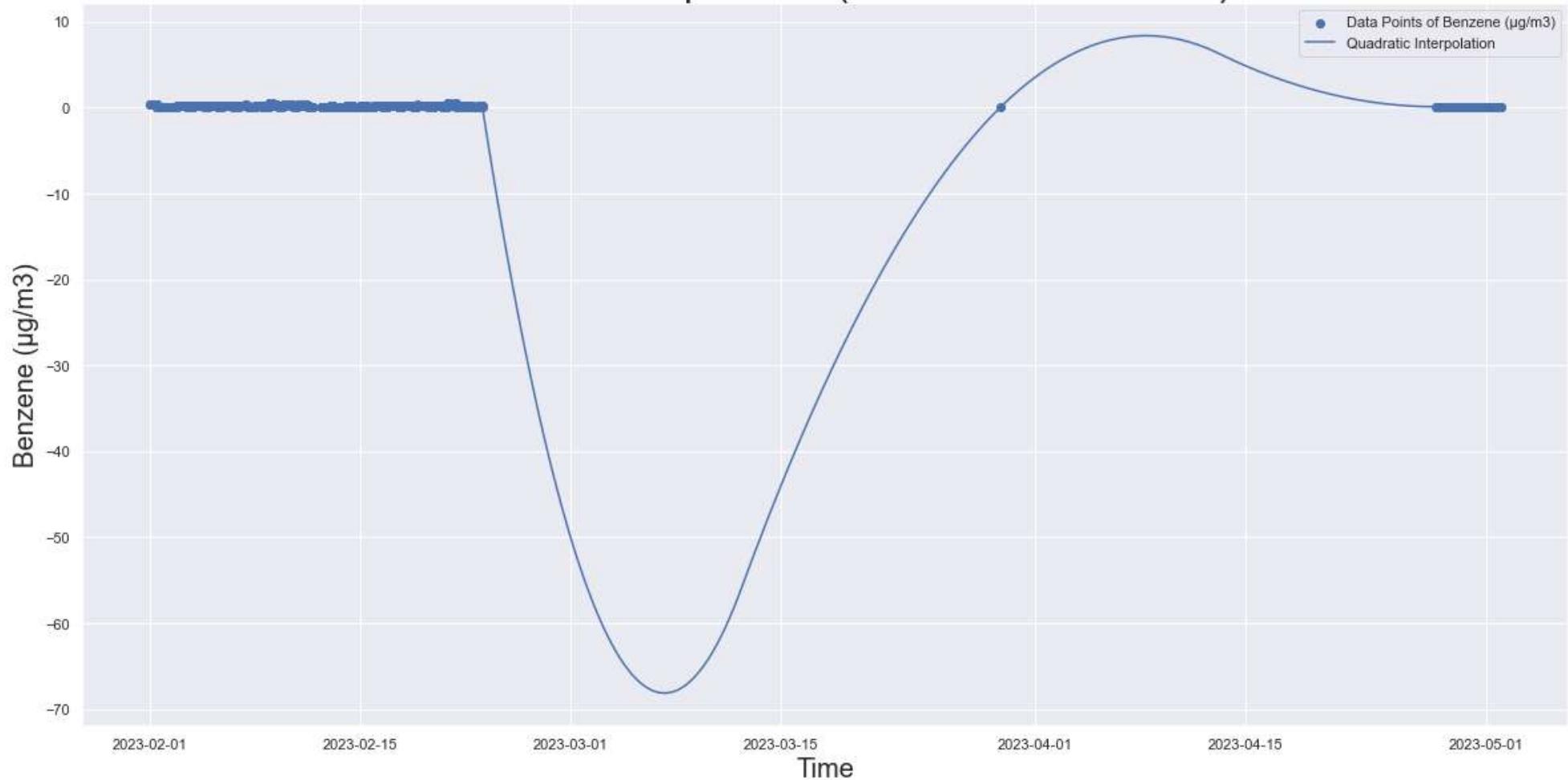


Linear Interpolation - Backward Filling (Data Points Excluded)

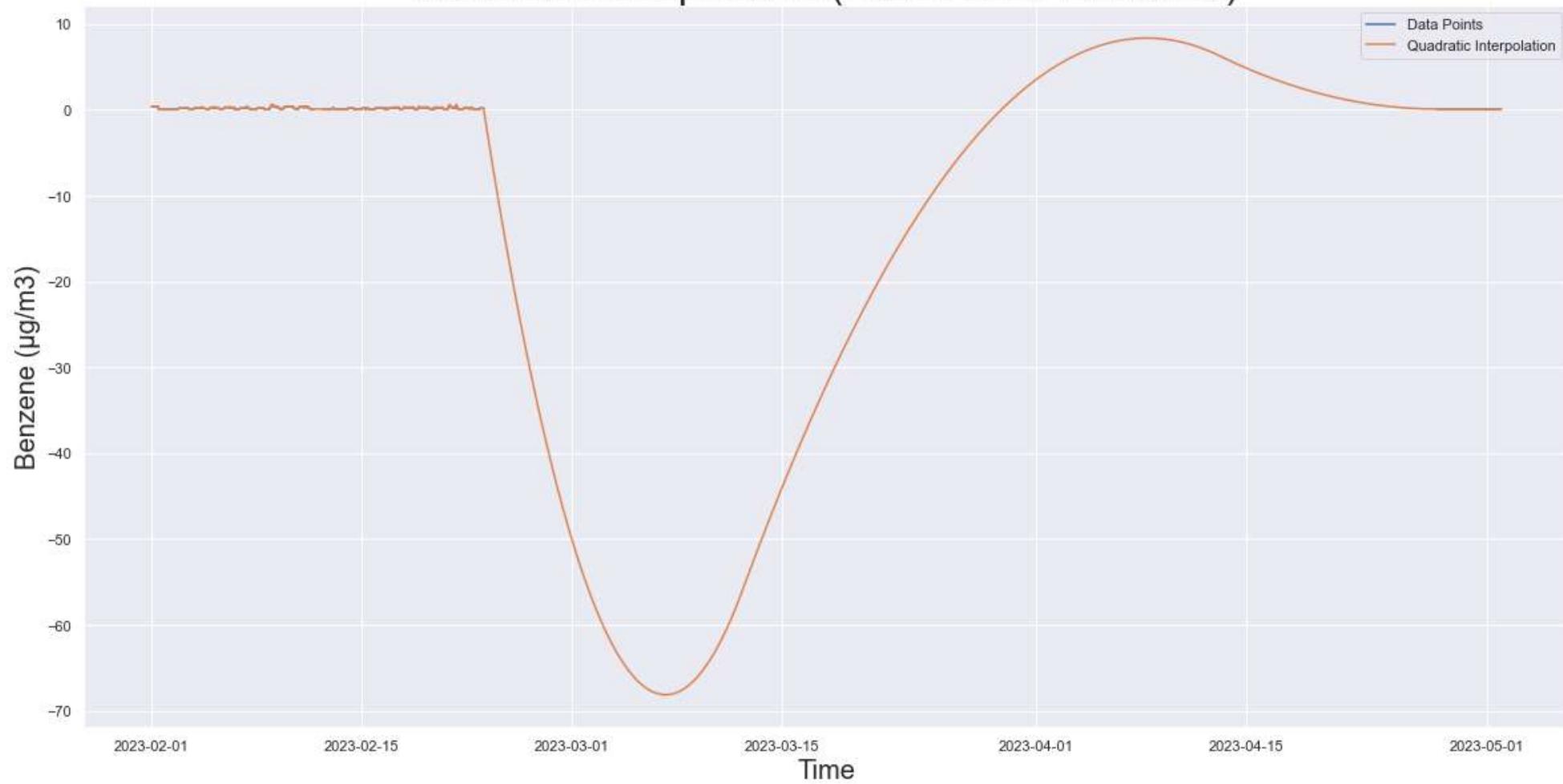


```
In [240]: interpolation_plot(df_n,df_quad,pollutant,"Quadratic Interpolation")
```

Quadratic Interpolation (Data Points Included)

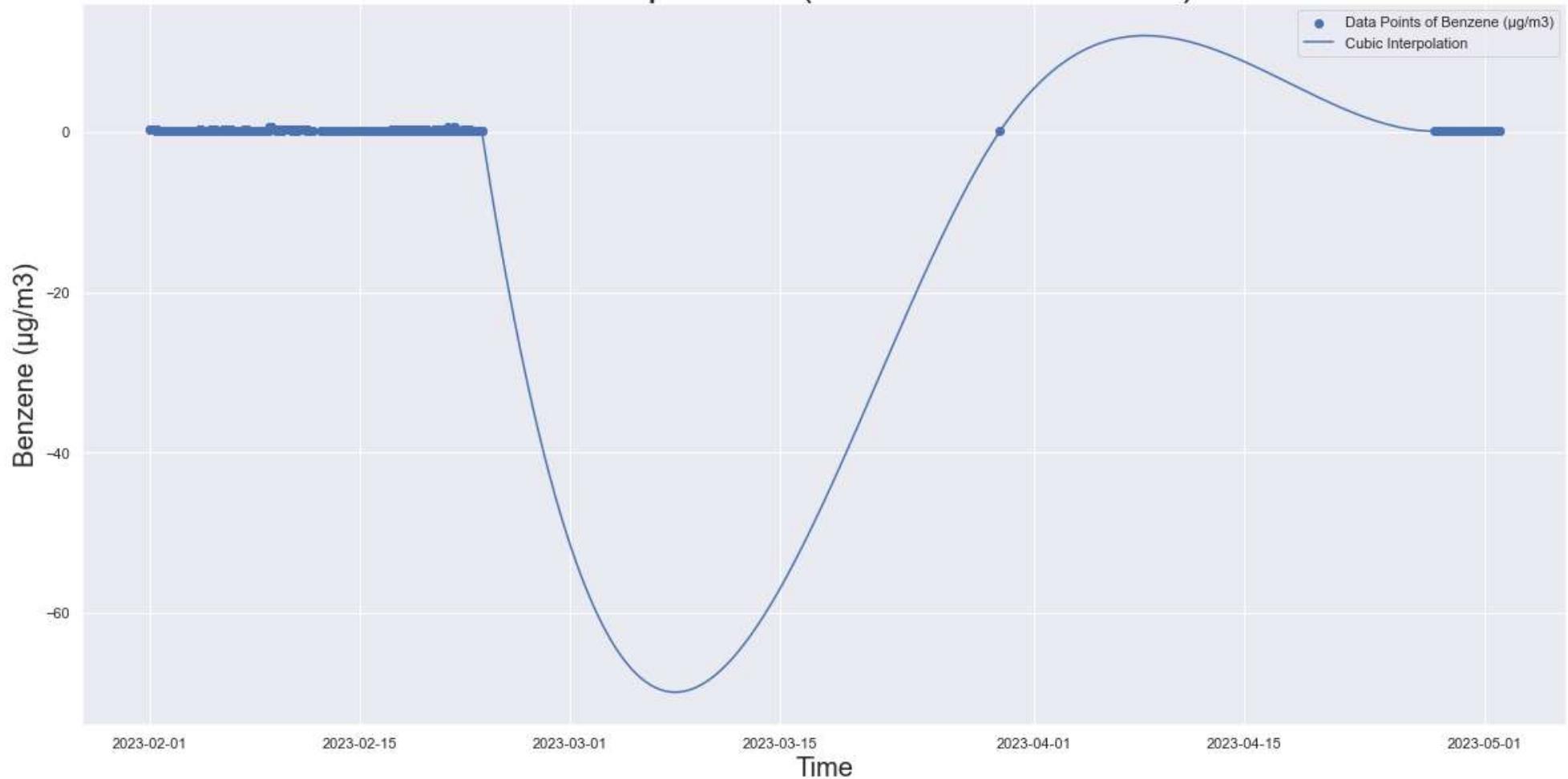


Quadratic Interpolation (Data Points Excluded)

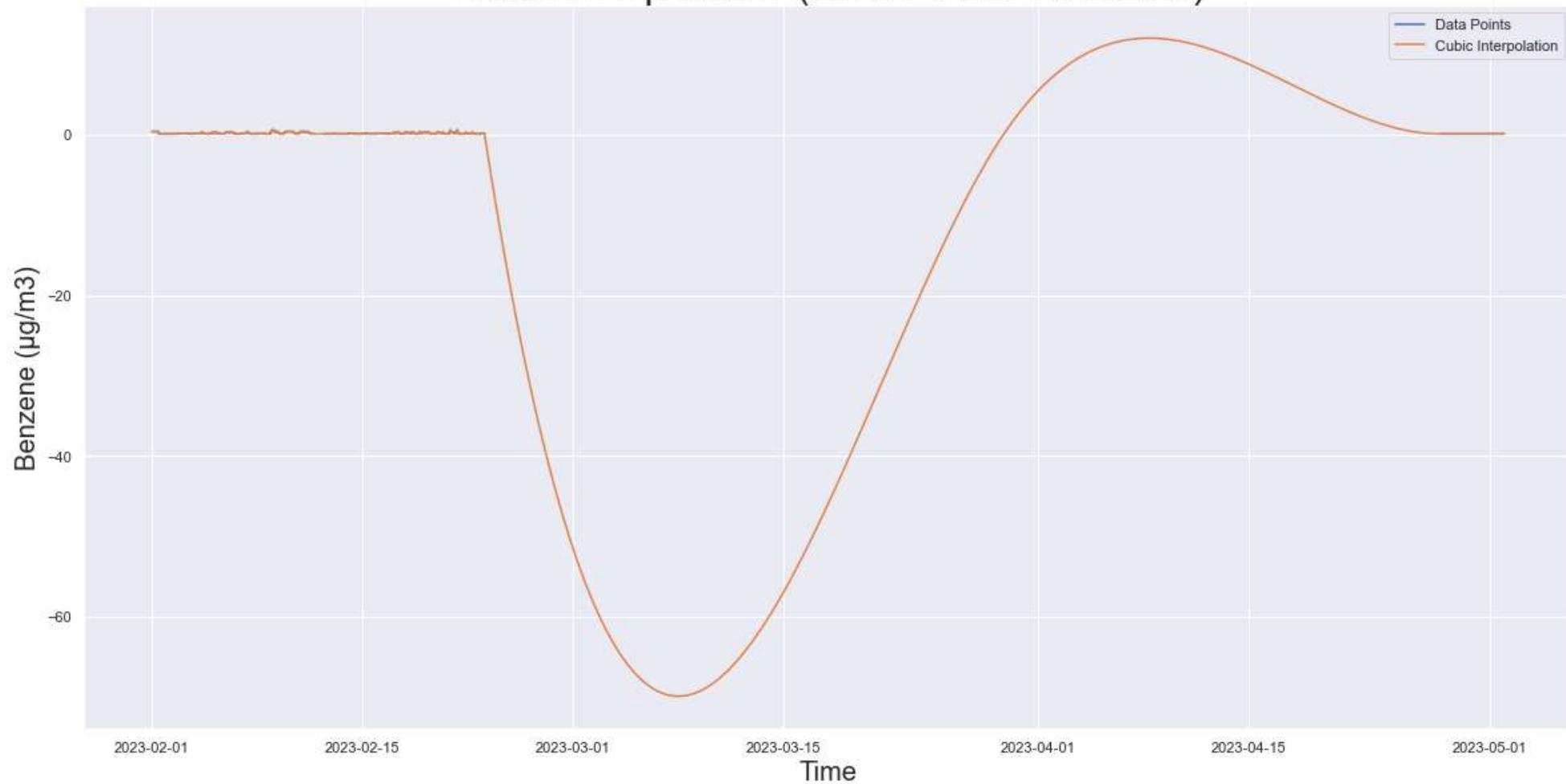


```
In [241]: interpolation_plot(df_n,df_cubic,pollutant,"Cubic Interpolation")
```

Cubic Interpolation (Data Points Included)

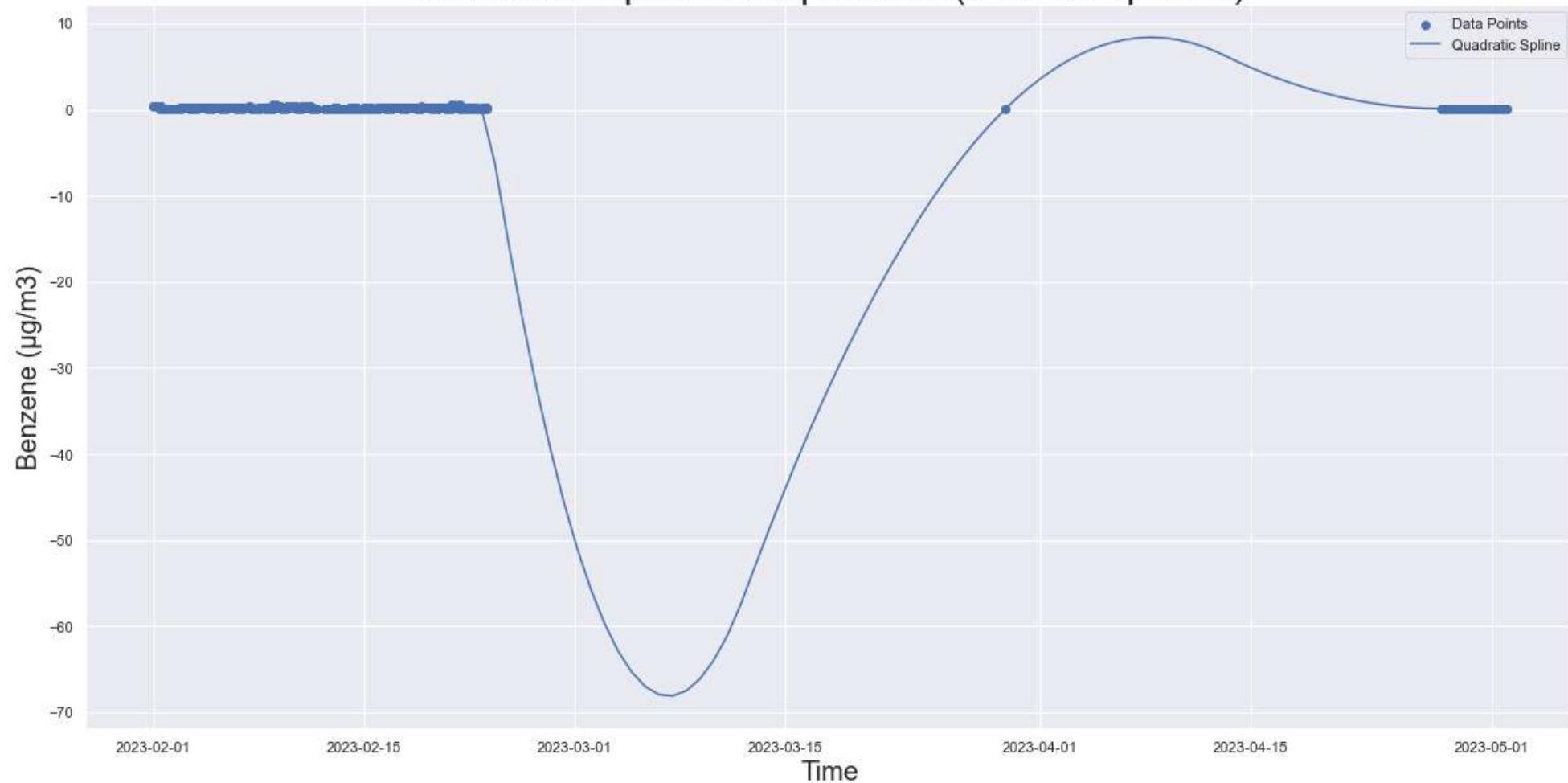


Cubic Interpolation (Data Points Excluded)

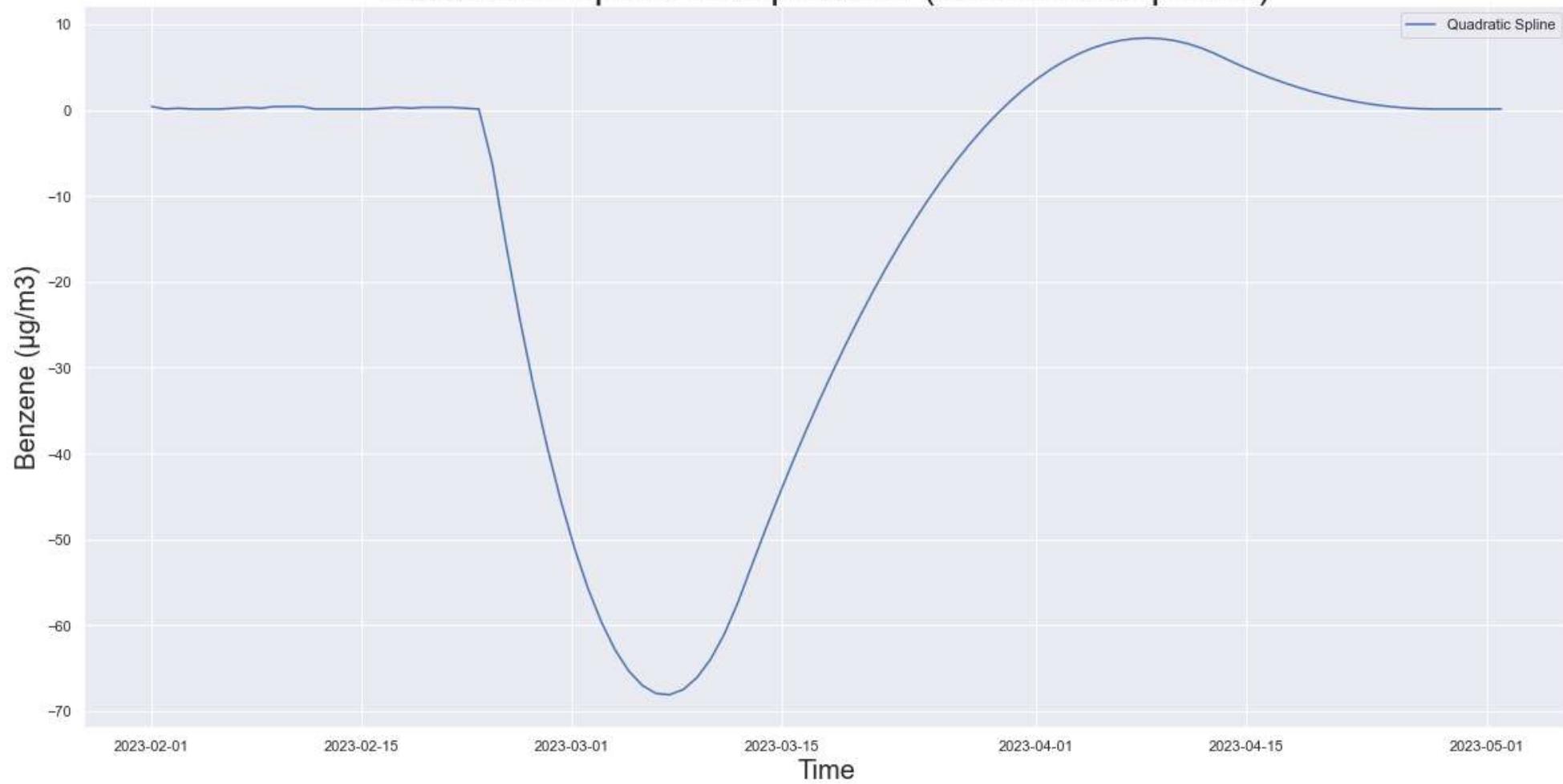


```
In [242]: quadraticspline(df_n,pollutant)
```

Quadratic Spline Interpolation (with datapoints)

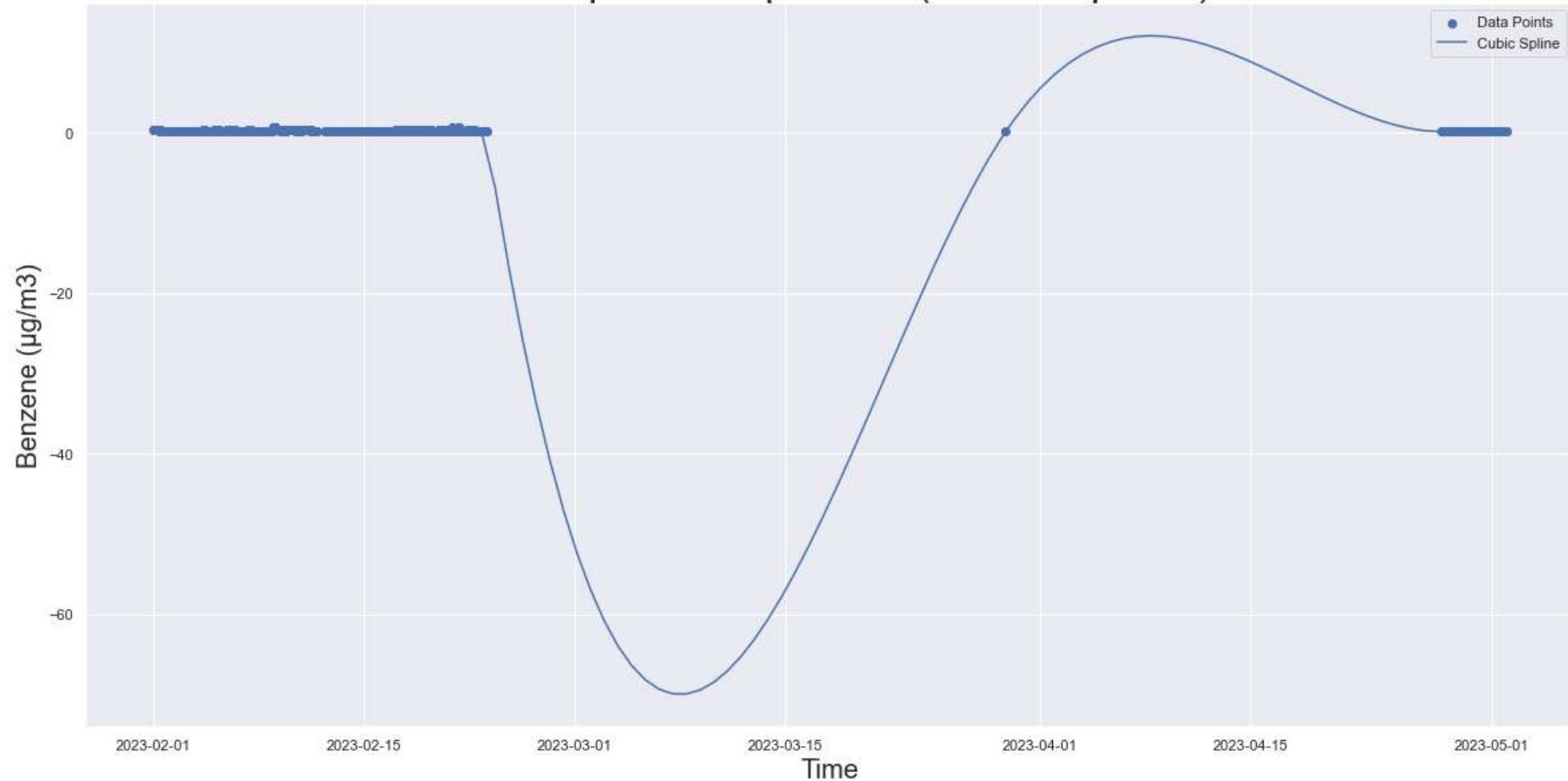


Quadratic Spline Interpolation (without datapoints)

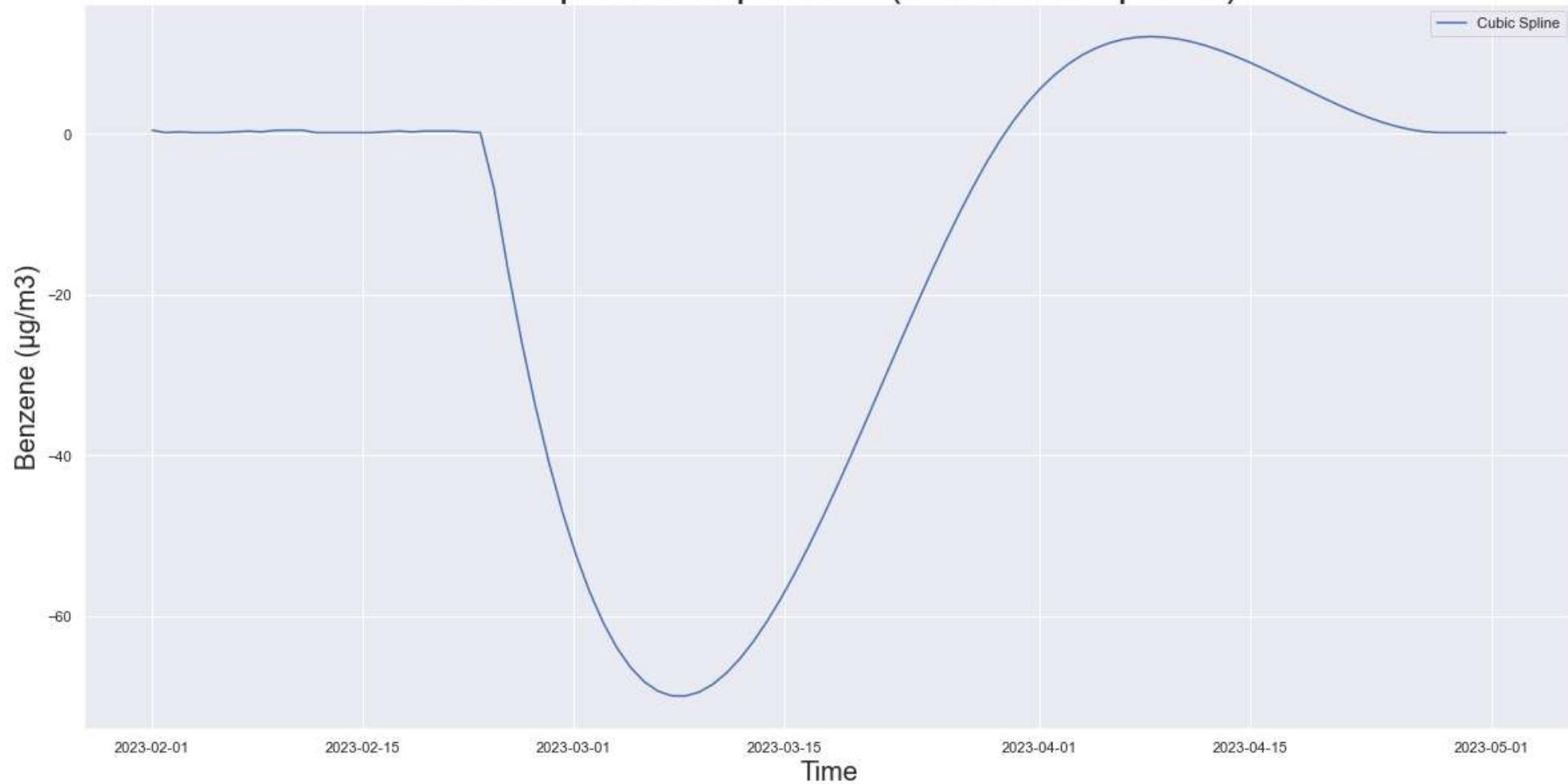


```
In [243]: cubicspline(df_n,pollutant)
```

Cubic Spline Interpolation (with datapoints)



Cubic Spline Interpolation (without datapoints)



As is evident from the plots, Cubic and Quadratic interpolations and spline introduce negative values which is practically not feasible. Therefore linear interpolation if used introduces unnecessary complexities. Mean is affected by the outliers as is evident from histogram and boxplots. Zero filling works best as most of the data is concentrated around zero and mean is also very close to zero.

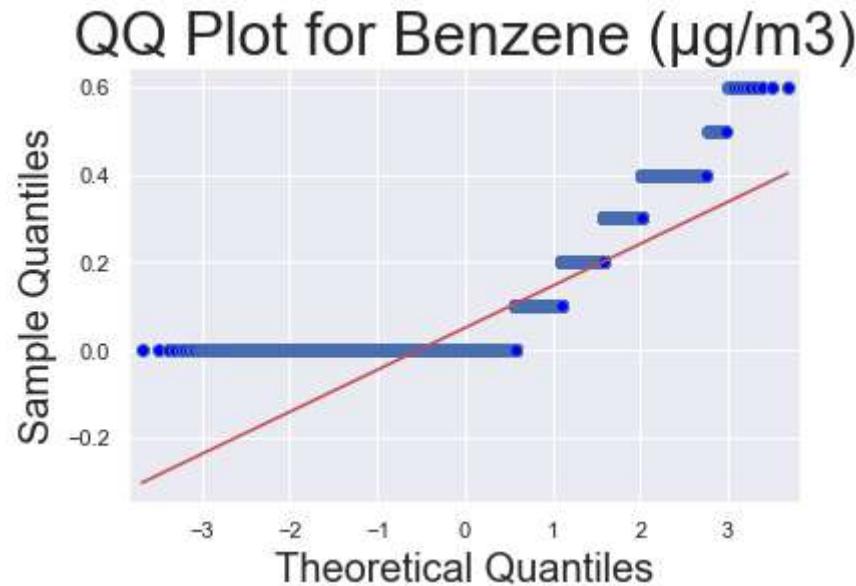
In [244]: `df_zero[pollutant].isnull().sum()`

Out[244]: 0

In [245]: `df_new[pollutant] = df_zero[pollutant]`

```
In [246]: qq_plot(df_new,pollutant)
```

<Figure size 1440x720 with 0 Axes>



```
In [247]: df_new.head()
```

Out[247]:

Time	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)	CO (mg/m^3)	SO2 ($\mu\text{g}/\text{m}^3$)	NH3 ($\mu\text{g}/\text{m}^3$)	Ozone ($\mu\text{g}/\text{m}^3$)	Benzene ($\mu\text{g}/\text{m}^3$)
2023-02-01 00:00:00	95.0	35.0	18.1	90.1	56.2	0.31	8.2	17.7	28.1	0.4
2023-02-01 00:15:00	95.0	35.0	18.1	88.0	55.1	0.33	8.2	18.3	27.1	0.4
2023-02-01 00:30:00	95.0	35.0	18.1	87.7	55.2	0.38	8.2	19.7	24.9	0.4
2023-02-01 00:45:00	122.0	34.0	18.1	88.9	55.7	0.38	8.2	21.3	21.9	0.4
2023-02-01 01:00:00	122.0	34.0	18.1	90.0	55.8	0.38	8.2	22.3	16.7	0.4

```
In [248]: df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8640 entries, 2023-02-01 00:00:00 to 2023-05-01 23:45:00
Freq: 15T
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   PM10 (\u00b5g/m\u00b3)    8640 non-null   float64
 1   PM2.5 (\u00b5g/m\u00b3)   8640 non-null   float64
 2   NO (\u00b5g/m\u00b3)      8640 non-null   float64
 3   NO2 (\u00b5g/m\u00b3)     8640 non-null   float64
 4   NOX (ppb)            8640 non-null   float64
 5   CO (mg/m\u00b3)         8640 non-null   float64
 6   SO2 (\u00b5g/m\u00b3)     8640 non-null   float64
 7   NH3 (\u00b5g/m\u00b3)     8640 non-null   float64
 8   Ozone (\u00b5g/m\u00b3)    8640 non-null   float64
 9   Benzene (\u00b5g/m\u00b3)  8640 non-null   float64
dtypes: float64(10)
memory usage: 1000.5 KB
```

```
In [249]: df_new.columns
```

```
Out[249]: Index(['PM10 (\u00b5g/m\u00b3)', 'PM2.5 (\u00b5g/m\u00b3)', 'NO (\u00b5g/m\u00b3)', 'NO2 (\u00b5g/m\u00b3)',  
                 'NOX (ppb)', 'CO (mg/m\u00b3)', 'SO2 (\u00b5g/m\u00b3)', 'NH3 (\u00b5g/m\u00b3)',  
                 'Ozone (\u00b5g/m\u00b3)', 'Benzene (\u00b5g/m\u00b3)'],
                 dtype='object')
```

```
In [250]: df_new.shape
```

```
Out[250]: (8640, 10)
```

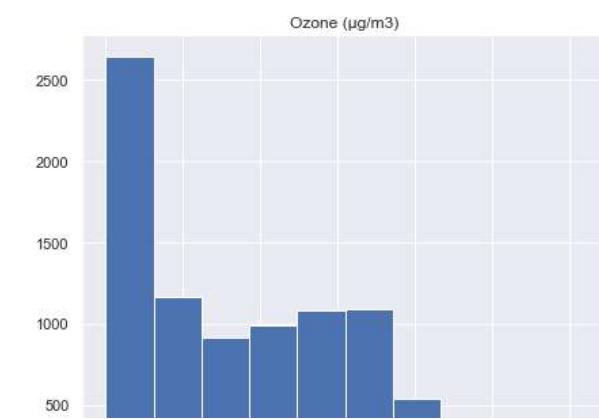
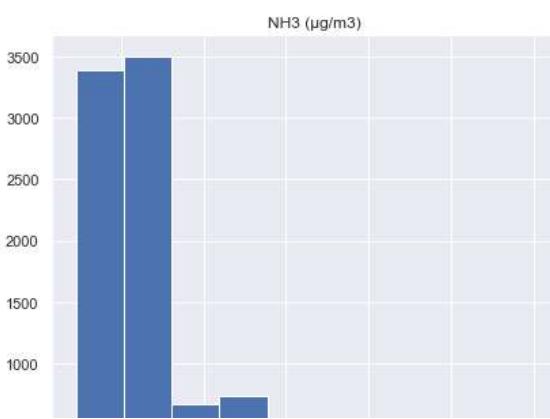
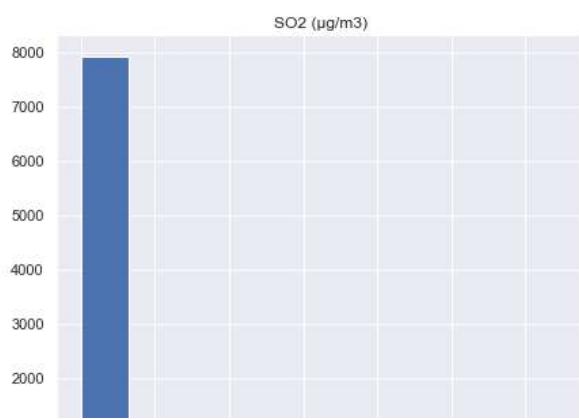
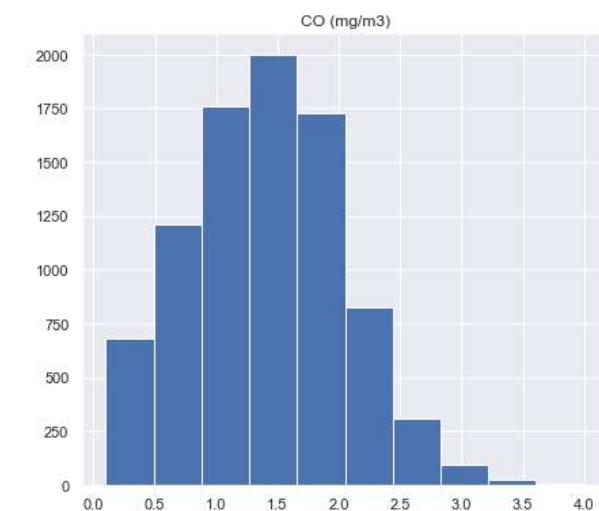
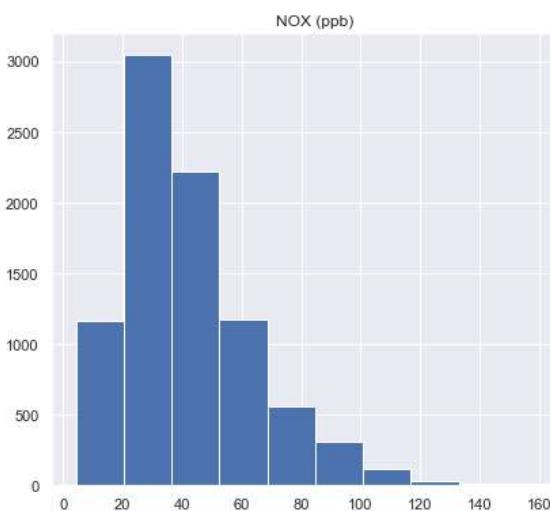
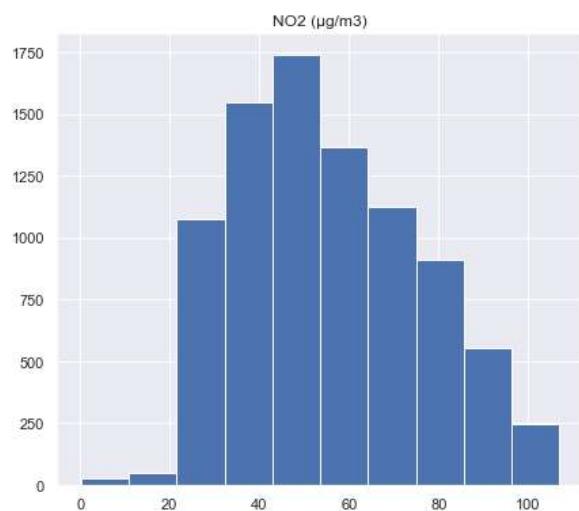
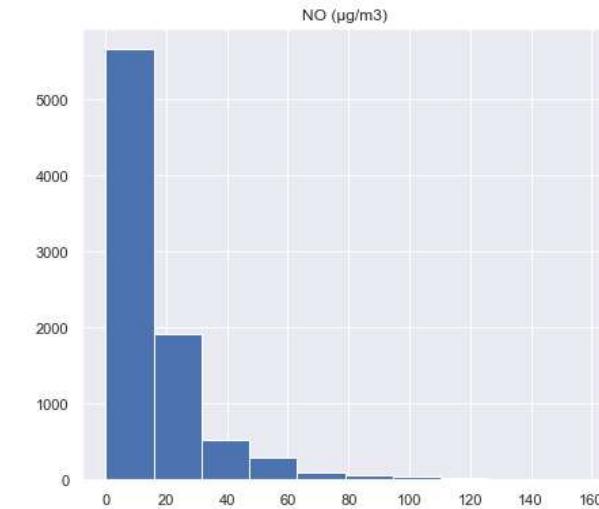
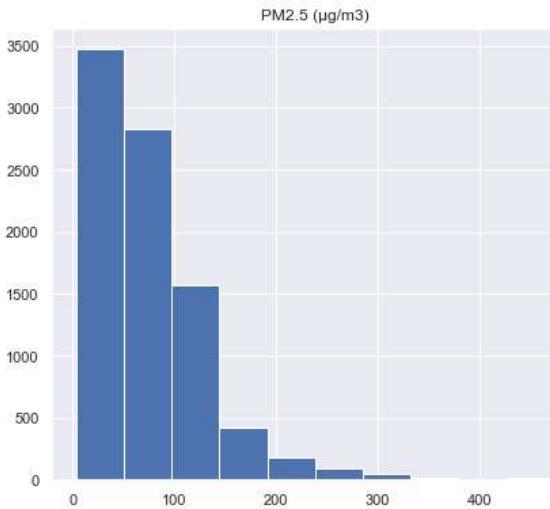
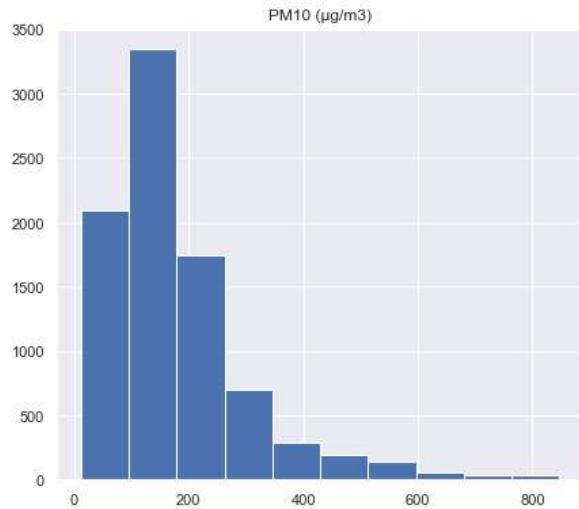
In [251]: df_new.describe()

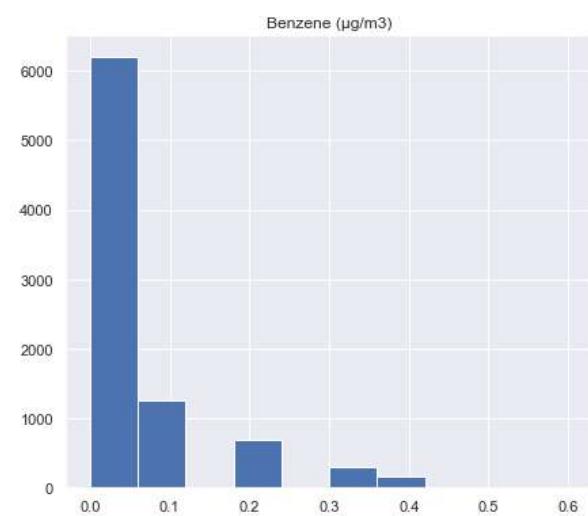
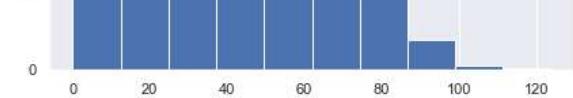
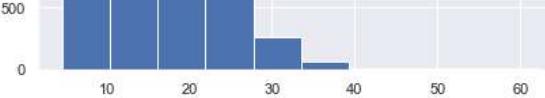
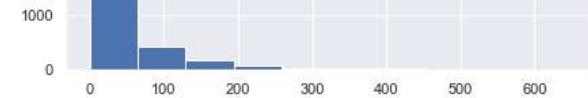
Out[251]:

	PM10 (µg/m3)	PM2.5 (µg/m3)	NO (µg/m3)	NO2 (µg/m3)	NOX (ppb)	CO (mg/m3)	SO2 (µg/m3)	NH3 (µg/m3)	Ozone (µg/m3)	Benzene (µg/m3)
count	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000
mean	177.463079	75.557350	14.940208	55.430689	42.235689	1.401927	31.926226	13.286956	35.193970	0.050231
std	124.773568	54.826850	17.862679	20.205531	22.210691	0.633087	39.027640	6.171697	26.867859	0.095712
min	12.000000	3.000000	0.100000	0.200000	4.200000	0.100000	0.100000	4.600000	0.100000	0.000000
25%	97.000000	36.821429	4.000000	39.300000	24.875000	0.950000	12.300000	9.500000	10.300000	0.000000
50%	151.900794	61.000000	7.500000	52.800000	37.286754	1.410000	22.800000	11.000000	31.700000	0.000000
75%	215.000000	101.000000	18.100000	70.700000	53.100000	1.850000	33.400000	14.000000	58.100000	0.100000
max	847.000000	474.000000	157.500000	106.900000	165.200000	4.000000	645.600000	62.400000	123.800000	0.600000

```
In [252]: df_new.hist(figsize=(25,30))
```

```
Out[252]: array([[<Axes: title={'center': 'PM10 (\u00b5g/m\u00b3)'>},  
   <Axes: title={'center': 'PM2.5 (\u00b5g/m\u00b3)'>},  
   <Axes: title={'center': 'NO (\u00b5g/m\u00b3)'>}],  
  [<Axes: title={'center': 'NO2 (\u00b5g/m\u00b3)'>},  
   <Axes: title={'center': 'NOX (ppb)'>},  
   <Axes: title={'center': 'CO (\u00b3g/m\u00b3)'>}],  
  [<Axes: title={'center': 'SO2 (\u00b5g/m\u00b3)'>},  
   <Axes: title={'center': 'NH3 (\u00b5g/m\u00b3)'>},  
   <Axes: title={'center': 'Ozone (\u00b5g/m\u00b3)'>}],  
  [<Axes: title={'center': 'Benzene (\u00b5g/m\u00b3)'>, <Axes: >, <Axes: >}],  
 dtype=object)
```

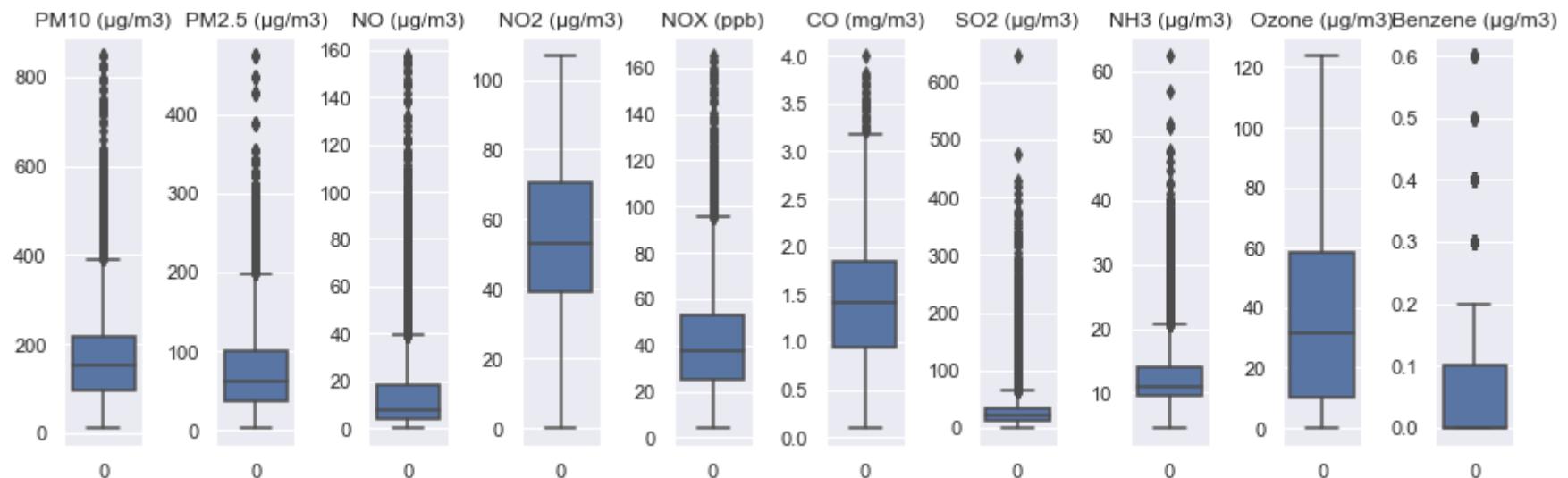


```
In [253]: # Create subplots with a grid layout
fig, axes = plt.subplots(1, len(df_new.columns), figsize=(12, 4))

# Iterate over the columns of the dataframe
for i in range(len(df_new.columns)):
    # Create a boxplot for each column
    sns.boxplot(data=df_new[df_new.columns[i]], ax=axes[i])
    axes[i].set_title(df_new.columns[i])

# Adjust the spacing between subplots
plt.tight_layout()

# Display the plots
plt.show()
```



```
In [254]: num_plots = len(df_new.columns) # Total number of countplots
plots_per_row = 2 # Number of countplots per row
num_rows = (num_plots + plots_per_row - 1) // plots_per_row # Calculate the number of rows

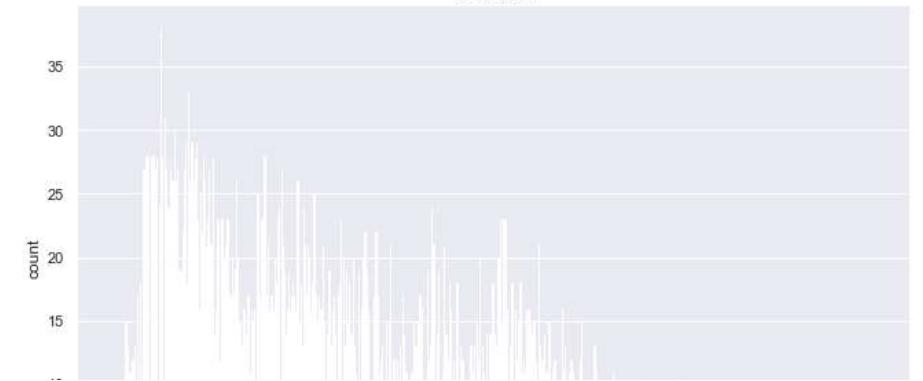
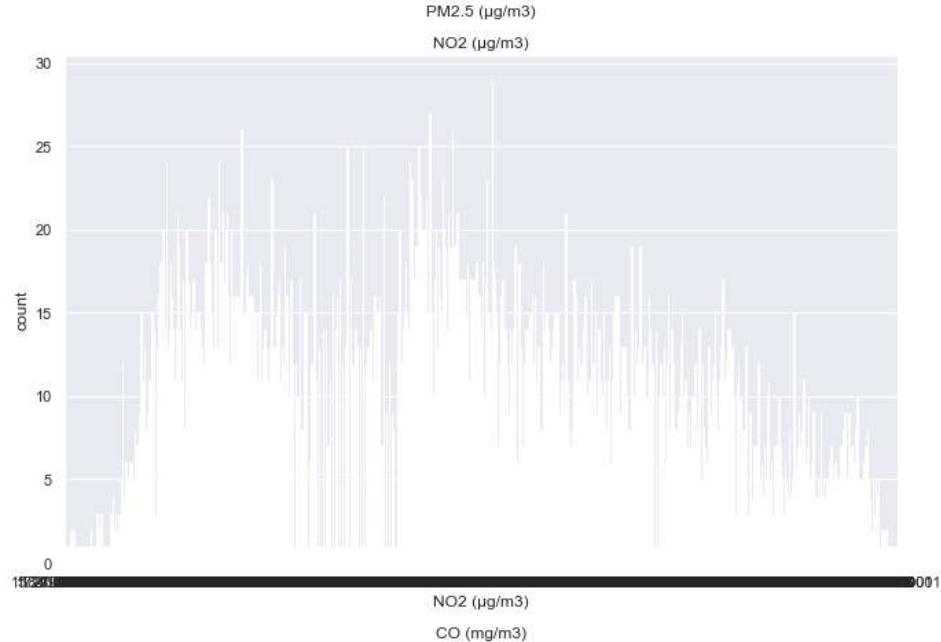
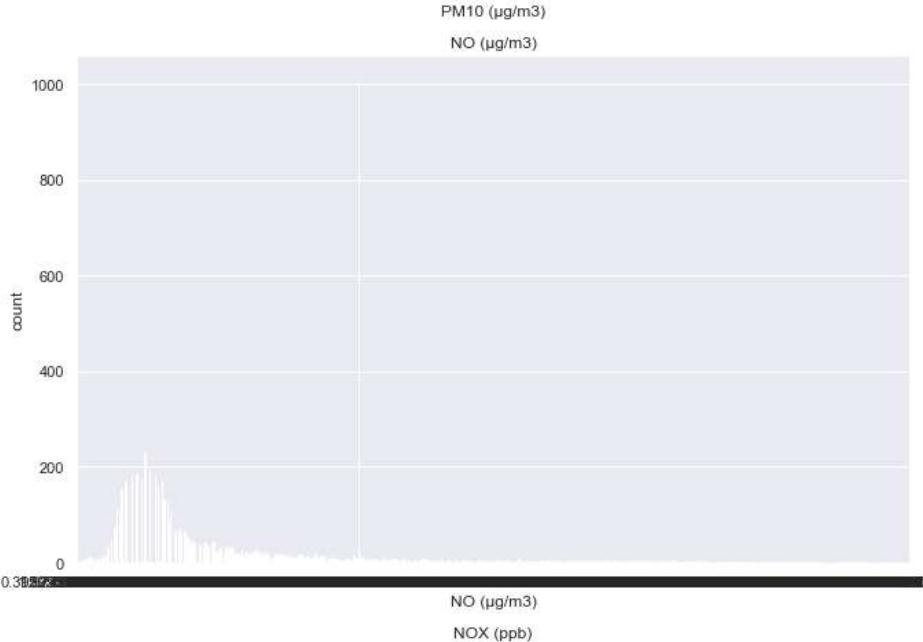
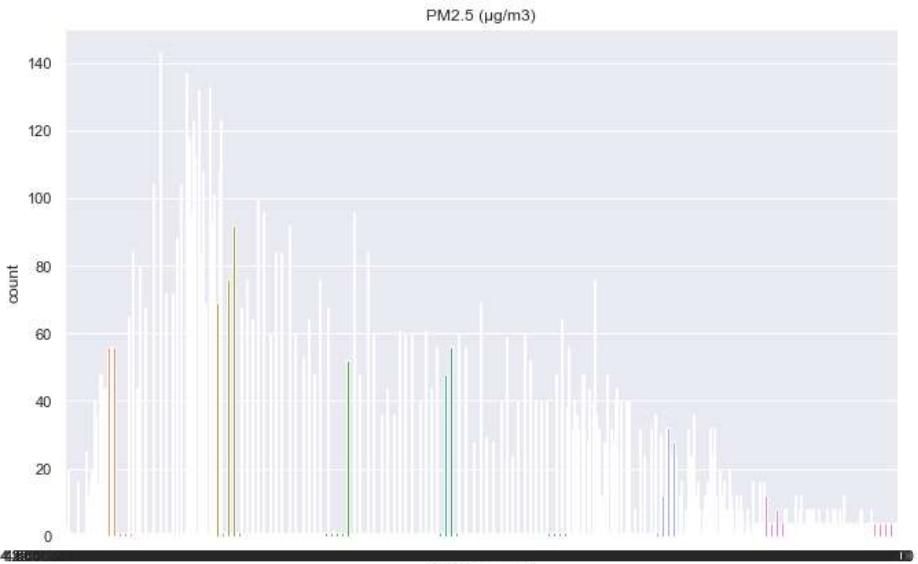
fig, axes = plt.subplots(num_rows, plots_per_row, figsize=(20,30)) # Create subplots with appropriate dimensions

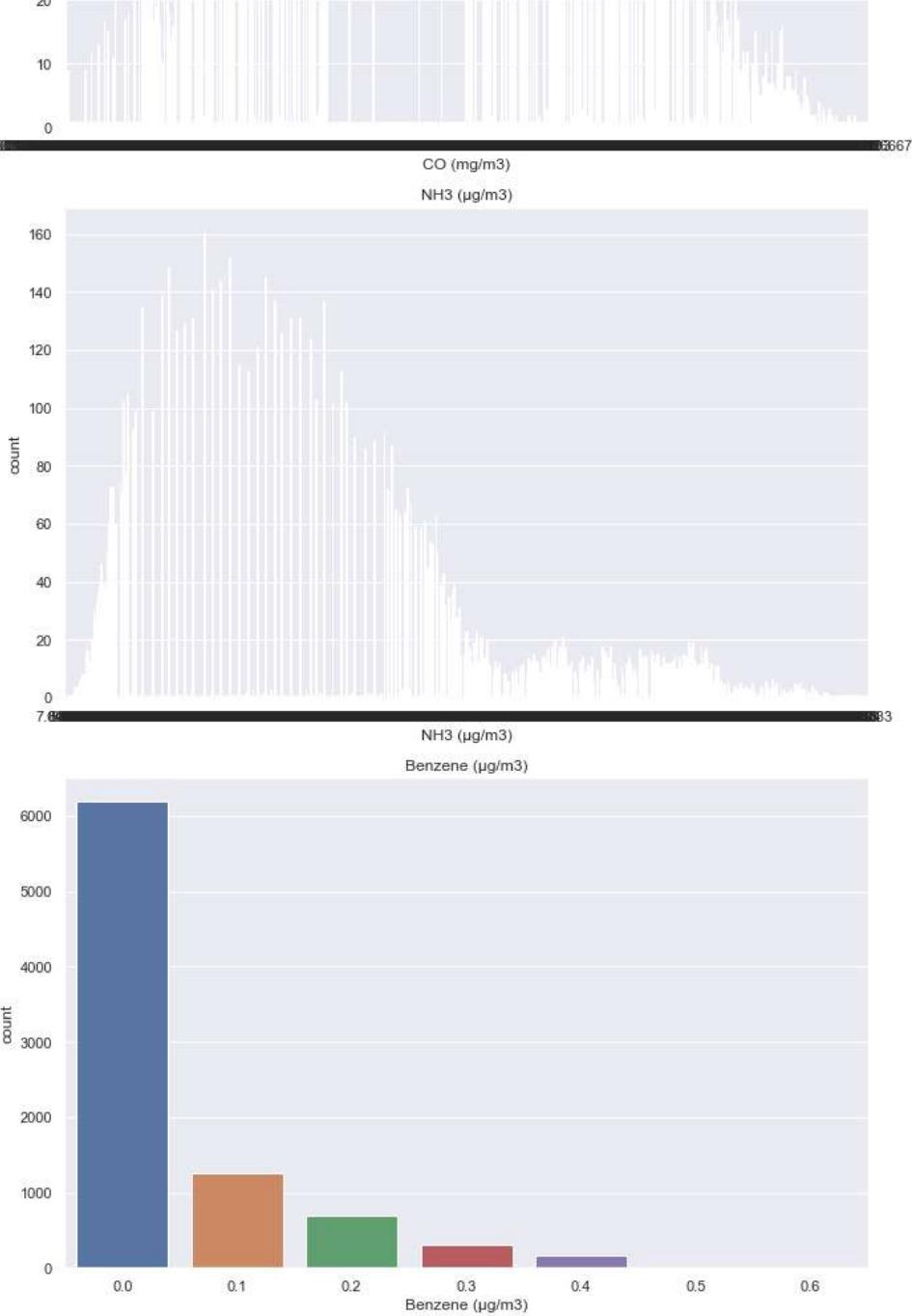
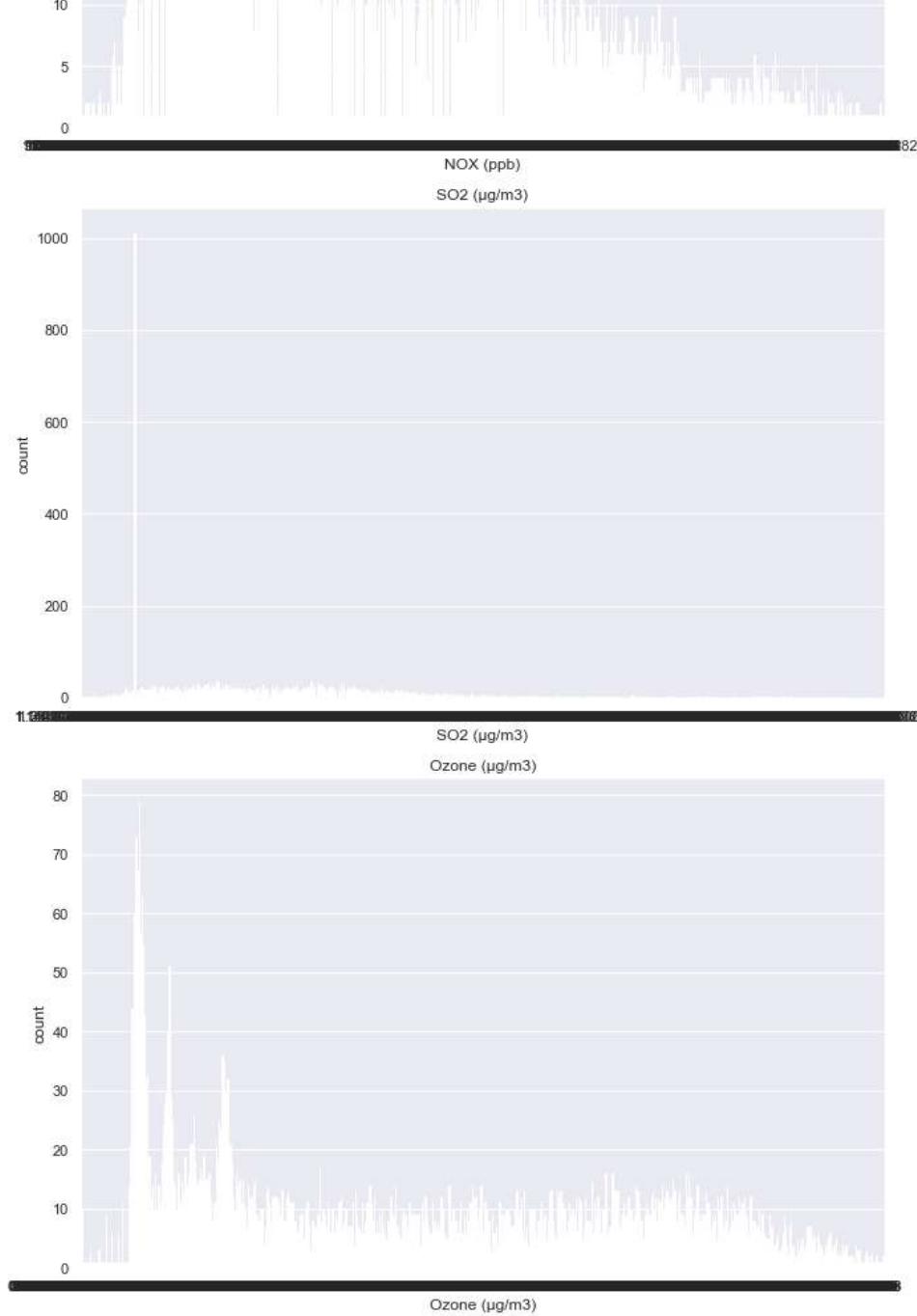
for i, column in enumerate(df_new.columns):
    row = i // plots_per_row # Calculate the row index
    col = i % plots_per_row # Calculate the column index

    sns.countplot(data=df_new, x=column, ax=axes[row, col]) # Create countplot in the specified subplot
    axes[row, col].set_title(column) # Set the title for the countplot

# Remove any unused subplots
if num_plots % plots_per_row != 0:
    for i in range(num_plots, num_rows * plots_per_row):
        fig.delaxes(axes.flatten()[i])

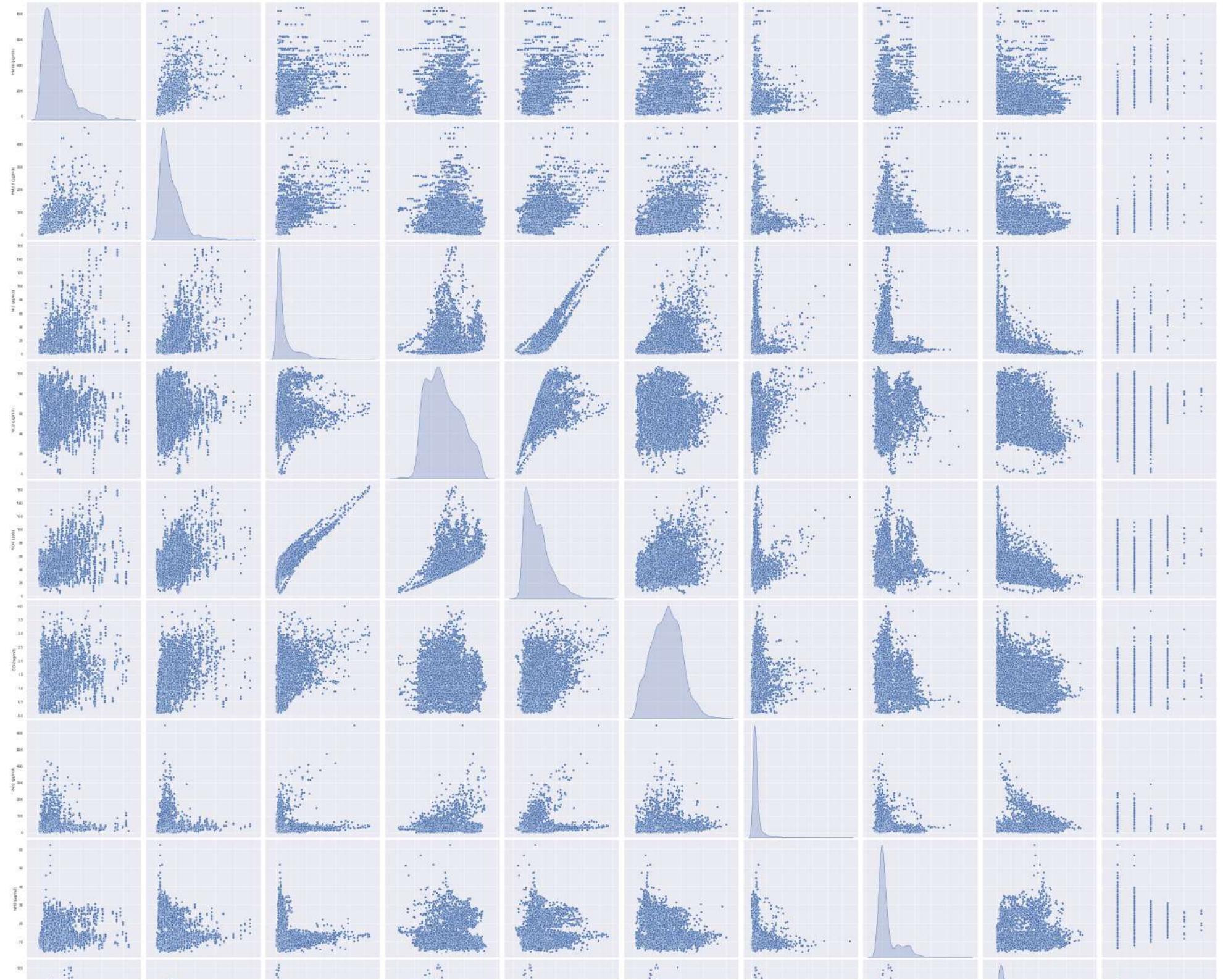
plt.tight_layout() # Adjust the spacing between subplots
plt.show() # Display the plots
```

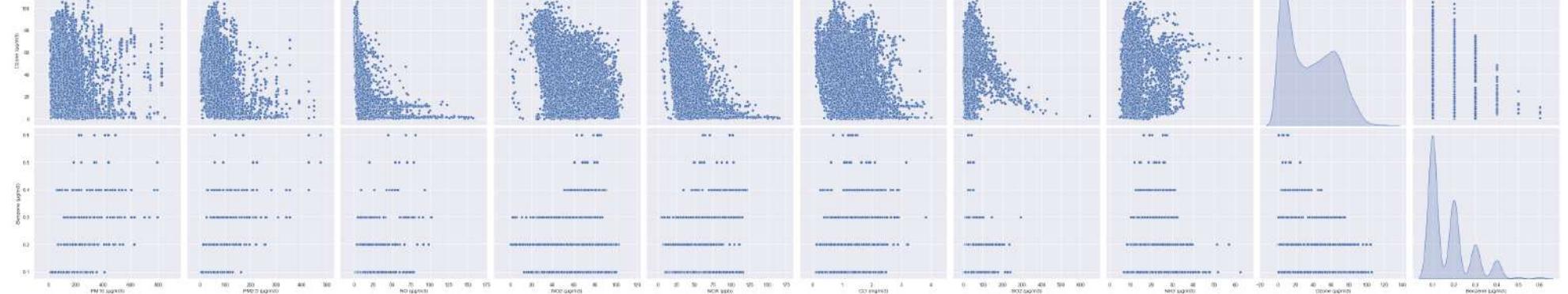





```
In [255]: sns.pairplot(df,diag_kind='kde', height=5)
```

```
Out[255]: <seaborn.axisgrid.PairGrid at 0x1a8e18dce50>
```



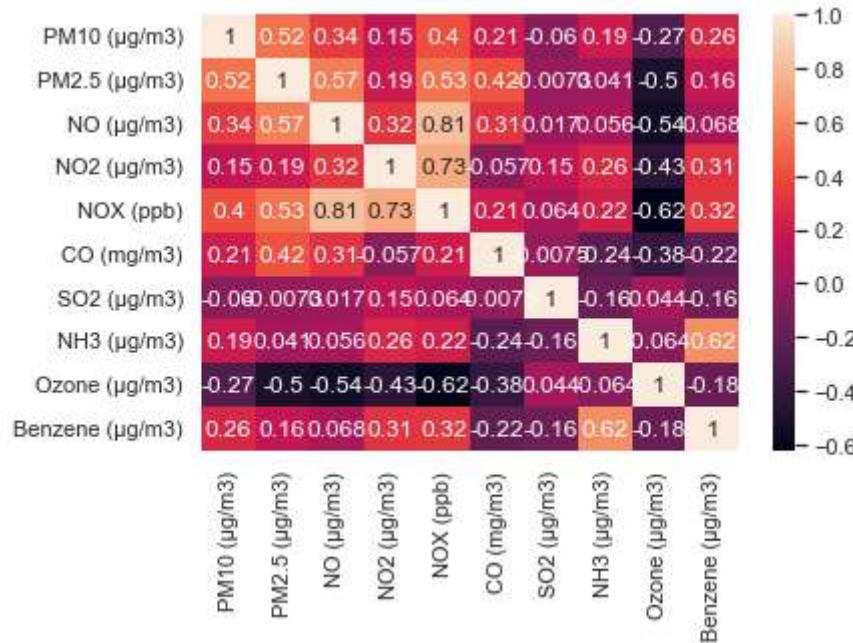
```
In [256]: corr = df_new.corr()
corr
```

Out[256]:

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)	CO (mg/m^3)	SO2 ($\mu\text{g}/\text{m}^3$)	NH3 ($\mu\text{g}/\text{m}^3$)	Ozone ($\mu\text{g}/\text{m}^3$)	Benzene ($\mu\text{g}/\text{m}^3$)
PM10 ($\mu\text{g}/\text{m}^3$)	1.000000	0.519669	0.341073	0.148992	0.403286	0.211036	-0.060239	0.192944	-0.274550	0.259963
PM2.5 ($\mu\text{g}/\text{m}^3$)	0.519669	1.000000	0.569069	0.193086	0.526319	0.417860	-0.007265	0.040620	-0.495977	0.159058
NO ($\mu\text{g}/\text{m}^3$)	0.341073	0.569069	1.000000	0.319533	0.811755	0.311163	0.016904	0.055734	-0.539470	0.067915
NO2 ($\mu\text{g}/\text{m}^3$)	0.148992	0.193086	0.319533	1.000000	0.729429	-0.056935	0.147611	0.261598	-0.433761	0.307349
NOX (ppb)	0.403286	0.526319	0.811755	0.729429	1.000000	0.211916	0.063710	0.220210	-0.621551	0.323194
CO (mg/m^3)	0.211036	0.417860	0.311163	-0.056935	0.211916	1.000000	-0.007528	-0.235739	-0.381699	-0.215772
SO2 ($\mu\text{g}/\text{m}^3$)	-0.060239	-0.007265	0.016904	0.147611	0.063710	-0.007528	1.000000	-0.164286	0.044367	-0.155067
NH3 ($\mu\text{g}/\text{m}^3$)	0.192944	0.040620	0.055734	0.261598	0.220210	-0.235739	-0.164286	1.000000	-0.063514	0.624589
Ozone ($\mu\text{g}/\text{m}^3$)	-0.274550	-0.495977	-0.539470	-0.433761	-0.621551	-0.381699	0.044367	-0.063514	1.000000	-0.183707
Benzene ($\mu\text{g}/\text{m}^3$)	0.259963	0.159058	0.067915	0.307349	0.323194	-0.215772	-0.155067	0.624589	-0.183707	1.000000

```
In [257]: sns.heatmap(corr, annot=True)
```

```
Out[257]: <Axes: >
```



White Noise

```
In [258]: df_wn = pd.DataFrame()
```

```
In [259]: for i in range(len(df_new.columns)):
    wn = np.random.normal(loc = df_new[df_new.columns[i]].mean(), scale = df_new[df_new.columns[i]].std(), size = len(df_new))
    # Try adding the white noise to the dataframe
    df_wn[df_new.columns[i] + " wn"] = wn
```

In [260]: df_wn

Out[260]:

	PM10 (μg/m3) wn	PM2.5 (μg/m3) wn	NO (μg/m3) wn	NO2 (μg/m3) wn	NOX (ppb) wn	CO (mg/m3) wn	SO2 (μg/m3) wn	NH3 (μg/m3) wn	Ozone (μg/m3) wn	Benzene (μg/m3) wn
0	206.513798	145.169649	-2.396097	59.384725	21.320561	1.037983	-9.597048	5.330299	7.997201	0.122885
1	276.310156	85.265108	36.358674	57.002439	87.849194	0.710377	43.826795	8.347168	37.452670	0.046962
2	279.094794	79.691081	28.370332	49.330469	45.739927	2.081828	11.118427	9.491626	57.711765	0.181411
3	95.202606	102.857141	17.211985	65.021463	44.274559	0.825866	46.486572	7.865709	26.869557	-0.114238
4	57.164866	3.411743	-6.156503	58.053354	49.430233	1.468697	148.983228	16.999388	-19.187788	0.032626
...
8635	276.233297	132.233638	18.260395	85.455752	77.782678	1.849191	77.633756	15.698730	-1.379101	0.116671
8636	99.523465	84.938659	12.115768	54.757796	63.872387	1.122778	27.284259	20.069233	45.667317	0.036854
8637	212.494947	0.830622	8.278516	57.342480	29.321998	1.612028	60.029613	19.268305	1.173904	0.280521
8638	382.931484	87.082302	20.836471	35.176806	53.207299	1.694597	38.544522	9.819373	61.375327	0.069414
8639	120.417471	157.408358	-6.724310	34.569112	51.112147	0.462106	-29.120559	11.704980	11.617337	0.034825

8640 rows × 10 columns

In [261]: df_wn.columns

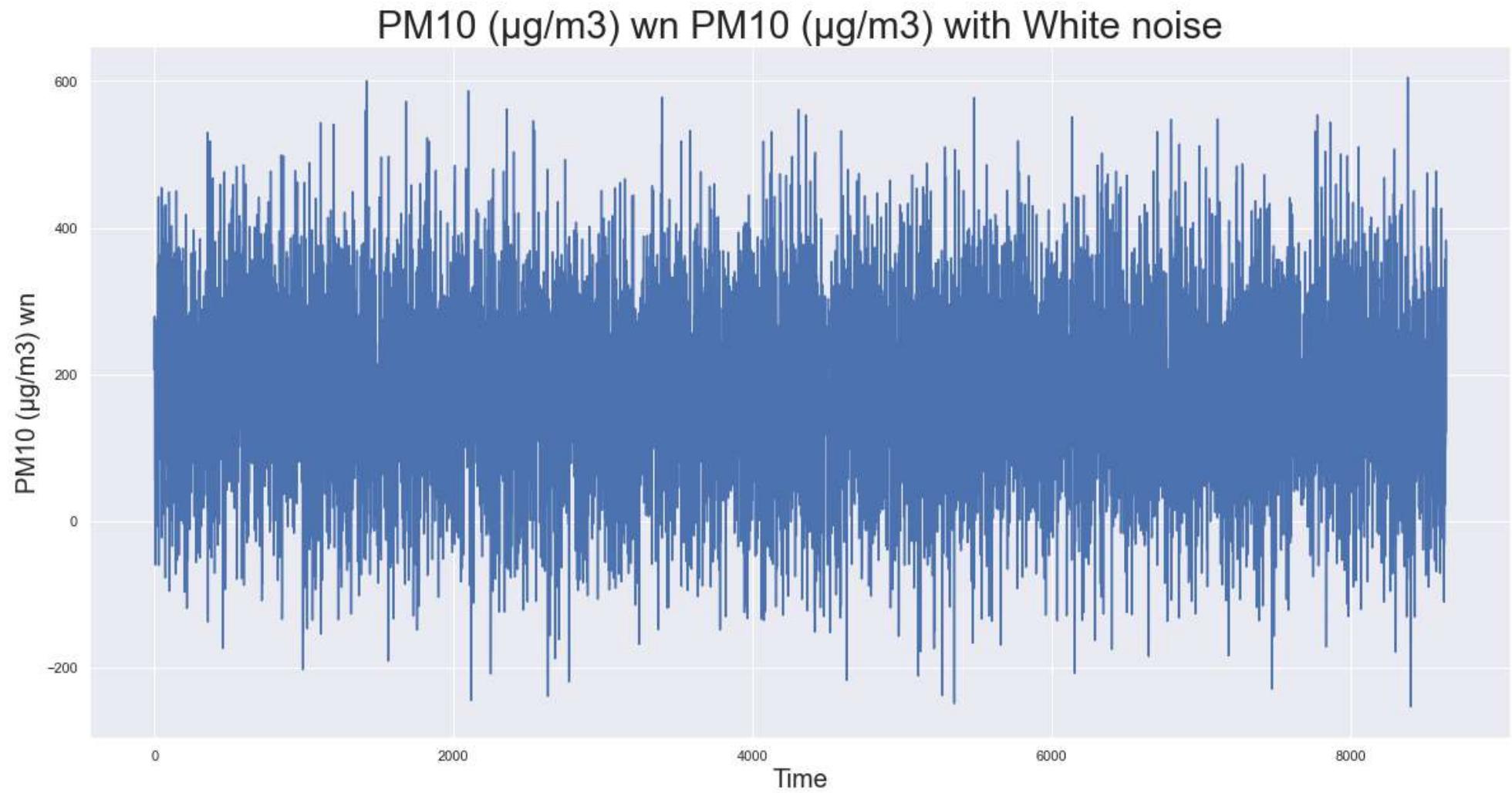
Out[261]: Index(['PM10 (μg/m3) wn', 'PM2.5 (μg/m3) wn', 'NO (μg/m3) wn', 'NO2 (μg/m3) wn', 'NOX (ppb) wn', 'CO (mg/m3) wn', 'SO2 (μg/m3) wn', 'NH3 (μg/m3) wn', 'Ozone (μg/m3) wn', 'Benzene (μg/m3) wn'], dtype='object')

In [262]: df_wn.describe()

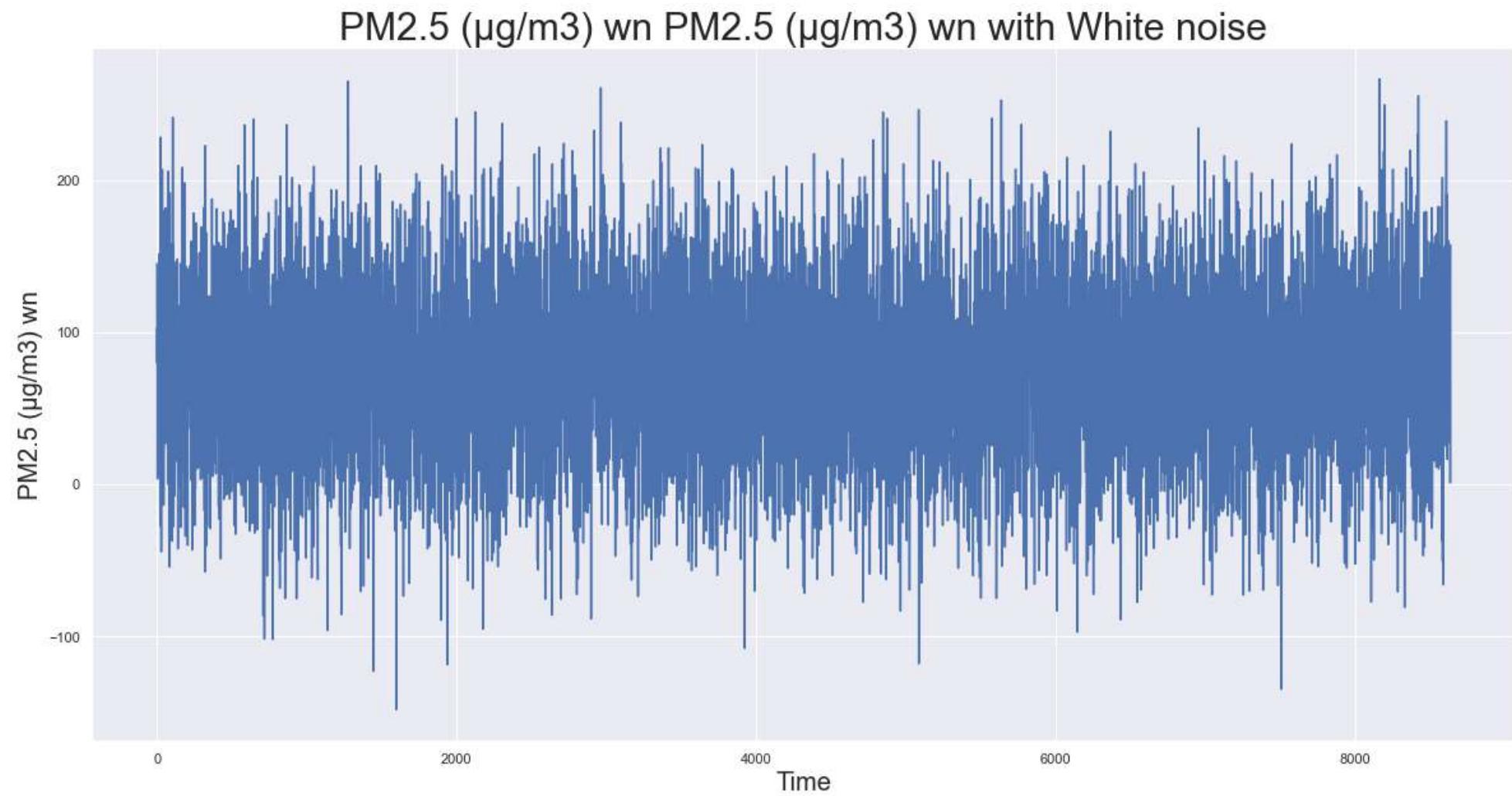
Out[262]:

	PM10 (µg/m3) wn	PM2.5 (µg/m3) wn	NO (µg/m3) wn	NO2 (µg/m3) wn	NOX (ppb) wn	CO (mg/m3) wn	SO2 (µg/m3) wn	NH3 (µg/m3) wn	Ozone (µg/m3) wn	Benzene (µg/m3) wn
count	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000	8640.000000
mean	177.304221	75.500183	14.973733	55.261013	42.380218	1.406480	32.340013	13.382062	35.302635	0.050341
std	124.531756	55.222278	18.168913	20.225826	22.503582	0.630952	38.733023	6.146825	26.460993	0.096284
min	-253.154959	-148.276611	-53.620422	-11.152614	-41.514988	-0.921743	-109.298716	-7.951207	-69.781695	-0.328448
25%	93.505712	37.729908	2.578048	41.569466	27.158178	0.985075	6.275895	9.251921	17.157959	-0.014429
50%	178.015011	75.960762	14.934623	55.310497	42.392093	1.409046	32.882310	13.432467	35.444198	0.051486
75%	261.416075	113.158971	27.251811	68.824366	57.676149	1.835147	58.300510	17.592174	53.081484	0.114950
max	605.228121	266.570008	84.626089	127.946446	126.830923	3.655358	160.334650	34.774396	143.576615	0.396047

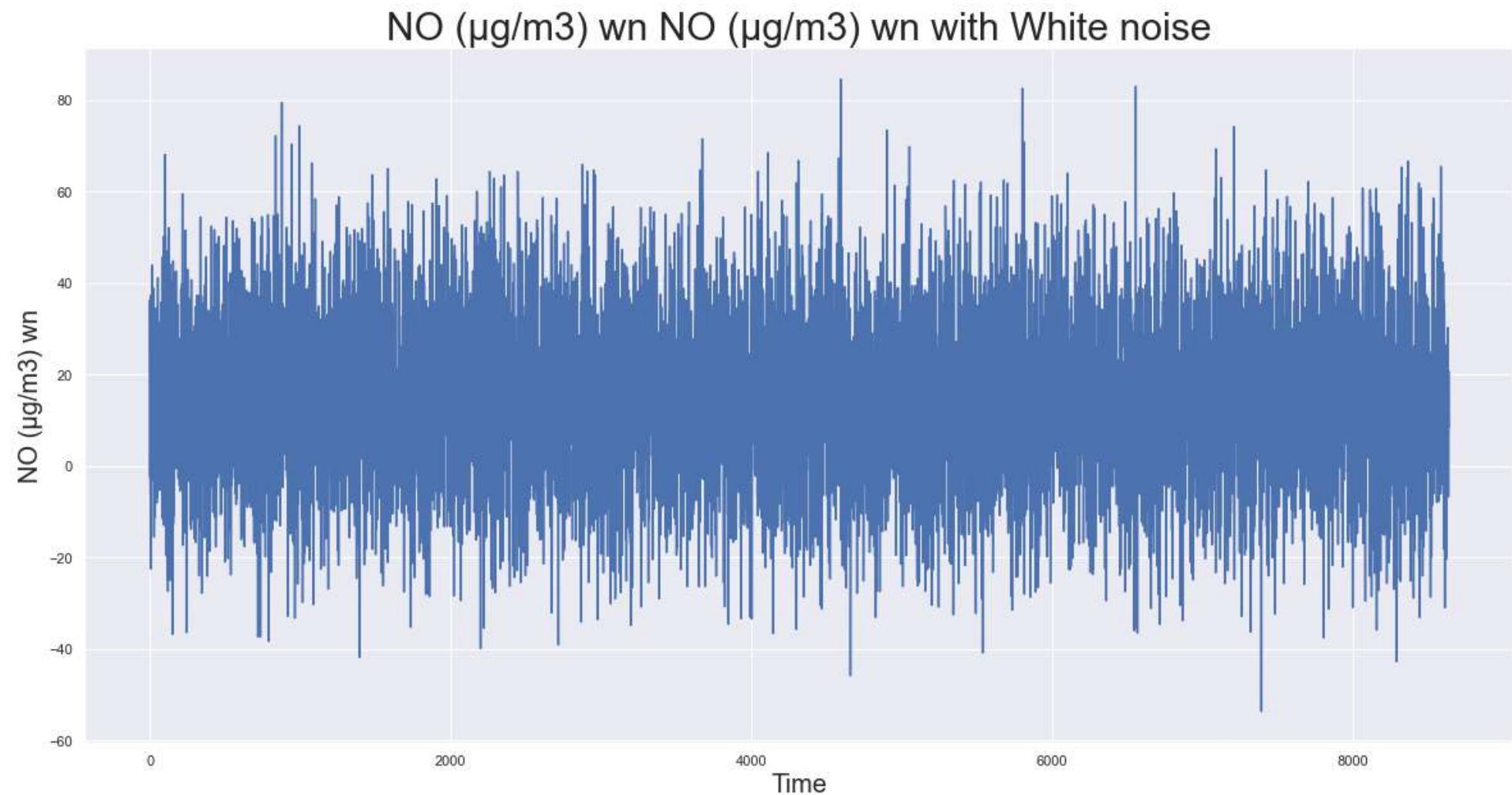
```
In [263]: simple_time_plot(df_wn,"PM10 (\u00b5g/m\u00b3) wn","PM10 (\u00b5g/m\u00b3) with White noise")
```



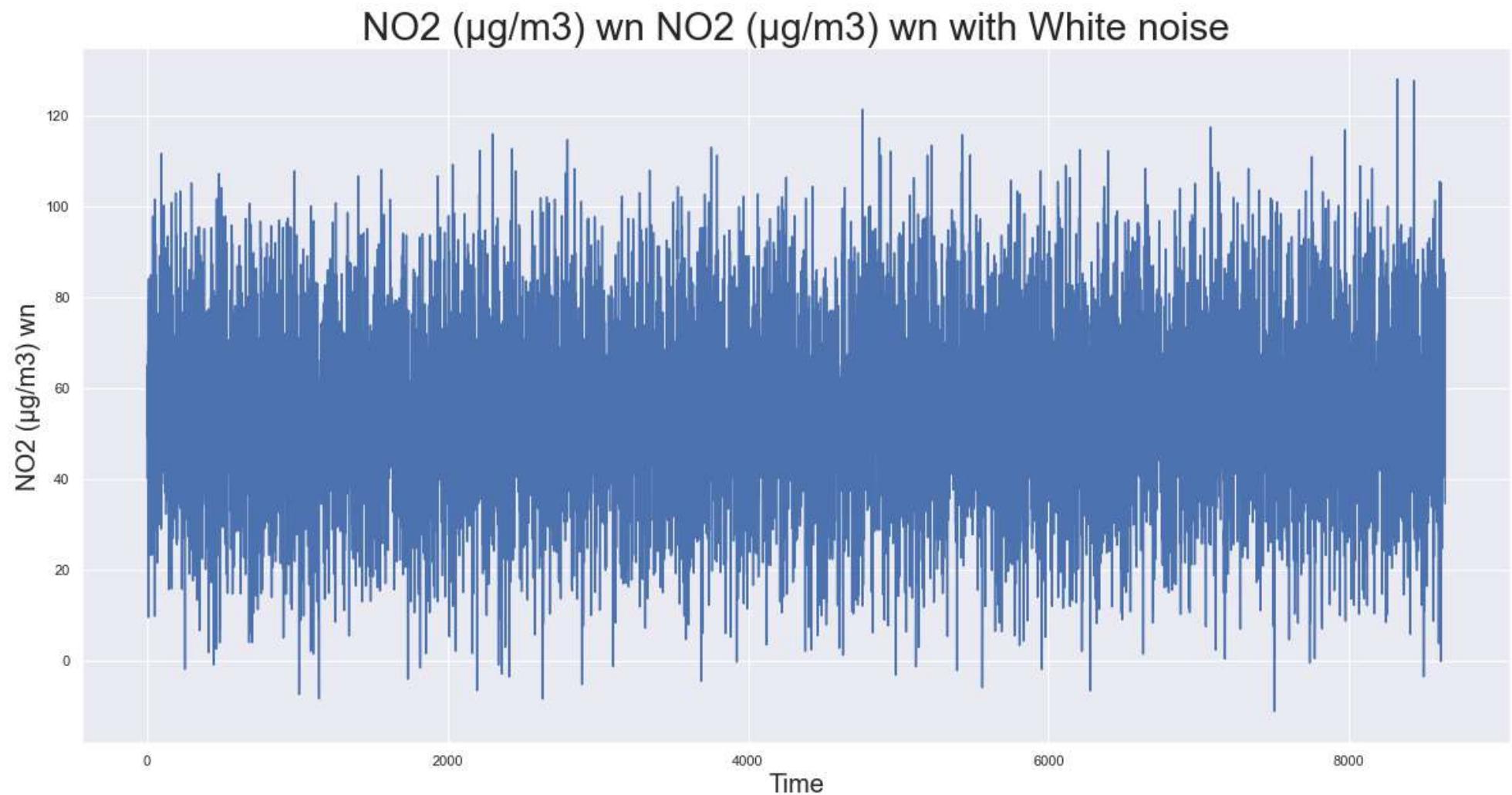
```
In [264]: simple_time_plot(df_wn,"PM2.5 (μg/m3) wn","PM2.5 (μg/m3) wn with White noise")
```



```
In [265]: simple_time_plot(df_wn,"NO (\u00b5g/m\u00b3) wn","NO (\u00b5g/m\u00b3) wn with White noise")
```

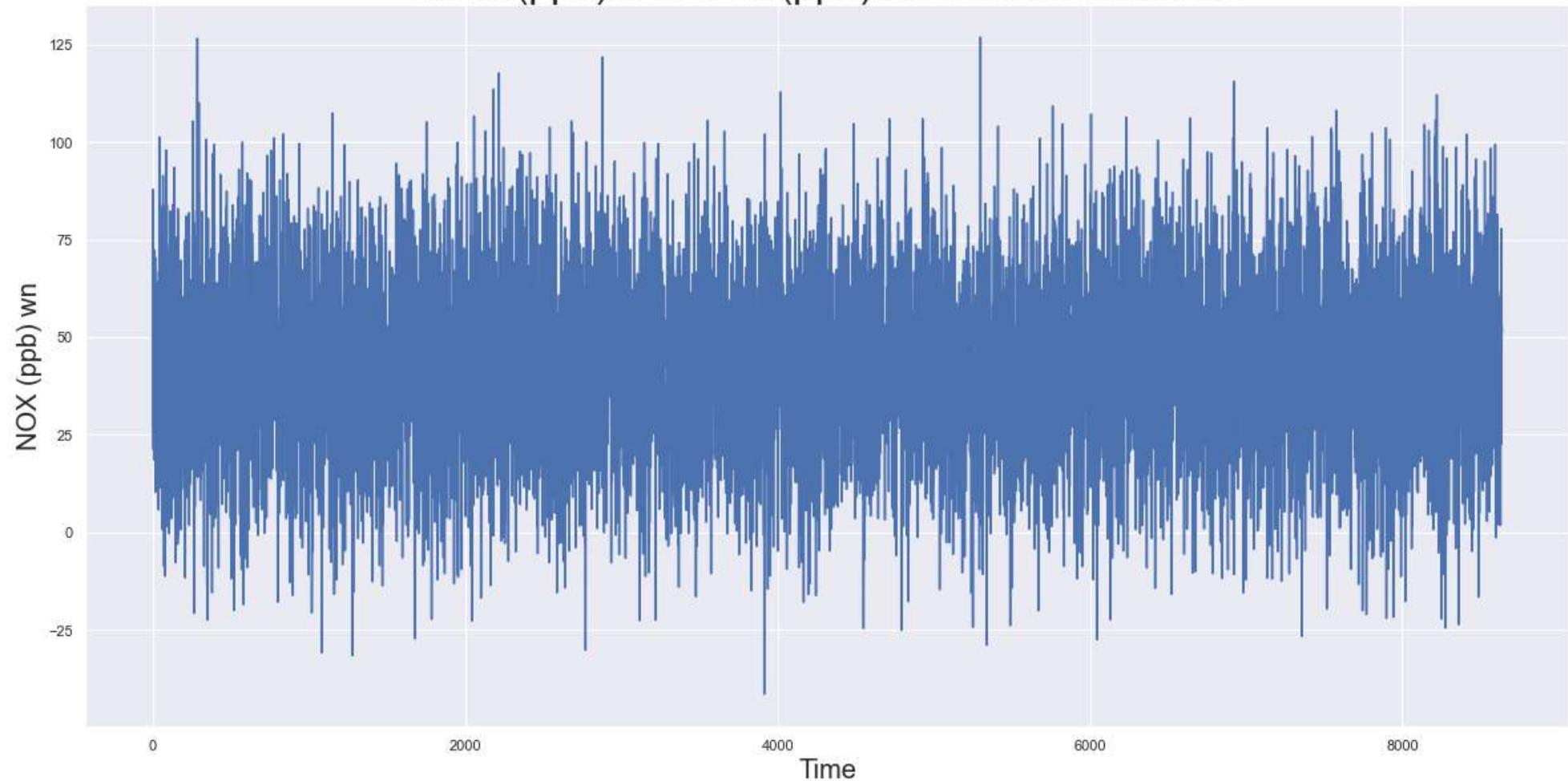


```
In [266]: simple_time_plot(df_wn,"NO2 (\u00b5g/m\u00b3) wn","NO2 (\u00b5g/m\u00b3) wn with White noise")
```

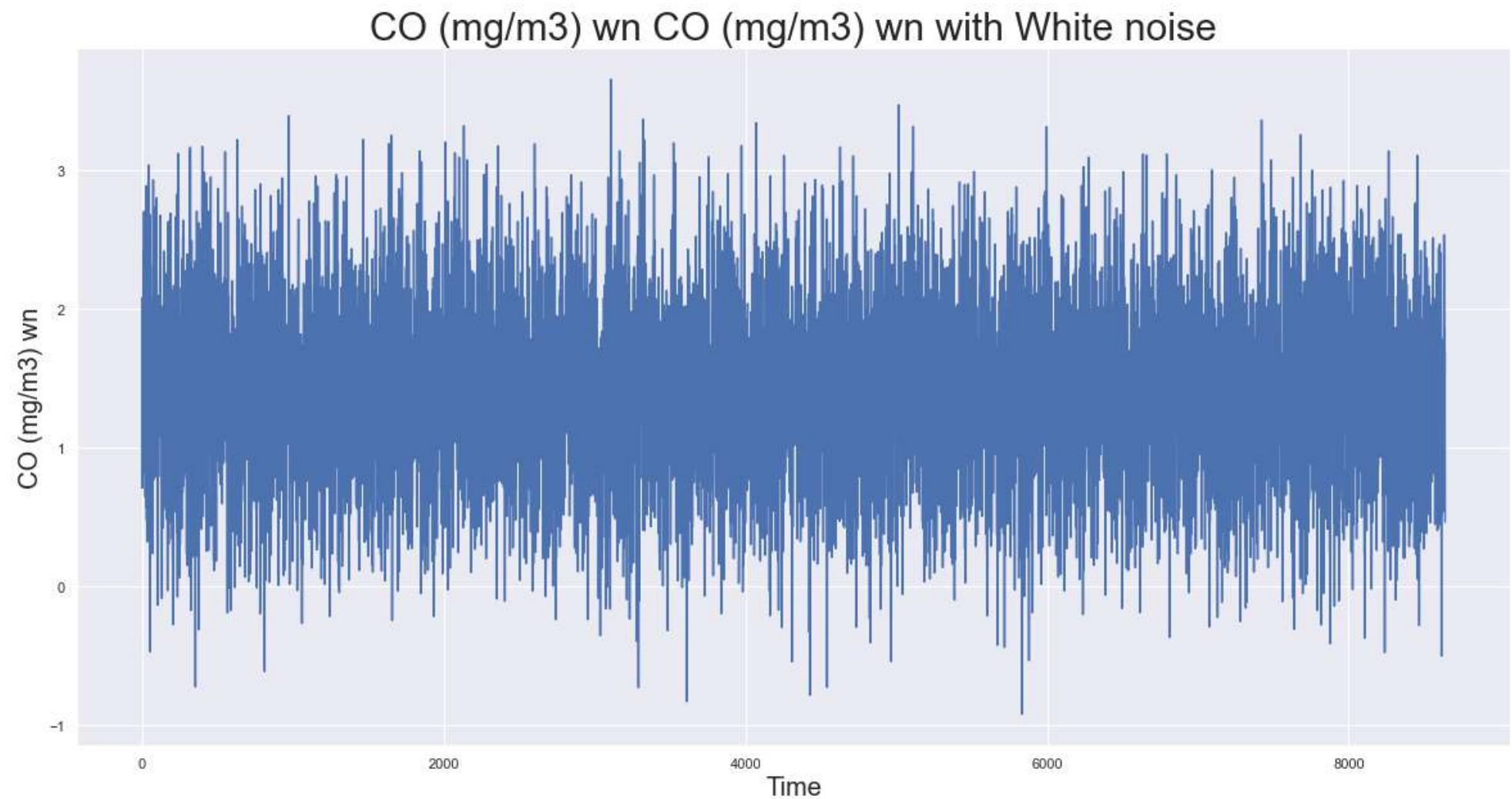


```
In [267]: simple_time_plot(df_wn,"NOX (ppb) wn","NOX (ppb) wn with White noise")
```

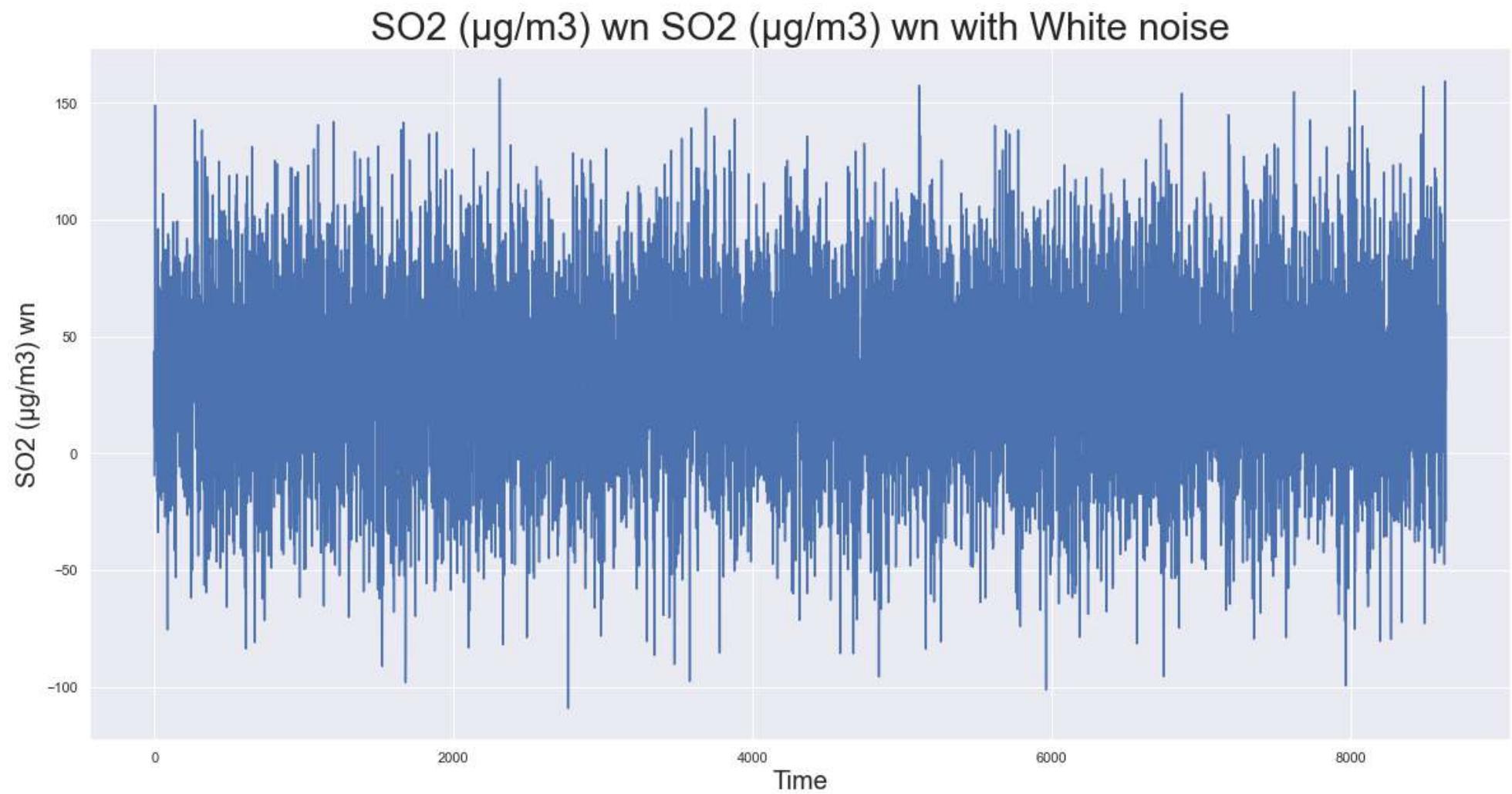
NOX (ppb) wn NOX (ppb) wn with White noise



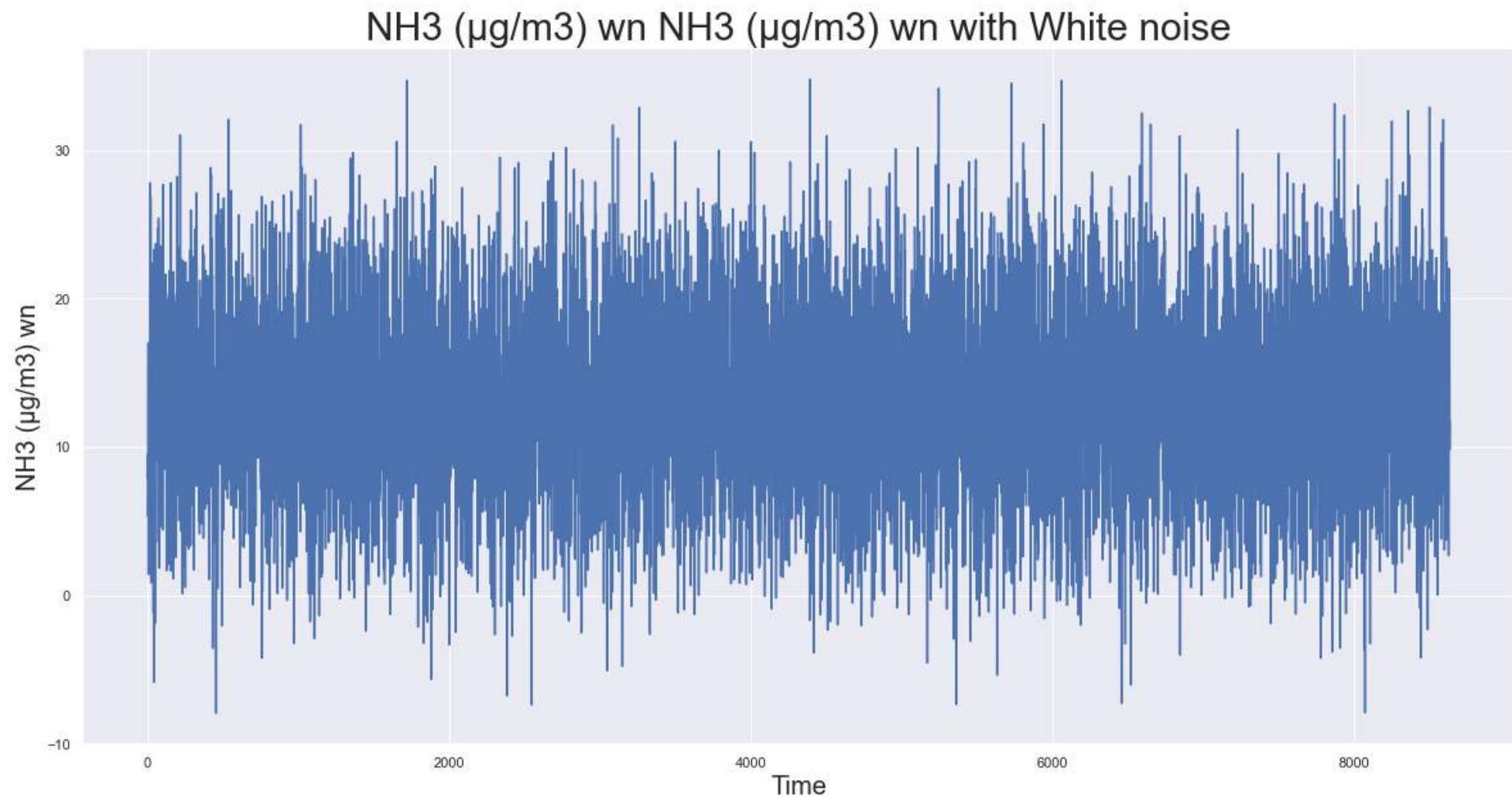
```
In [268]: simple_time_plot(df_wn,"CO (mg/m3) wn","CO (mg/m3) wn with White noise")
```



```
In [269]: simple_time_plot(df_wn,"SO2 (\u00b5g/m\u00b3) wn","SO2 (\u00b5g/m\u00b3) wn with White noise")
```

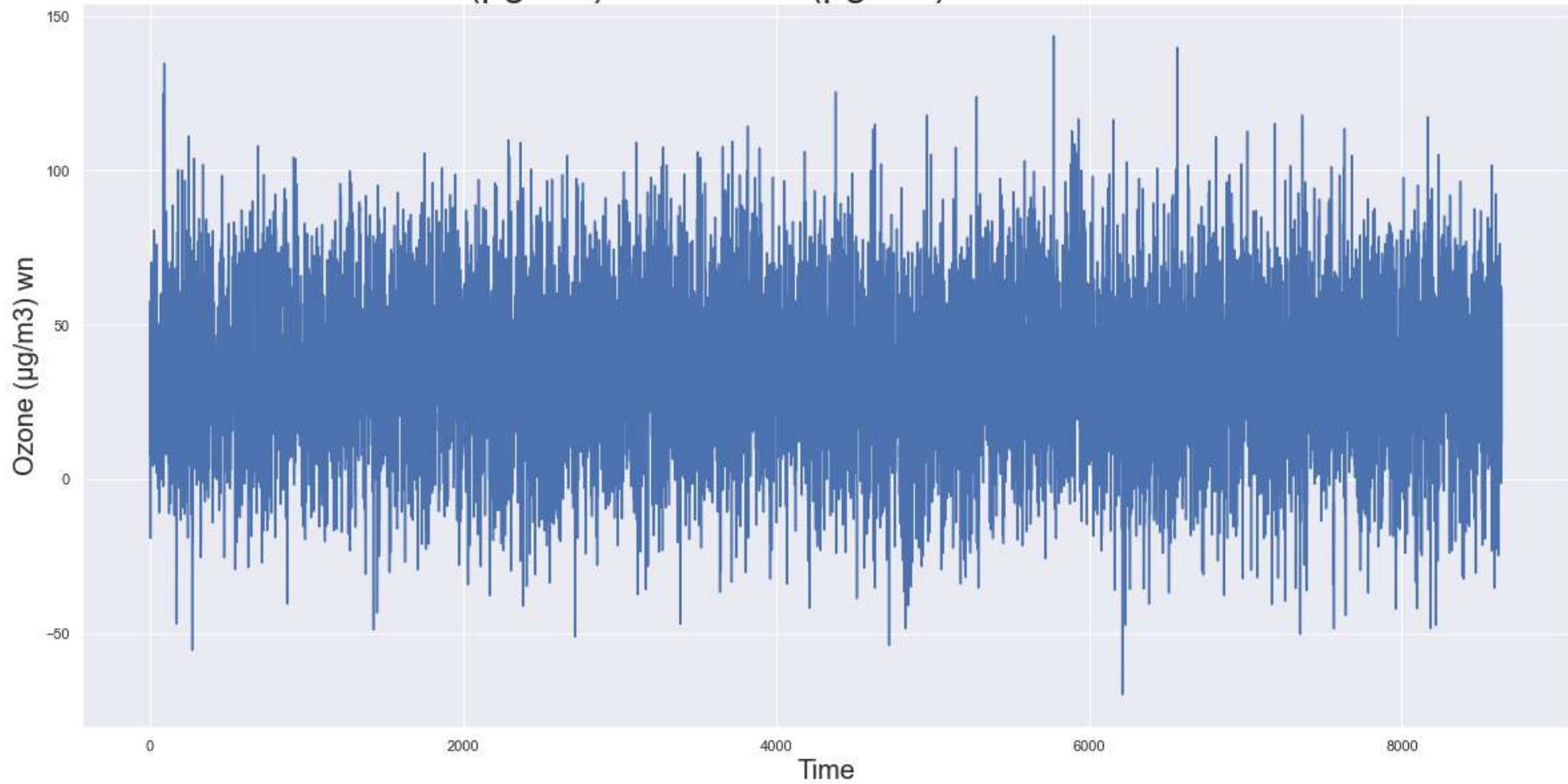


```
In [270]: simple_time_plot(df_wn,"NH3 (\u00b5g/m\u00b3) wn","NH3 (\u00b5g/m\u00b3) wn with White noise")
```

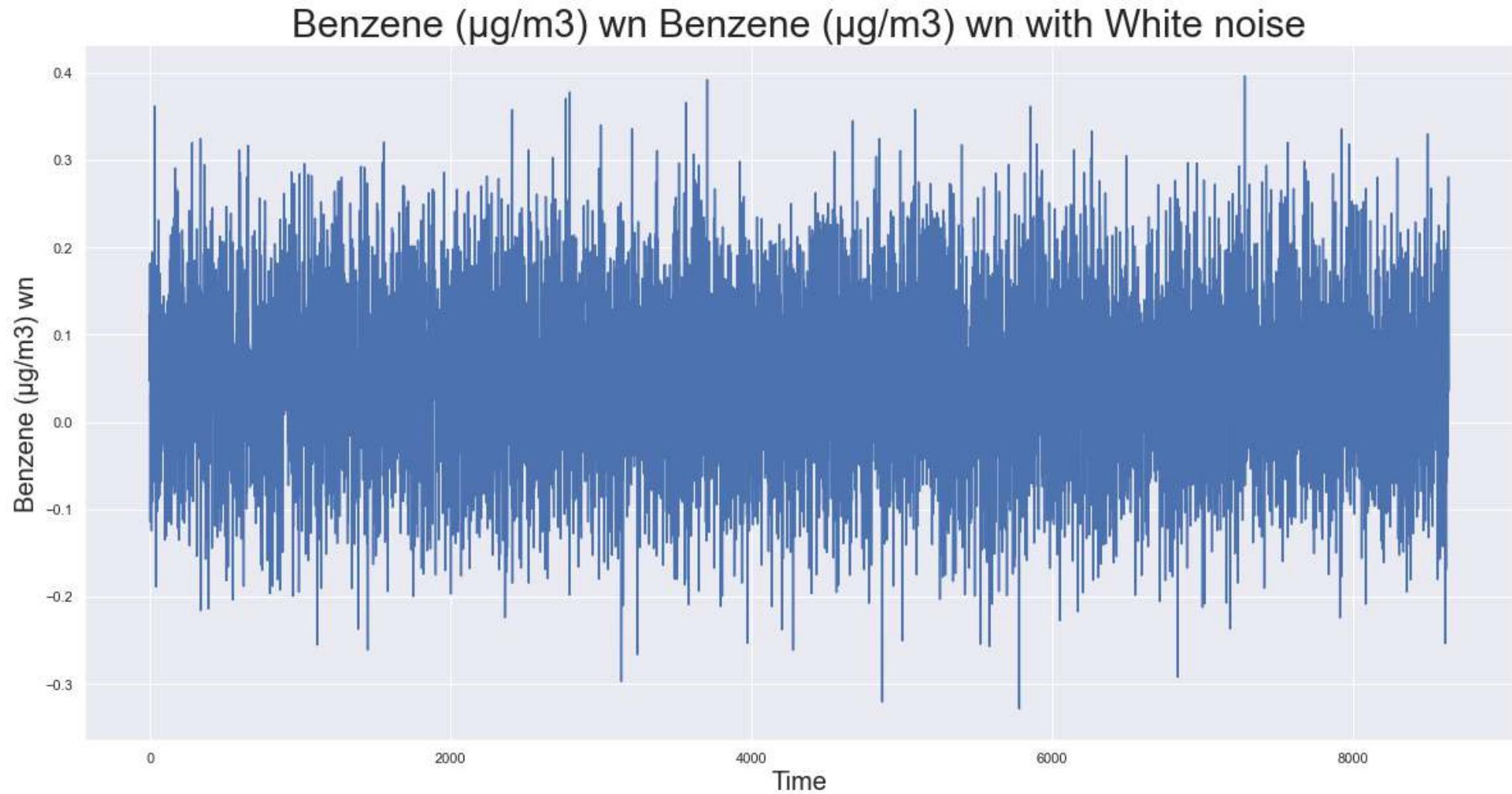


```
In [271]: simple_time_plot(df_wn,"Ozone (μg/m3) wn","Ozone (μg/m3) wn with White noise")
```

Ozone ($\mu\text{g}/\text{m}^3$) wn Ozone ($\mu\text{g}/\text{m}^3$) wn with White noise



```
In [272]: simple_time_plot(df_wn,"Benzene (\u00b5g/m\u00b3) wn","Benzene (\u00b5g/m\u00b3) wn with White noise")
```



A special type of time-series, where values tend to persist over time and the difference between periods are simply white noise

Stationarity

"weak-form" stationarity or "covariance" stationarity

constant mean and sigma and covariance

For white noise, mean is always zero and sigma is constant and covariance is also zero

Augmented Dickey-Fuller (ADF) test

```
In [273]: sts.adfuller(df_new["PM10 (\mu g/m³]"))
```

```
Out[273]: (-9.023313848430973,
 5.697654959032442e-15,
 36,
 8603,
 {'1%': -3.43110345490302,
 '5%': -2.861876021468662,
 '10%': -2.566948859294898},
 87720.11218958281)
```

```
In [274]: sts.adfuller(df_wn["PM10 (\mu g/m³) wn"])
```

```
Out[274]: (-94.52996183746103,
 0.0,
 0,
 8639,
 {'1%': -3.4311071760751575,
 '5%': -2.8618746209792243,
 '10%': -2.5669481138040715},
 107416.02615276945)
```

```
In [275]: sts.adfuller(df_new["PM2.5 (\mu g/m³]"))
```

```
Out[275]: (-11.159054400107523,
 2.8178026254645535e-20,
 36,
 8603,
 {'1%': -3.43110345490302,
 '5%': -2.861876021468662,
 '10%': -2.566948859294898},
 75382.9619636433)
```

```
In [276]: sts.adfuller(df_wn["PM2.5 (\u00b5g/m\u00b3) wn"])
```

```
Out[276]: (-47.961819424399636,  
 0.0,  
 3,  
 8636,  
 {'1%': -3.4311074391835352,  
 '5%': -2.8618747372406377,  
 '10%': -2.5669481756908668},  
 93412.65247267464)
```

```
In [277]: sts.adfuller(df_new["NO (\u00b5g/m\u00b3)"])
```

```
Out[277]: (-14.799960366282331,  
 2.1069740025829943e-27,  
 11,  
 8628,  
 {'1%': -3.4311081417006526,  
 '5%': -2.8618750476664037,  
 '10%': -2.5669483409327674},  
 41899.65745690938)
```

```
In [278]: sts.adfuller(df_wn["NO (\u00b5g/m\u00b3) wn"])
```

```
Out[278]: (-94.24156879830569,  
 0.0,  
 0,  
 8639,  
 {'1%': -3.4311071760751575,  
 '5%': -2.8618746209792243,  
 '10%': -2.5669481138040715},  
 74307.366875165)
```

```
In [279]: sts.adfuller(df_new["NO2 (\u00b5g/m\u00b3)"])
```

```
Out[279]: (-9.181725236972362,  
 2.2425355810086067e-15,  
 21,  
 8618,  
 {'1%': -3.431109021681739,  
 '5%': -2.8618754365092136,  
 '10%': -2.5669485479166507},  
 43364.67117055162)
```

```
In [280]: sts.adfuller(df_wn["NO2 (µg/m³) wn"])
```

```
Out[280]: (-93.19856081024928,
 0.0,
 0,
 8639,
 {'1%': -3.4311071760751575,
 '5%': -2.8618746209792243,
 '10%': -2.5669481138040715},
 76146.28336080667)
```

```
In [281]: sts.adfuller(df_new["NOX (ppb)"])
```

```
Out[281]: (-12.767251588723852,
 7.908914084843848e-24,
 25,
 8614,
 {'1%': -3.4311093742464234,
 '5%': -2.8618755922991688,
 '10%': -2.5669486308447933},
 41623.58945747933)
```

```
In [282]: sts.adfuller(df_wn["NOX (ppb) wn"])
```

```
Out[282]: (-93.03639890277302,
 0.0,
 0,
 8639,
 {'1%': -3.4311071760751575,
 '5%': -2.8618746209792243,
 '10%': -2.5669481138040715},
 77998.31667915579)
```

```
In [283]: sts.adfuller(df_new["CO (mg/m³)"])
```

```
Out[283]: (-9.976113391269292,
 2.175422556079248e-17,
 8,
 8631,
 {'1%': -3.4311078781040734,
 '5%': -2.861874931189293,
 '10%': -2.56694827893115},
 -6533.580690127992)
```

```
In [284]: sts.adfuller(df_wn["CO (mg/m3) wn"])
```

```
Out[284]: (-64.01839229217944,  
 0.0,  
 1,  
 8638,  
 {'1%': -3.431107263757638,  
 '5%': -2.8618746597240543,  
 '10%': -2.566948134428226},  
 16489.617206038143)
```

```
In [285]: sts.adfuller(df_new["SO2 (\mu g/m3)"])
```

```
Out[285]: (-14.153436999187724,  
 2.1419881217514085e-26,  
 20,  
 8619,  
 {'1%': -3.4311089335917146,  
 '5%': -2.8618753975843223,  
 '10%': -2.5669485271966446},  
 70924.85623175491)
```

```
In [286]: sts.adfuller(df_wn["SO2 (\mu g/m3) wn"])
```

```
Out[286]: (-66.84869100587073,  
 0.0,  
 1,  
 8638,  
 {'1%': -3.431107263757638,  
 '5%': -2.8618746597240543,  
 '10%': -2.566948134428226},  
 87333.70030649567)
```

```
In [287]: sts.adfuller(df_new["NH3 (\mu g/m3)"])
```

```
Out[287]: (-3.0598986426622434,  
 0.029666749029596005,  
 33,  
 8606,  
 {'1%': -3.4311100803593018,  
 '5%': -2.8618759043136137,  
 '10%': -2.5669487969323943},  
 24679.78171252983)
```

```
In [288]: sts.adfuller(df_wn["NH3 (\mu g/m3) wn"])
```

```
Out[288]: (-66.70716954299634,
 0.0,
 1,
 8638,
 {'1%': -3.431107263757638,
 '5%': -2.8618746597240543,
 '10%': -2.566948134428226},
 55654.78311685787)
```

```
In [289]: sts.adfuller(df_new["Ozone (\mu g/m3)"])
```

```
Out[289]: (-20.95894625642859,
 0.0,
 34,
 8605,
 {'1%': -3.431110168715755,
 '5%': -2.8618759433562184,
 '10%': -2.5669488177150632},
 52693.31913985891)
```

```
In [290]: sts.adfuller(df_wn["Ozone (\mu g/m3) wn"])
```

```
Out[290]: (-94.11936012452333,
 0.0,
 0,
 8639,
 {'1%': -3.4311071760751575,
 '5%': -2.8618746209792243,
 '10%': -2.5669481138040715},
 80784.67276734595)
```

```
In [291]: sts.adfuller(df_new["Benzene (\mu g/m3)"])
```

```
Out[291]: (-5.859900865713501,
 3.4298988816517216e-07,
 37,
 8602,
 {'1%': -3.4311104339084104,
 '5%': -2.861876060538507,
 '10%': -2.5669488800920677},
 -48023.67268175674)
```

```
In [292]: sts.adfuller(df_wn["Benzene (\mu g/m3) wn"])
```

```
Out[292]: (-93.59171849356801,  
 0.0,  
 0,  
 8639,  
 {'1%': -3.4311071760751575,  
 '5%': -2.8618746209792243,  
 '10%': -2.5669481138040715},  
 -15853.027009371668)
```

For white noises, p-values are very low, approximately close to zero which is obvious.

The p-value helps in determining whether there is enough evidence to reject the null hypothesis and conclude that the time series is stationary. Typically, if the p-value is less than a predetermined significance level (such as 0.05 or 0.01), we reject the null hypothesis and consider the time series to be stationary. Conversely, if the p-value is greater than the significance level, we fail to reject the null hypothesis and conclude that the time series is non-stationary.

In the given outputs, the p-values are like 2.38370370455499e-15, which are extremely small. This indicates that the probability of observing such a significant test statistic under the null hypothesis is very low. Since the p-value is much smaller than the significance level of 0.05, we reject the null hypothesis and conclude that the time series (PM10) is stationary. Similarly all the other are stationary as well because p_values are very low.

Seasonality

certain trends will appear on a cyclical basis

Decomposition: Split into 3 effects:

Trend -> Pattern :Seasonal -> Cyclical effects :Residual -> Error of prediction

"Naive" Decomposition: can be additive or multiplicative

```
In [293]: def seasonality(df,parameter,typeseason):
    # Resample the data to the desired frequency (15 minutes in this example)
    df_resampled = df[parameter].resample('15Min').mean()

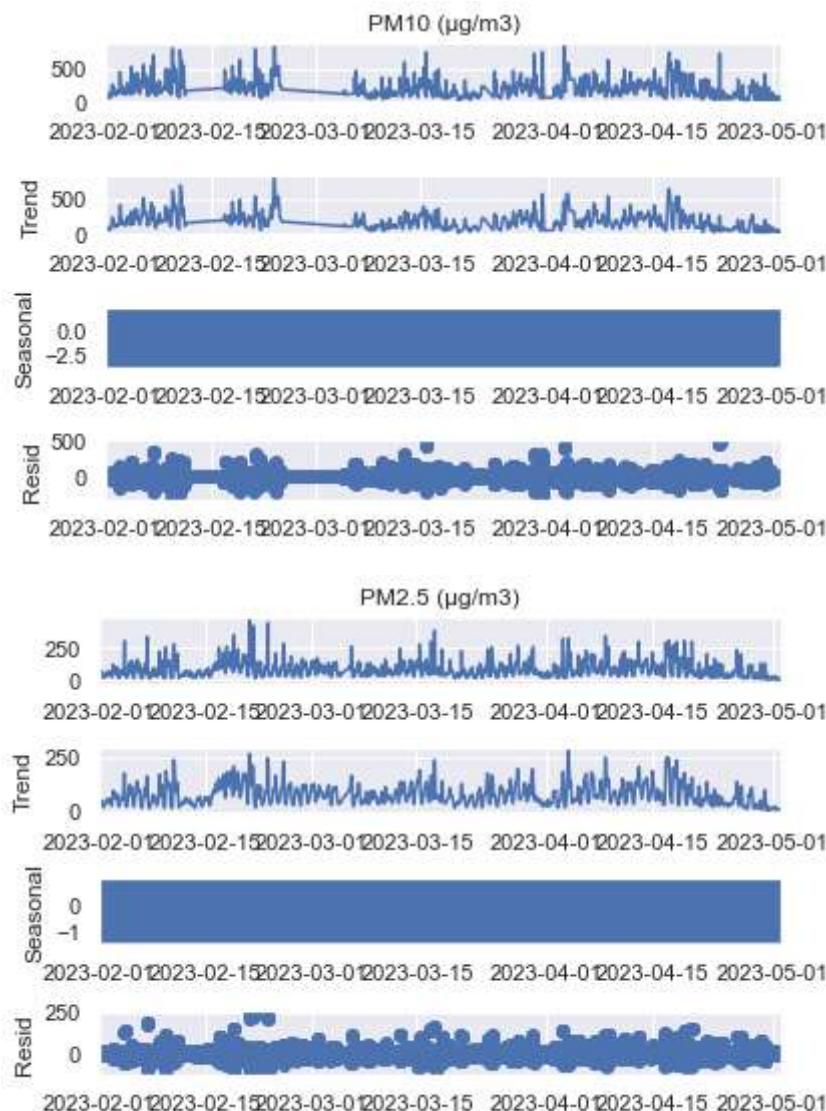
    # Perform seasonal decomposition on the resampled data
    s_dec_additive = seasonal_decompose(df_resampled, model=typeseason, period=15)

    # Plot the decomposition results
    s_dec_additive.plot()
    plt.show()

### The difference between true values and predictions for any period is Residual
```

Seasonality - "Additive"

```
In [294]: for i in range(len(df_new.columns)//2):
    seasonality(df_new,df_new.columns[i], "additive")
```

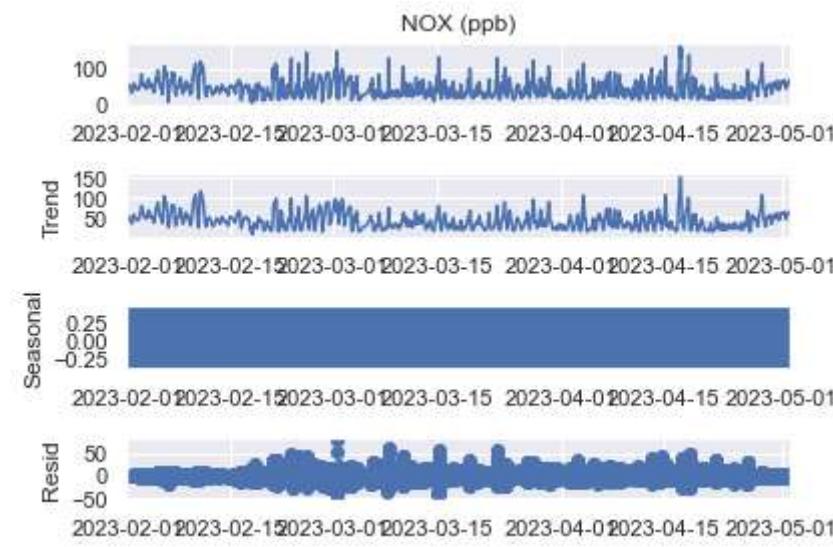


NO ($\mu\text{g}/\text{m}^3$)

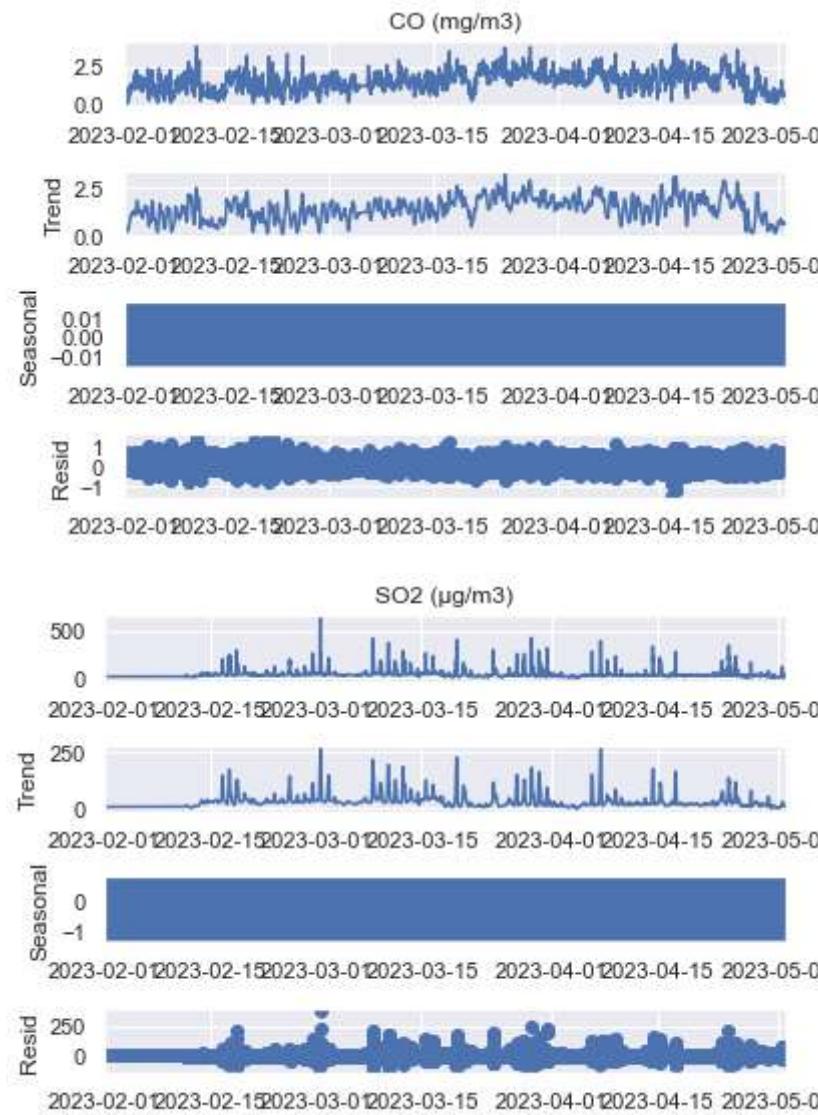


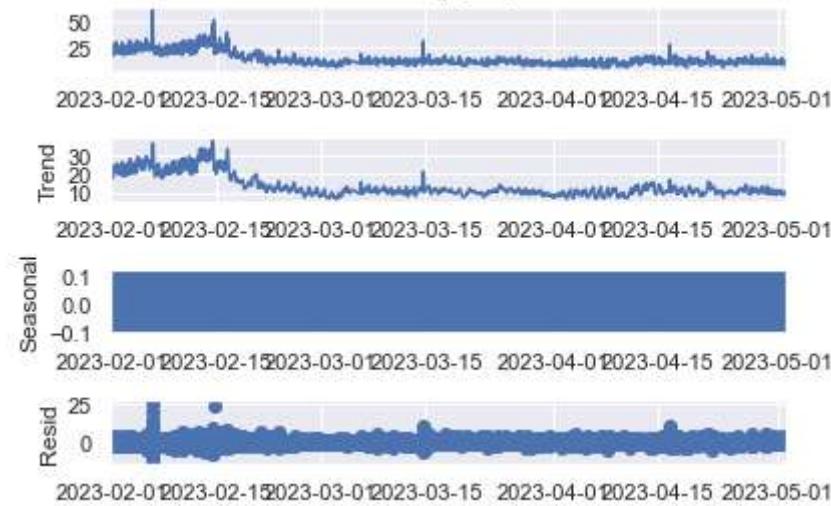
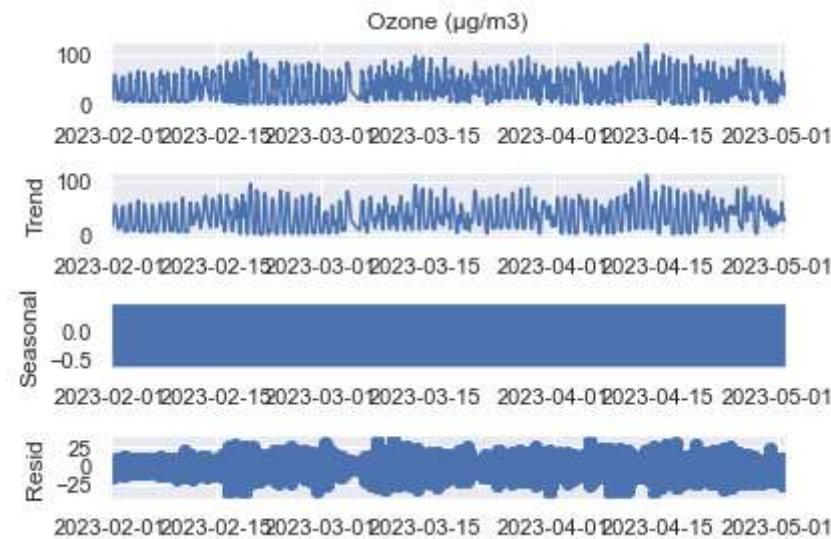
NO₂ ($\mu\text{g}/\text{m}^3$)

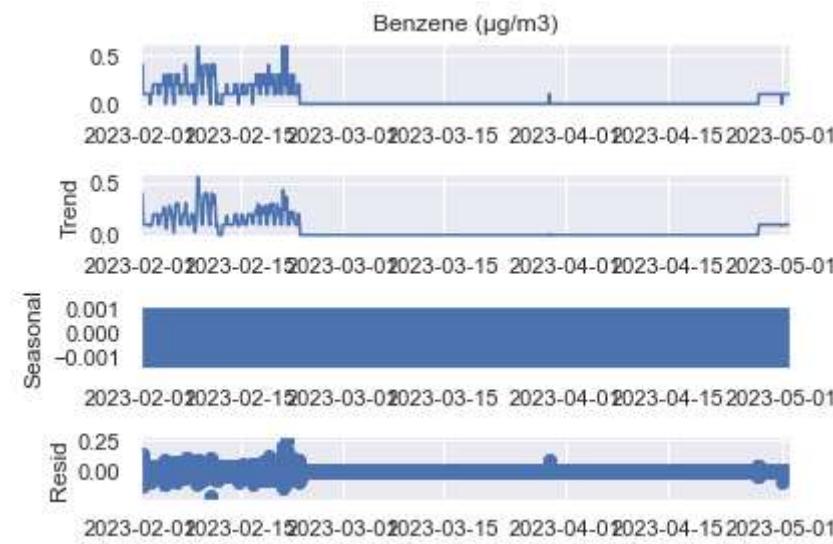




```
In [295]: for i in range(len(df_new.columns)//2 ,len(df_new.columns)):  
    seasonality(df_new,df_new.columns[i], "additive")
```

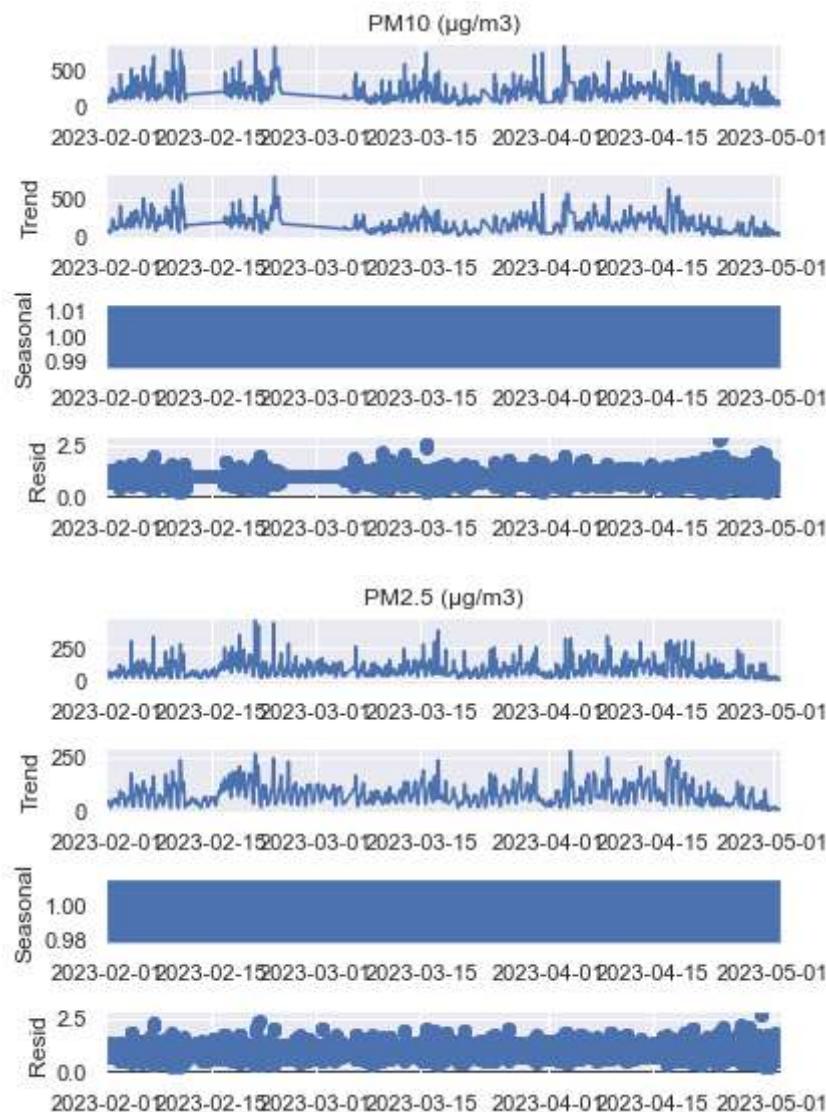


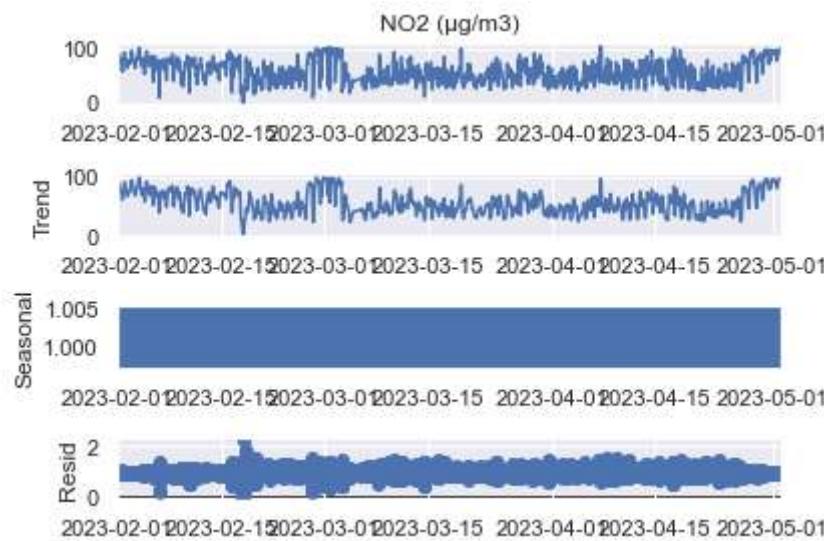
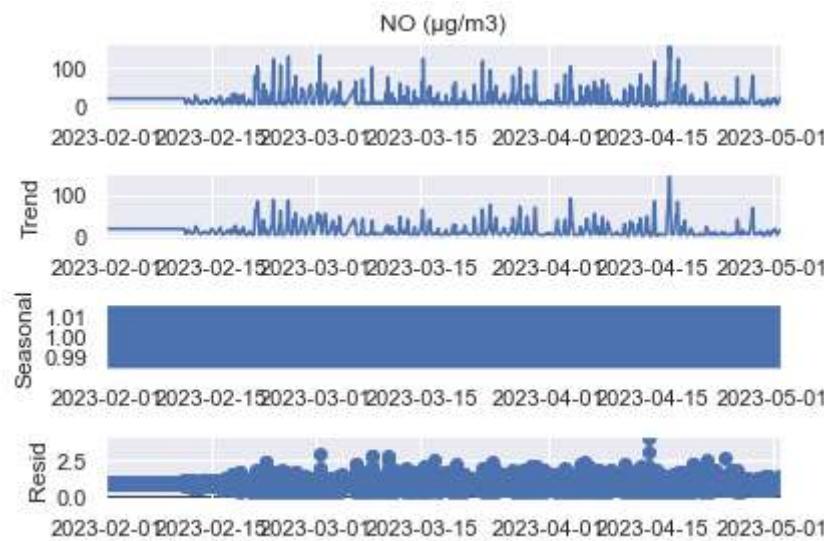
NH₃ ($\mu\text{g}/\text{m}^3$)Ozone ($\mu\text{g}/\text{m}^3$)

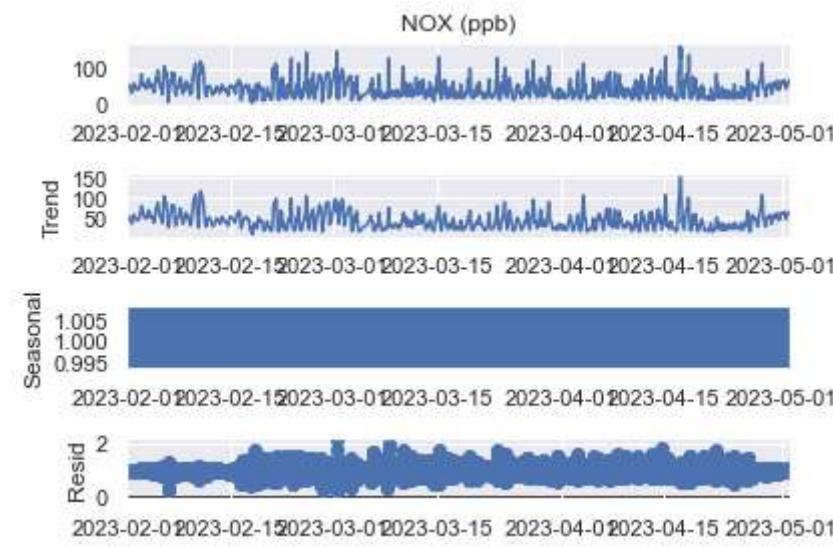


Seasonality - Multiplicative

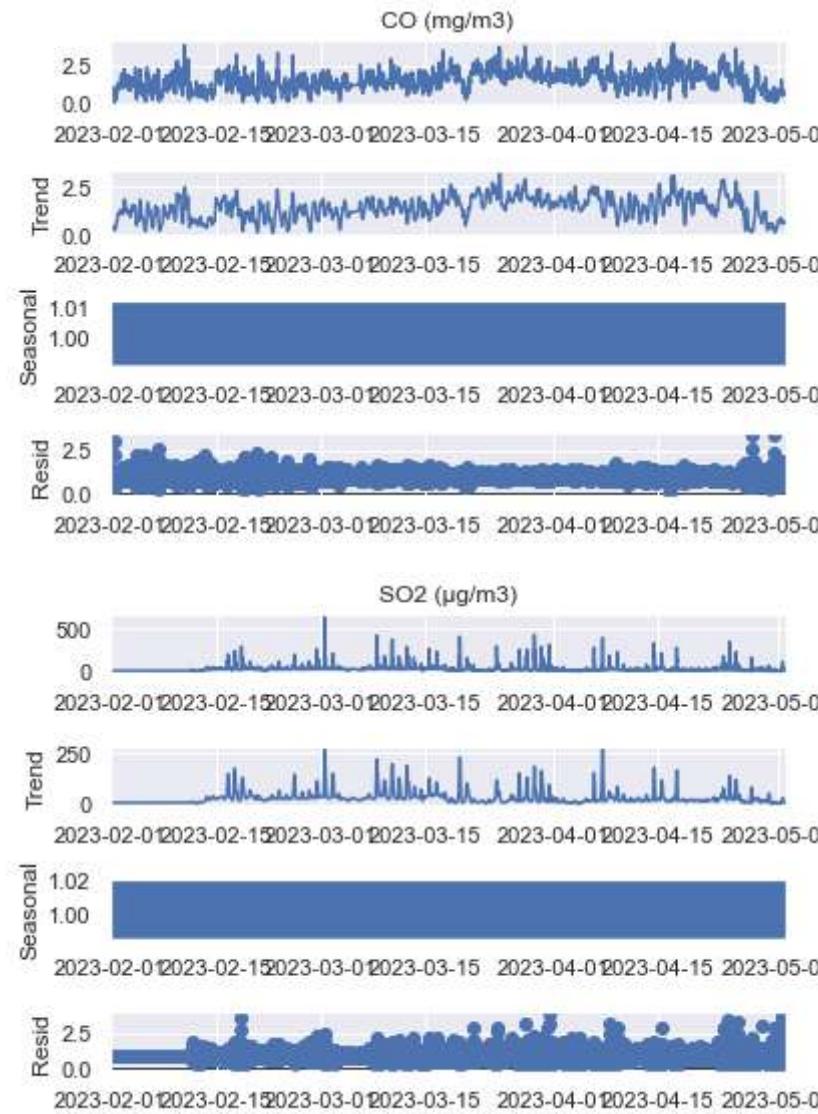
```
In [296]: for i in range(len(df_new.columns)//2):
    seasonality(df_new,df_new.columns[i], "multiplicative")
```

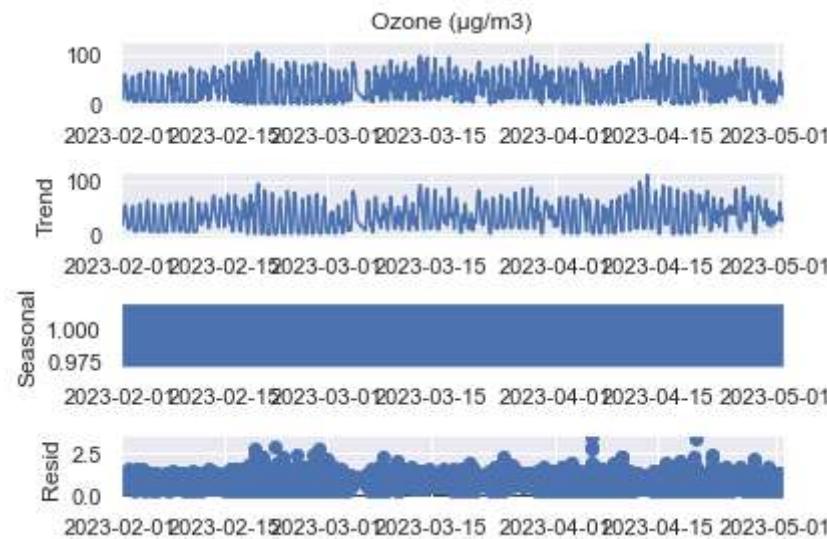
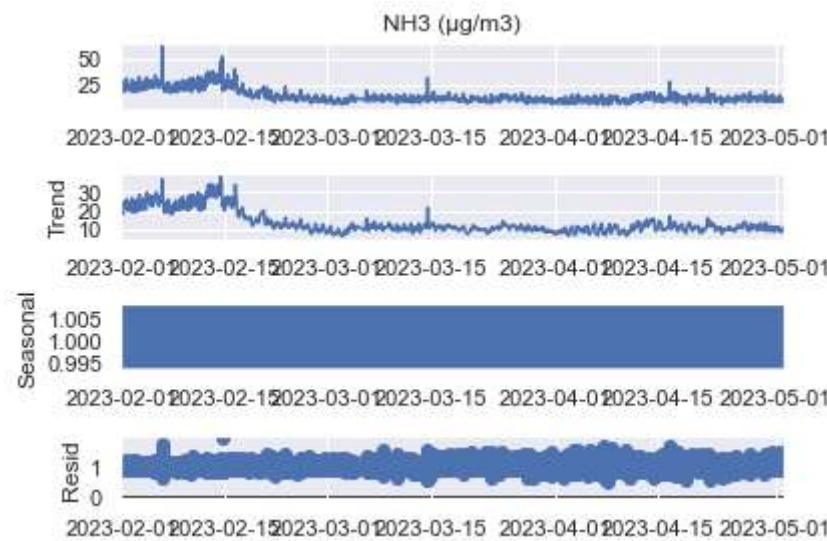






```
In [297]: for i in range(len(df_new.columns)//2 ,len(df_new.columns)-1):
    seasonality(df_new,df_new.columns[i], "multiplicative")
```





Benzene has some negative values so multiplicative seasonality cannot be plotted for it.

Correlation measures the similarity in the change of values of two series

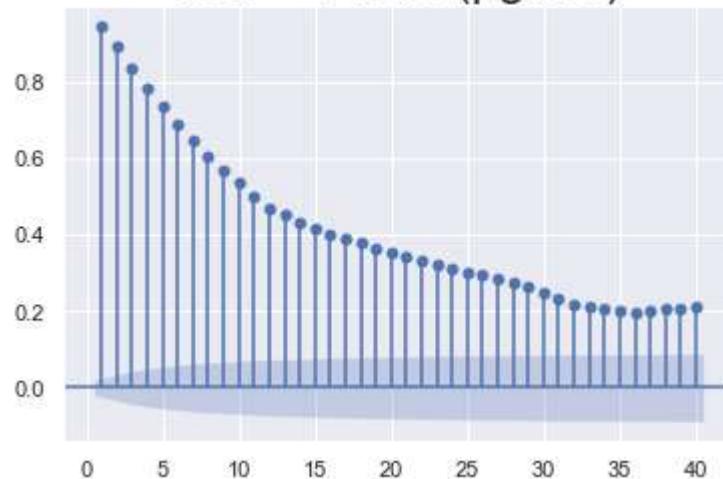
Autocorrelation is the correlation between a sequence and itself

ACF plots

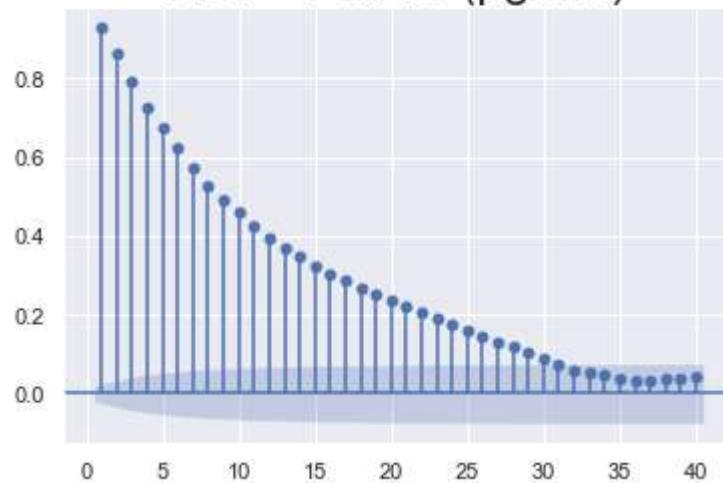
```
In [298]: def acf_plots(df,parameter):
    sgt.plot_acf(df[parameter], lags = 40, zero = False)
    plt.title("ACF - "+ parameter, size = 24)
    plt.show()
    # The greater the distance in time, the more unlikely it is that this autocorrelation persists
```

```
In [299]: for i in range(len(df_new.columns)//2):
    acf_plots(df_new,df_new.columns[i])
```

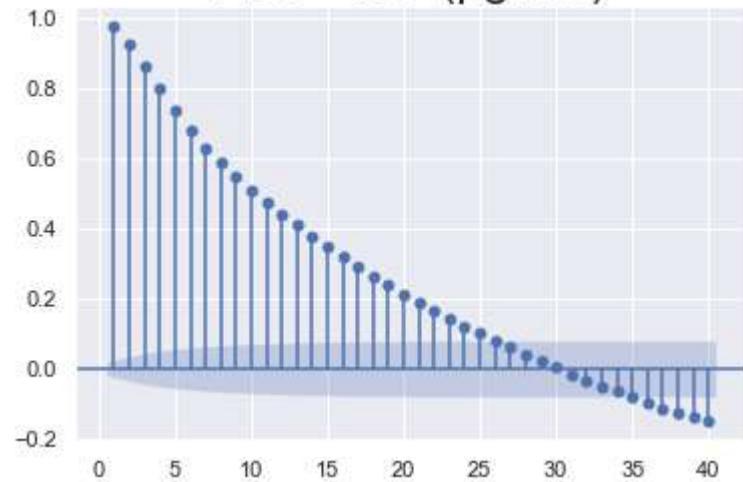
ACF - PM10 ($\mu\text{g}/\text{m}^3$)



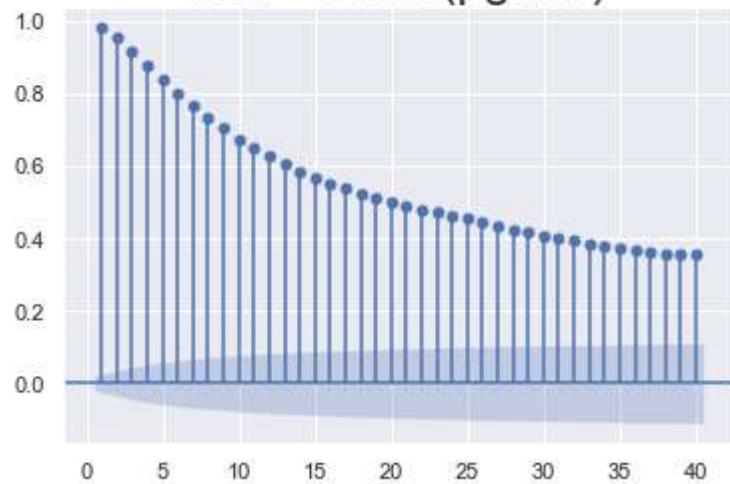
ACF - PM2.5 ($\mu\text{g}/\text{m}^3$)



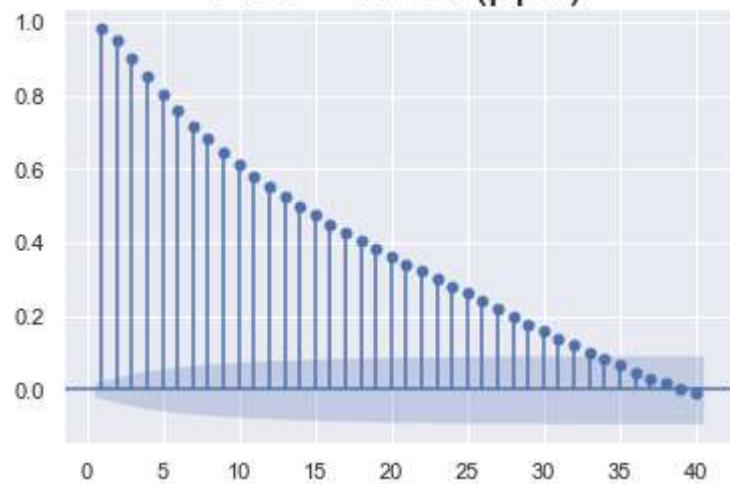
ACF - NO ($\mu\text{g}/\text{m}^3$)



ACF - NO₂ ($\mu\text{g}/\text{m}^3$)

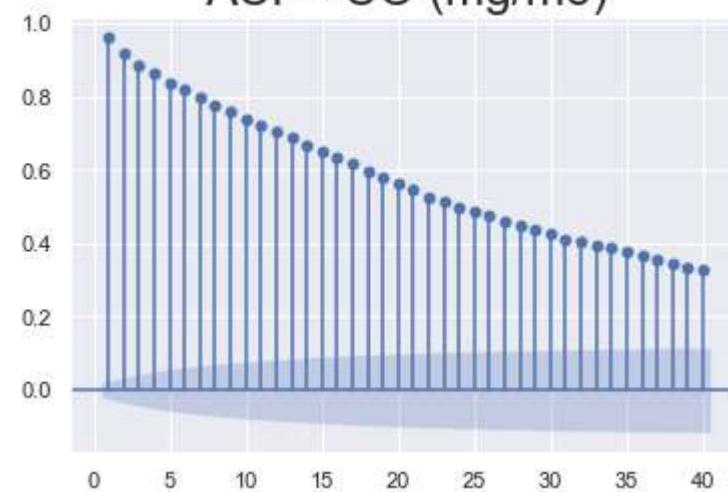


ACF - NOX (ppb)

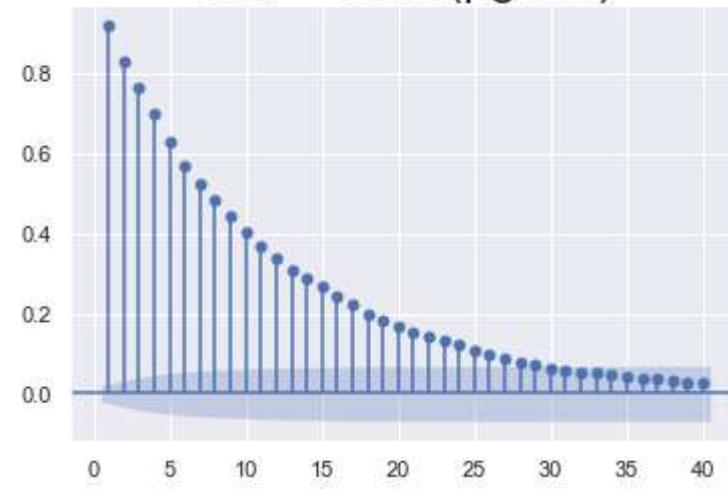


```
In [300]: for i in range(len(df_new.columns)//2 ,len(df_new.columns)):
    acf_plots(df_new,df_new.columns[i])
```

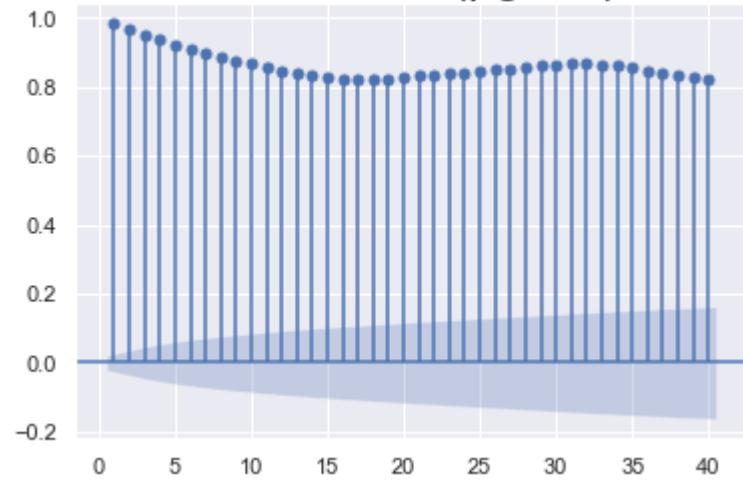
ACF - CO (mg/m3)



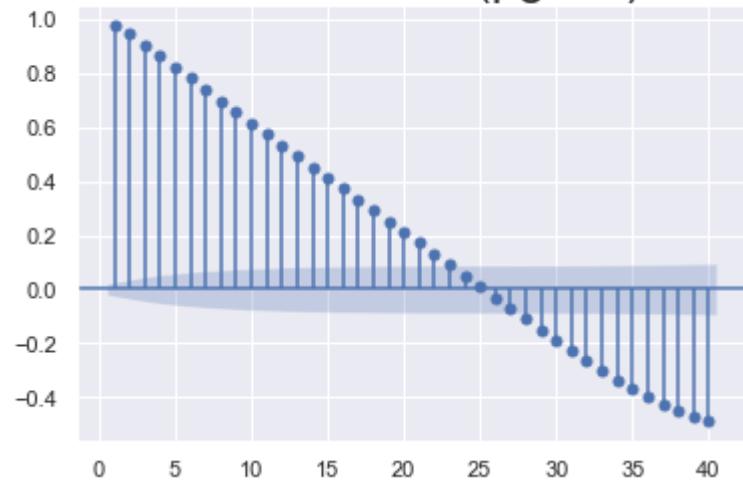
ACF - SO2 (μg/m3)



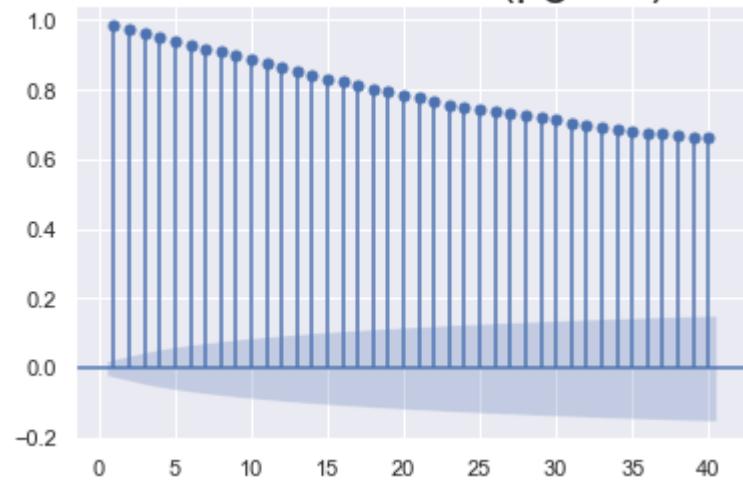
ACF - NH₃ ($\mu\text{g}/\text{m}^3$)



ACF - Ozone ($\mu\text{g}/\text{m}^3$)



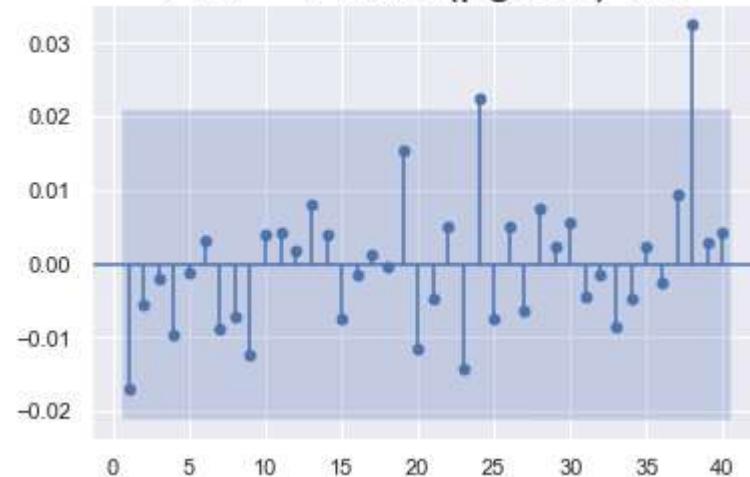
ACF - Benzene ($\mu\text{g}/\text{m}^3$)



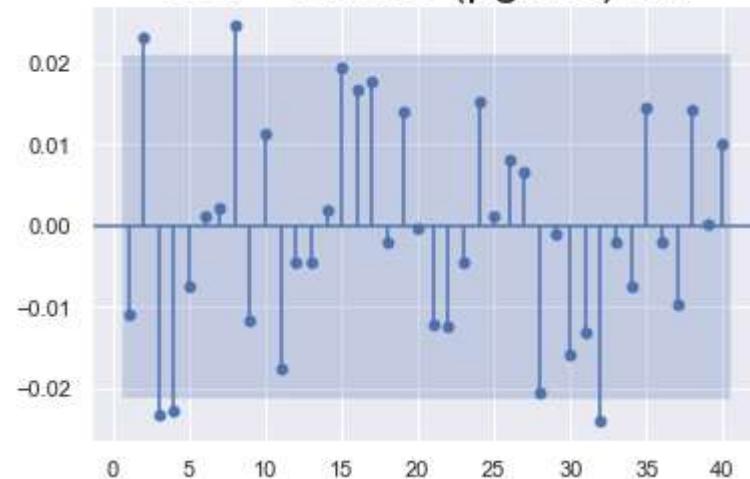
ACF plots for White noise values

```
In [301]: for i in range(len(df_wn.columns)//2):  
    acf_plots(df_wn,df_wn.columns[i])
```

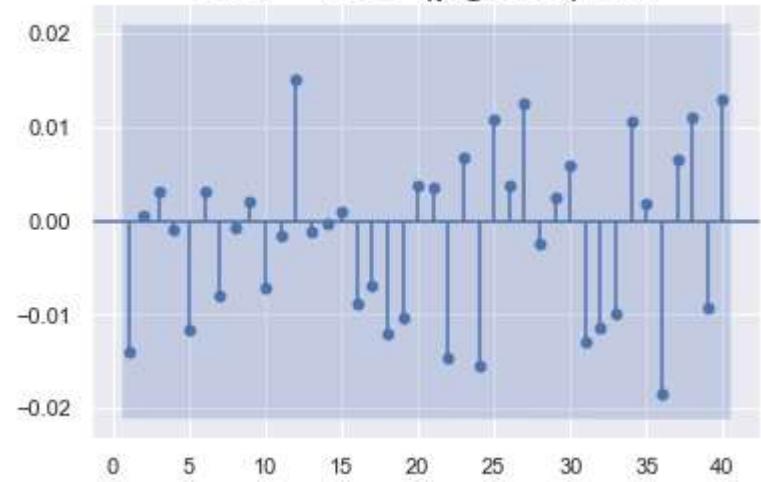
ACF - PM10 ($\mu\text{g}/\text{m}^3$) wn



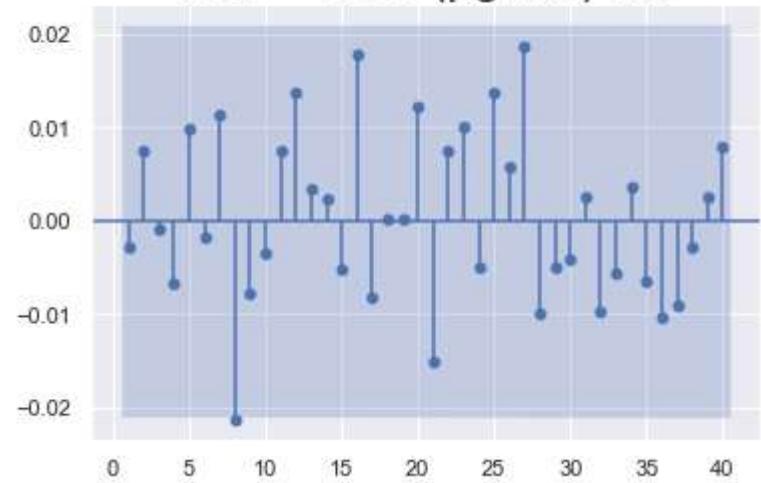
ACF - PM2.5 ($\mu\text{g}/\text{m}^3$) wn



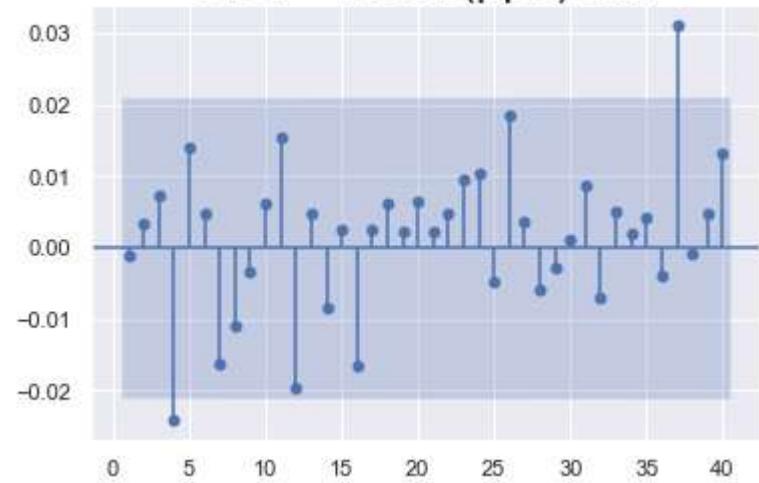
ACF - NO ($\mu\text{g}/\text{m}^3$) wn



ACF - NO₂ ($\mu\text{g}/\text{m}^3$) wn

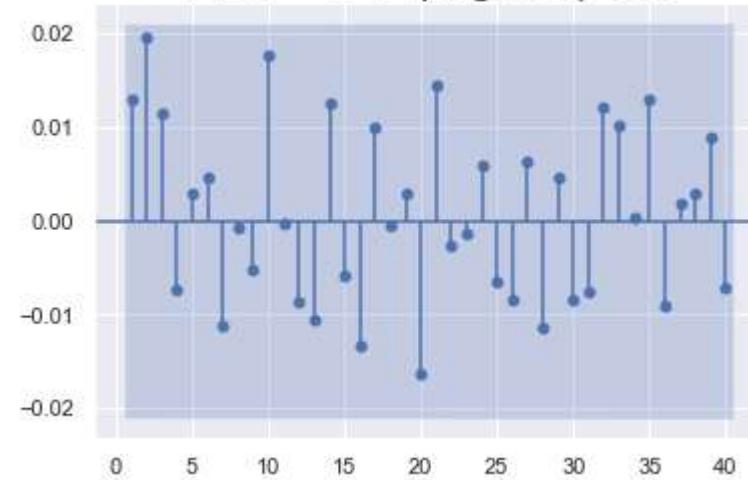


ACF - NOX (ppb) wn

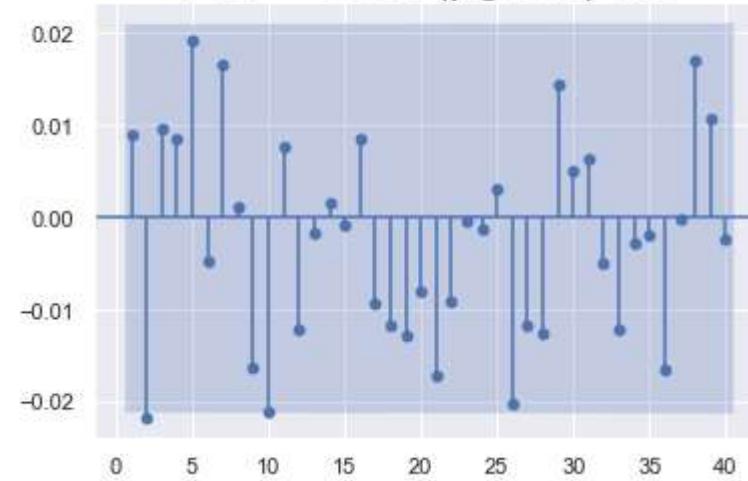


```
In [302]: for i in range(len(df_wn.columns)//2 ,len(df_wn.columns)):  
    acf_plots(df_wn,df_wn.columns[i])
```

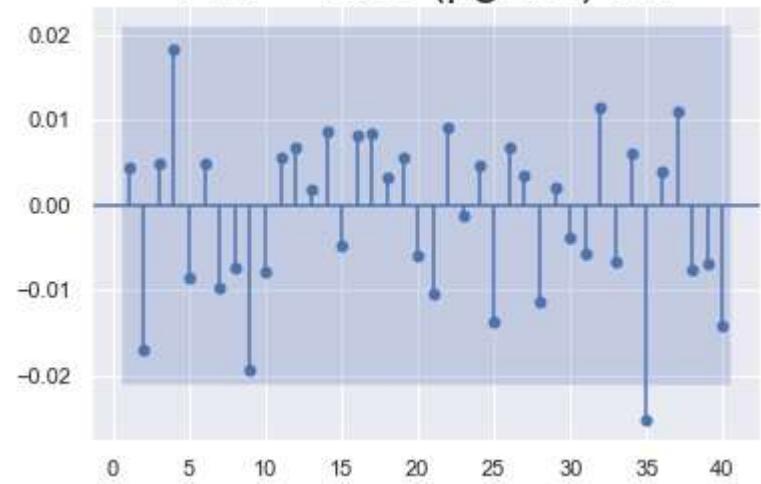
ACF - CO (mg/m³) wn



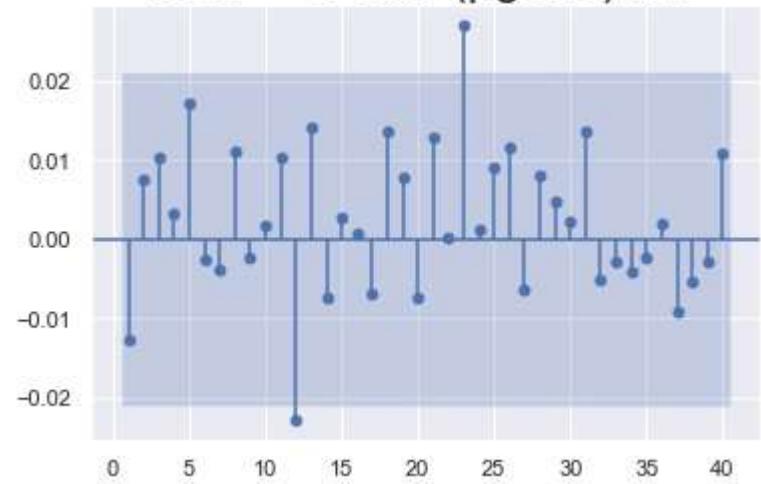
ACF - SO₂ (μg/m³) wn



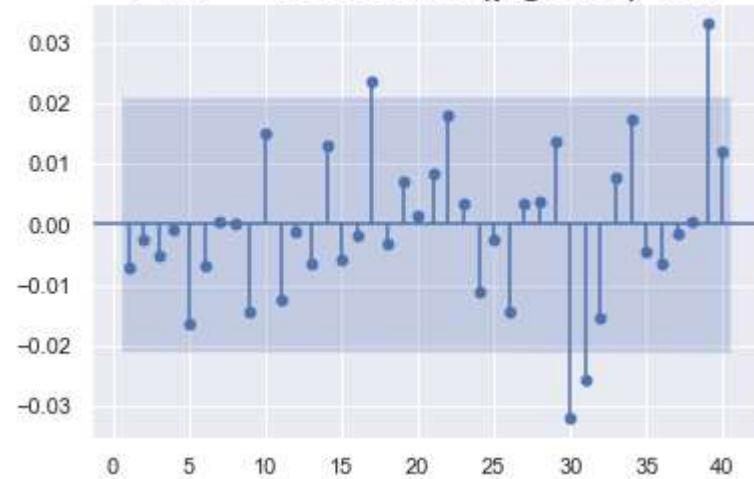
ACF - NH₃ ($\mu\text{g}/\text{m}^3$) wn



ACF - Ozone ($\mu\text{g}/\text{m}^3$) wn



ACF - Benzene ($\mu\text{g}/\text{m}^3$) wn



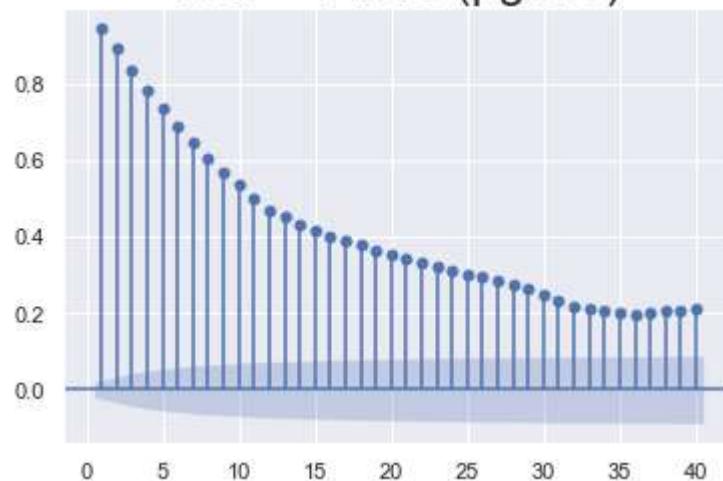
PACF plots

```
In [303]: def pacf_plots(df,parameter):
    sgt.plot_pacf(df[parameter], lags = 40, zero = False, method = ('ols'))
    plt.title("PACF - "+ parameter, size = 24)
    plt.show()

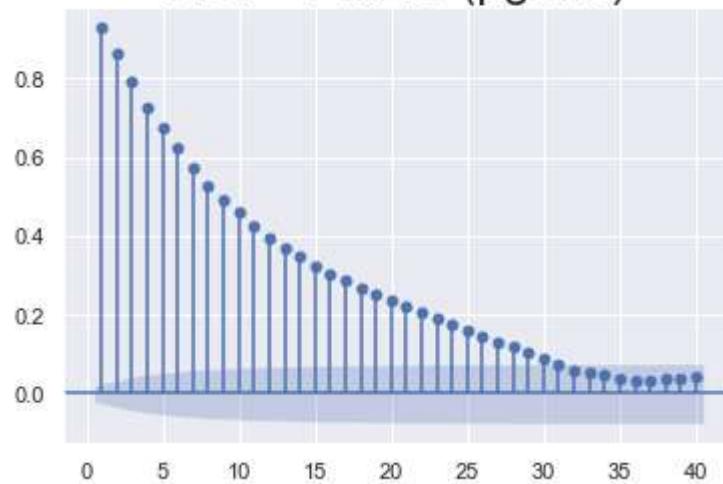
# It cancels out all additional channels a previous period value affects the present one
# Coefficients of different sizes can be observed
# Coefficients that are negative
# coefficients which are not significant
# Coefficient values will be extremely close to zero
# There impact on the model is minimal, so they are not relevant to us
# determine how many Lags model must include
# We would not always be able to spot such a convenient pattern in our data
```

```
In [304]: for i in range(len(df_new.columns)//2):
    acf_plots(df_new,df_new.columns[i])
```

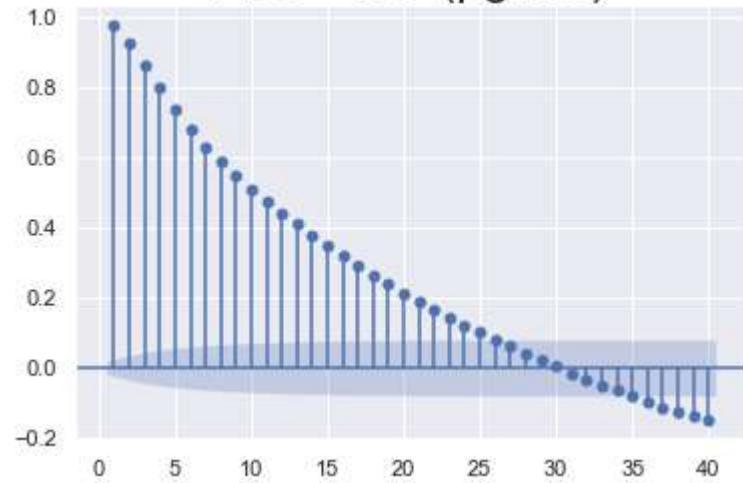
ACF - PM10 ($\mu\text{g}/\text{m}^3$)



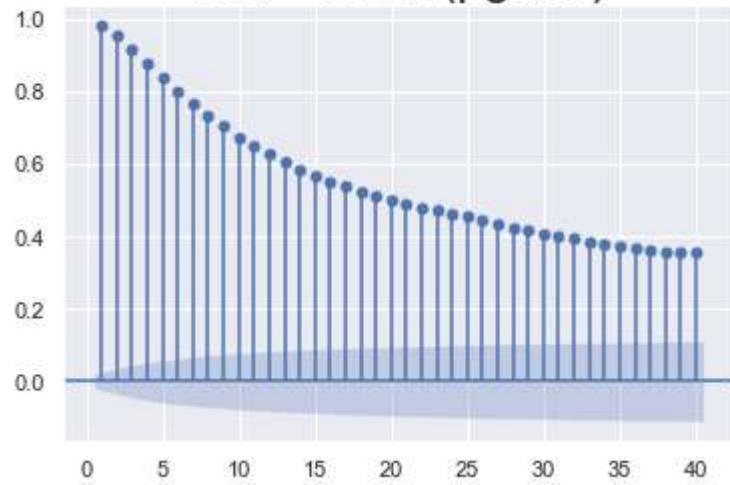
ACF - PM2.5 ($\mu\text{g}/\text{m}^3$)



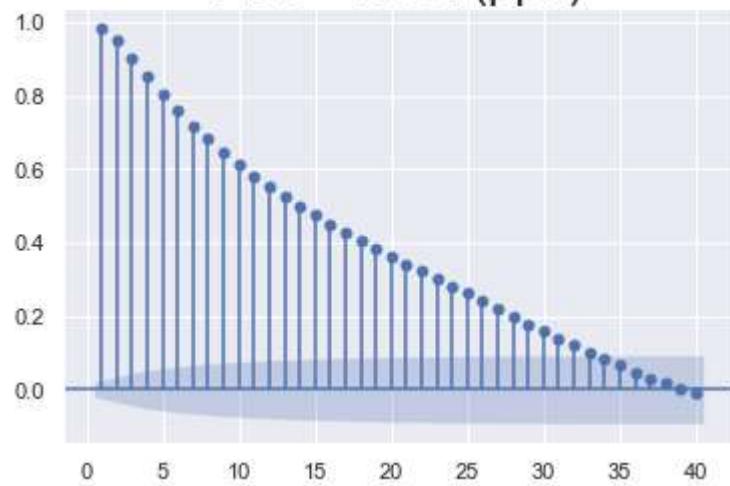
ACF - NO ($\mu\text{g}/\text{m}^3$)



ACF - NO₂ ($\mu\text{g}/\text{m}^3$)

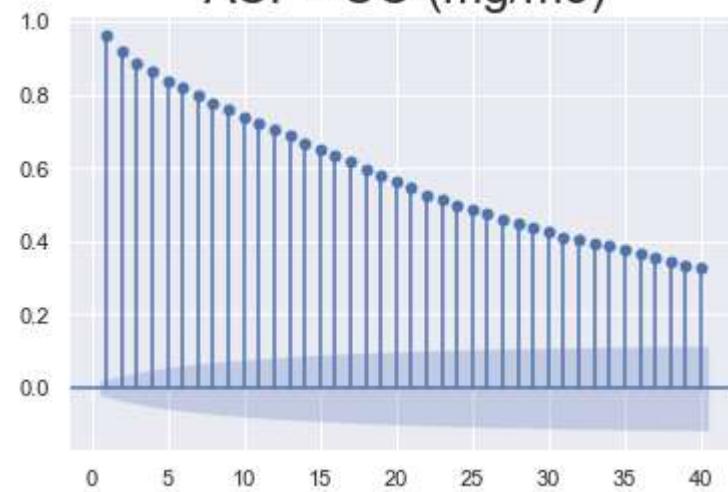


ACF - NOX (ppb)

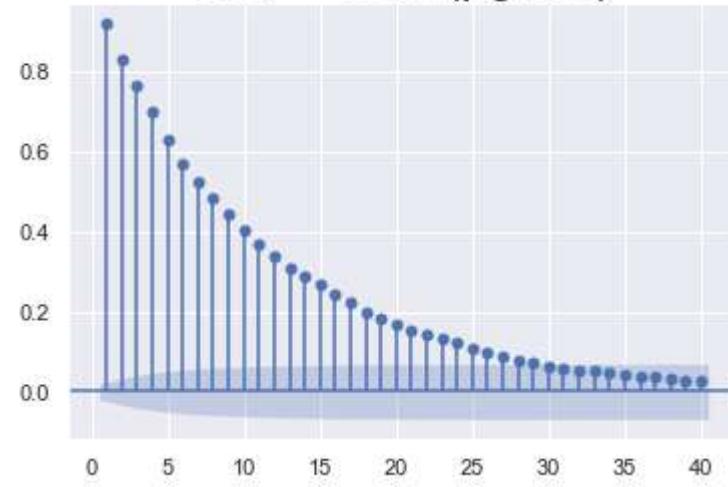


```
In [305]: for i in range(len(df_new.columns)//2 ,len(df_new.columns)):
    acf_plots(df_new,df_new.columns[i])
```

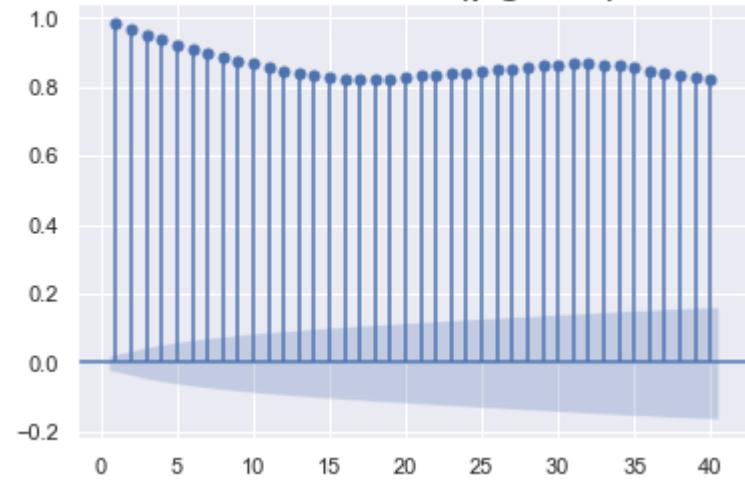
ACF - CO (mg/m3)



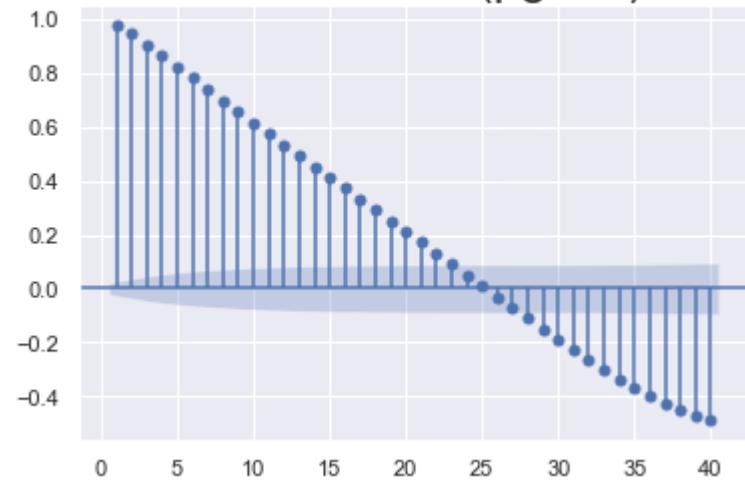
ACF - SO2 (μg/m3)



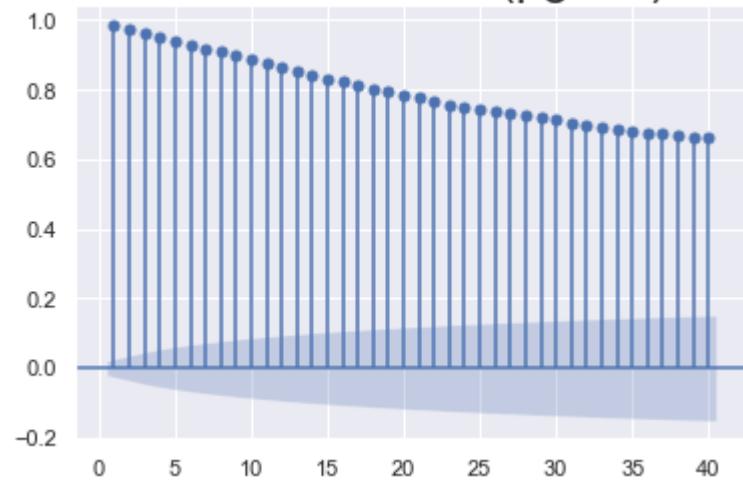
ACF - NH₃ ($\mu\text{g}/\text{m}^3$)



ACF - Ozone ($\mu\text{g}/\text{m}^3$)



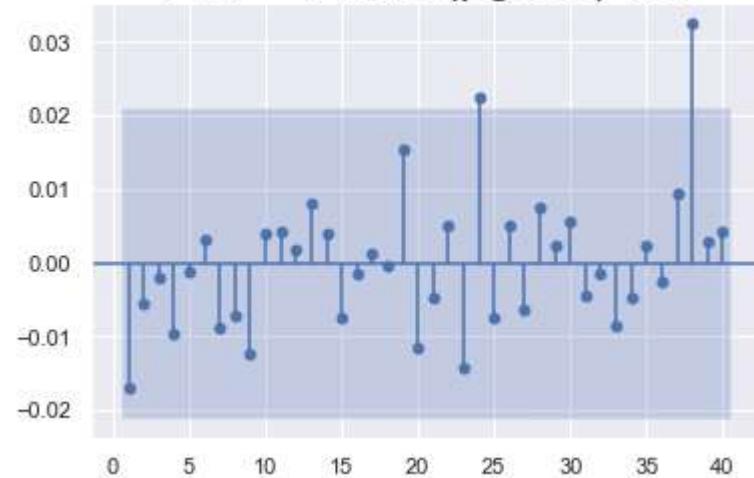
ACF - Benzene ($\mu\text{g}/\text{m}^3$)



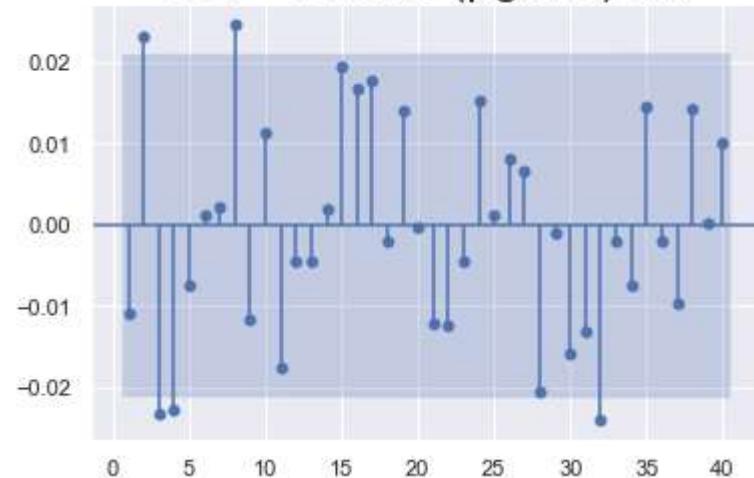
PACF plots for white noise values

```
In [306]: for i in range(len(df_wn.columns)//2):  
    acf_plots(df_wn,df_wn.columns[i])
```

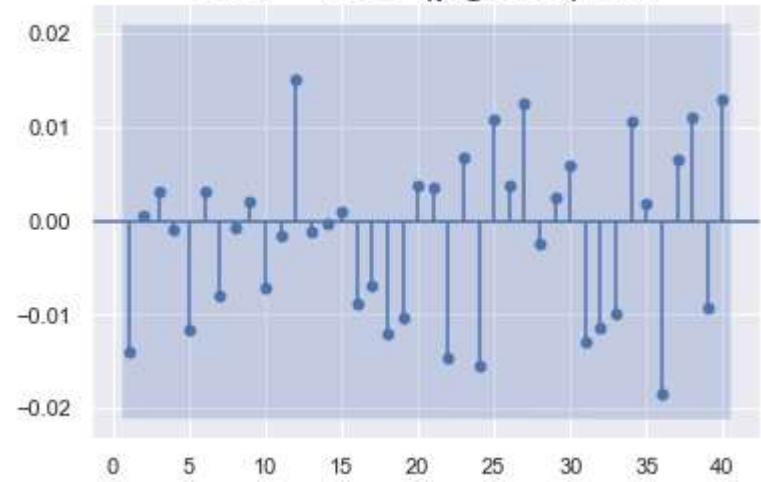
ACF - PM10 ($\mu\text{g}/\text{m}^3$) wn



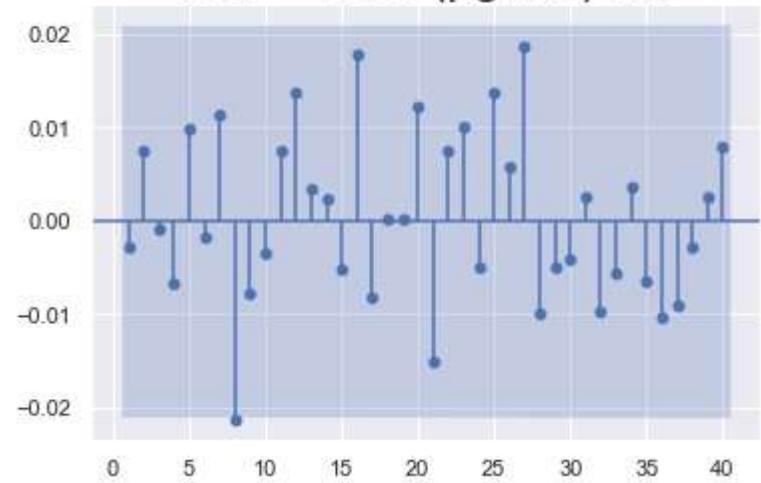
ACF - PM2.5 ($\mu\text{g}/\text{m}^3$) wn



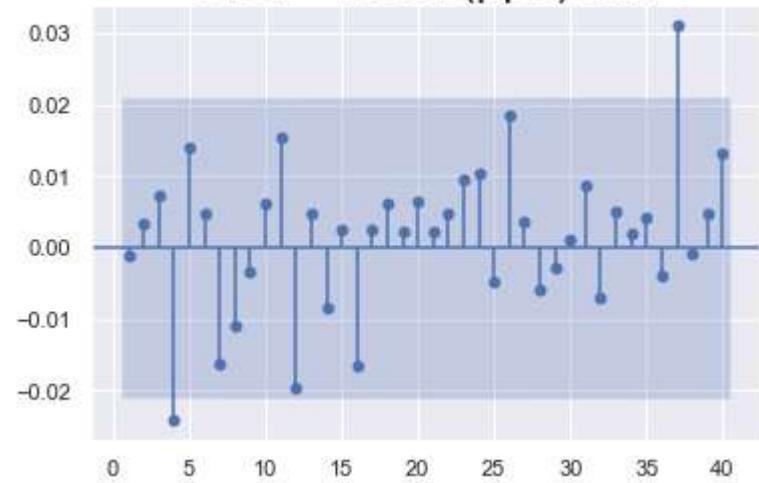
ACF - NO ($\mu\text{g}/\text{m}^3$) wn



ACF - NO₂ ($\mu\text{g}/\text{m}^3$) wn

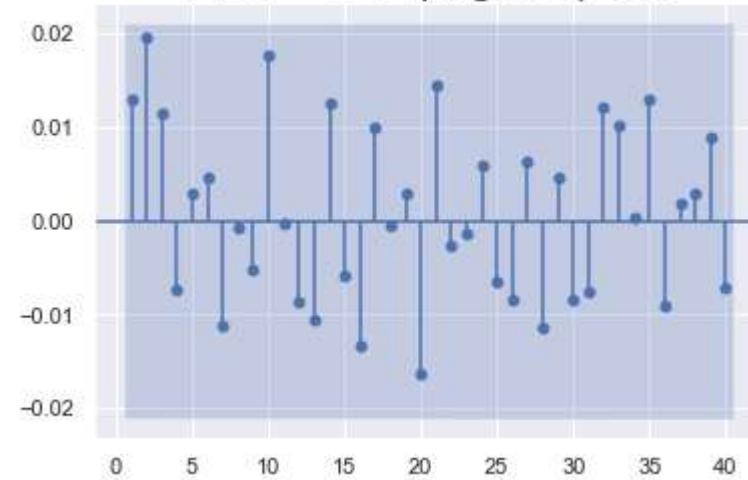


ACF - NOX (ppb) wn

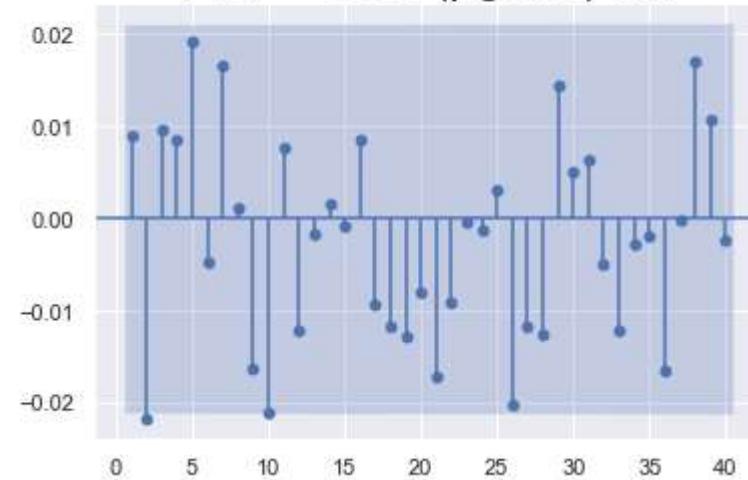


```
In [307]: for i in range(len(df_wn.columns)//2 ,len(df_wn.columns)):
    acf_plots(df_wn,df_wn.columns[i])
```

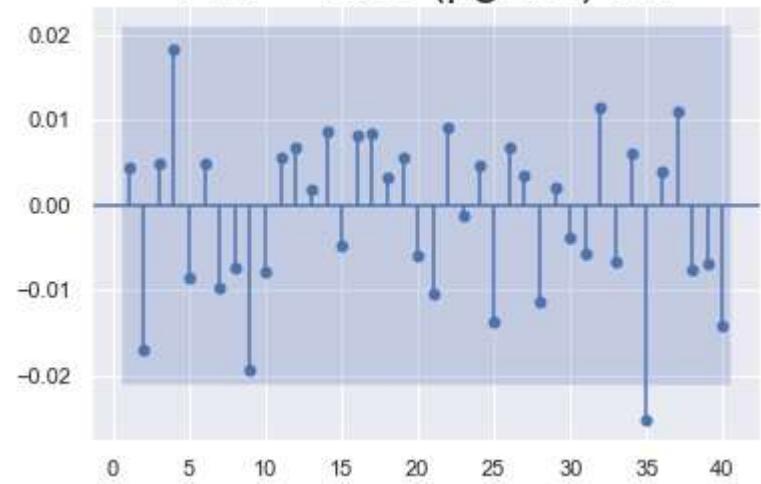
ACF - CO (mg/m³) wn



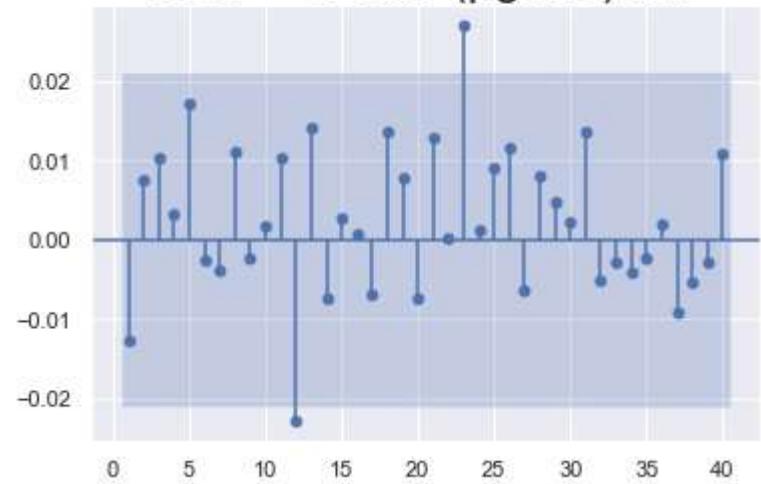
ACF - SO₂ ($\mu\text{g}/\text{m}^3$) wn



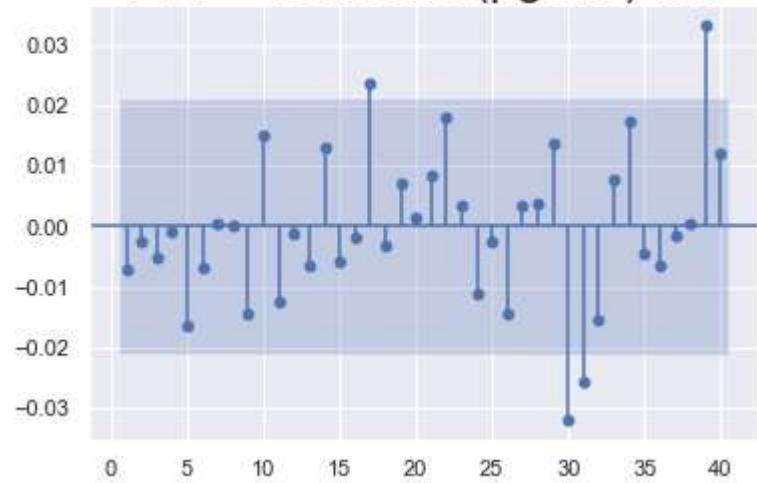
ACF - NH₃ ($\mu\text{g}/\text{m}^3$) wn



ACF - Ozone ($\mu\text{g}/\text{m}^3$) wn



ACF - Benzene ($\mu\text{g}/\text{m}^3$) wn



PACF: $X_{(t-2)} \rightarrow X_{(t)}$

cancels out $X_{(t-2)} \rightarrow X_{(t-1)} \rightarrow X_{(t)}$

direct effect considered

ACF: $X_{(t-2)} \rightarrow X_{(t)}$

$X_{(t-2)} \rightarrow X_{(t-1)} \rightarrow X_{(t)}$

considers all the effects

Picking the correct model in Time-Series analysis

1. Significant coefficients

2. Parsimonious (as simple as possible)

To check significantly better predictions compared to simpler model over complex model-

Use Log-Likelihood Ratio Test

same total number of lags \rightarrow No LLR

Information criteria (AIC,BIC)

Residuals - White Noise

A linear model, where current period values are a sum of past outcomes multiplied by a numeric factor

$X_{(t-1)}$ = The values of X during the previous period

phi = Any numeric constant by which we multiply the lagged variable [-1,1]

If mod phi greater than 1, the values keep on increasing and the system might blow up

epsilon t is residual- The difference between our prediction for period "t" and the correct value

$$X_t = C + \phi * X_{t-1} + \epsilon_t$$

How many lag values ?

More lags means more complicated model

We use ACF and PACF to determine the appropriate number of lags in a model

```
In [308]: # Define the number of rows in each split dataframe
split_size = 96

# Calculate the number of splits needed
days = len(df) // split_size

# Create a list to store the split dataframes
df_days = []

# Use a for loop to split the dataframe
for i in range(days):
    start_idx = i * split_size
    end_idx = start_idx + split_size

    # Extract the rows for the current split
    df_new_day = df_new[start_idx:end_idx]

    # Append the split dataframe to the list
    df_days.append(df_new_day)
```

```
In [309]: len(df_days)
```

```
Out[309]: 90
```

Generally, model with higher Log-Likelihood and Lower information criterion is preferred

LLR Test

```
In [310]: def LLR_test(mod_1, mod_2, DF=1):
    L1 = mod_1.fit().llf
    L2 = mod_2.fit().llf
    LR = (2*(L2-L1))
    p = chi2.sf(LR, DF).round(3)
    return p
```

Autoregressive Model

```
In [311]: def fit_ar_model(data, column_name, order):
    model = sm.tsa.AR(data[column_name])
    ar_model = model.fit(maxlag=order, method='mle')
    return ar_model
```

```
In [312]: order = 1 # Example order of the AR model

ar_models = {}
for column in df_new.columns: # List of column names
    ar_models[column] = fit_ar_model(df_new, column, order)
```

```
In [313]: # Extract the lag coefficients from each AR model
lag_coefs = []
for column in df_new.columns:
    lag_coefs.append(ar_models[column].params[1:])

# Stack the lag coefficients horizontally to create the VAR input
var_input = np.hstack(lag_coefs)

# Fit the VAR model
var_model = VAR(df_new)
var_result = var_model.fit(order)
```

In [314]: var_result.summary()

Out[314]: Summary of Regression Results

=====

Model: VAR
Method: OLS
Date: Wed, 28, Jun, 2023
Time: 04:16:02

No. of Equations: 10.0000 BIC: 14.1154
Nobs: 8639.00 HQIC: 14.0561
Log likelihood: -183055. FPE: 1.23364e+06
AIC: 14.0255 Det(Omega_mle): 1.21804e+06

Results for equation PM10 ($\mu\text{g}/\text{m}^3$)

=====

	coefficient	std. error	t-stat	prob
const	-2.871378	2.663747	-1.078	0.281
L1.PM10 ($\mu\text{g}/\text{m}^3$)	0.892708	0.004207	212.189	0.000
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.208022	0.011031	18.858	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	-0.239749	0.061842	-3.877	0.000
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.094234	0.046256	-2.037	0.042
L1.NOX (ppb)	0.251124	0.072463	3.466	0.001
L1.CO (mg/m^3)	0.441973	0.832057	0.531	0.595
L1.SO2 ($\mu\text{g}/\text{m}^3$)	-0.027786	0.011309	-2.457	0.014
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.345745	0.090154	3.835	0.000
L1.Ozone ($\mu\text{g}/\text{m}^3$)	0.010826	0.021848	0.496	0.620
L1.Benzene ($\mu\text{g}/\text{m}^3$)	-6.401409	6.483445	-0.987	0.323

=====

Results for equation PM2.5 ($\mu\text{g}/\text{m}^3$)

=====

	coefficient	std. error	t-stat	prob
const	8.753989	1.303791	6.714	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.002084	0.002059	-1.012	0.312
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.843452	0.005399	156.222	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	0.203958	0.030269	6.738	0.000
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.072427	0.022640	-3.199	0.001
L1.NOX (ppb)	-0.018333	0.035468	-0.517	0.605
L1.CO (mg/m^3)	5.420190	0.407256	13.309	0.000
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.012515	0.005535	2.261	0.024
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.083626	0.044127	1.895	0.058
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.129757	0.010694	-12.134	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	12.750542	3.173371	4.018	0.000

=====

Results for equation NO ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	2.308878	0.245298	9.413	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	0.000718	0.000387	1.854	0.064
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.016351	0.001016	-16.097	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	0.994193	0.005695	174.577	0.000
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.004244	0.004260	-0.996	0.319
L1.NOX (ppb)	-0.024924	0.006673	-3.735	0.000
L1.CO (mg/m ³)	0.779946	0.076622	10.179	0.000
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.005000	0.001041	4.801	0.000
L1.NH3 ($\mu\text{g}/\text{m}^3$)	-0.001212	0.008302	-0.146	0.884
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.034064	0.002012	-16.931	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	2.709161	0.597045	4.538	0.000

Results for equation NO2 ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	3.837970	0.227521	16.869	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.001753	0.000359	-4.878	0.000
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.006359	0.000942	-6.749	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	0.001934	0.005282	0.366	0.714
L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.955950	0.003951	241.960	0.000
L1.NOX (ppb)	0.002078	0.006189	0.336	0.737
L1.CO (mg/m ³)	-0.092099	0.071069	-1.296	0.195
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.014920	0.000966	15.445	0.000
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.024112	0.007700	3.131	0.002
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.041214	0.001866	-22.086	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	1.242332	0.553775	2.243	0.025

Results for equation NOX (ppb)

	coefficient	std. error	t-stat	prob
const	3.844053	0.248824	15.449	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.000265	0.000393	-0.673	0.501
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.016314	0.001030	-15.833	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	0.001595	0.005777	0.276	0.782
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.022867	0.004321	-5.292	0.000
L1.NOX (ppb)	0.971336	0.006769	143.500	0.000
L1.CO (mg/m ³)	0.677632	0.077723	8.719	0.000
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.013047	0.001056	12.350	0.000

L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.015002	0.008421	1.781	0.075
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.051711	0.002041	-25.338	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	2.894526	0.605626	4.779	0.000

Results for equation CO (mg/m^3)

	coefficient	std. error	t-stat	prob
const	0.167019	0.011362	14.700	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	0.000056	0.000018	3.118	0.002
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.000267	0.000047	-5.667	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	-0.000255	0.000264	-0.965	0.335
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.000397	0.000197	-2.014	0.044
L1.NOX (ppb)	-0.000192	0.000309	-0.622	0.534
L1.CO (mg/m^3)	0.952169	0.003549	268.297	0.000
L1.SO2 ($\mu\text{g}/\text{m}^3$)	-0.000129	0.000048	-2.680	0.007
L1.NH3 ($\mu\text{g}/\text{m}^3$)	-0.000747	0.000385	-1.943	0.052
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.001030	0.000093	-11.051	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	-0.108807	0.027654	-3.935	0.000

Results for equation SO2 ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	0.441933	1.033134	0.428	0.669
L1.PM10 ($\mu\text{g}/\text{m}^3$)	0.002028	0.001632	1.243	0.214
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.012116	0.004278	2.832	0.005
L1.NO ($\mu\text{g}/\text{m}^3$)	0.008653	0.023985	0.361	0.718
L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.038773	0.017940	2.161	0.031
L1.NOX (ppb)	-0.076777	0.028105	-2.732	0.006
L1.CO (mg/m^3)	0.850969	0.322713	2.637	0.008
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.920539	0.004386	209.864	0.000
L1.NH3 ($\mu\text{g}/\text{m}^3$)	-0.011300	0.034966	-0.323	0.747
L1.Ozone ($\mu\text{g}/\text{m}^3$)	0.021540	0.008474	2.542	0.011
L1.Benzene ($\mu\text{g}/\text{m}^3$)	-0.285070	2.514603	-0.113	0.910

Results for equation NH3 ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	0.247951	0.071158	3.485	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.000048	0.000112	-0.430	0.667
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.001140	0.000295	-3.870	0.000

L1.NO ($\mu\text{g}/\text{m}^3$)	0.002610	0.001652	1.580	0.114
L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.001743	0.001236	1.410	0.158
L1.NOX (ppb)	0.000152	0.001936	0.078	0.938
L1.CO (mg/m^3)	0.036340	0.022227	1.635	0.102
L1.SO2 ($\mu\text{g}/\text{m}^3$)	-0.000768	0.000302	-2.541	0.011
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.973141	0.002408	404.071	0.000
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.000609	0.000584	-1.044	0.297
L1.Benzene ($\mu\text{g}/\text{m}^3$)	1.113182	0.173196	6.427	0.000

Results for equation Ozone ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	-3.854470	0.369634	-10.428	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.000422	0.000584	-0.722	0.470
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.017308	0.001531	11.308	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	-0.007417	0.008581	-0.864	0.387
L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.027430	0.006419	4.274	0.000
L1.NOX (ppb)	0.001623	0.010055	0.161	0.872
L1.CO (mg/m^3)	0.093928	0.115460	0.814	0.416
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.018621	0.001569	11.866	0.000
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.027364	0.012510	2.187	0.029
L1.Ozone ($\mu\text{g}/\text{m}^3$)	1.002230	0.003032	330.584	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	-0.483965	0.899674	-0.538	0.591

Results for equation Benzene ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	-0.000378	0.001011	-0.374	0.708
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.000004	0.000002	-2.205	0.027
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.000003	0.000004	0.637	0.524
L1.NO ($\mu\text{g}/\text{m}^3$)	-0.000130	0.000023	-5.523	0.000
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.000064	0.000018	-3.635	0.000
L1.NOX (ppb)	0.000129	0.000027	4.689	0.000
L1.CO (mg/m^3)	0.000762	0.000316	2.413	0.016
L1.SO2 ($\mu\text{g}/\text{m}^3$)	-0.000005	0.000004	-1.266	0.206
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.000190	0.000034	5.553	0.000
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.000039	0.000008	-4.722	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	0.975188	0.002460	396.464	0.000

Correlation matrix of residuals

PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)	CO (mg/m^3)	SO2 ($\mu\text{g}/\text{m}^3$)	NH3 ($\mu\text{g}/\text{m}^3$)	Ozone
-----------------------------------	------------------------------------	---------------------------------	----------------------------------	-----------	-------------------------------	----------------------------------	----------------------------------	-------

In [315]: # Correlation matrix of residuals

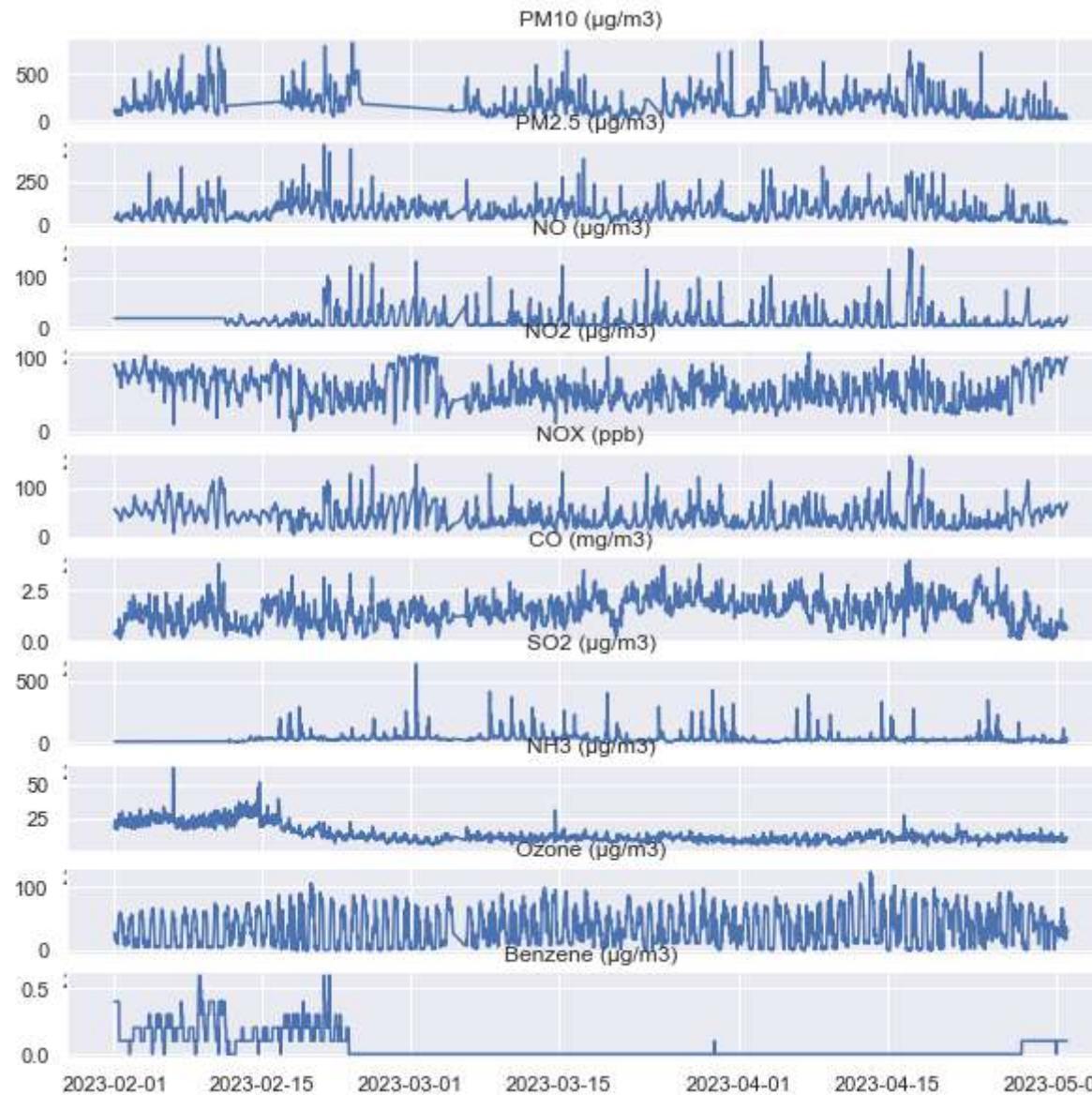
#	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)
# PM10 ($\mu\text{g}/\text{m}^3$)	1.000000	-0.006425	0.034820	0.000808	0.039149
# PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.006425	1.000000	0.088805	0.100111	0.130629
# NO ($\mu\text{g}/\text{m}^3$)	0.034820	0.088805	1.000000	0.113358	0.852916
# NO2 ($\mu\text{g}/\text{m}^3$)	0.000808	0.100111	0.113358	1.000000	0.574285
# NOX (ppb)	0.039149	0.130629	0.852916	0.574285	1.000000
# CO (mg/m^3)	-0.008890	-0.070251	0.038088	-0.008029	0.029488
# SO2 ($\mu\text{g}/\text{m}^3$)	-0.005590	0.003422	0.194194	0.145064	0.229492
# NH3 ($\mu\text{g}/\text{m}^3$)	-0.001271	0.010250	-0.009239	-0.070100	-0.043572
# Ozone ($\mu\text{g}/\text{m}^3$)	-0.017473	-0.010157	-0.131053	-0.147780	-0.189803
# Benzene ($\mu\text{g}/\text{m}^3$)	0.026593	0.079896	0.019926	0.026682	0.042896
#	CO (mg/m^3)	SO2 ($\mu\text{g}/\text{m}^3$)	NH3 ($\mu\text{g}/\text{m}^3$)	Ozone ($\mu\text{g}/\text{m}^3$)	Benzene ($\mu\text{g}/\text{m}^3$)
#	-0.008890	-0.005590	-0.001271	-0.017473	0.026593
#	-0.070251	0.003422	0.010250	-0.010157	0.079896
#	0.038088	0.194194	-0.009239	-0.131053	0.019926
#	-0.008029	0.145064	-0.070100	-0.147780	0.026682
#	0.029488	0.229492	-0.043572	-0.189803	0.042896
#	1.000000	-0.002795	0.029230	-0.193322	-0.037005
#	-0.002795	1.000000	-0.033758	-0.501710	-0.003558
#	0.029230	-0.033758	1.000000	0.026097	0.029184
#	-0.193322	-0.501710	0.026097	1.000000	-0.006481
#	-0.037005	-0.003558	0.029184	-0.006481	1.000000

```
In [316]: # Plot the VAR model diagnostics
```

```
var_result.plot()
```

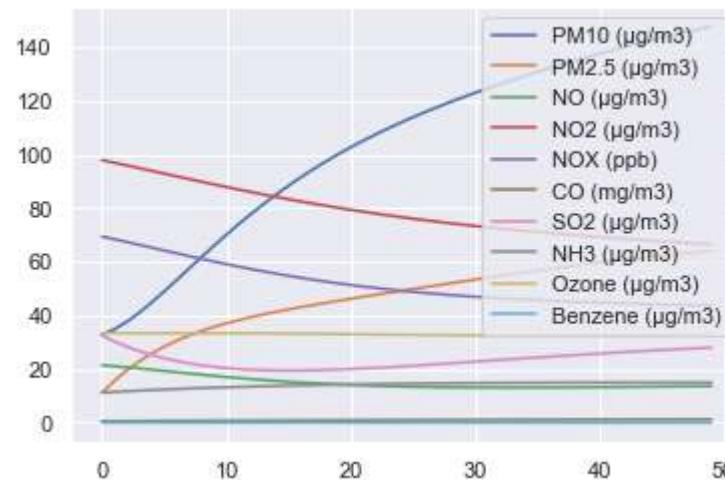
```
# Show the plot
```

```
plt.show()
```



```
In [317]: forecasted_data = var_result.forecast(var_result.y, steps=50)
```

```
In [318]: plt.plot(forecasted_data[:, 0], label=df.columns[2])
plt.plot(forecasted_data[:, 1], label=df.columns[3])
plt.plot(forecasted_data[:, 2], label=df.columns[4])
plt.plot(forecasted_data[:, 3], label=df.columns[5])
plt.plot(forecasted_data[:, 4], label=df.columns[6])
plt.plot(forecasted_data[:, 5], label=df.columns[7])
plt.plot(forecasted_data[:, 6], label=df.columns[8])
plt.plot(forecasted_data[:, 7], label=df.columns[9])
plt.plot(forecasted_data[:, 8], label=df.columns[10])
plt.plot(forecasted_data[:, 9], label=df.columns[11])
plt.legend()
plt.show()
```



Autoregressive Moving Average Model

```
In [319]: order = 1 # Example order of the ARMA model

arma_models = []
for column in df_new.columns: # List of column names
    model = sm.tsa.ARMA(df_new[column], order=(order, order))
    arma_models.append(model)

# Fit the individual ARMA models
arma_results = [model.fit() for model in arma_models]

# Extract the lag coefficients from each ARMA model
lag_coefs = []
for result in arma_results:
    lag_coefs.append(result.params[1:])

# Stack the lag coefficients horizontally to create the VAR input
var_input = np.hstack(lag_coefs)

# Fit the VAR model
var_model = VAR(df_new)
var_result = var_model.fit(order)
```

In [320]: `var_result.summary()`

Out[320]: Summary of Regression Results

Model:	VAR			
Method:	OLS			
Date:	Wed, 28, Jun, 2023			
Time:	04:16:19			
<hr/>				
No. of Equations:	10.0000	BIC:	14.1154	
Nobs:	8639.00	HQIC:	14.0561	
Log likelihood:	-183055.	FPE:	1.23364e+06	
AIC:	14.0255	Det(Omega_mle):	1.21804e+06	
<hr/>				
Results for equation PM10 ($\mu\text{g}/\text{m}^3$)				
	coefficient	std. error	t-stat	prob
const	-2.871378	2.663747	-1.078	0.281
L1.PM10 ($\mu\text{g}/\text{m}^3$)	0.892708	0.004207	212.189	0.000
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.208022	0.011031	18.858	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	-0.239749	0.061842	-3.877	0.000
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.094234	0.046256	-2.037	0.042
L1.NOX (ppb)	0.251124	0.072463	3.466	0.001
L1.CO (mg/m^3)	0.441973	0.832057	0.531	0.595
L1.SO2 ($\mu\text{g}/\text{m}^3$)	-0.027786	0.011309	-2.457	0.014
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.345745	0.090154	3.835	0.000
L1.Ozone ($\mu\text{g}/\text{m}^3$)	0.010826	0.021848	0.496	0.620
L1.Benzene ($\mu\text{g}/\text{m}^3$)	-6.401409	6.483445	-0.987	0.323
<hr/>				
Results for equation PM2.5 ($\mu\text{g}/\text{m}^3$)				
	coefficient	std. error	t-stat	prob
const	8.753989	1.303791	6.714	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.002084	0.002059	-1.012	0.312
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.843452	0.005399	156.222	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	0.203958	0.030269	6.738	0.000
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.072427	0.022640	-3.199	0.001
L1.NOX (ppb)	-0.018333	0.035468	-0.517	0.605
L1.CO (mg/m^3)	5.420190	0.407256	13.309	0.000
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.012515	0.005535	2.261	0.024
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.083626	0.044127	1.895	0.058
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.129757	0.010694	-12.134	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	12.750542	3.173371	4.018	0.000
<hr/>				

Results for equation NO ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	2.308878	0.245298	9.413	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	0.000718	0.000387	1.854	0.064
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.016351	0.001016	-16.097	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	0.994193	0.005695	174.577	0.000
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.004244	0.004260	-0.996	0.319
L1.NOX (ppb)	-0.024924	0.006673	-3.735	0.000
L1.CO (mg/m ³)	0.779946	0.076622	10.179	0.000
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.005000	0.001041	4.801	0.000
L1.NH3 ($\mu\text{g}/\text{m}^3$)	-0.001212	0.008302	-0.146	0.884
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.034064	0.002012	-16.931	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	2.709161	0.597045	4.538	0.000

Results for equation NO2 ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	3.837970	0.227521	16.869	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.001753	0.000359	-4.878	0.000
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.006359	0.000942	-6.749	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	0.001934	0.005282	0.366	0.714
L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.955950	0.003951	241.960	0.000
L1.NOX (ppb)	0.002078	0.006189	0.336	0.737
L1.CO (mg/m ³)	-0.092099	0.071069	-1.296	0.195
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.014920	0.000966	15.445	0.000
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.024112	0.007700	3.131	0.002
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.041214	0.001866	-22.086	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	1.242332	0.553775	2.243	0.025

Results for equation NOX (ppb)

	coefficient	std. error	t-stat	prob
const	3.844053	0.248824	15.449	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.000265	0.000393	-0.673	0.501
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.016314	0.001030	-15.833	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	0.001595	0.005777	0.276	0.782
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.022867	0.004321	-5.292	0.000
L1.NOX (ppb)	0.971336	0.006769	143.500	0.000
L1.CO (mg/m ³)	0.677632	0.077723	8.719	0.000
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.013047	0.001056	12.350	0.000

L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.015002	0.008421	1.781	0.075
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.051711	0.002041	-25.338	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	2.894526	0.605626	4.779	0.000

Results for equation CO (mg/m^3)

	coefficient	std. error	t-stat	prob
const	0.167019	0.011362	14.700	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	0.000056	0.000018	3.118	0.002
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.000267	0.000047	-5.667	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	-0.000255	0.000264	-0.965	0.335
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.000397	0.000197	-2.014	0.044
L1.NOX (ppb)	-0.000192	0.000309	-0.622	0.534
L1.CO (mg/m^3)	0.952169	0.003549	268.297	0.000
L1.SO2 ($\mu\text{g}/\text{m}^3$)	-0.000129	0.000048	-2.680	0.007
L1.NH3 ($\mu\text{g}/\text{m}^3$)	-0.000747	0.000385	-1.943	0.052
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.001030	0.000093	-11.051	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	-0.108807	0.027654	-3.935	0.000

Results for equation SO2 ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	0.441933	1.033134	0.428	0.669
L1.PM10 ($\mu\text{g}/\text{m}^3$)	0.002028	0.001632	1.243	0.214
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.012116	0.004278	2.832	0.005
L1.NO ($\mu\text{g}/\text{m}^3$)	0.008653	0.023985	0.361	0.718
L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.038773	0.017940	2.161	0.031
L1.NOX (ppb)	-0.076777	0.028105	-2.732	0.006
L1.CO (mg/m^3)	0.850969	0.322713	2.637	0.008
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.920539	0.004386	209.864	0.000
L1.NH3 ($\mu\text{g}/\text{m}^3$)	-0.011300	0.034966	-0.323	0.747
L1.Ozone ($\mu\text{g}/\text{m}^3$)	0.021540	0.008474	2.542	0.011
L1.Benzene ($\mu\text{g}/\text{m}^3$)	-0.285070	2.514603	-0.113	0.910

Results for equation NH3 ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	0.247951	0.071158	3.485	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.000048	0.000112	-0.430	0.667
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.001140	0.000295	-3.870	0.000

L1.NO ($\mu\text{g}/\text{m}^3$)	0.002610	0.001652	1.580	0.114
L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.001743	0.001236	1.410	0.158
L1.NOX (ppb)	0.000152	0.001936	0.078	0.938
L1.CO (mg/m^3)	0.036340	0.022227	1.635	0.102
L1.SO2 ($\mu\text{g}/\text{m}^3$)	-0.000768	0.000302	-2.541	0.011
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.973141	0.002408	404.071	0.000
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.000609	0.000584	-1.044	0.297
L1.Benzene ($\mu\text{g}/\text{m}^3$)	1.113182	0.173196	6.427	0.000

Results for equation Ozone ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	-3.854470	0.369634	-10.428	0.000
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.000422	0.000584	-0.722	0.470
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.017308	0.001531	11.308	0.000
L1.NO ($\mu\text{g}/\text{m}^3$)	-0.007417	0.008581	-0.864	0.387
L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.027430	0.006419	4.274	0.000
L1.NOX (ppb)	0.001623	0.010055	0.161	0.872
L1.CO (mg/m^3)	0.093928	0.115460	0.814	0.416
L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.018621	0.001569	11.866	0.000
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.027364	0.012510	2.187	0.029
L1.Ozone ($\mu\text{g}/\text{m}^3$)	1.002230	0.003032	330.584	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	-0.483965	0.899674	-0.538	0.591

Results for equation Benzene ($\mu\text{g}/\text{m}^3$)

	coefficient	std. error	t-stat	prob
const	-0.000378	0.001011	-0.374	0.708
L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.000004	0.000002	-2.205	0.027
L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.000003	0.000004	0.637	0.524
L1.NO ($\mu\text{g}/\text{m}^3$)	-0.000130	0.000023	-5.523	0.000
L1.NO2 ($\mu\text{g}/\text{m}^3$)	-0.000064	0.000018	-3.635	0.000
L1.NOX (ppb)	0.000129	0.000027	4.689	0.000
L1.CO (mg/m^3)	0.000762	0.000316	2.413	0.016
L1.SO2 ($\mu\text{g}/\text{m}^3$)	-0.000005	0.000004	-1.266	0.206
L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.000190	0.000034	5.553	0.000
L1.Ozone ($\mu\text{g}/\text{m}^3$)	-0.000039	0.000008	-4.722	0.000
L1.Benzene ($\mu\text{g}/\text{m}^3$)	0.975188	0.002460	396.464	0.000

Correlation matrix of residuals

PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)	CO (mg/m^3)	SO2 ($\mu\text{g}/\text{m}^3$)	NH3 ($\mu\text{g}/\text{m}^3$)	Ozone
-----------------------------------	------------------------------------	---------------------------------	----------------------------------	-----------	-------------------------------	----------------------------------	----------------------------------	-------

In [321]: # Correlation matrix of residuals

	PM10 ($\mu\text{g}/\text{m}^3$)	PM2.5 ($\mu\text{g}/\text{m}^3$)	NO ($\mu\text{g}/\text{m}^3$)	NO2 ($\mu\text{g}/\text{m}^3$)	NOX (ppb)
# PM10 ($\mu\text{g}/\text{m}^3$)	1.000000	-0.006425	0.034820	0.000808	0.039149
# PM2.5 ($\mu\text{g}/\text{m}^3$)	-0.006425	1.000000	0.088805	0.100111	0.130629
# NO ($\mu\text{g}/\text{m}^3$)	0.034820	0.088805	1.000000	0.113358	0.852916
# NO2 ($\mu\text{g}/\text{m}^3$)	0.000808	0.100111	0.113358	1.000000	0.574285
# NOX (ppb)	0.039149	0.130629	0.852916	0.574285	1.000000
# CO (mg/m^3)	-0.008890	-0.070251	0.038088	-0.008029	0.029488
# SO2 ($\mu\text{g}/\text{m}^3$)	-0.005590	0.003422	0.194194	0.145064	0.229492
# NH3 ($\mu\text{g}/\text{m}^3$)	-0.001271	0.010250	-0.009239	-0.070100	-0.043572
# Ozone ($\mu\text{g}/\text{m}^3$)	-0.017473	-0.010157	-0.131053	-0.147780	-0.189803
# Benzene ($\mu\text{g}/\text{m}^3$)	0.026593	0.079896	0.019926	0.026682	0.042896

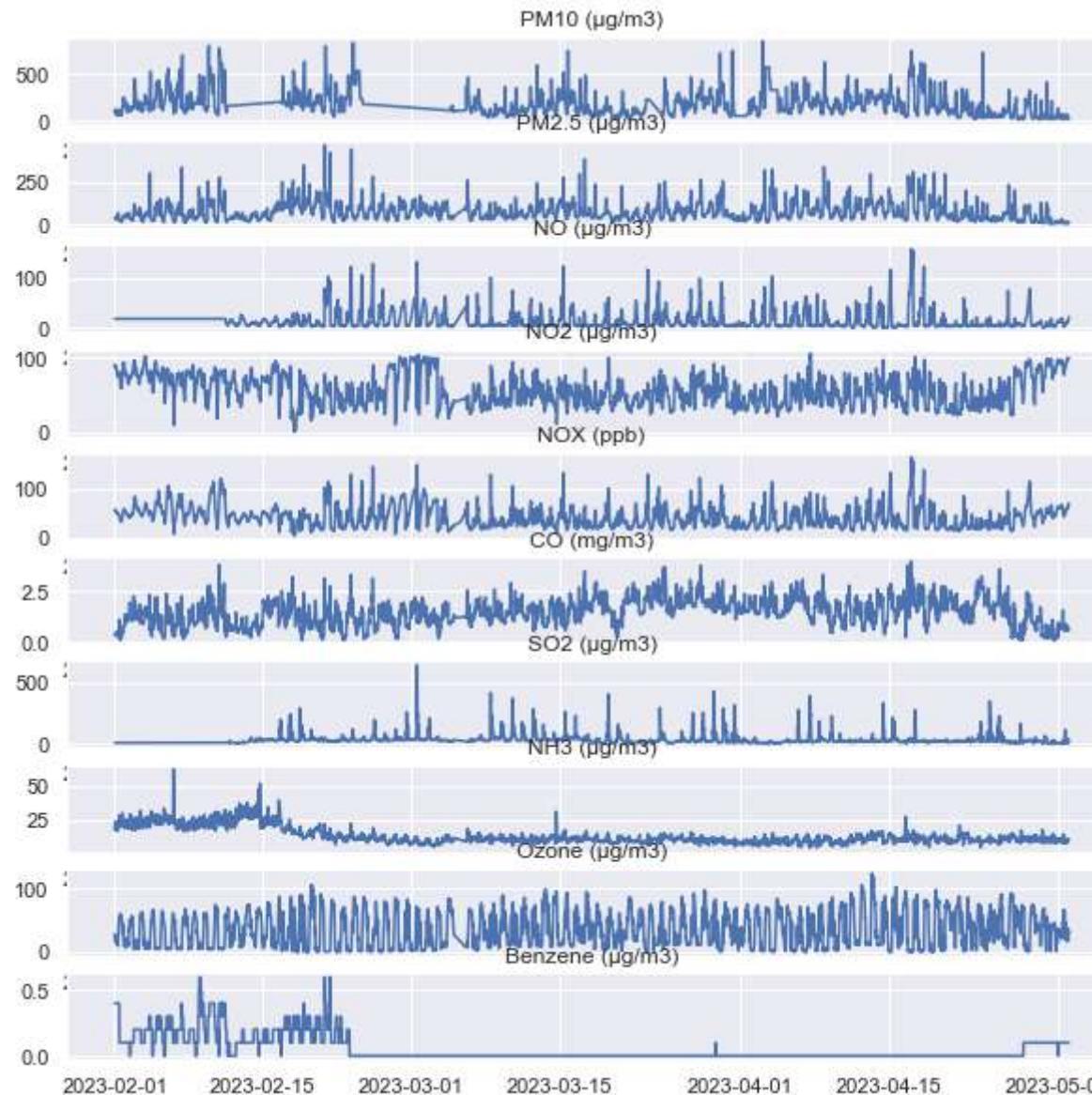
	CO (mg/m^3)	SO2 ($\mu\text{g}/\text{m}^3$)	NH3 ($\mu\text{g}/\text{m}^3$)	Ozone ($\mu\text{g}/\text{m}^3$)	Benzene ($\mu\text{g}/\text{m}^3$)
#	-0.008890	-0.005590	-0.001271	-0.017473	0.026593
#	-0.070251	0.003422	0.010250	-0.010157	0.079896
#	0.038088	0.194194	-0.009239	-0.131053	0.019926
#	-0.008029	0.145064	-0.070100	-0.147780	0.026682
#	0.029488	0.229492	-0.043572	-0.189803	0.042896
#	1.000000	-0.002795	0.029230	-0.193322	-0.037005
#	-0.002795	1.000000	-0.033758	-0.501710	-0.003558
#	0.029230	-0.033758	1.000000	0.026097	0.029184
#	-0.193322	-0.501710	0.026097	1.000000	-0.006481
#	-0.037005	-0.003558	0.029184	-0.006481	1.000000

```
In [322]: # Plot the VAR model diagnostics
```

```
var_result.plot()
```

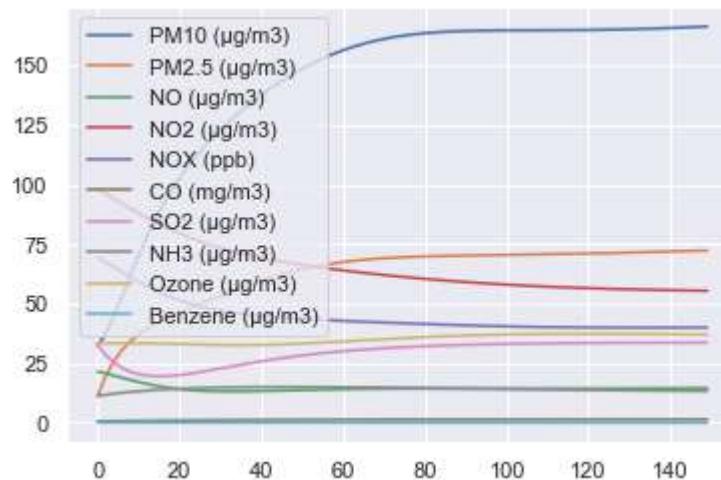
```
# Show the plot
```

```
plt.show()
```



```
In [323]: forecasted_data = var_result.forecast(var_result.y, steps=150)
```

```
In [324]: plt.plot(forecasted_data[:, 0], label=df.columns[2])
plt.plot(forecasted_data[:, 1], label=df.columns[3])
plt.plot(forecasted_data[:, 2], label=df.columns[4])
plt.plot(forecasted_data[:, 3], label=df.columns[5])
plt.plot(forecasted_data[:, 4], label=df.columns[6])
plt.plot(forecasted_data[:, 5], label=df.columns[7])
plt.plot(forecasted_data[:, 6], label=df.columns[8])
plt.plot(forecasted_data[:, 7], label=df.columns[9])
plt.plot(forecasted_data[:, 8], label=df.columns[10])
plt.plot(forecasted_data[:, 9], label=df.columns[11])
plt.legend()
plt.show()
```



```
In [325]: # LLR_test(mod_1, mod_2, DF=1):
```

```
In [326]: def ARIMA_model(df,col,steps,order):
```

```
    # Define the ARIMA model with the chosen values of p, d, and q
    model = sm.tsa.ARIMA(df[col], order=order)

    # Fit the model to the data
    model_fit = model.fit()

    # Print the model summary
    print(model_fit.summary())

    # Fit the ARIMA model
    model = ARIMA(df[col], order=(1,0,1))
    model_fit = model.fit()

    # Evaluate the model
    residuals = pd.DataFrame(model_fit.resid)

    # Plotting diagnostics
    model_fit.plot_diagnostics(figsize=(12, 8))
    plt.show()
```

```
In [327]: def ARIMA_forecast(df,col,steps,order):
```

```
    # Define the ARIMA model with the chosen values of p, d, and q
    model = sm.tsa.ARIMA(df[col], order=order)

    # Fit the model to the data
    model_fit = model.fit()

    forecasted_data = model_fit.forecast(steps=steps)
    return forecasted_data[0].tolist()
```

```
In [328]: ARIMA_model(df_new,df_new.columns[0],5,(2,0,1))
```

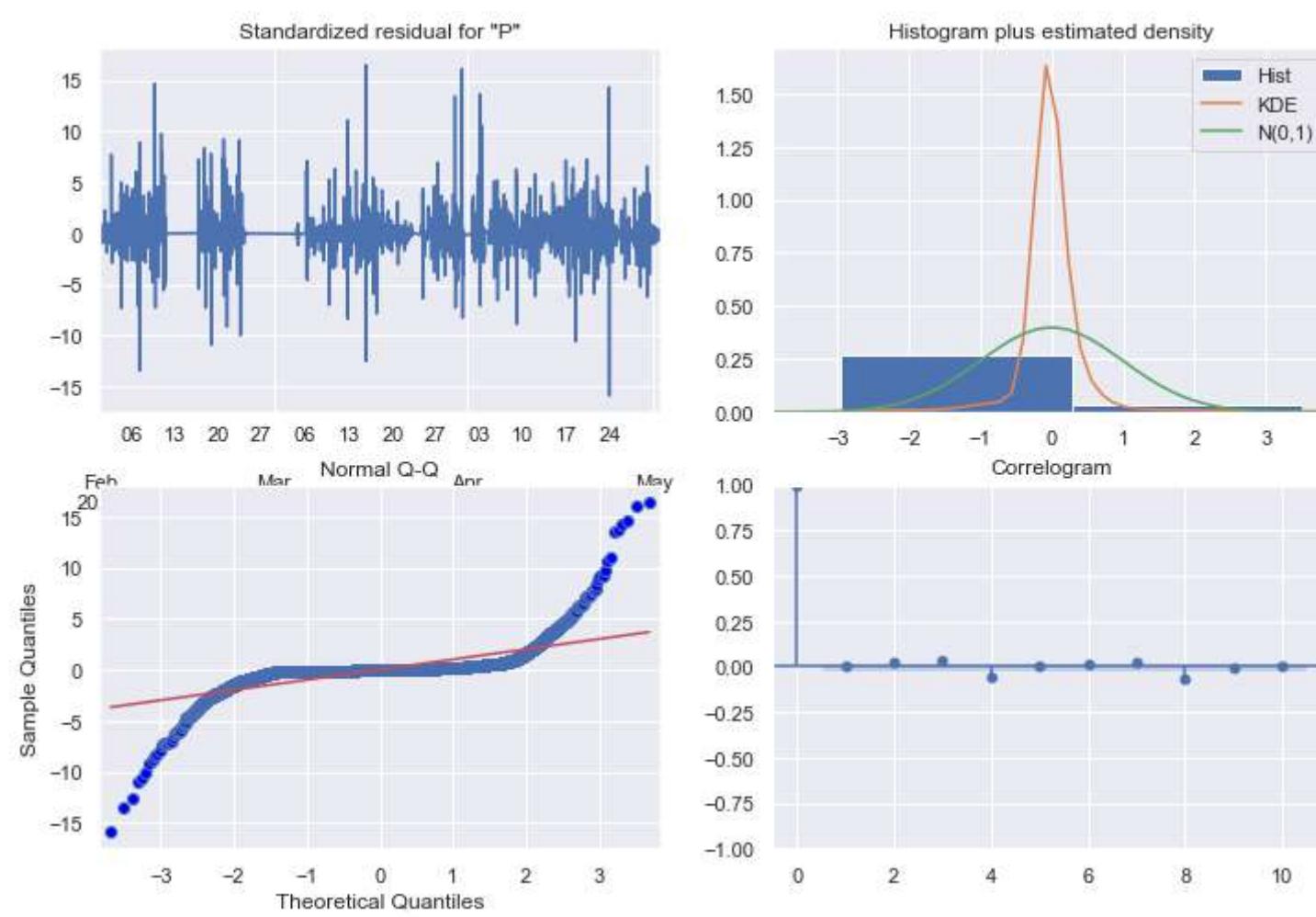
ARMA Model Results

```
=====
Dep. Variable: PM10 ( $\mu\text{g}/\text{m}^3$ ) No. Observations: 8640
Model: ARMA(2, 1) Log Likelihood: -44172.539
Method: css-mle S.D. of innovations: 40.185
Date: Wed, 28 Jun 2023 AIC: 88355.078
Time: 04:16:25 BIC: 88390.399
Sample: 02-01-2023 HQIC: 88367.121
- 05-01-2023
=====
```

	coef	std err	z	P> z	[0.025	0.975]
const	177.0745	7.442	23.795	0.000	162.489	191.660
ar.L1.PM10 ($\mu\text{g}/\text{m}^3$)	1.4684	0.119	12.294	0.000	1.234	1.702
ar.L2.PM10 ($\mu\text{g}/\text{m}^3$)	-0.4984	0.113	-4.420	0.000	-0.719	-0.277
ma.L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.4822	0.122	-3.937	0.000	-0.722	-0.242

Roots

```
=====
Real Imaginary Modulus Frequency
-----
AR.1 1.0687 +0.0000j 1.0687 0.0000
AR.2 1.8775 +0.0000j 1.8775 0.0000
MA.1 2.0737 +0.0000j 2.0737 0.0000
-----
```



```
In [329]: ARIMA_model(df_new,df_new.columns[0],5,(3,0,1))
```

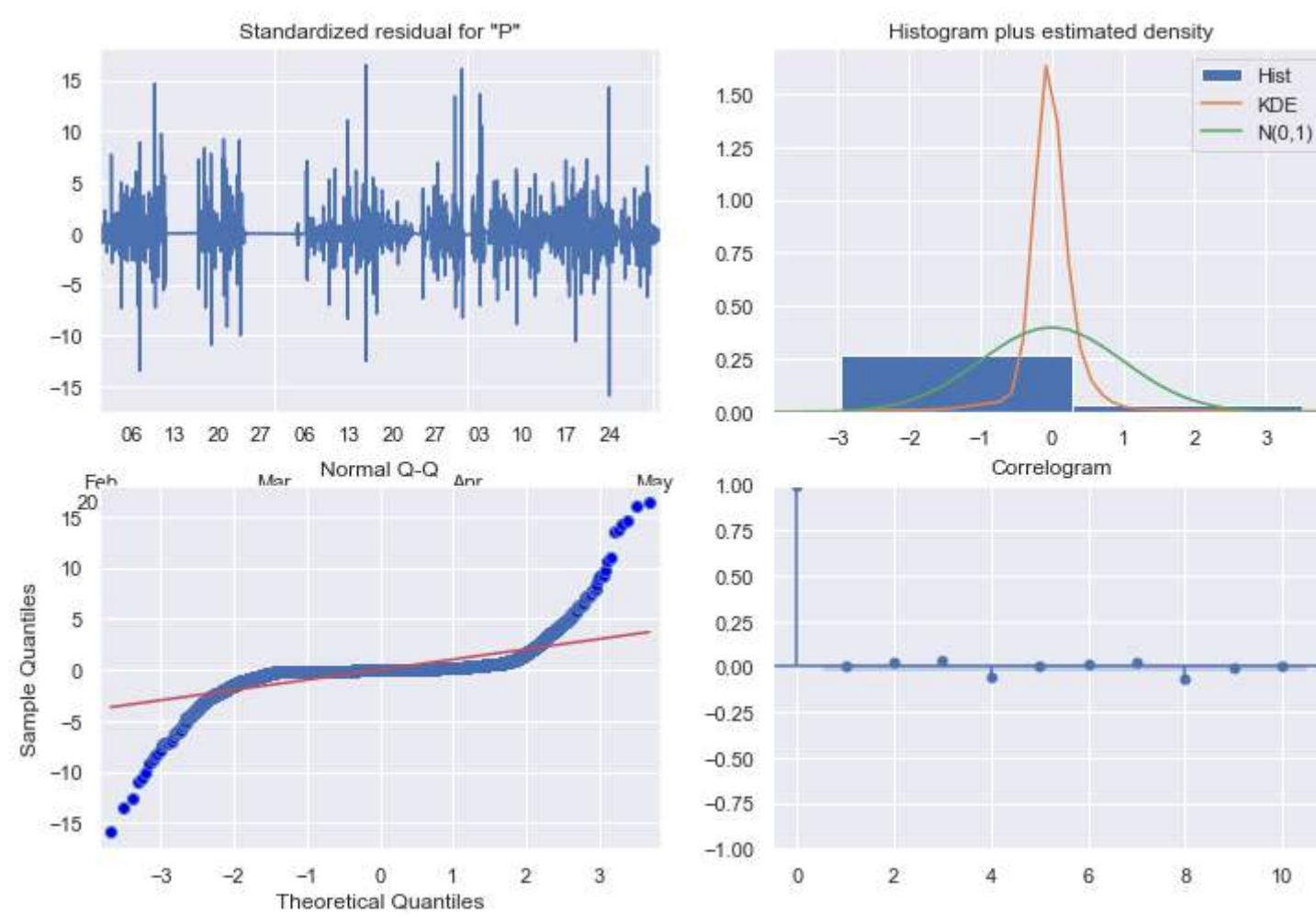
ARMA Model Results

```
=====
Dep. Variable: PM10 ( $\mu\text{g}/\text{m}^3$ ) No. Observations: 8640
Model: ARMA(3, 1) Log Likelihood: -44171.293
Method: css-mle S.D. of innovations: 40.179
Date: Wed, 28 Jun 2023 AIC: 88354.586
Time: 04:16:30 BIC: 88396.971
Sample: 02-01-2023 HQIC: 88369.038
- 05-01-2023
=====
```

	coef	std err	z	P> z	[0.025	0.975]
const	177.0697	7.437	23.811	0.000	162.494	191.645
ar.L1.PM10 ($\mu\text{g}/\text{m}^3$)	1.2334	0.169	7.278	0.000	0.901	1.565
ar.L2.PM10 ($\mu\text{g}/\text{m}^3$)	-0.2523	0.169	-1.497	0.134	-0.583	0.078
ar.L3.PM10 ($\mu\text{g}/\text{m}^3$)	-0.0245	0.013	-1.824	0.068	-0.051	0.002
ma.L1.PM10 ($\mu\text{g}/\text{m}^3$)	-0.2522	0.169	-1.490	0.136	-0.584	0.080

Roots

```
=====
Real Imaginary Modulus Frequency
-----
AR.1 1.0686 +0.0000j 1.0686 0.0000
AR.2 2.7140 +0.0000j 2.7140 0.0000
AR.3 -14.0943 +0.0000j 14.0943 0.5000
MA.1 3.9648 +0.0000j 3.9648 0.0000
-----
```



```
In [330]: ARIMA_model(df_new,df_new.columns[1],5,(1,0,1))
```

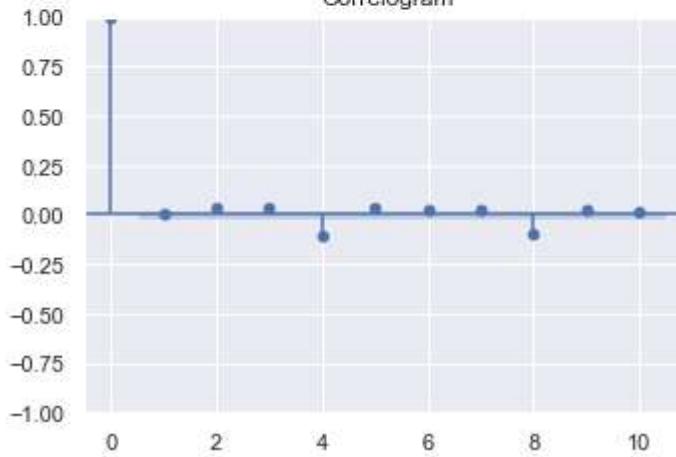
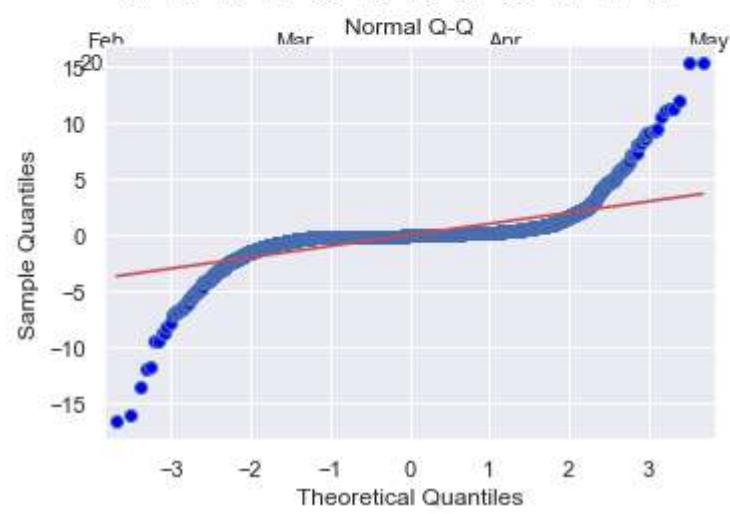
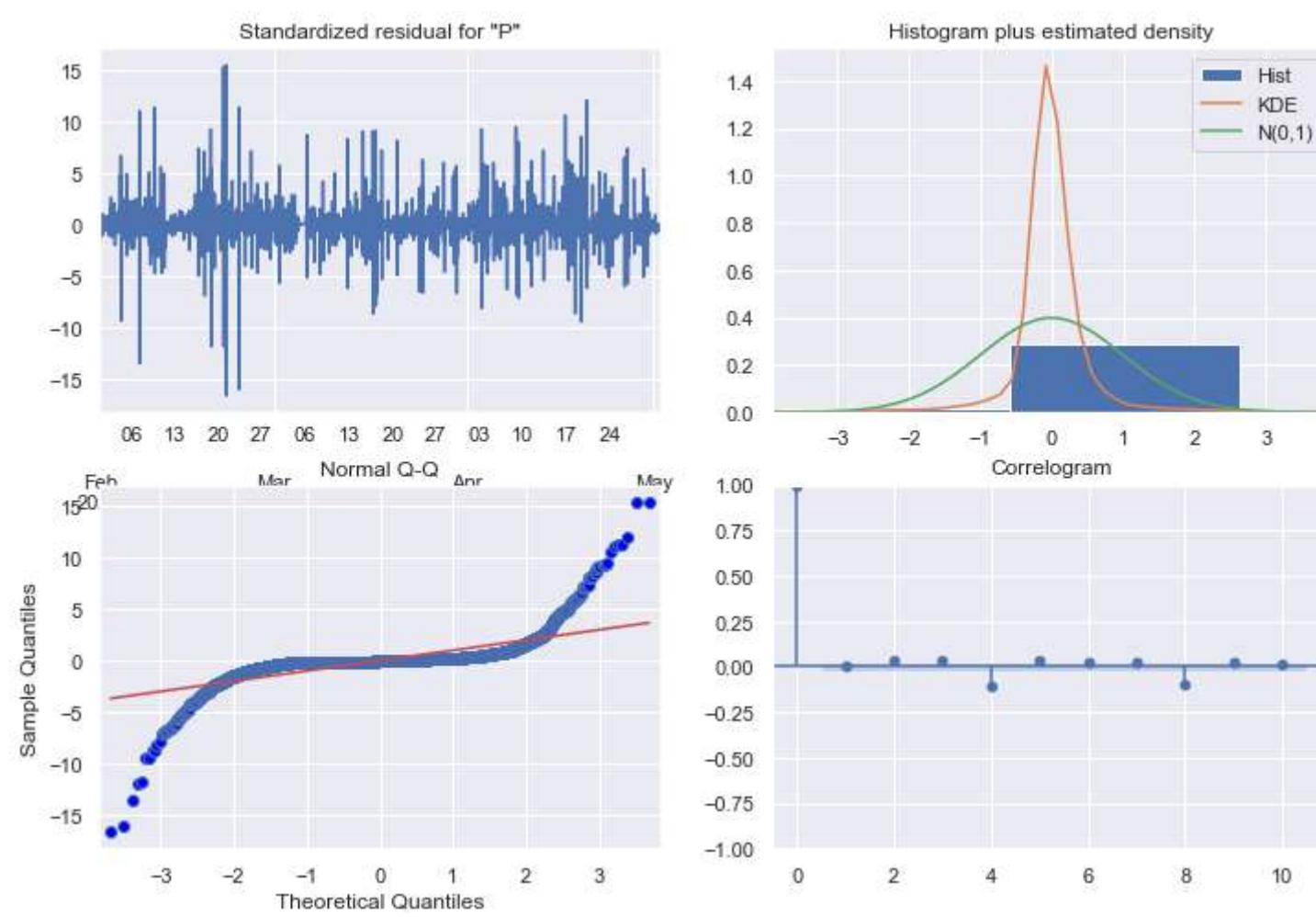
ARMA Model Results

Dep. Variable: PM2.5 ($\mu\text{g}/\text{m}^3$) No. Observations: 8640
Model: ARMA(1, 1) Log Likelihood -38129.560
Method: css-mle S.D. of innovations 19.967
Date: Wed, 28 Jun 2023 AIC 76267.119
Time: 04:16:34 BIC 76295.376
Sample: 02-01-2023 HQIC 76276.754
- 05-01-2023

	coef	std err	z	P> z	[0.025	0.975]
const	75.3956	3.022	24.947	0.000	69.472	81.319
ar.L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.9265	0.004	214.177	0.000	0.918	0.935
ma.L1.PM2.5 ($\mu\text{g}/\text{m}^3$)	0.0360	0.011	3.228	0.001	0.014	0.058

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0794	+0.0000j	1.0794	0.0000
MA.1	-27.7829	+0.0000j	27.7829	0.5000



```
In [331]: ARIMA_model(df_new,df_new.columns[2],5,(1,0,1))
```

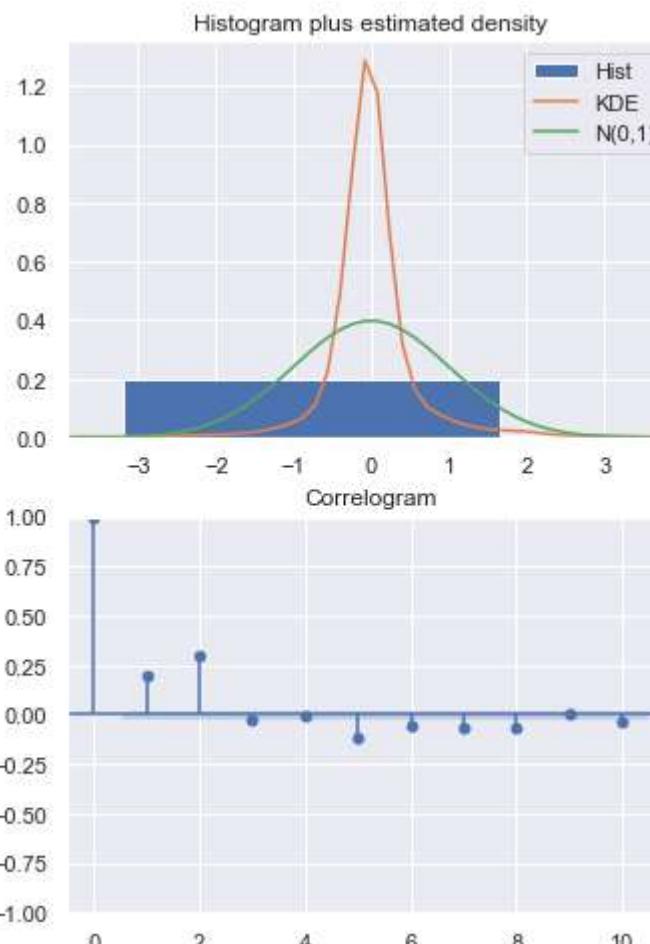
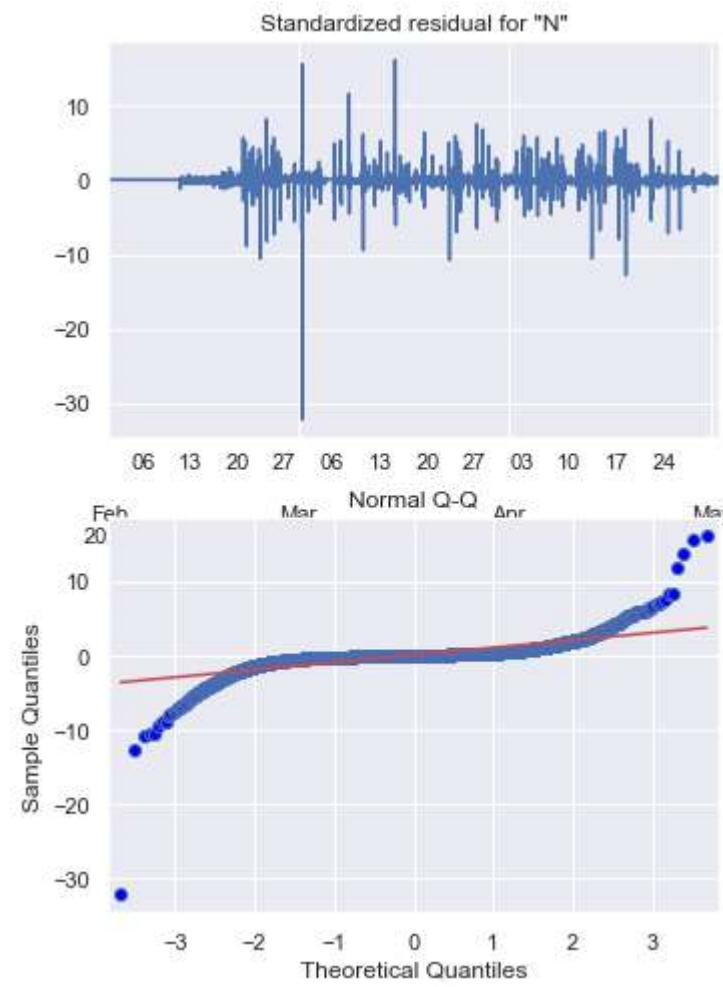
ARMA Model Results

```
=====
Dep. Variable: NO ( $\mu\text{g}/\text{m}^3$ ) No. Observations: 8640
Model: ARMA(1, 1) Log Likelihood: -21672.281
Method: css-mle S.D. of innovations: 2.972
Date: Wed, 28 Jun 2023 AIC: 43352.561
Time: 04:16:38 BIC: 43380.818
Sample: 02-01-2023 HQIC: 43362.196
- 05-01-2023
=====
```

	coef	std err	z	P> z	[0.025	0.975]
const	14.9741	1.444	10.372	0.000	12.145	17.804
ar.L1.NO ($\mu\text{g}/\text{m}^3$)	0.9649	0.003	340.672	0.000	0.959	0.970
ma.L1.NO ($\mu\text{g}/\text{m}^3$)	0.5883	0.007	82.654	0.000	0.574	0.602

Roots

```
=====
Real Imaginary Modulus Frequency
-----
AR.1 1.0363 +0.0000j 1.0363 0.0000
MA.1 -1.6998 +0.0000j 1.6998 0.5000
-----
```



```
In [332]: ARIMA_model(df_new,df_new.columns[3],5,(1,0,1))
```

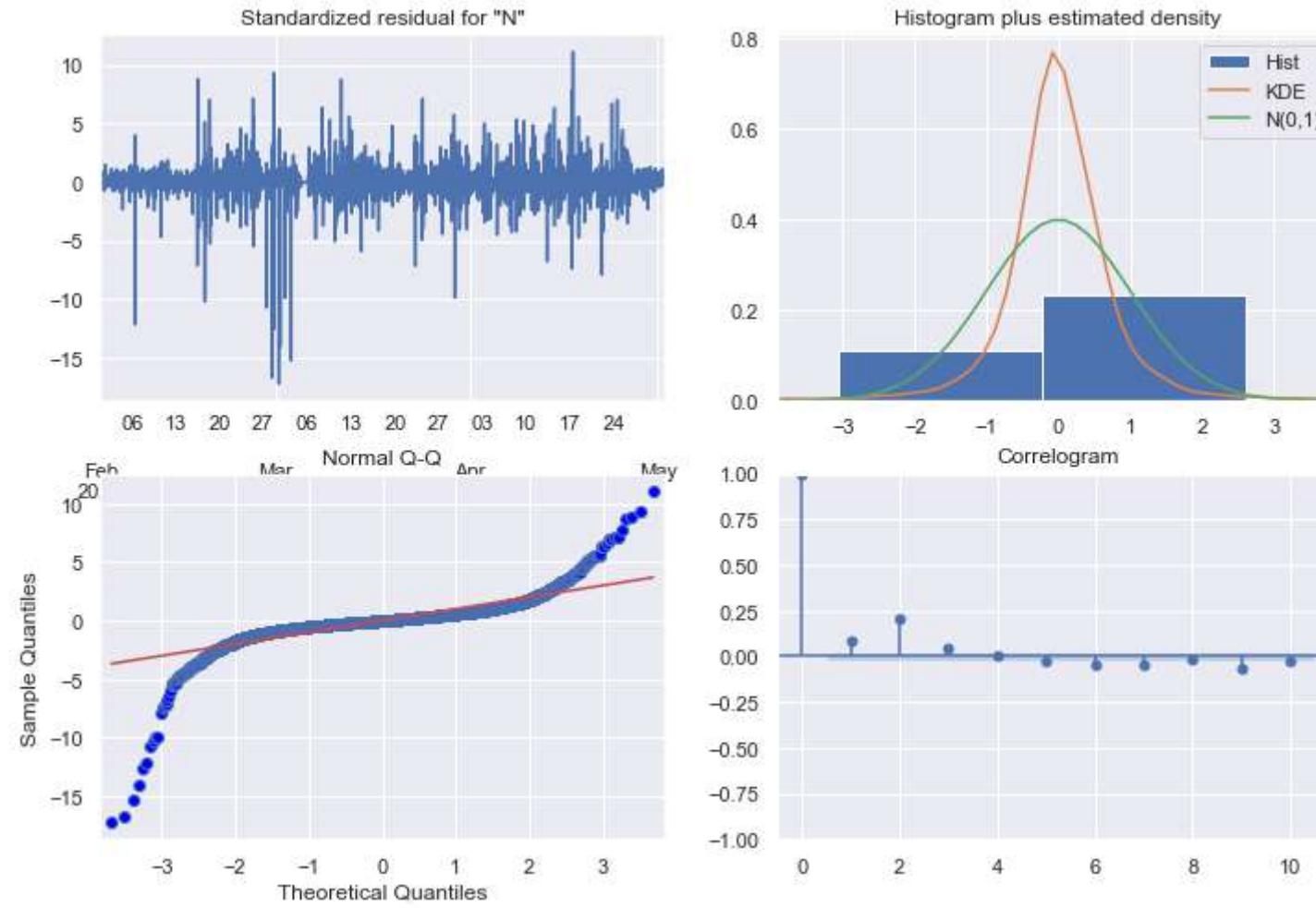
ARMA Model Results

Dep. Variable: NO2 ($\mu\text{g}/\text{m}^3$) No. Observations: 8640
Model: ARMA(1, 1) Log Likelihood -22007.653
Method: css-mle S.D. of innovations 3.090
Date: Wed, 28 Jun 2023 AIC 44023.307
Time: 04:16:43 BIC 44051.563
Sample: 02-01-2023 HQIC 44032.941
- 05-01-2023

	coef	std err	z	P> z	[0.025	0.975]
const	55.7962	1.979	28.191	0.000	51.917	59.675
ar.L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.9758	0.002	410.218	0.000	0.971	0.980
ma.L1.NO2 ($\mu\text{g}/\text{m}^3$)	0.4458	0.009	51.861	0.000	0.429	0.463

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0248	+0.0000j	1.0248	0.0000
MA.1	-2.2431	+0.0000j	2.2431	0.5000



```
In [333]: ARIMA_model(df_new,df_new.columns[4],5,(1,0,1))
```

ARMA Model Results

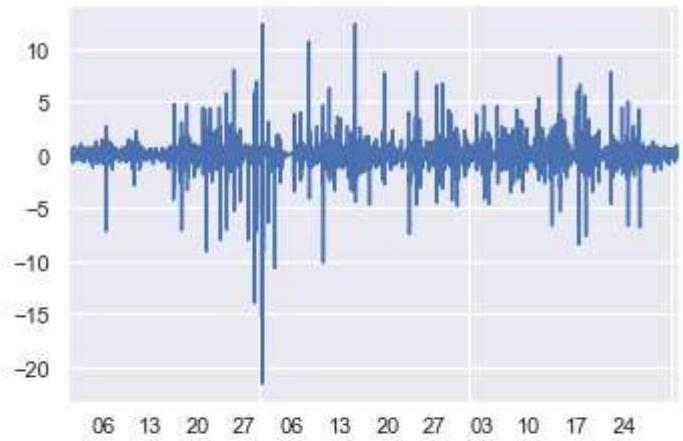
```
=====
Dep. Variable:          NOX (ppb)    No. Observations:             8640
Model:                 ARMA(1, 1)    Log Likelihood:            -21574.264
Method:                css-mle     S.D. of innovations:       2.938
Date:                  Wed, 28 Jun 2023   AIC:                   43156.528
Time:                  04:16:49      BIC:                   43184.785
Sample:                02-01-2023   HQIC:                  43166.163
                           - 05-01-2023
=====
```

	coef	std err	z	P> z	[0.025	0.975]
const	42.4348	2.158	19.664	0.000	38.205	46.664
ar.L1.NOX (ppb)	0.9757	0.002	413.558	0.000	0.971	0.980
ma.L1.NOX (ppb)	0.6653	0.007	98.091	0.000	0.652	0.679

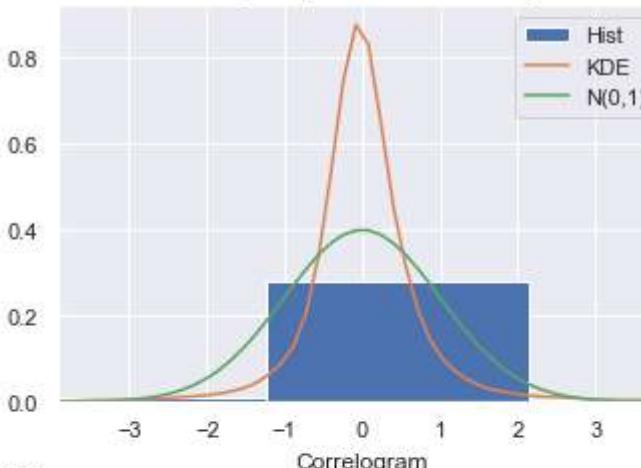
Roots

```
=====
          Real           Imaginary        Modulus        Frequency
-----
AR.1      1.0249         +0.0000j    1.0249         0.0000
MA.1     -1.5030         +0.0000j    1.5030         0.5000
-----
```

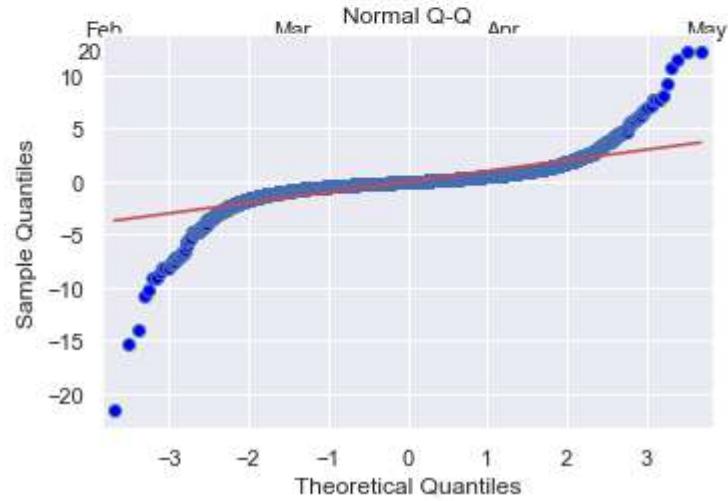
Standardized residual for "N"



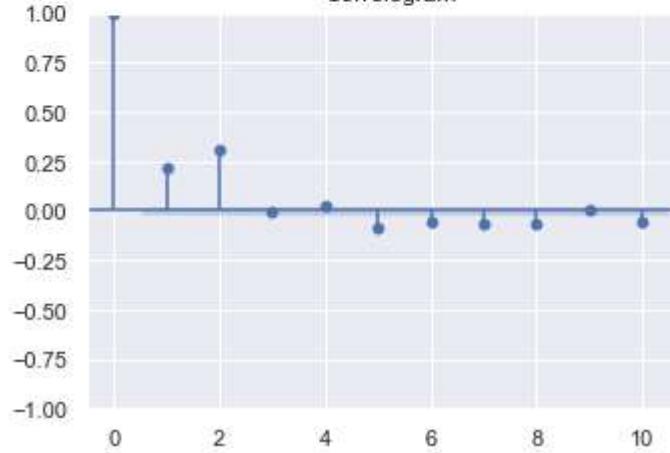
Histogram plus estimated density



Normal Q-Q



Correlogram



In [334]: ARIMA_model(df_new,df_new.columns[5],5,(1,0,1))

ARMA Model Results

=====

Dep. Variable:	CO (mg/m3)	No. Observations:	8640
Model:	ARMA(1, 1)	Log Likelihood:	3214.542
Method:	css-mle	S.D. of innovations:	0.167
Date:	Wed, 28 Jun 2023	AIC:	-6421.083
Time:	04:16:54	BIC:	-6392.826
Sample:	02-01-2023 - 05-01-2023	HQIC:	-6411.448

=====

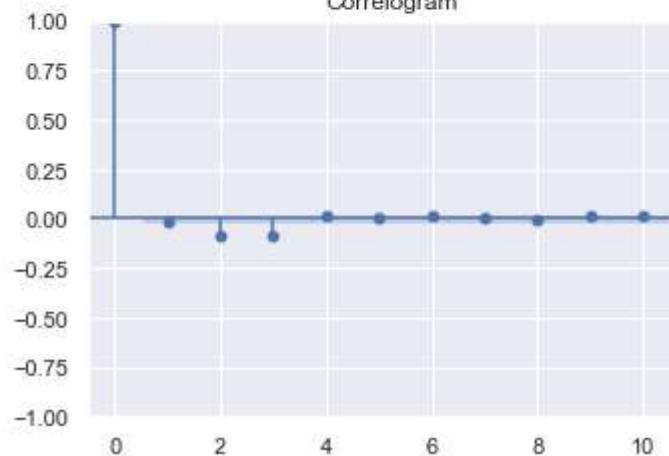
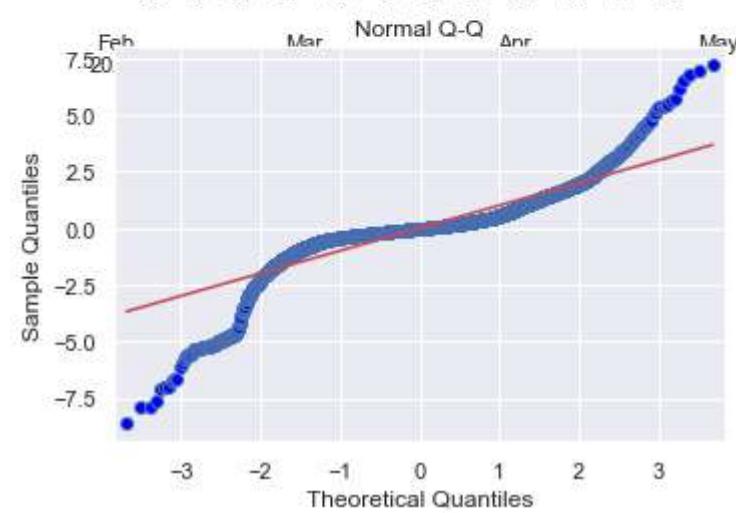
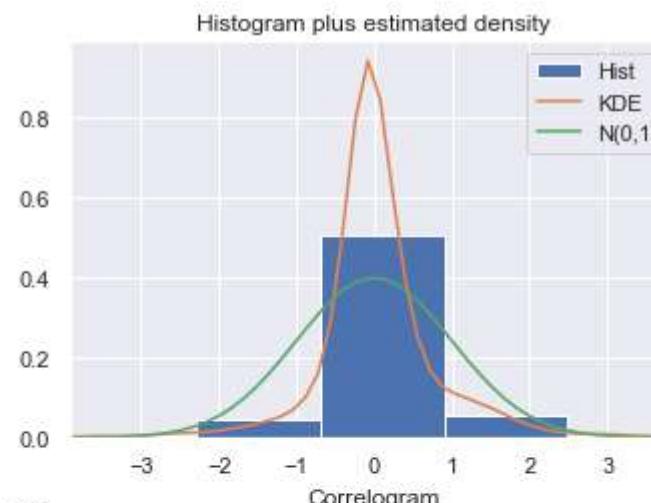
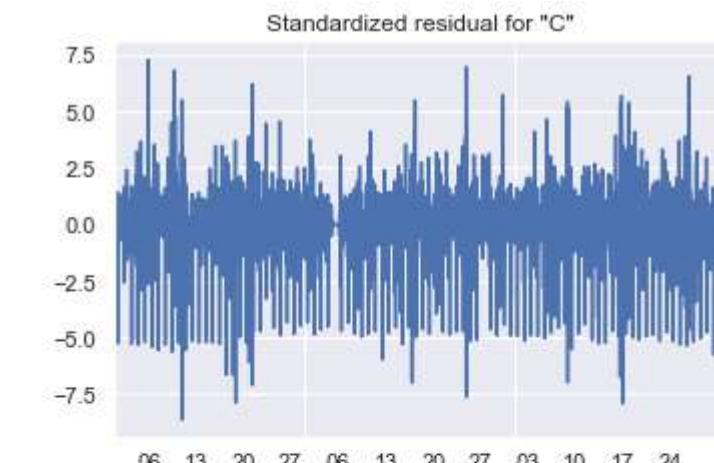
	coef	std err	z	P> z	[0.025	0.975]
const	1.3974	0.044	31.652	0.000	1.311	1.484
ar.L1.CO (mg/m3)	0.9531	0.003	280.950	0.000	0.946	0.960
ma.L1.CO (mg/m3)	0.1573	0.012	13.330	0.000	0.134	0.180

Roots

=====

	Real	Imaginary	Modulus	Frequency
AR.1	1.0492	+0.0000j	1.0492	0.0000
MA.1	-6.3579	+0.0000j	6.3579	0.5000

=====



```
In [335]: ARIMA_model(df_new,df_new.columns[6],5,(1,0,1))
```

ARMA Model Results

=====

Dep. Variable:	SO2 ($\mu\text{g}/\text{m}^3$)	No. Observations:	8640
Model:	ARMA(1, 1)	Log Likelihood:	-35679.058
Method:	css-mle	S.D. of innovations:	15.036
Date:	Wed, 28 Jun 2023	AIC:	71366.117
Time:	04:16:58	BIC:	71394.374
Sample:	02-01-2023 - 05-01-2023	HQIC:	71375.752

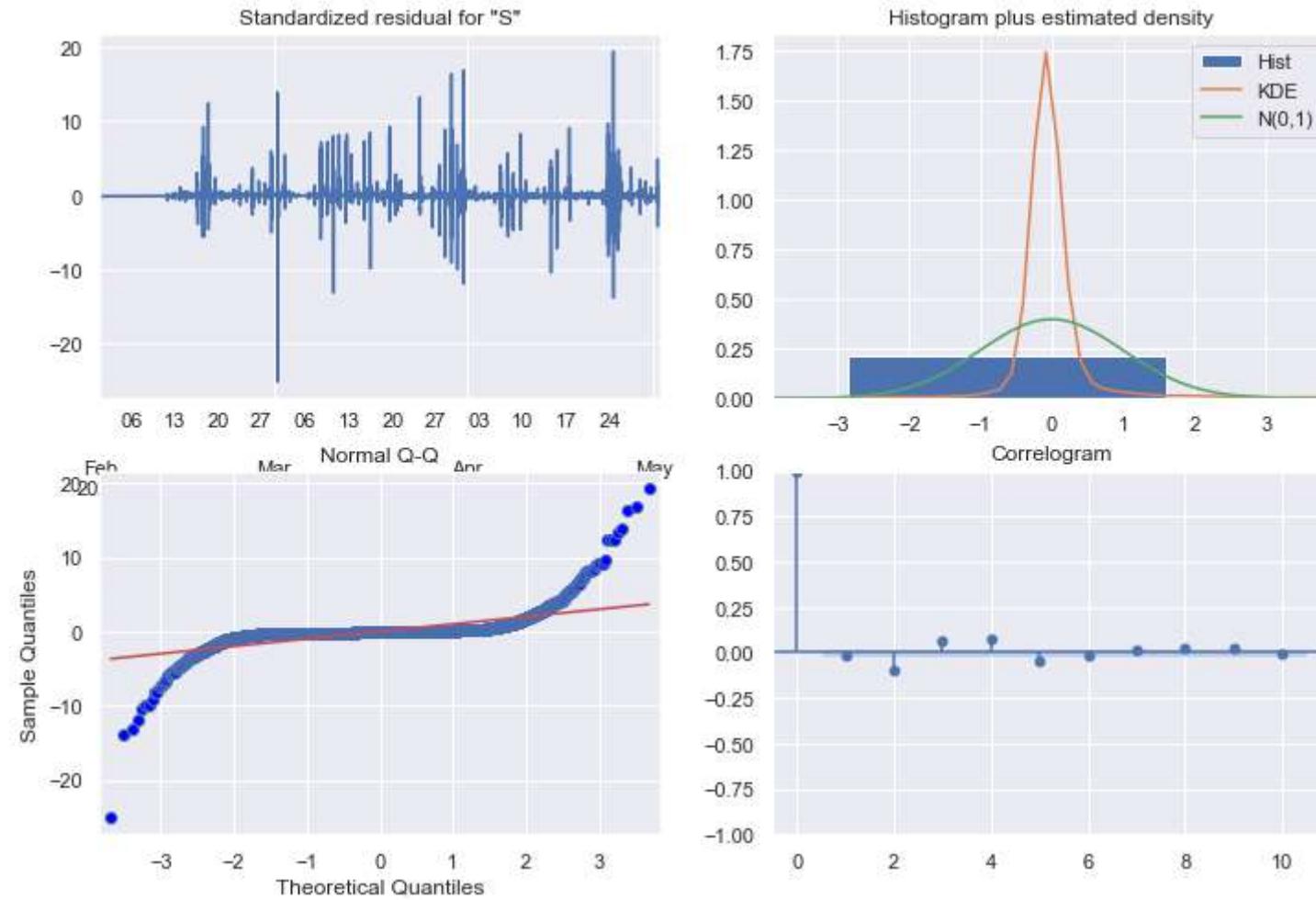
=====

	coef	std err	z	P> z	[0.025	0.975]
const	31.9086	1.805	17.677	0.000	28.371	35.447
ar.L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.8950	0.005	168.735	0.000	0.885	0.905
ma.L1.SO2 ($\mu\text{g}/\text{m}^3$)	0.1728	0.013	13.116	0.000	0.147	0.199

Roots

=====

	Real	Imaginary	Modulus	Frequency
AR.1	1.1173	+0.0000j	1.1173	0.0000
MA.1	-5.7863	+0.0000j	5.7863	0.5000



In [336]: ARIMA_model(df_new,df_new.columns[7],5,(1,0,1))

ARMA Model Results

=====

Dep. Variable:	NH3 ($\mu\text{g}/\text{m}^3$)	No. Observations:	8640
Model:	ARMA(1, 1)	Log Likelihood:	-12630.004
Method:	css-mle	S.D. of innovations:	1.044
Date:	Wed, 28 Jun 2023	AIC:	25268.009
Time:	04:17:03	BIC:	25296.265
Sample:	02-01-2023 - 05-01-2023	HQIC:	25277.643

=====

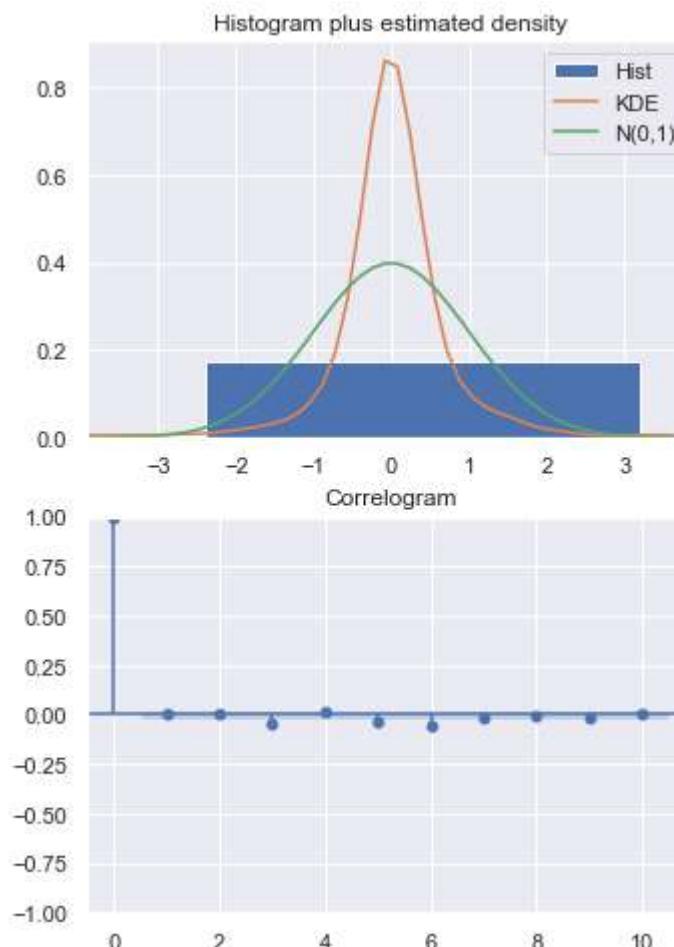
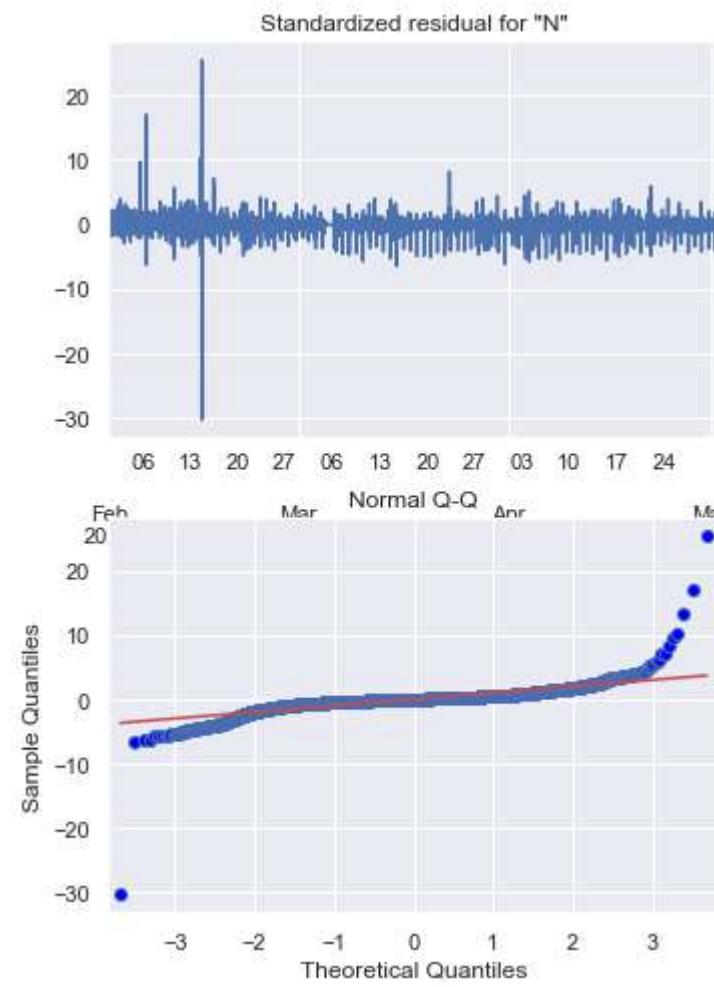
	coef	std err	z	P> z	[0.025	0.975]
const	13.3005	0.704	18.881	0.000	11.920	14.681
ar.L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.9826	0.002	488.126	0.000	0.979	0.987
ma.L1.NH3 ($\mu\text{g}/\text{m}^3$)	0.0960	0.011	8.921	0.000	0.075	0.117

Roots

=====

	Real	Imaginary	Modulus	Frequency
AR.1	1.0177	+0.0000j	1.0177	0.0000
MA.1	-10.4158	+0.0000j	10.4158	0.5000

=====



In [337]: ARIMA_model(df_new,df_new.columns[8],5,(1,0,1))

ARMA Model Results

=====

Dep. Variable:	Ozone ($\mu\text{g}/\text{m}^3$)	No. Observations:	8640
Model:	ARMA(1, 1)	Log Likelihood:	-26602.582
Method:	css-mle	S.D. of innovations:	5.259
Date:	Wed, 28 Jun 2023	AIC:	53213.165
Time:	04:17:07	BIC:	53241.422
Sample:	02-01-2023 - 05-01-2023	HQIC:	53222.800

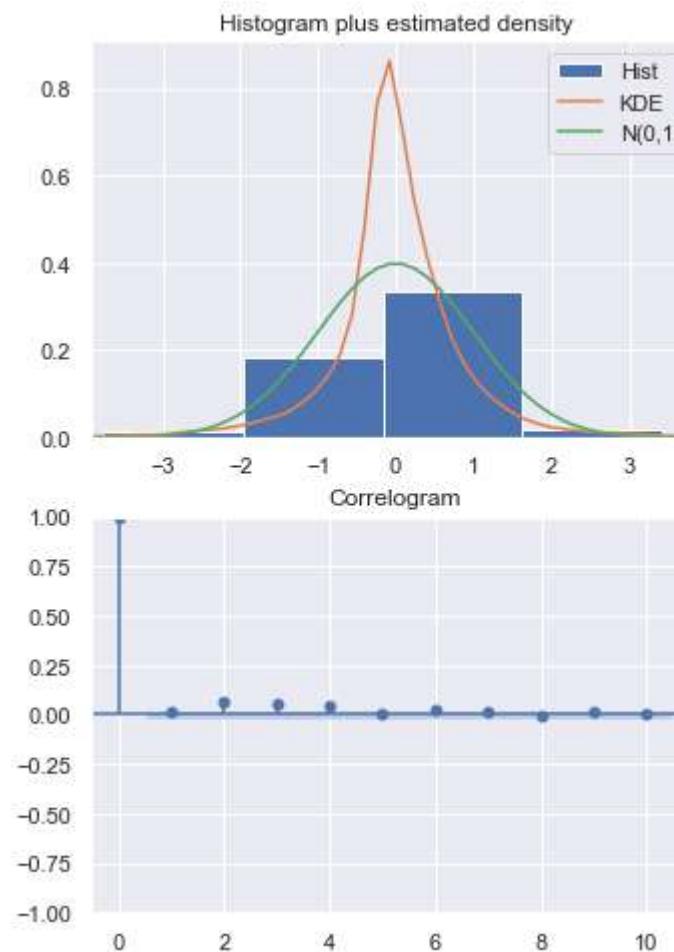
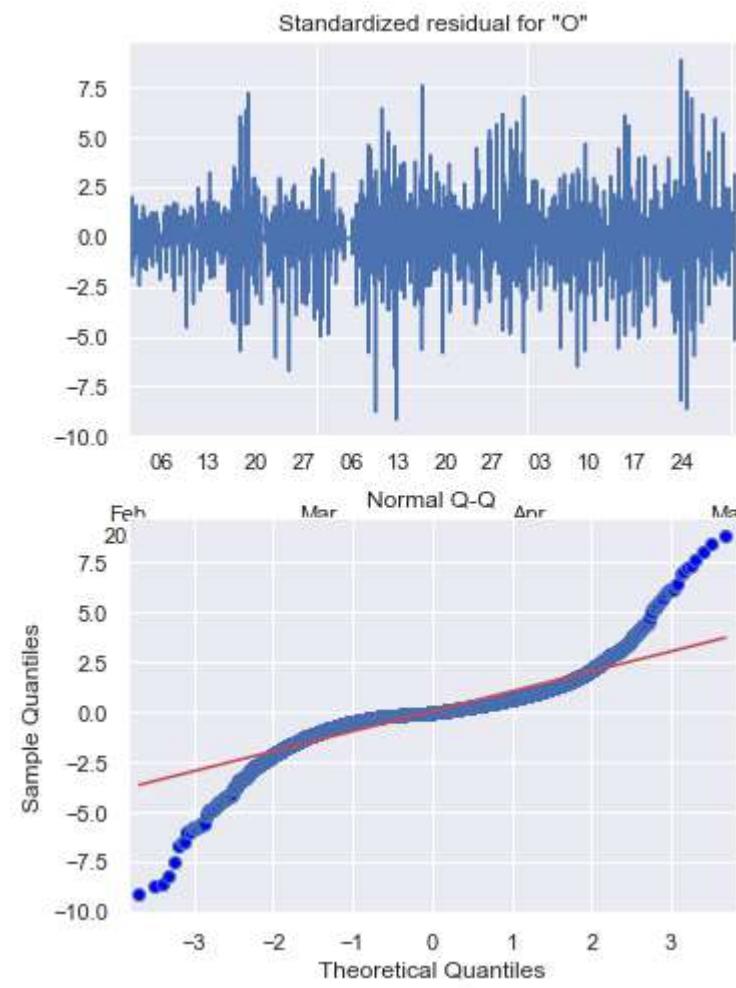
=====

	coef	std err	z	P> z	[0.025	0.975]
const	35.1692	2.198	16.002	0.000	30.862	39.477
ar.L1.Ozone ($\mu\text{g}/\text{m}^3$)	0.9659	0.003	342.275	0.000	0.960	0.971
ma.L1.Ozone ($\mu\text{g}/\text{m}^3$)	0.3292	0.010	33.098	0.000	0.310	0.349

=====
Roots
=====

	Real	Imaginary	Modulus	Frequency
AR.1	1.0353	+0.0000j	1.0353	0.0000
MA.1	-3.0377	+0.0000j	3.0377	0.5000

=====



```
In [338]: ARIMA_model(df_new,df_new.columns[9],5,(1,0,1))
```

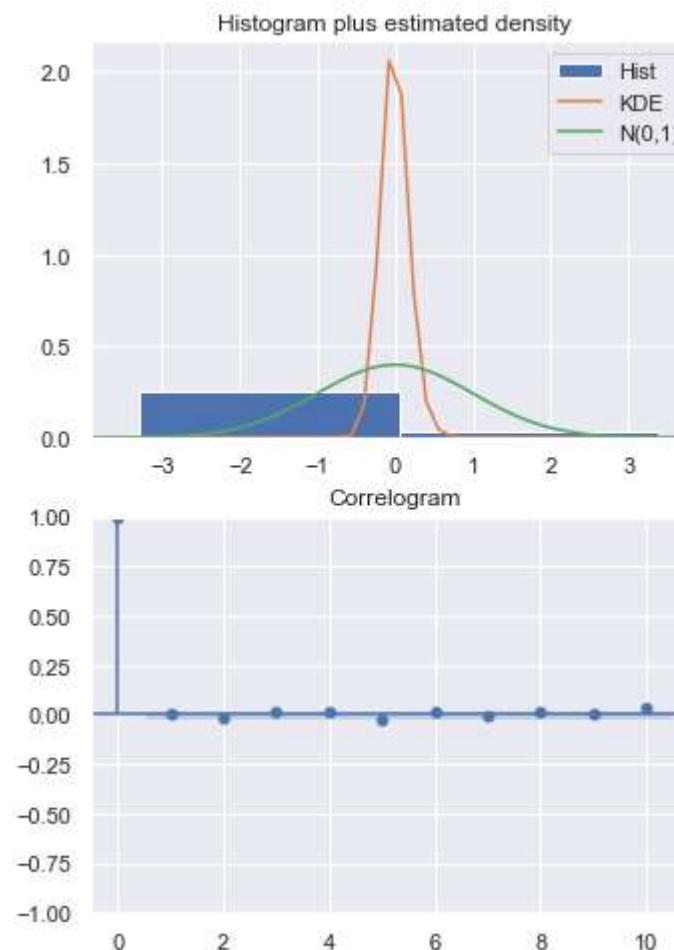
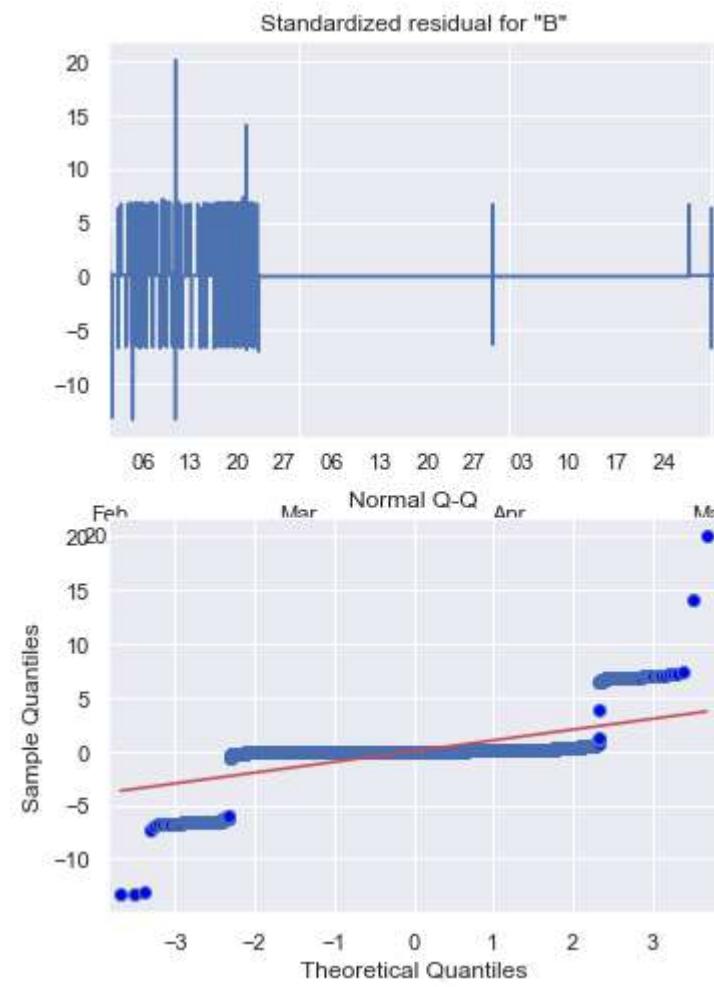
ARMA Model Results

```
=====
Dep. Variable: Benzene ( $\mu\text{g}/\text{m}^3$ ) No. Observations: 8640
Model: ARMA(1, 1) Log Likelihood: 24087.804
Method: css-mle S.D. of innovations: 0.015
Date: Wed, 28 Jun 2023 AIC: -48167.607
Time: 04:17:13 BIC: -48139.351
Sample: 02-01-2023 HQIC: -48157.973
- 05-01-2023
=====
```

	coef	std err	z	P> z	[0.025	0.975]
const	0.0545	0.015	3.750	0.000	0.026	0.083
ar.L1.Benzene ($\mu\text{g}/\text{m}^3$)	0.9896	0.002	616.076	0.000	0.986	0.993
ma.L1.Benzene ($\mu\text{g}/\text{m}^3$)	-0.0481	0.011	-4.363	0.000	-0.070	-0.027

Roots

```
=====
Real Imaginary Modulus Frequency
-----
AR.1 1.0105 +0.0000j 1.0105 0.0000
MA.1 20.7810 +0.0000j 20.7810 0.0000
-----
```



Forecasting values

```
In [339]: new_data = []
step = 100

forecast_1 = ARIMA_forecast(df_new,df_new.columns[0],step,(1,0,1))
new_data.append(forecast_1)

forecast_2 = ARIMA_forecast(df_new,df_new.columns[1],step,(1,0,1))
new_data.append(forecast_2)

forecast_3 = ARIMA_forecast(df_new,df_new.columns[2],step,(1,0,1))
new_data.append(forecast_3)

forecast_4 = ARIMA_forecast(df_new,df_new.columns[3],step,(1,0,1))
new_data.append(forecast_4)

forecast_5 = ARIMA_forecast(df_new,df_new.columns[4],step,(1,0,1))
new_data.append(forecast_5)

forecast_6 = ARIMA_forecast(df_new,df_new.columns[5],step,(1,0,1))
new_data.append(forecast_6)

forecast_7 = ARIMA_forecast(df_new,df_new.columns[6],step,(1,0,1))
new_data.append(forecast_7)

forecast_8 = ARIMA_forecast(df_new,df_new.columns[7],step,(1,0,1))
new_data.append(forecast_8)

forecast_9 = ARIMA_forecast(df_new,df_new.columns[8],step,(1,0,1))
new_data.append(forecast_9)

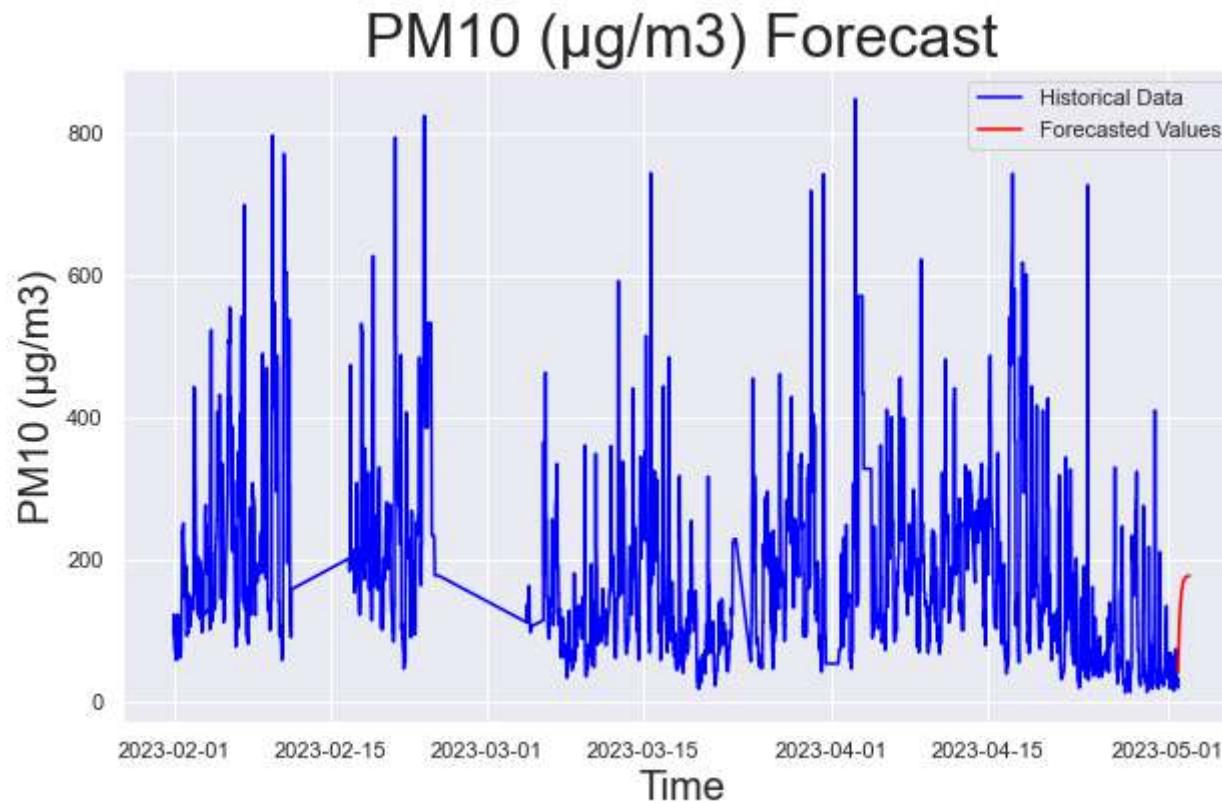
forecast_10 = ARIMA_forecast(df_new,df_new.columns[9],step,(1,0,1))
new_data.append(forecast_10)
```

```
In [340]: new_data_T = [list(row) for row in zip(*new_data)]
```

```
In [341]: df_created = pd.DataFrame(df_new, index=pd.date_range('2023-02-01', periods=8640, freq='15T'))  
  
forecasted_index = pd.date_range('2023-05-02', periods=step, freq='15T')  
# Generating datetime index for the forecasted data  
  
df_forecasted = pd.DataFrame(new_data_T, index=forecasted_index, columns=df_new.columns)  
  
# Concatenating the original DataFrame and the forecasted DataFrame row-wise  
df_created = pd.concat([df_new, df_forecasted], axis=0)
```

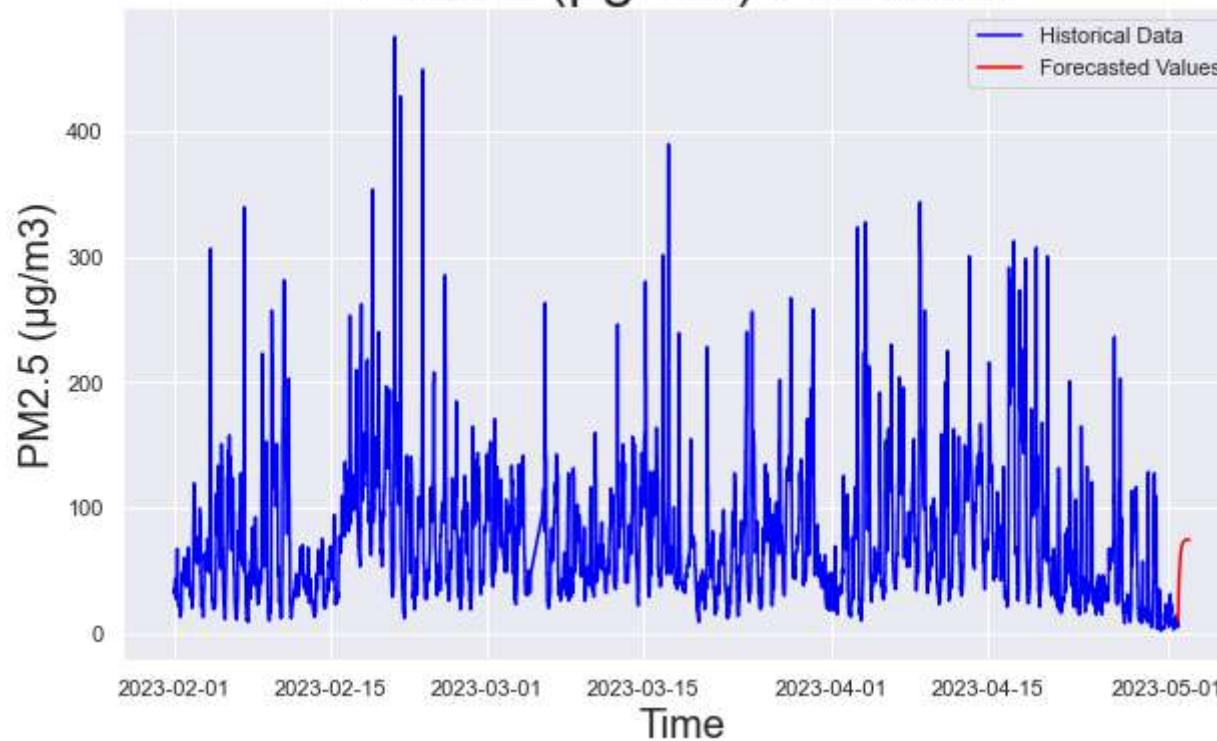
```
In [342]: def forecast(i):  
    # Plotting parameters  
  
    plt.figure(figsize=(10, 6))  
    plt.title(df_created.columns[i]+" Forecast", fontsize = 30)  
    plt.xlabel('Time', fontsize = 20)  
    plt.ylabel(df_created.columns[i], fontsize=20)  
  
    # Plotting historical data in blue  
    plt.plot(df_created.index[:8640], df_created[df_created.columns[i]][:8640], color='blue', label='Historical Data')  
  
    # Plotting forecasted values in red  
    plt.plot(df_created.index[8640:], df_created[df_created.columns[i]][8640:], color='red', label='Forecasted Values')  
  
    plt.legend()  
    plt.show()
```

In [343]: `forecast(0)`

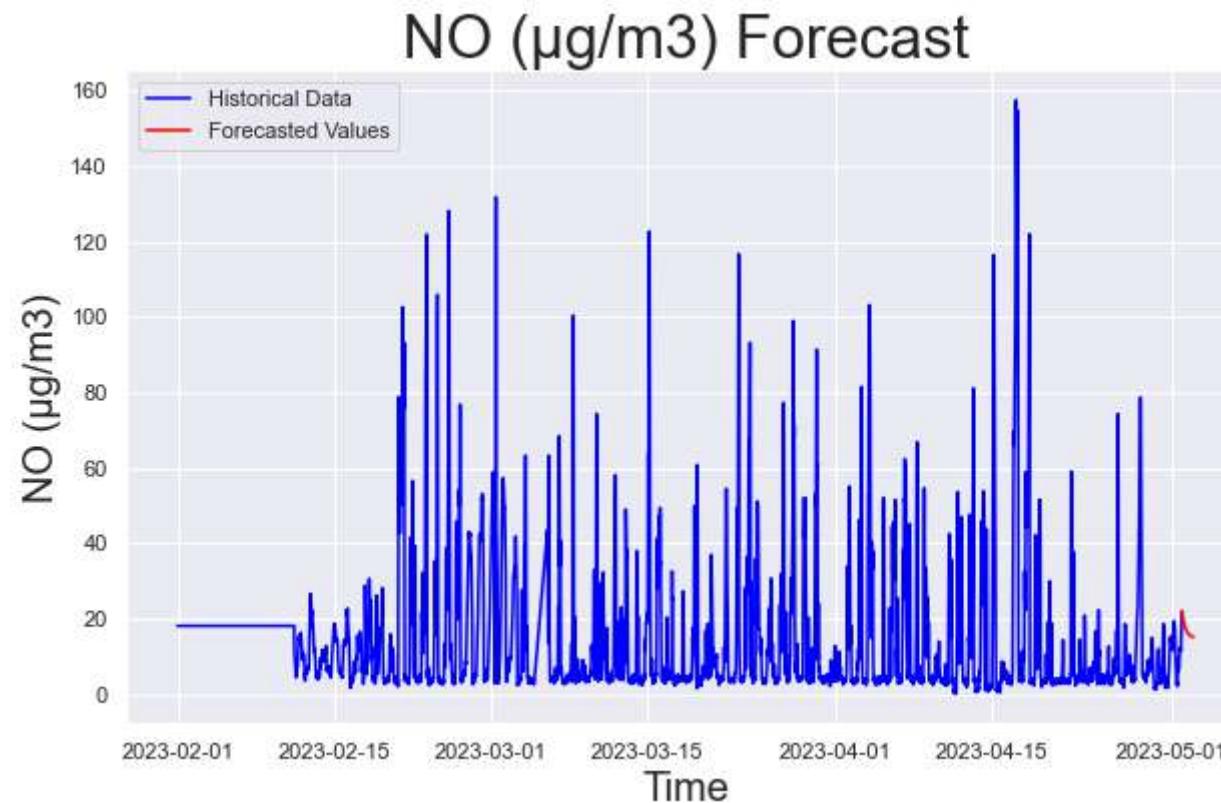


In [344]: `forecast(1)`

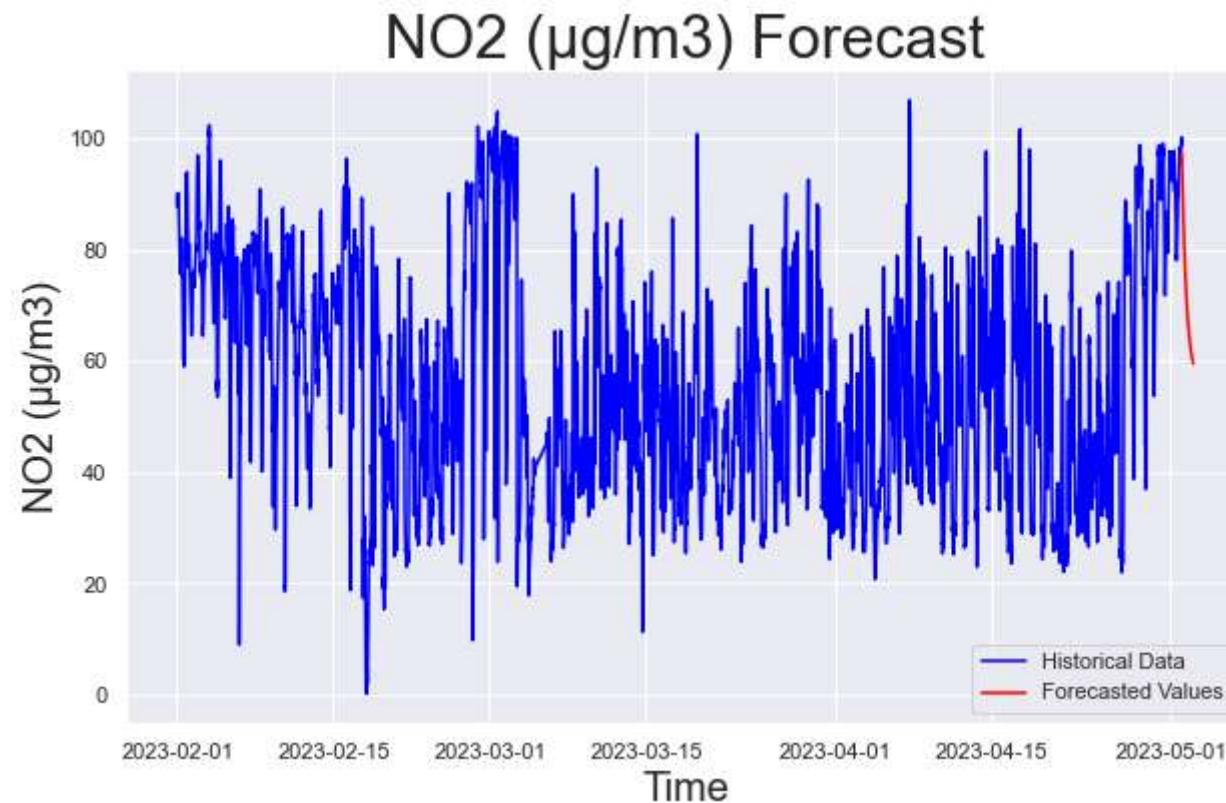
PM2.5 ($\mu\text{g}/\text{m}^3$) Forecast



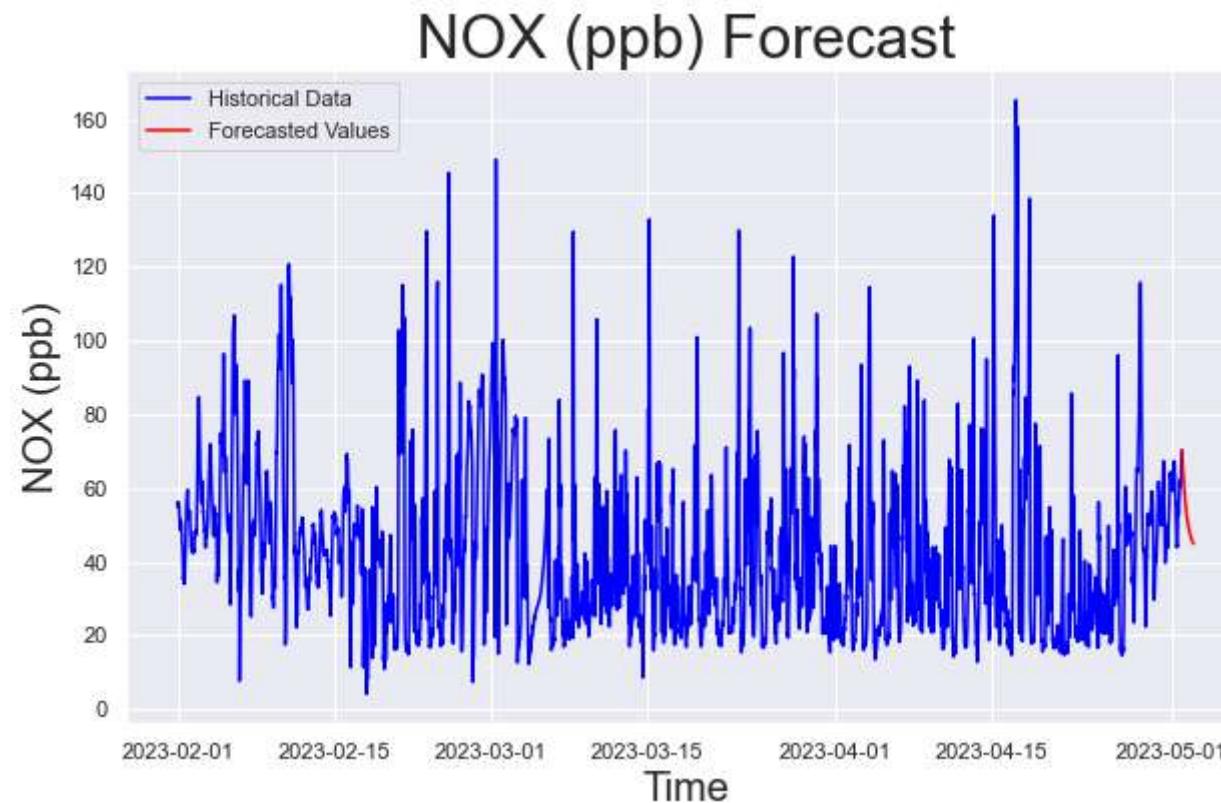
In [345]: `forecast(2)`



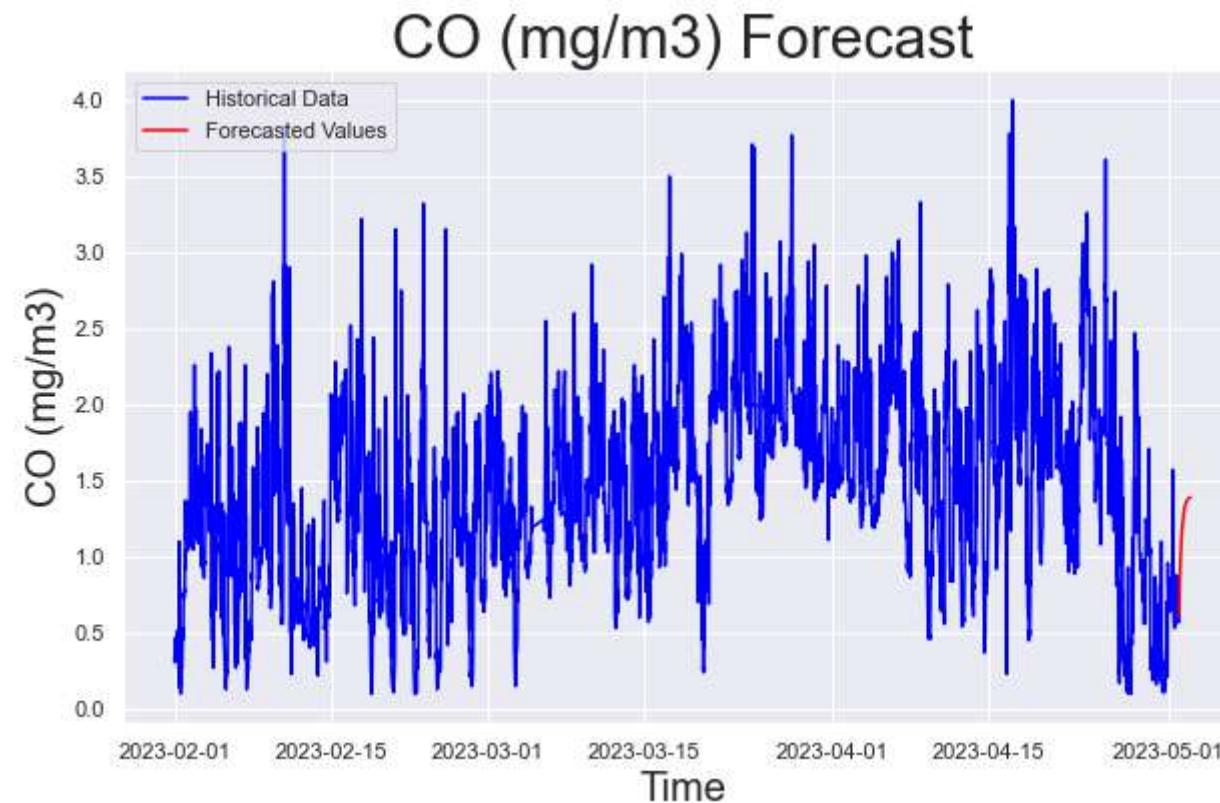
In [346]: `forecast(3)`



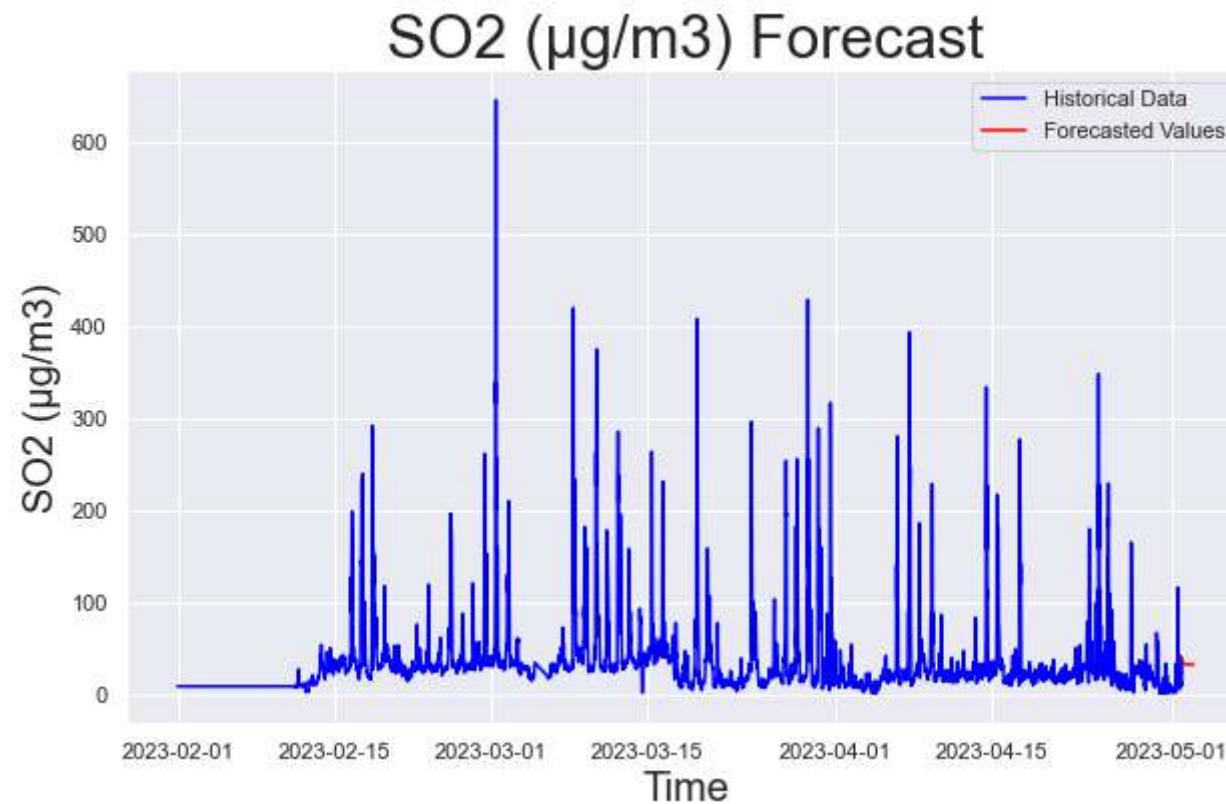
In [347]: `forecast(4)`



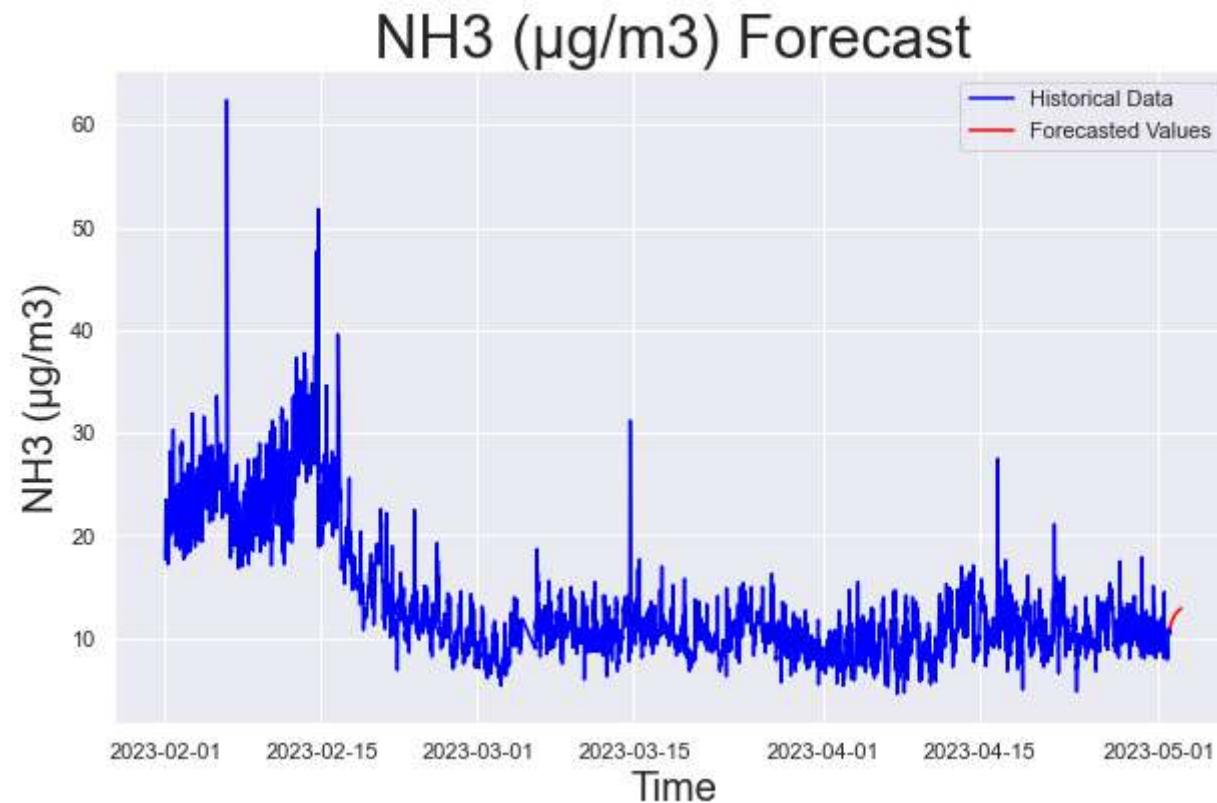
In [348]: `forecast(5)`



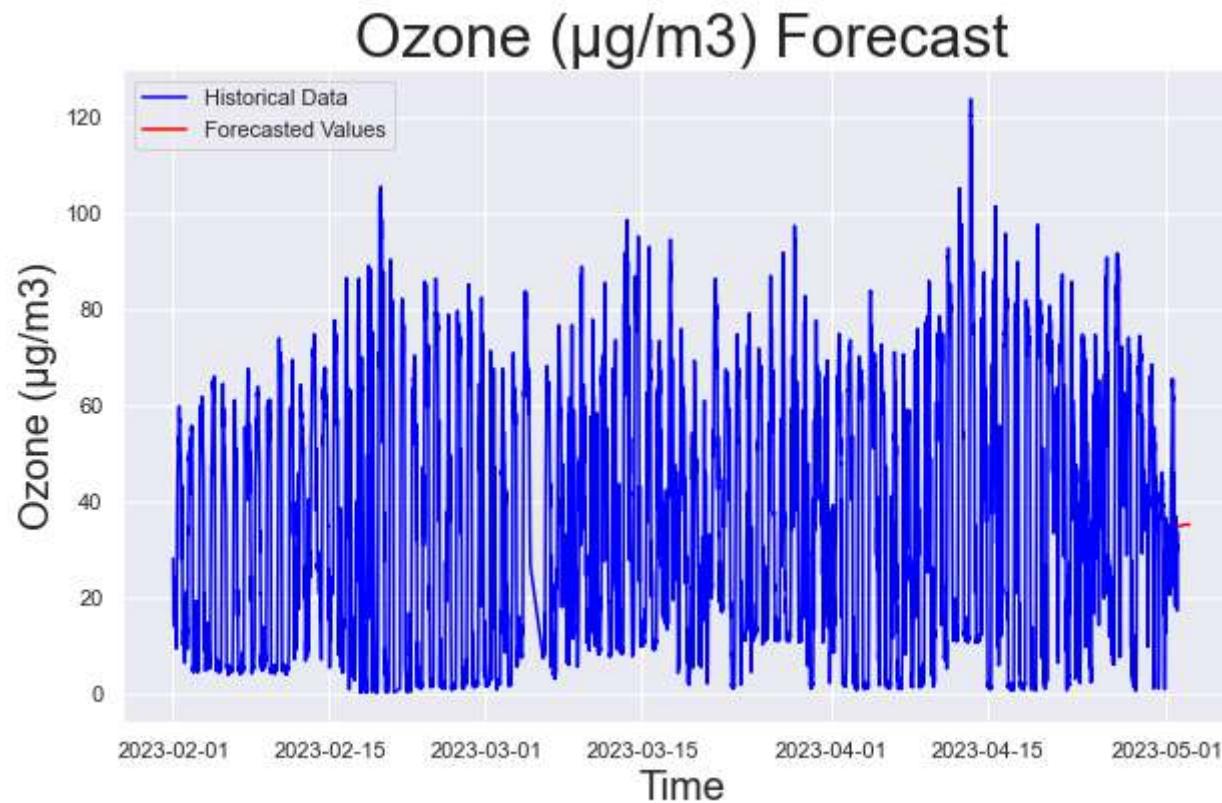
In [349]: `forecast(6)`



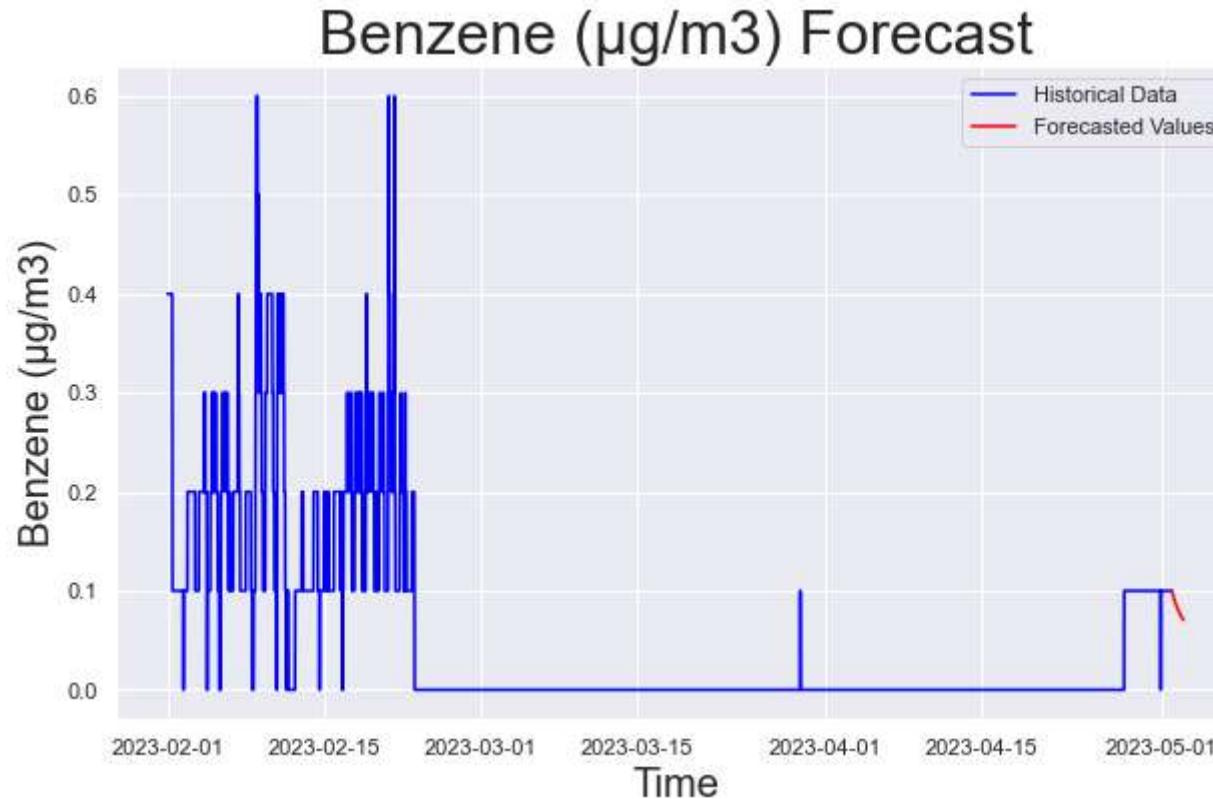
In [350]: `forecast(7)`



In [351]: `forecast(8)`



In [352]: `forecast(9)`



Conditional Probability Calculation

In [353]:

```
i = 0

# Weighted Combination of Air Polluting Factors
weights = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
combined_data = df_new.mul(weights).sum(axis=1)

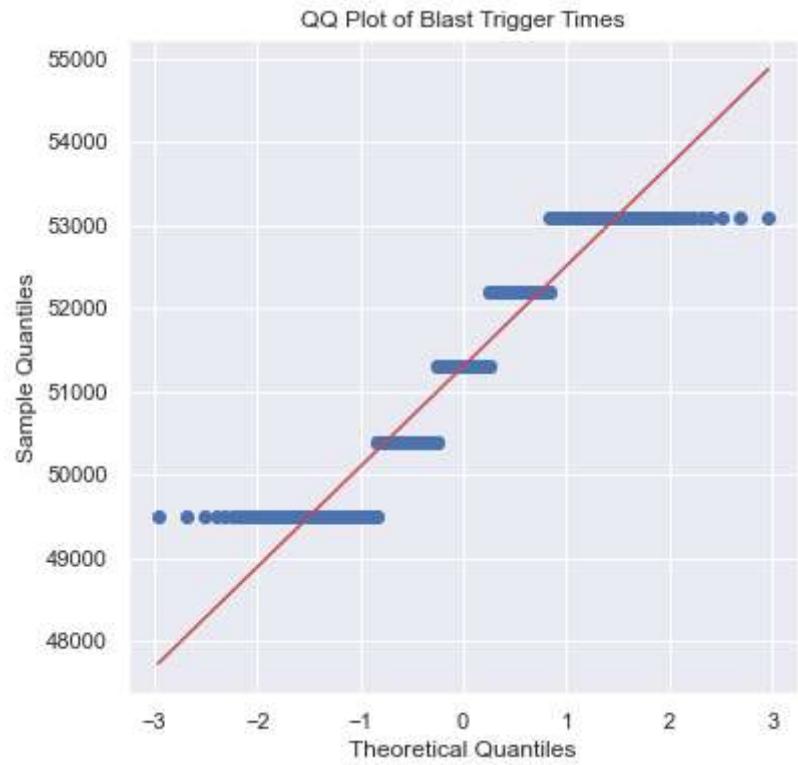
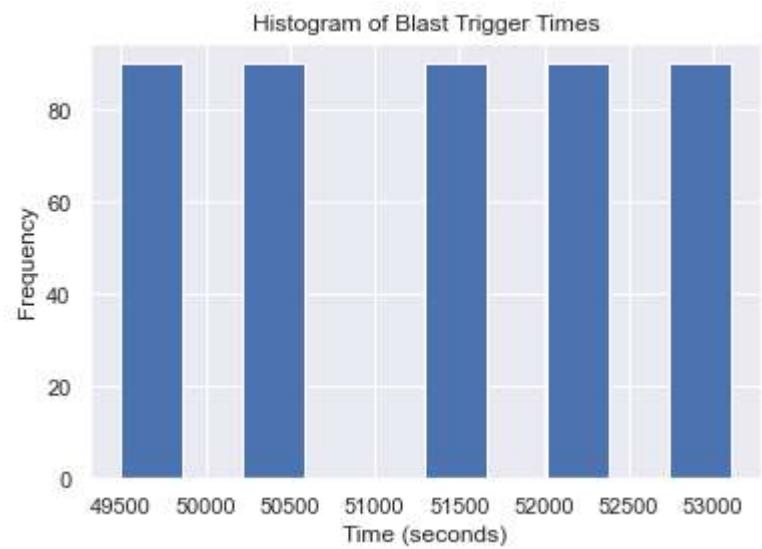
# Detecting Blasting Time
blasting_data = df_new.between_time('13:45:00', '14:45:00')

blasting_impact = blasting_data[ df_new.columns[i] ]
# Plotting Histogram of Blast Trigger Times
blast_trigger_times = blasting_data.index.time
blast_trigger_numeric = [t.hour * 3600 + t.minute * 60 + t.second for t in blast_trigger_times]
plt.hist(blast_trigger_numeric, bins='auto')
plt.xlabel('Time (seconds)')
plt.ylabel('Frequency')
plt.title('Histogram of Blast Trigger Times')
plt.show()

# QQ Plot and Distribution Type
plt.figure(figsize=(6, 6))
stats.probplot(blast_trigger_numeric, dist="norm", plot=plt)
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.title('QQ Plot of Blast Trigger Times')
plt.show()

# Probability of Blast Happening during a Specific Time Interval
start_time = pd.to_datetime('14:15:00').time()
end_time = pd.to_datetime('14:30:00').time()
blast_interval_data = blasting_data.between_time(start_time, end_time)
probability_blast = len(blast_interval_data) / len(blasting_data)

print(f"Probability of blast happening during {start_time} to {end_time}: {probability_blast}")
```



Probability of blast happening during 14:15:00 to 14:30:00: 0.4

Probability of blast happening during 14:15:00 to 14:30:00: 0.4

In []:

In []: