# assignment1

June 27, 2023

## 1 ASSIGNMENT 1

## 2 Name =Bachhav Aryan Kishor

## 3 Roll No=210253

To plot a multi-variate time series, we can utilize various graph plotting techniques such as line plots, scatter plots, area plots, or stacked area plots. These techniques can help visualize the patterns and relationships between different variables over time.

When dealing with NA values in a time series, simply replacing them with 0 values may not be the best strategy. Replacing NA values with 0 can distort the actual data and introduce bias in the analysis. It is generally recommended to handle NA values appropriately based on the nature of the data and the specific analysis being performed. For now we can replace NA values with mean values . Changing the unit of NOX and CO to ug/m3

1) if we drop NA values then plot the graph we can see their major gap graph so we should fill those value with mean or interpolation

changing sign of CO as we can see CO g=has unit mg/m3 we have to update it to ug/m3 and same for NOX we have to update that too

[454]:
```
import pandas as pd

df = pd.read_csv('Open pit blasting 01-02-2023 000000 To 01-05-2023 235959.
 ↪csv', na_values='NA')

# Convert the column to datetime type, handling "Min" as missing value
df['start'] = pd.to_datetime(df['From'], format='%Y-%m-%d %H:%M:%S',␣
 ↪errors='coerce')
df['end'] = pd.to_datetime(df['To (Interval: 15M)'], format='%Y-%m-%d %H:%M:
 ↪%S', errors='coerce')
df = df.drop(['From', '#', 'To (Interval: 15M)'], axis=1)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM10 (µg/m3)':␣
 ↪'PM10'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM2.5 (µg/m3)':␣
 ↪'PM2.5'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO2 (µg/m3)':␣
 ↪'NO2'}, inplace=True)
```

```
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO (µg/m3)':␣
 ↪'NO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  CO (mg/m3)':␣
 ↪'CO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  SO2 (µg/m3)':␣
 ↪'SO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NH3 (µg/m3)':␣
 ↪'NH3'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Ozone (µg/m3)':␣
 ↪'Ozone'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Benzene (µg/m3)':␣
 ↪'Benzene'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NOX (ppb)':␣
 ↪'NOX'}, inplace=True)

df.set_index('start', inplace=True)
df['CO'] = df['CO'] * 1000
df['NOX'] = df['NOX'] * 1.23

# Drop rows with NA values
df.dropna(inplace=True)

df
```

[454]:

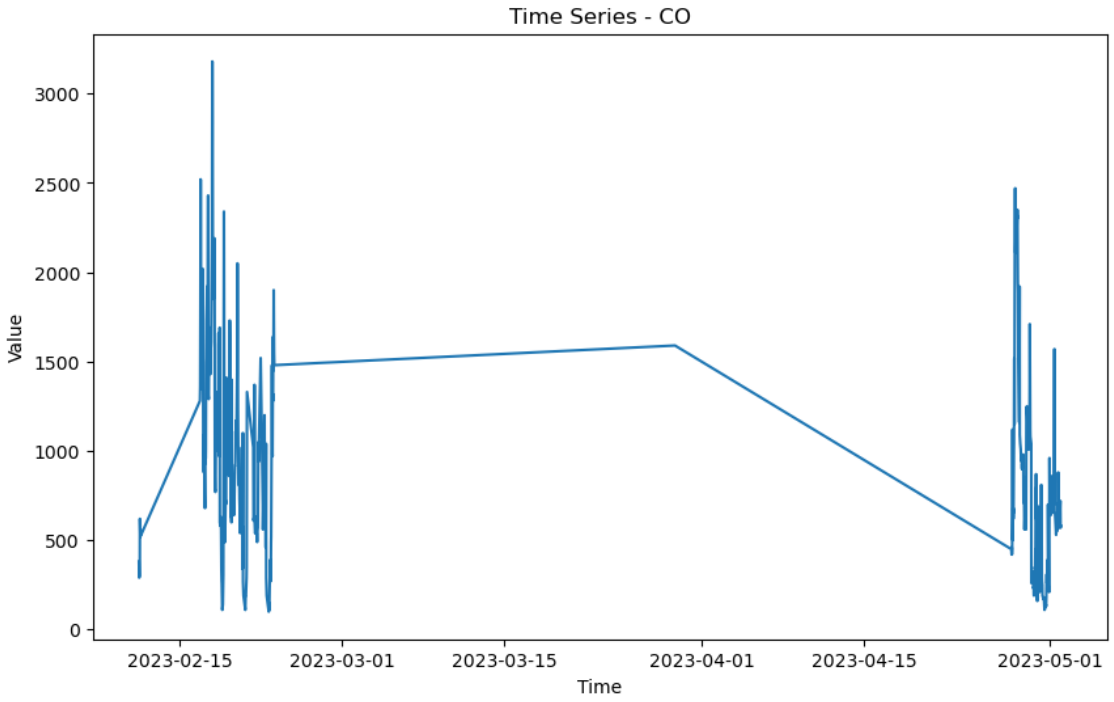
| start | PM10 | PM2.5 | NO | NO2 | NOX | CO | SO2 | NH3 | Ozone | Benzene | end |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2023-02-11 11:00:00 | 110.0 | 26.0 | 18.1 | 74.9 | 67.035 | 380.0 | 8.2 | 32.0 | 59.5 | 0.3 | 2023-02-11 11:15:00 |
| 2023-02-11 11:15:00 | 110.0 | 26.0 | 14.4 | 73.1 | 62.238 | 350.0 | 8.2 | 32.0 | 56.7 | 0.2 | 2023-02-11 11:30:00 |
| 2023-02-11 11:30:00 | 110.0 | 26.0 | 11.5 | 72.4 | 58.794 | 290.0 | 7.9 | 31.8 | 52.0 | 0.2 | 2023-02-11 11:45:00 |
| 2023-02-11 12:30:00 | 110.0 | 20.0 | 9.9 | 65.2 | 52.644 | 300.0 | 7.5 | 22.8 | 52.4 | 0.2 | 2023-02-11 12:45:00 |
| 2023-02-11 12:45:00 | 90.0 | 13.0 | 9.8 | 67.8 | 54.120 | 300.0 | 7.8 | 22.9 | 54.5 | 0.2 | 2023-02-11 13:00:00 |
| … | … | … | … | … | … | … | … | … | … | … | … |
| 2023-05-01 22:30:00 | 28.0 | 7.0 | 17.1 | 99.0 | 81.918 | 720.0 | 9.5 | 10.9 | | | |
| 2023-05-01 22:45:00 | 19.0 | 11.0 | 17.9 | 100.0 | 83.394 | 630.0 | 10.0 | 10.7 | | | |
| 2023-05-01 23:00:00 | 19.0 | 11.0 | 17.9 | 100.0 | 83.271 | 570.0 | 10.0 | 10.4 | | | |
| 2023-05-01 23:15:00 | 19.0 | 11.0 | 19.6 | 100.2 | 85.116 | 580.0 | 9.9 | 10.5 | | | |
| 2023-05-01 23:30:00 | 19.0 | 11.0 | 20.8 | 100.2 | 86.346 | 580.0 | 9.5 | 10.8 | | | |

```
2023-05-01 22:30:00   17.2      0.1 2023-05-01 22:45:00
2023-05-01 22:45:00   26.1      0.1 2023-05-01 23:00:00
2023-05-01 23:00:00   30.9      0.1 2023-05-01 23:15:00
2023-05-01 23:15:00   29.6      0.1 2023-05-01 23:30:00
2023-05-01 23:30:00   30.0      0.1 2023-05-01 23:45:00

[766 rows x 11 columns]
```

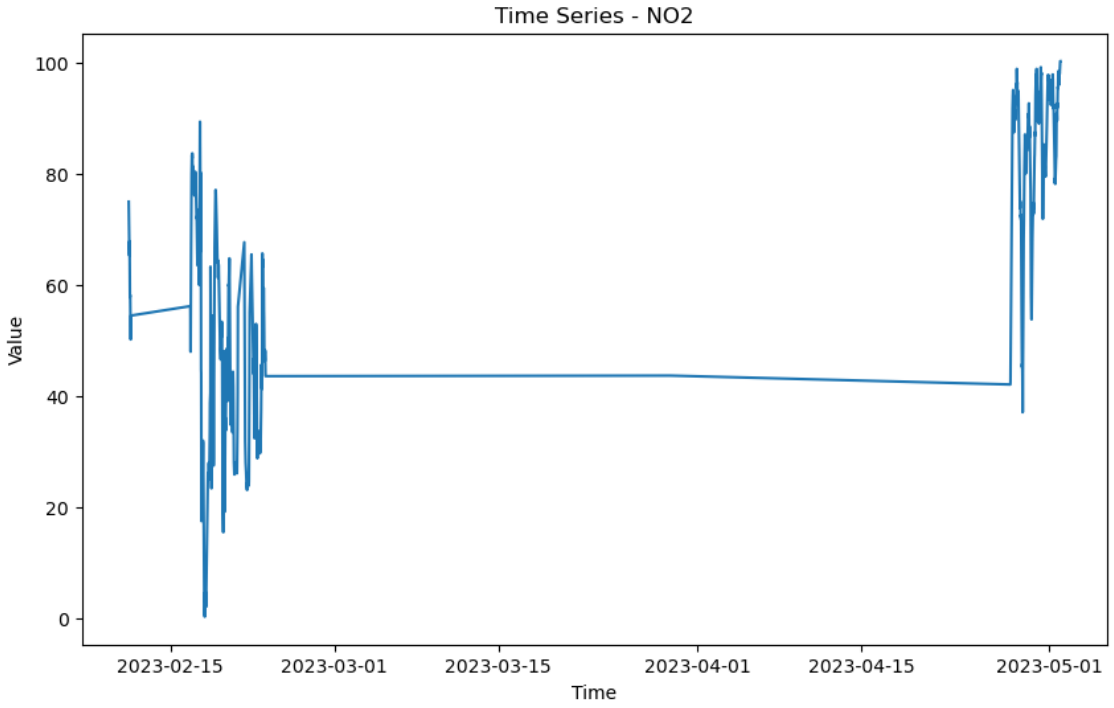
[455]:
```python
from statsmodels.tsa.stattools import adfuller

# Columns of interest
columns_of_interest = ['CO', 'NO2', 'Ozone', 'PM10', 'PM2.5', 'SO2', 'NOX']

# Iterate over each column
for column in columns_of_interest:
    # Perform ADF test
    result = adfuller(df[column])

    # Extract ADF test statistics and p-value
    adf_statistic = result[0]
    p_value = result[1]

    # Print the ADF test statistics and p-value
    print(f'Column: {column}')
    print(f'ADF Statistic: {adf_statistic}')
    print(f'p-value: {p_value}')

    # Define a threshold for p-value to determine stationarity
    alpha = 0.05

    # Check the result of the ADF test
    if p_value < alpha:
        print('The time series is stationary (reject the null hypothesis).')
    else:
        print('The time series is non-stationary (fail to reject the null␣
 ↪hypothesis).')

    # Plot the time series
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column])
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.title(f'Time Series - {column}')
    plt.show()
```

```
Column: CO
ADF Statistic: -4.795006817798647
```

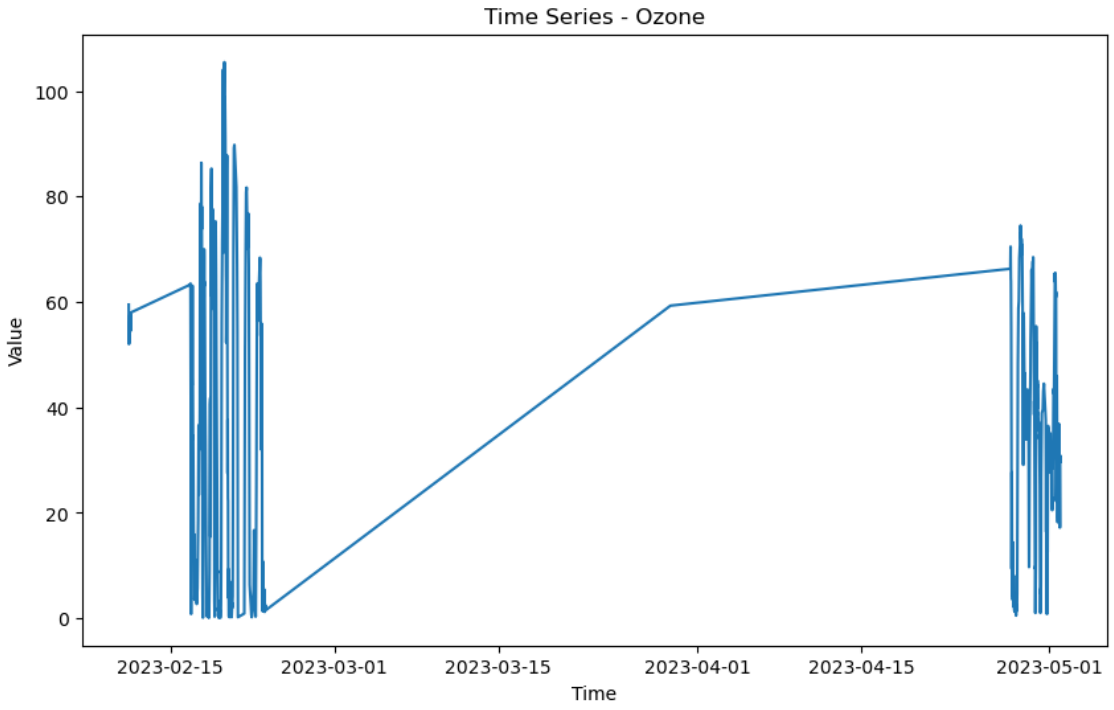
p-value: 5.559479615493005e-05
The time series is stationary (reject the null hypothesis).



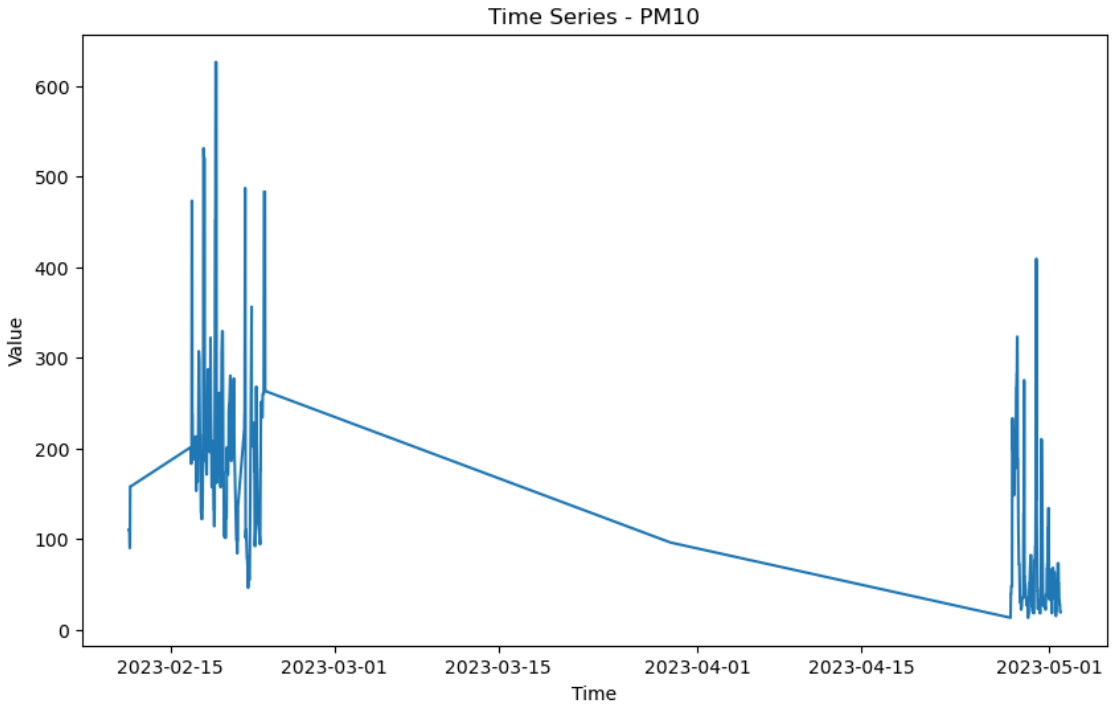

Column: NO2
ADF Statistic: -2.669483450396587
p-value: 0.07946945787859017
The time series is non-stationary (fail to reject the null hypothesis).



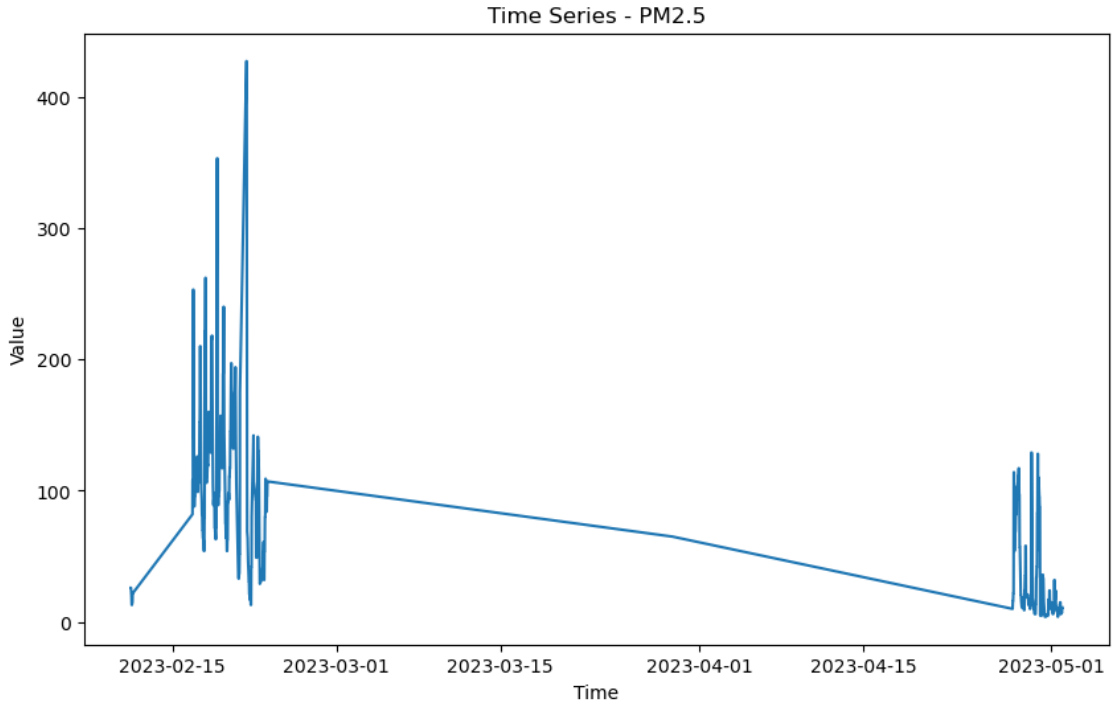

```
Column: Ozone
ADF Statistic: -5.938215856782174
p-value: 2.2938480345721162e-07
The time series is stationary (reject the null hypothesis).
```



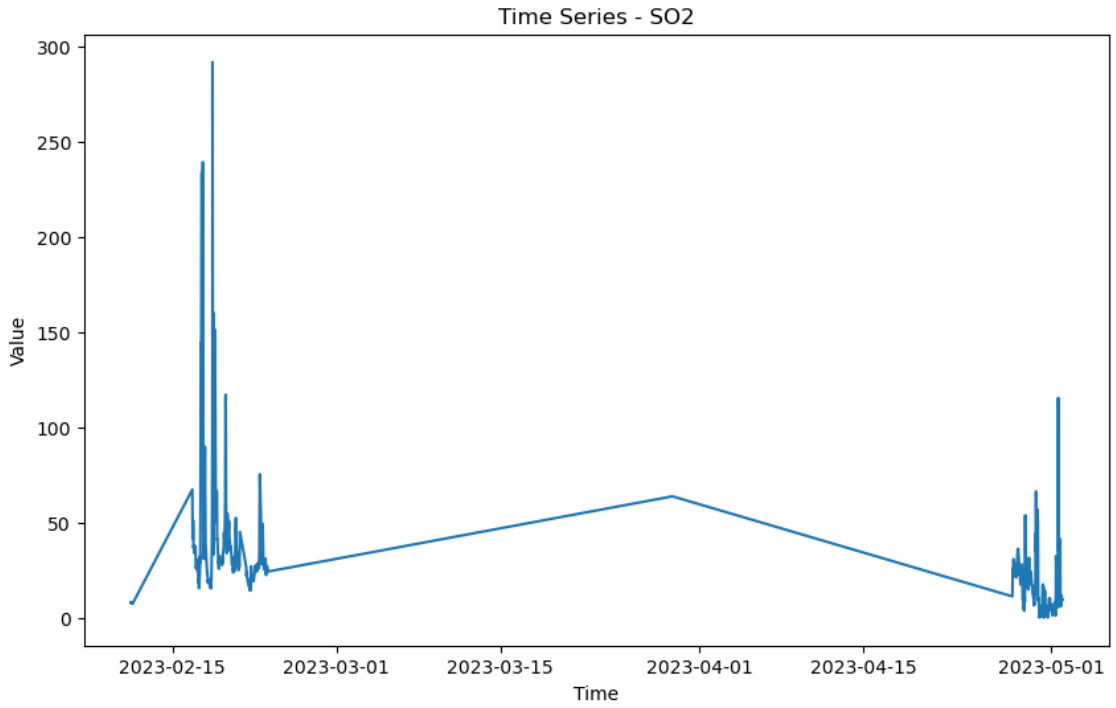
```
Column: PM10
ADF Statistic: -4.199738418758473
p-value: 0.0006595723184032765
The time series is stationary (reject the null hypothesis).
```



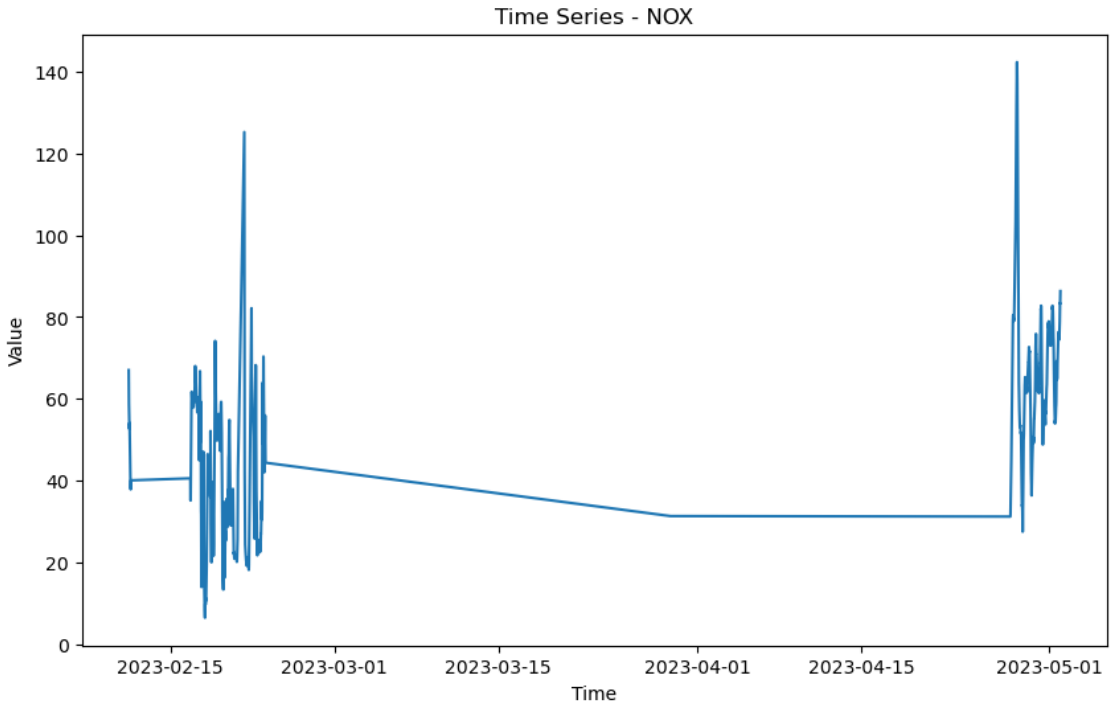

```
Column: PM2.5
ADF Statistic: -5.472515263800854
p-value: 2.3811819753775418e-06
The time series is stationary (reject the null hypothesis).
```




```
Column: SO2
ADF Statistic: -5.129047864811531
p-value: 1.2251136004615971e-05
The time series is stationary (reject the null hypothesis).
```



```
Column: NOX
ADF Statistic: -4.090277270832267
p-value: 0.0010049918708811116
The time series is stationary (reject the null hypothesis).
```




2)lets try to fill NA values with zero

[457]:
```
import pandas as pd

df = pd.read_csv('Open pit blasting 01-02-2023 000000 To 01-05-2023 235959.
 ↪csv', na_values='NA')

# Convert the column to datetime type, handling "Min" as missing value
df['start'] = pd.to_datetime(df['From'], format='%Y-%m-%d %H:%M:%S',␣
 ↪errors='coerce')
df['end'] = pd.to_datetime(df['To (Interval: 15M)'], format='%Y-%m-%d %H:%M:
 ↪%S', errors='coerce')
df = df.drop(['From', '#', 'To (Interval: 15M)'], axis=1)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM10 (µg/m3)':␣
 ↪'PM10'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM2.5 (µg/m3)':␣
 ↪'PM2.5'}, inplace=True)
```

```
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO2 (µg/m3)':␣
 ↪'NO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO (µg/m3)':␣
 ↪'NO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  CO (mg/m3)':␣
 ↪'CO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  SO2 (µg/m3)':␣
 ↪'SO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NH3 (µg/m3)':␣
 ↪'NH3'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Ozone (µg/m3)':␣
 ↪'Ozone'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Benzene (µg/m3)':␣
 ↪'Benzene'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NOX (ppb)':␣
 ↪'NOX'}, inplace=True)

columns_to_impute = ['NO2', 'NO', 'SO2', 'PM10', 'PM2.5', 'NOX', 'NH3', 'CO',␣
 ↪'Ozone', 'Benzene']  # Add the column names you want to handle

# Handle missing values for each column
for column in columns_to_impute:
    df[column].fillna(0, inplace=True)  # Fill NA values with zero

df.set_index('start', inplace=True)
df['CO'] = df['CO'] * 1000
df['NOX'] = df['NOX'] * 1.23
df = df.iloc[:-3]

df
```

[457]:

| start | PM10 | PM2.5 | NO | NO2 | NOX | CO | SO2 | NH3 | \ |
|---|---|---|---|---|---|---|---|---|---|
| 2023-02-01 00:00:00 | 95.0 | 35.0 | 0.0 | 90.1 | 69.126 | 310.0 | 0.0 | 17.7 | |
| 2023-02-01 00:15:00 | 95.0 | 35.0 | 0.0 | 88.0 | 67.773 | 330.0 | 0.0 | 18.3 | |
| 2023-02-01 00:30:00 | 95.0 | 35.0 | 0.0 | 87.7 | 67.896 | 380.0 | 0.0 | 19.7 | |
| 2023-02-01 00:45:00 | 122.0 | 34.0 | 0.0 | 88.9 | 68.511 | 380.0 | 0.0 | 21.3 | |
| 2023-02-01 01:00:00 | 122.0 | 34.0 | 0.0 | 90.0 | 68.634 | 380.0 | 0.0 | 22.3 | |
| … | … | … | … | … | … | … | … | … | |
| 2023-05-01 22:45:00 | 19.0 | 11.0 | 17.9 | 100.0 | 83.394 | 630.0 | 10.0 | 10.7 | |
| 2023-05-01 23:00:00 | 19.0 | 11.0 | 17.9 | 100.0 | 83.271 | 570.0 | 10.0 | 10.4 | |
| 2023-05-01 23:15:00 | 19.0 | 11.0 | 19.6 | 100.2 | 85.116 | 580.0 | 9.9 | 10.5 | |
| 2023-05-01 23:30:00 | 19.0 | 11.0 | 20.8 | 100.2 | 86.346 | 580.0 | 9.5 | 10.8 | |
| 2023-05-01 23:45:00 | 32.0 | 6.0 | 21.8 | 98.8 | 86.469 | 0.0 | 0.0 | 11.0 | |

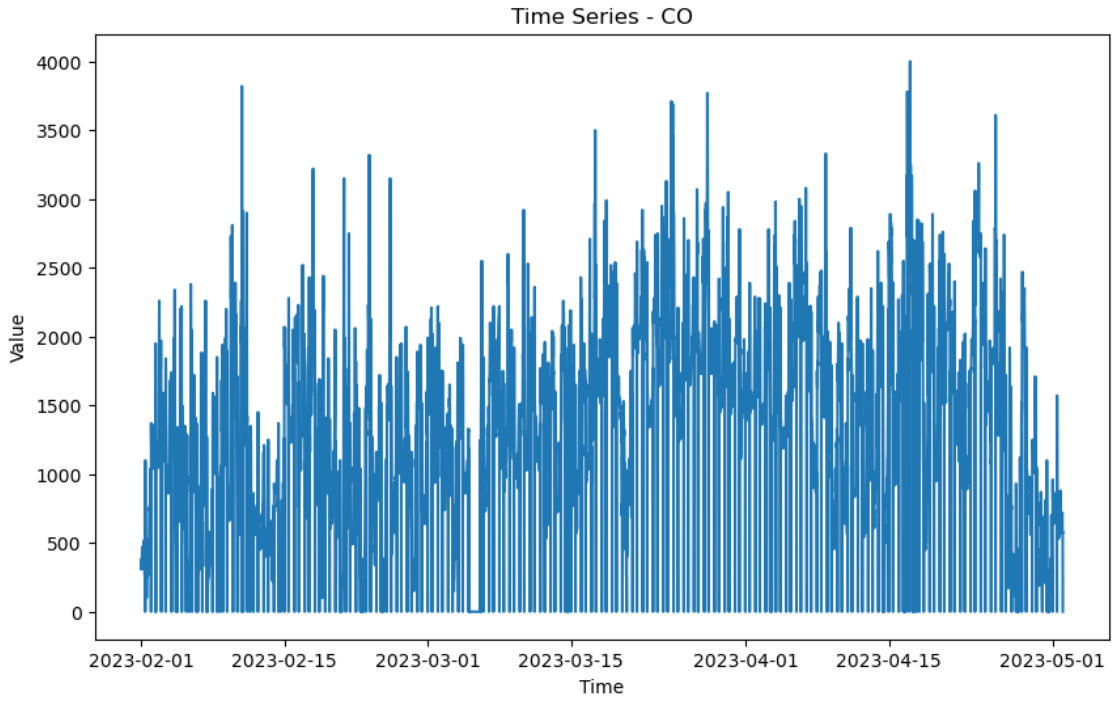
| start | Ozone | Benzene | end |
|---|---|---|---|

```
start
2023-02-01 00:00:00   28.1      0.4 2023-02-01 00:15:00
2023-02-01 00:15:00   27.1      0.4 2023-02-01 00:30:00
2023-02-01 00:30:00   24.9      0.4 2023-02-01 00:45:00
2023-02-01 00:45:00   21.9      0.4 2023-02-01 01:00:00
2023-02-01 01:00:00   16.7      0.4 2023-02-01 01:15:00
…                     …         …                   …
2023-05-01 22:45:00   26.1      0.1 2023-05-01 23:00:00
2023-05-01 23:00:00   30.9      0.1 2023-05-01 23:15:00
2023-05-01 23:15:00   29.6      0.1 2023-05-01 23:30:00
2023-05-01 23:30:00   30.0      0.1 2023-05-01 23:45:00
2023-05-01 23:45:00   33.5      0.1 2023-05-02 00:00:00

[8640 rows x 11 columns]
```

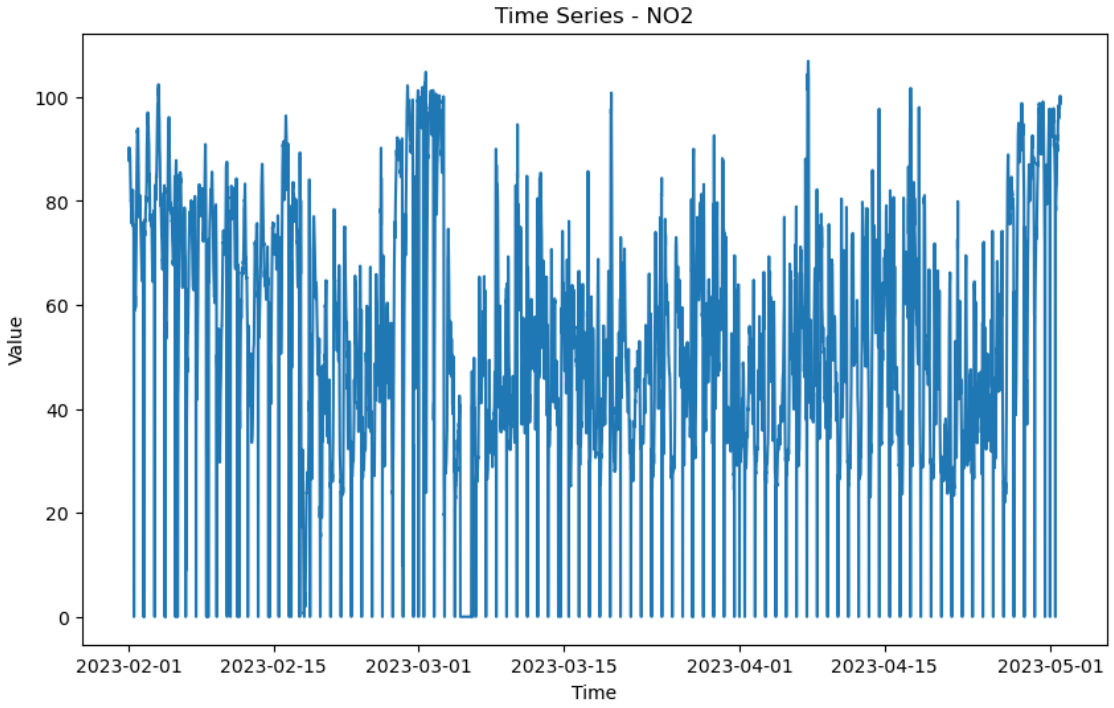
[458]:
```python
from statsmodels.tsa.stattools import adfuller

# Columns of interest
columns_of_interest = ['CO', 'NO2', 'Ozone', 'PM10', 'PM2.5', 'SO2', 'NOX']

# Iterate over each column
for column in columns_of_interest:
    # Perform ADF test
    result = adfuller(df[column])

    # Extract ADF test statistics and p-value
    adf_statistic = result[0]
    p_value = result[1]

    # Print the ADF test statistics and p-value
    print(f'Column: {column}')
    print(f'ADF Statistic: {adf_statistic}')
    print(f'p-value: {p_value}')

    # Define a threshold for p-value to determine stationarity
    alpha = 0.05

    # Check the result of the ADF test
    if p_value < alpha:
        print('The time series is stationary (reject the null hypothesis).')
    else:
        print('The time series is non-stationary (fail to reject the null␣
 ↪hypothesis).')

    # Plot the time series
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column])
```

```
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.title(f'Time Series - {column}')
    plt.show()
```

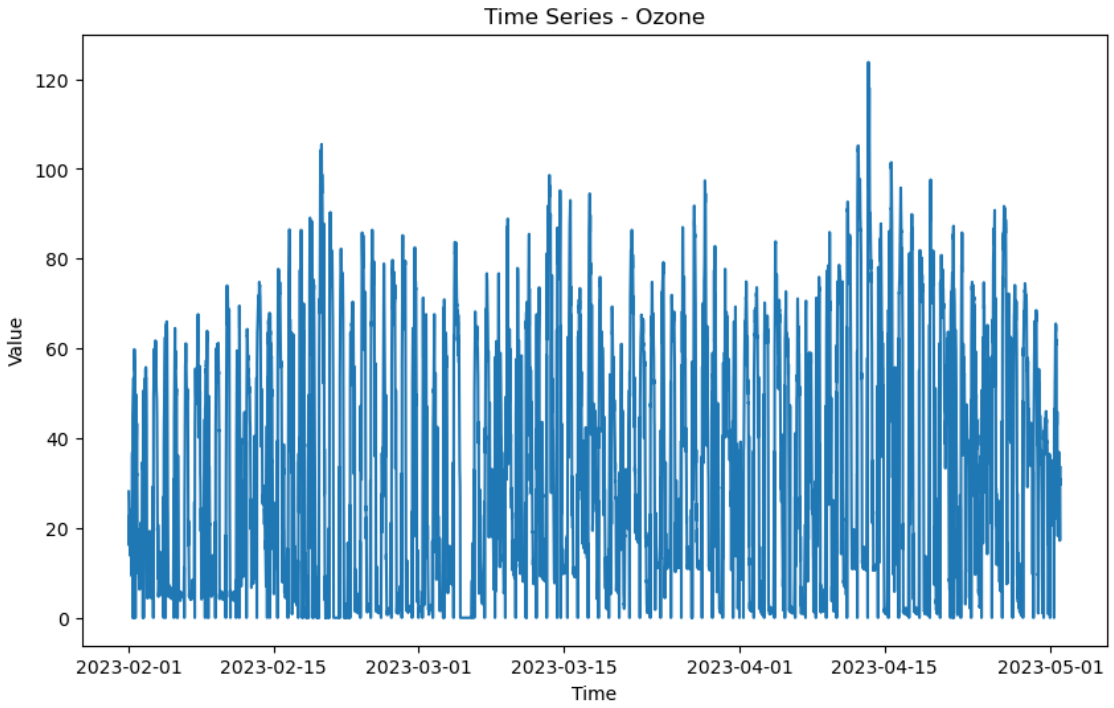
```
Column: CO
ADF Statistic: -11.565786172653722
p-value: 3.188917672012004e-21
The time series is stationary (reject the null hypothesis).
```



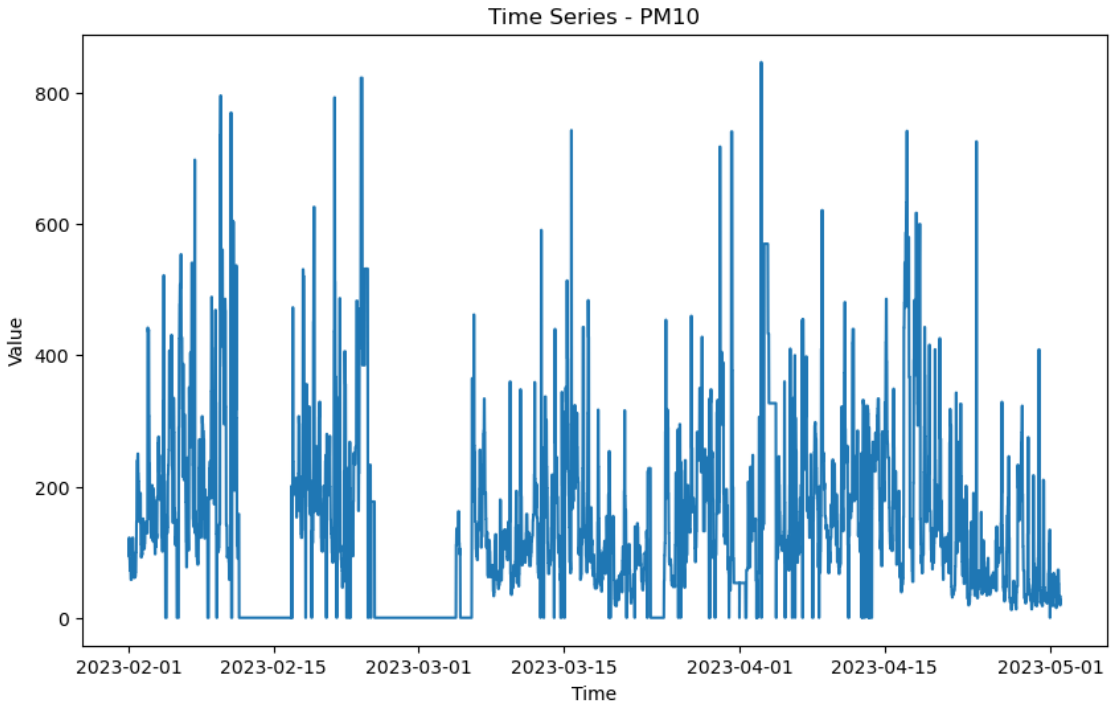

```
Column: NO2
ADF Statistic: -9.747911502361072
p-value: 8.161771045682678e-17
The time series is stationary (reject the null hypothesis).
```



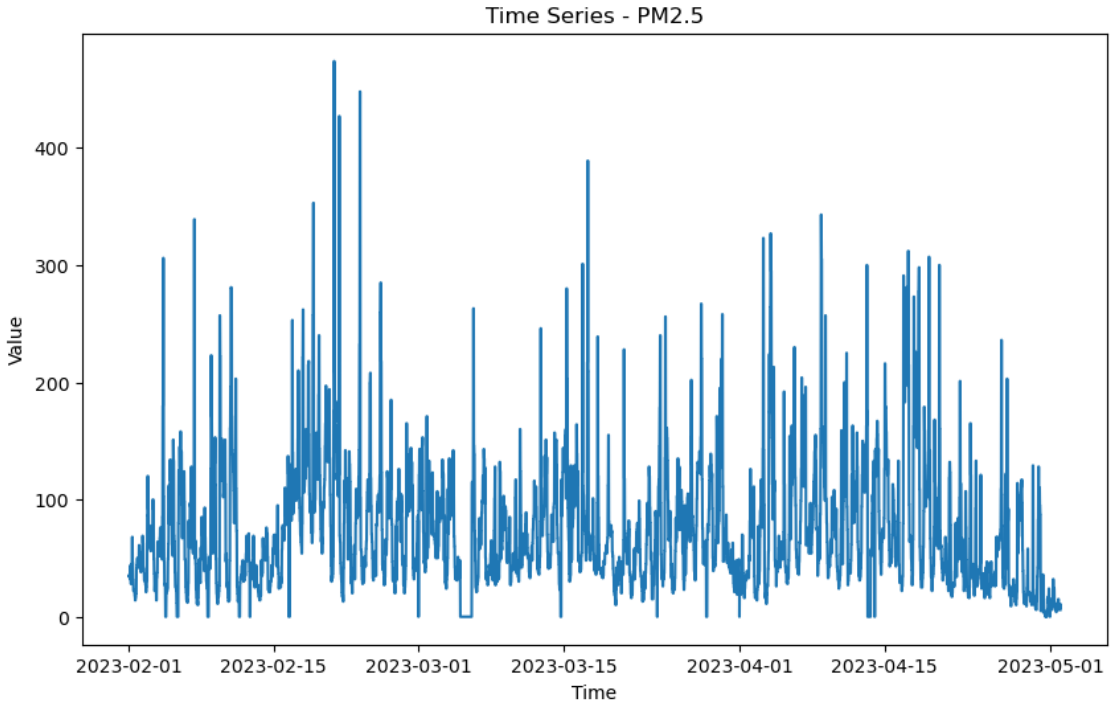

```
Column: Ozone
ADF Statistic: -20.190679760899208
p-value: 0.0
The time series is stationary (reject the null hypothesis).
```



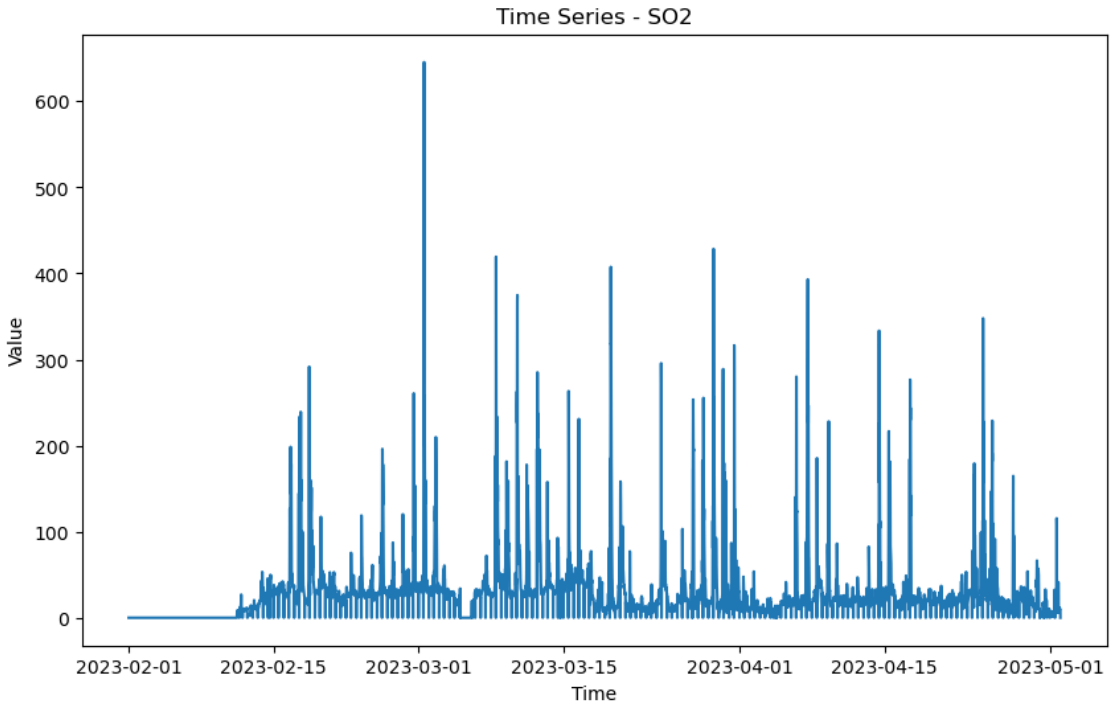
```
Column: PM10
ADF Statistic: -7.861551031137817
p-value: 5.267732385009768e-12
The time series is stationary (reject the null hypothesis).
```



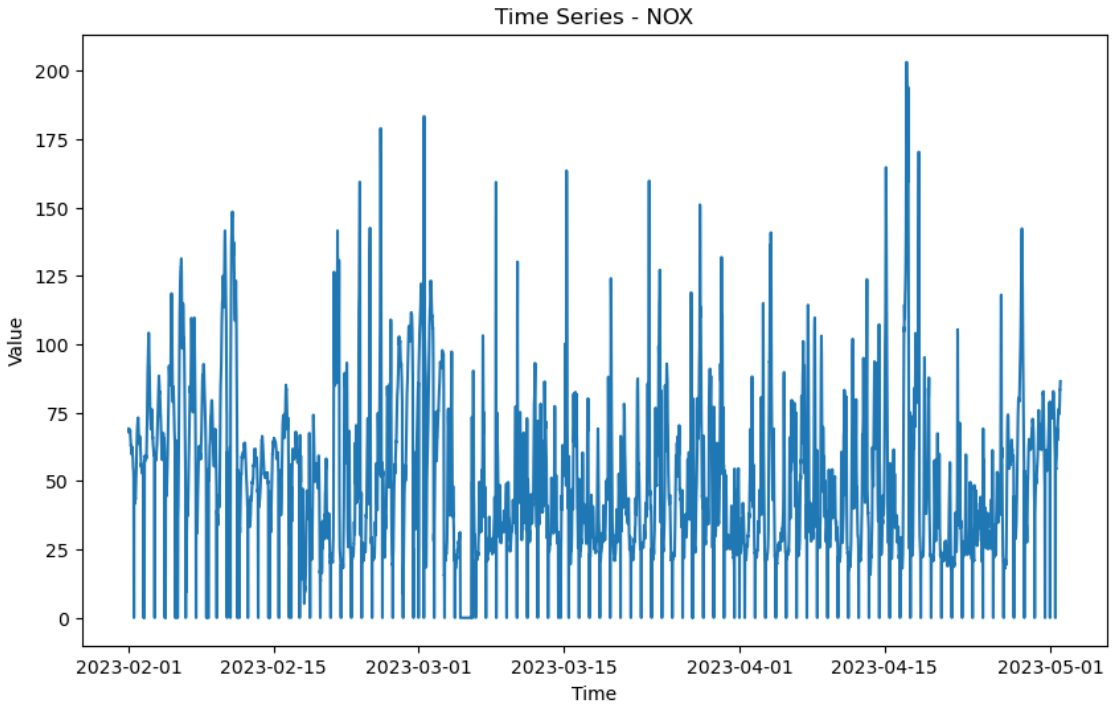

```
Column: PM2.5
ADF Statistic: -11.222738135871086
p-value: 1.994837137328599e-20
The time series is stationary (reject the null hypothesis).
```




```
Column: SO2
ADF Statistic: -16.305332562061427
p-value: 3.235738942509796e-29
The time series is stationary (reject the null hypothesis).
```



```
Column: NOX
ADF Statistic: -13.37126175082574
p-value: 5.1923245175574805e-25
The time series is stationary (reject the null hypothesis).
```




as you can see the graph get distorted if we fill NA with zero if should be related

3) let fill NA with mean values

[459]:
```
import pandas as pd

df = pd.read_csv('Open pit blasting 01-02-2023 000000 To 01-05-2023 235959.
 ↪csv', na_values='NA')

# Convert the column to datetime type, handling "Min" as missing value
df['start'] = pd.to_datetime(df['From'], format='%Y-%m-%d %H:%M:%S',␣
 ↪errors='coerce')
df['end'] = pd.to_datetime(df['To (Interval: 15M)'], format='%Y-%m-%d %H:%M:
 ↪%S', errors='coerce')
df = df.drop(['From', '#', 'To (Interval: 15M)'], axis=1)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM10 (µg/m3)':␣
 ↪'PM10'}, inplace=True)
```

```
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM2.5 (µg/m3)':␣
 ↪'PM2.5'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO2 (µg/m3)':␣
 ↪'NO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO (µg/m3)':␣
 ↪'NO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  CO (mg/m3)':␣
 ↪'CO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  SO2 (µg/m3)':␣
 ↪'SO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NH3 (µg/m3)':␣
 ↪'NH3'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Ozone (µg/m3)':␣
 ↪'Ozone'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Benzene (µg/m3)':␣
 ↪'Benzene'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NOX (ppb)':␣
 ↪'NOX'}, inplace=True)

columns_to_impute = ['NO2', 'NO', 'SO2', 'PM10', 'PM2.5', 'NOX','NH3', 'CO',␣
 ↪'Ozone','Benzene']  # Add the column names you want to handle

# Handle missing values for each column
for column in columns_to_impute:
    df[column].fillna(df[column].mean(), inplace=True)

df.set_index('start', inplace=True)
df['CO'] = df['CO'] * 1000
df['NOX'] = df['NOX'] * 1.23
df = df.iloc[:-3]

df
```

[459]:

| start | PM10 | PM2.5 | NO | NO2 | NOX | CO |
|---|---|---|---|---|---|---|
| 2023-02-01 00:00:00 | 95.0 | 35.0 | 14.667274 | 90.1 | 69.126 | 310.000000 |
| 2023-02-01 00:15:00 | 95.0 | 35.0 | 14.667274 | 88.0 | 67.773 | 330.000000 |
| 2023-02-01 00:30:00 | 95.0 | 35.0 | 14.667274 | 87.7 | 67.896 | 380.000000 |
| 2023-02-01 00:45:00 | 122.0 | 34.0 | 14.667274 | 88.9 | 68.511 | 380.000000 |
| 2023-02-01 01:00:00 | 122.0 | 34.0 | 14.667274 | 90.0 | 68.634 | 380.000000 |
| … | … | … | … | … | … | … |
| 2023-05-01 22:45:00 | 19.0 | 11.0 | 17.900000 | 100.0 | 83.394 | 630.000000 |
| 2023-05-01 23:00:00 | 19.0 | 11.0 | 17.900000 | 100.0 | 83.271 | 570.000000 |
| 2023-05-01 23:15:00 | 19.0 | 11.0 | 19.600000 | 100.2 | 85.116 | 580.000000 |
| 2023-05-01 23:30:00 | 19.0 | 11.0 | 20.800000 | 100.2 | 86.346 | 580.000000 |
| 2023-05-01 23:45:00 | 32.0 | 6.0 | 21.800000 | 98.8 | 86.469 | 1408.695225 |

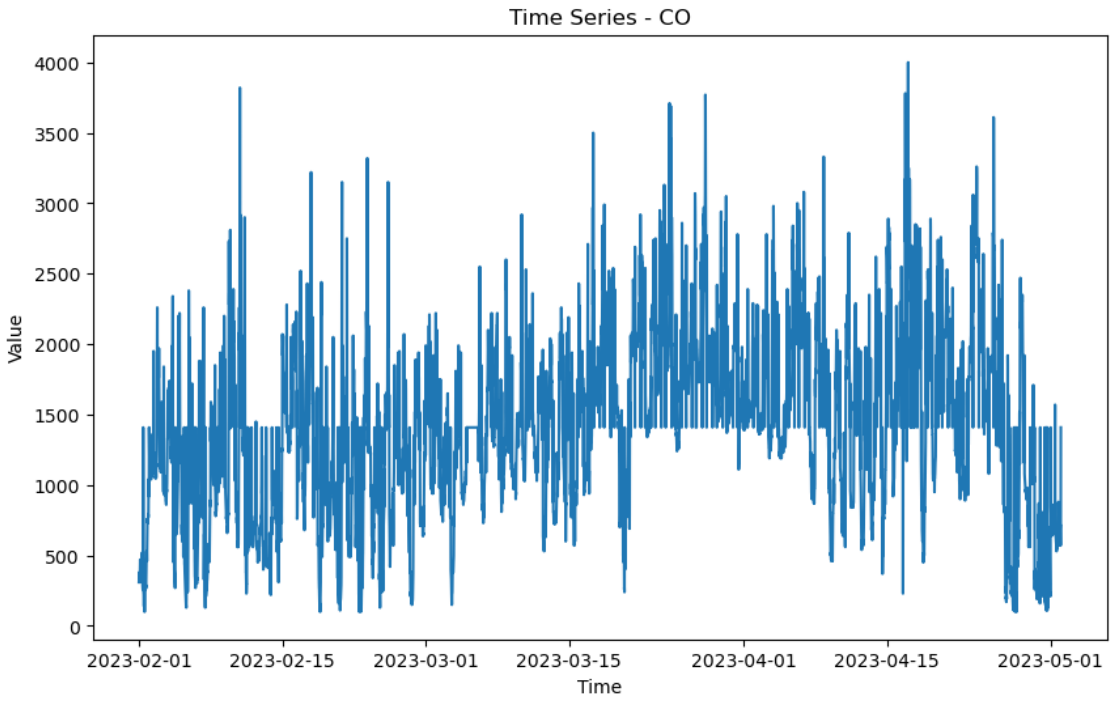
```
                          SO2   NH3  Ozone  Benzene                 end
start
2023-02-01 00:00:00  34.312991  17.7   28.1      0.4 2023-02-01 00:15:00
2023-02-01 00:15:00  34.312991  18.3   27.1      0.4 2023-02-01 00:30:00
2023-02-01 00:30:00  34.312991  19.7   24.9      0.4 2023-02-01 00:45:00
2023-02-01 00:45:00  34.312991  21.3   21.9      0.4 2023-02-01 01:00:00
2023-02-01 01:00:00  34.312991  22.3   16.7      0.4 2023-02-01 01:15:00
…                          …    …     …        …                   …
2023-05-01 22:45:00  10.000000  10.7   26.1      0.1 2023-05-01 23:00:00
2023-05-01 23:00:00  10.000000  10.4   30.9      0.1 2023-05-01 23:15:00
2023-05-01 23:15:00   9.900000  10.5   29.6      0.1 2023-05-01 23:30:00
2023-05-01 23:30:00   9.500000  10.8   30.0      0.1 2023-05-01 23:45:00
2023-05-01 23:45:00  34.312991  11.0   33.5      0.1 2023-05-02 00:00:00

[8640 rows x 11 columns]
```

As you can see now graph is more relatable checking staationarity of every coloumn in the dataset which is found to be stationary so we can apply ARIMA model

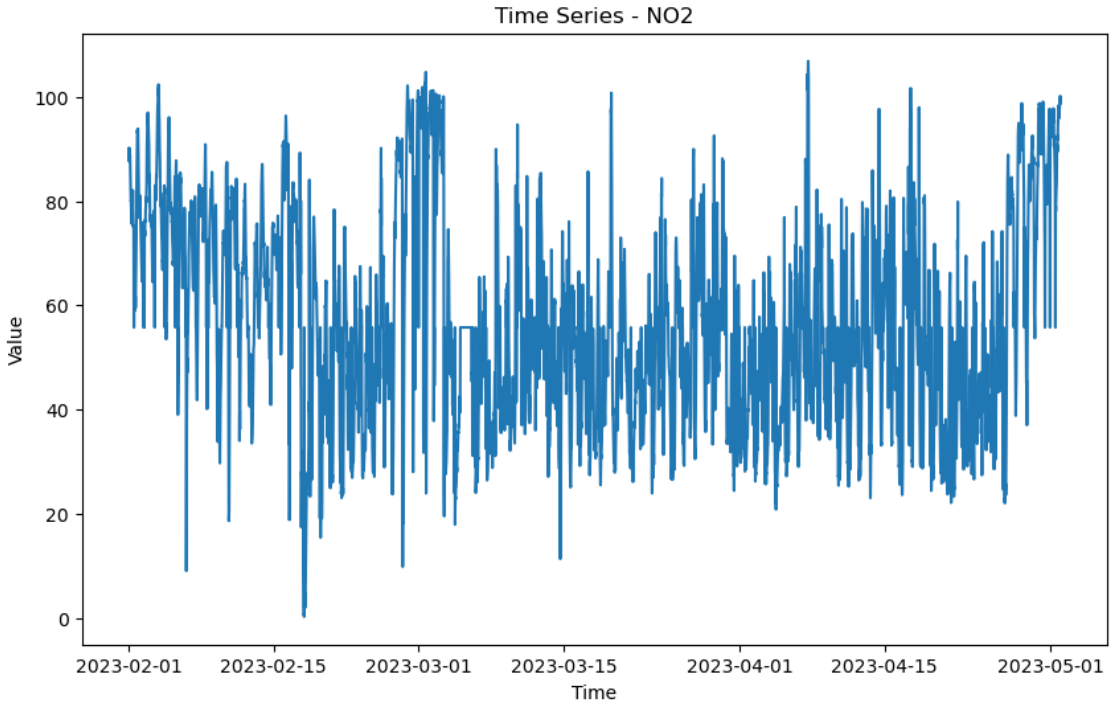
[462]:
```python
from statsmodels.tsa.stattools import adfuller

# Columns of interest
columns_of_interest = ['CO', 'NO2', 'Ozone', 'PM10', 'PM2.5', 'SO2', 'NOX']

# Iterate over each column
for column in columns_of_interest:
    # Perform ADF test
    result = adfuller(df[column])

    # Extract ADF test statistics and p-value
    adf_statistic = result[0]
    p_value = result[1]

    # Print the ADF test statistics and p-value
    print(f'Column: {column}')
    print(f'ADF Statistic: {adf_statistic}')
    print(f'p-value: {p_value}')

    # Define a threshold for p-value to determine stationarity
    alpha = 0.05

    # Check the result of the ADF test
    if p_value < alpha:
        print('The time series is stationary (reject the null hypothesis).')
    else:
```

```
        print('The time series is non-stationary (fail to reject the null␣
↪hypothesis).')

    # Plot the time series
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column])
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.title(f'Time Series - {column}')
    plt.show()
```

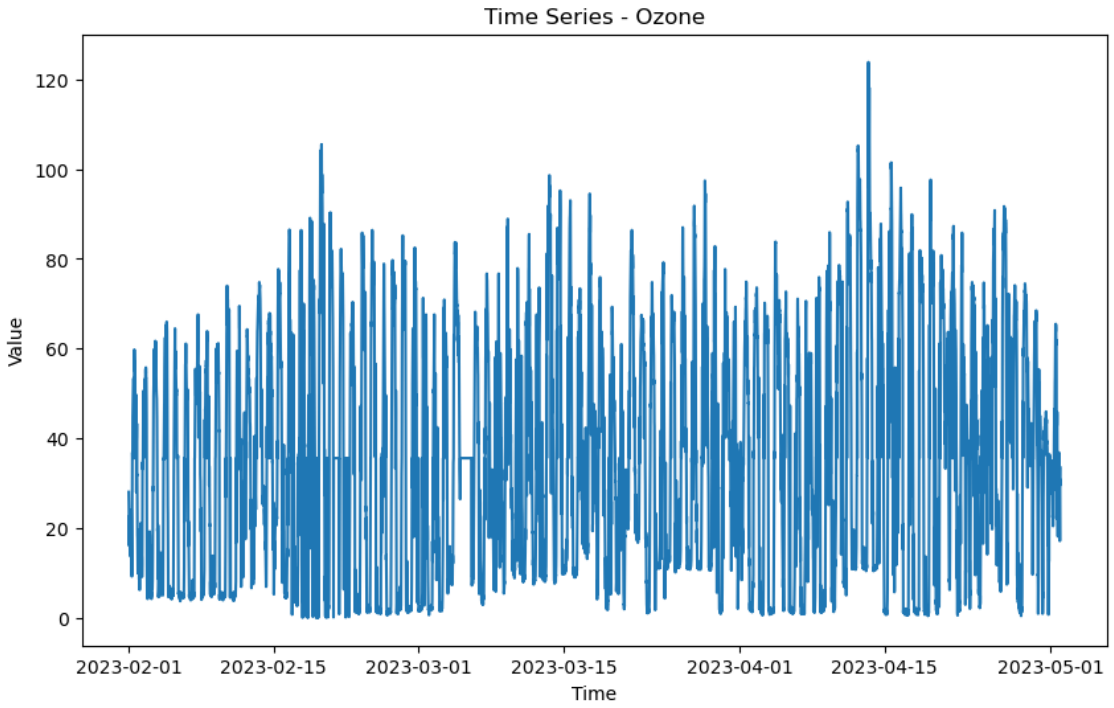
```
Column: CO
ADF Statistic: -10.603976786188195
p-value: 6.069262939917717e-19
The time series is stationary (reject the null hypothesis).
```



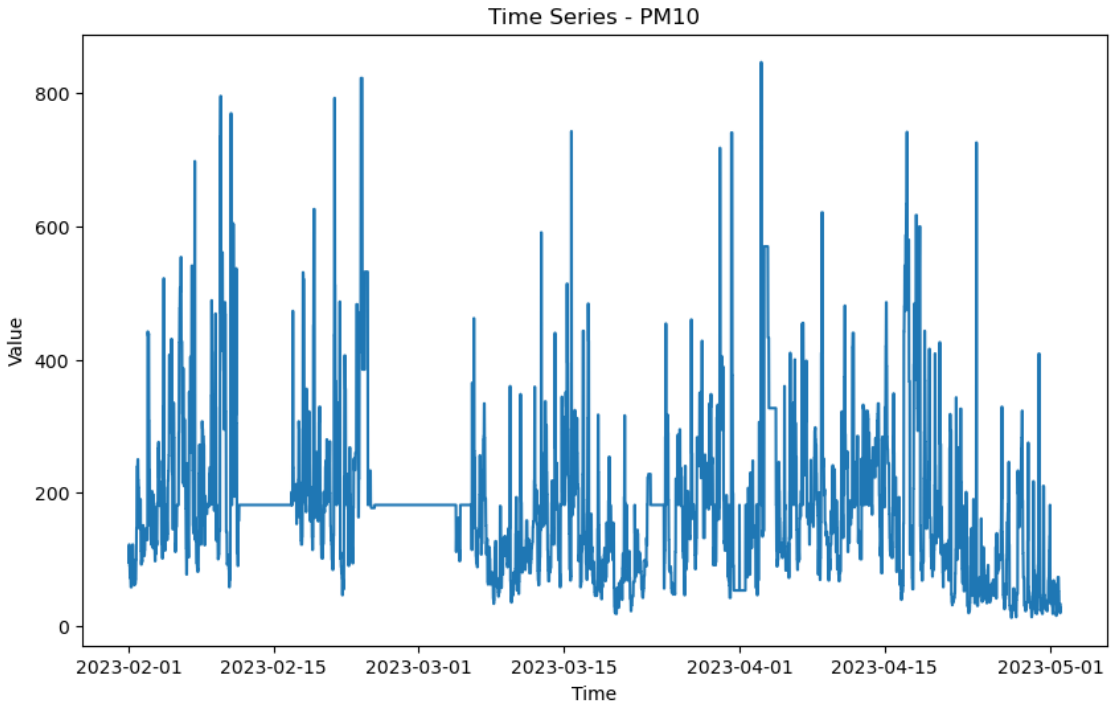

```
Column: NO2
ADF Statistic: -9.293484582542694
p-value: 1.1627850836300296e-15
The time series is stationary (reject the null hypothesis).
```



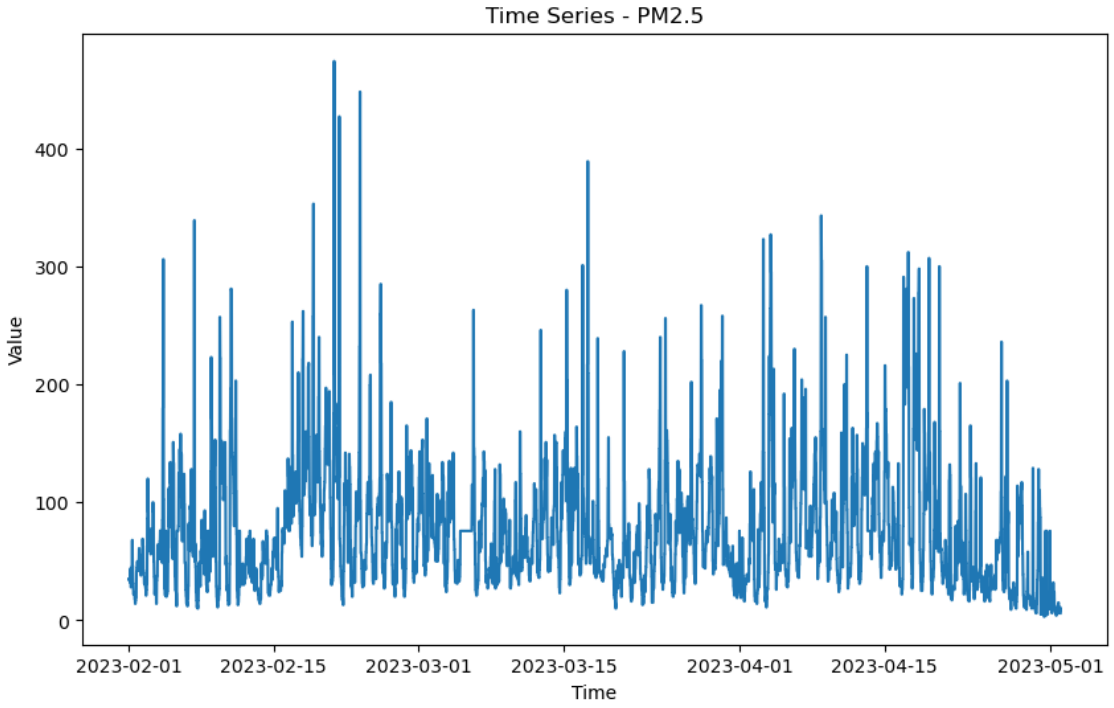

```
Column: Ozone
ADF Statistic: -20.915452969224916
p-value: 0.0
The time series is stationary (reject the null hypothesis).
```



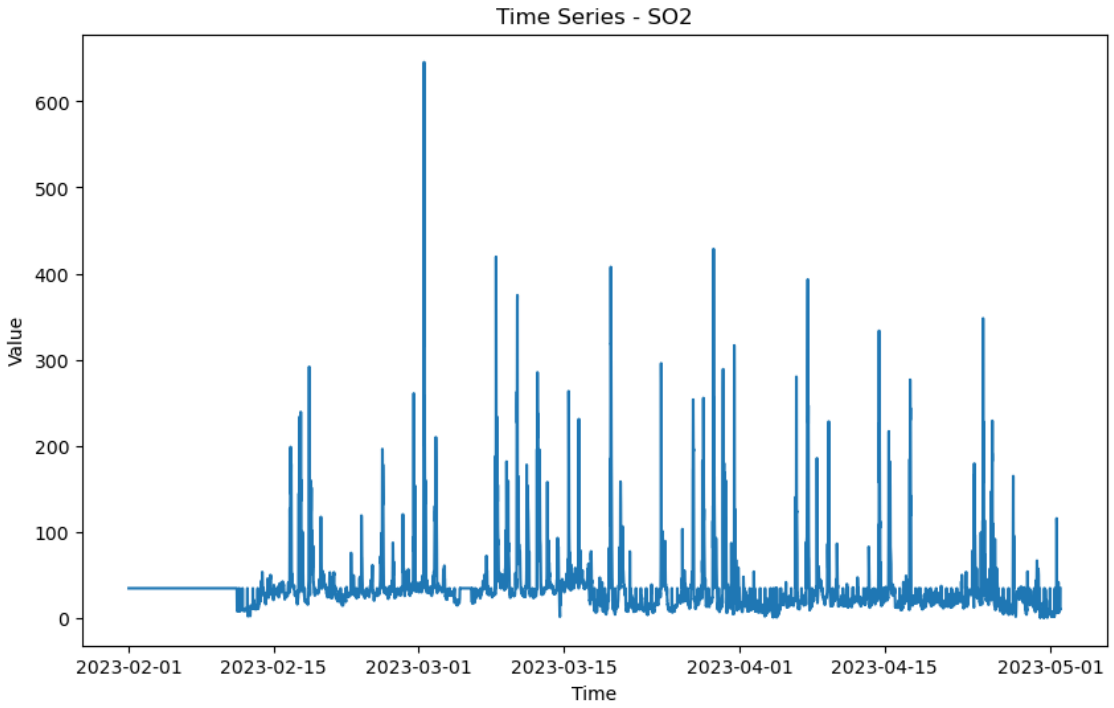
```
Column: PM10
ADF Statistic: -9.171195417168066
p-value: 2.3858111649750564e-15
The time series is stationary (reject the null hypothesis).
```



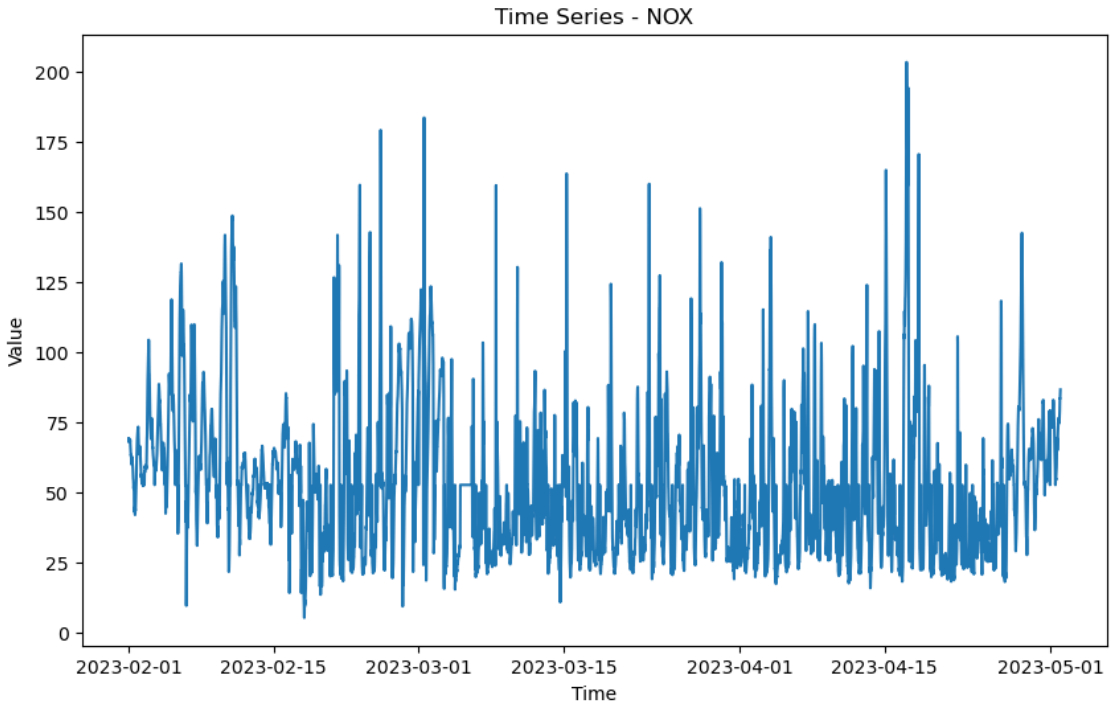

```
Column: PM2.5
ADF Statistic: -11.367904851617654
p-value: 9.130110293524146e-21
The time series is stationary (reject the null hypothesis).
```




```
Column: SO2
ADF Statistic: -18.460262821835652
p-value: 2.1472698712602428e-30
The time series is stationary (reject the null hypothesis).
```



```
Column: NOX
ADF Statistic: -13.666213297307369
p-value: 1.4890109085185823e-25
The time series is stationary (reject the null hypothesis).
```

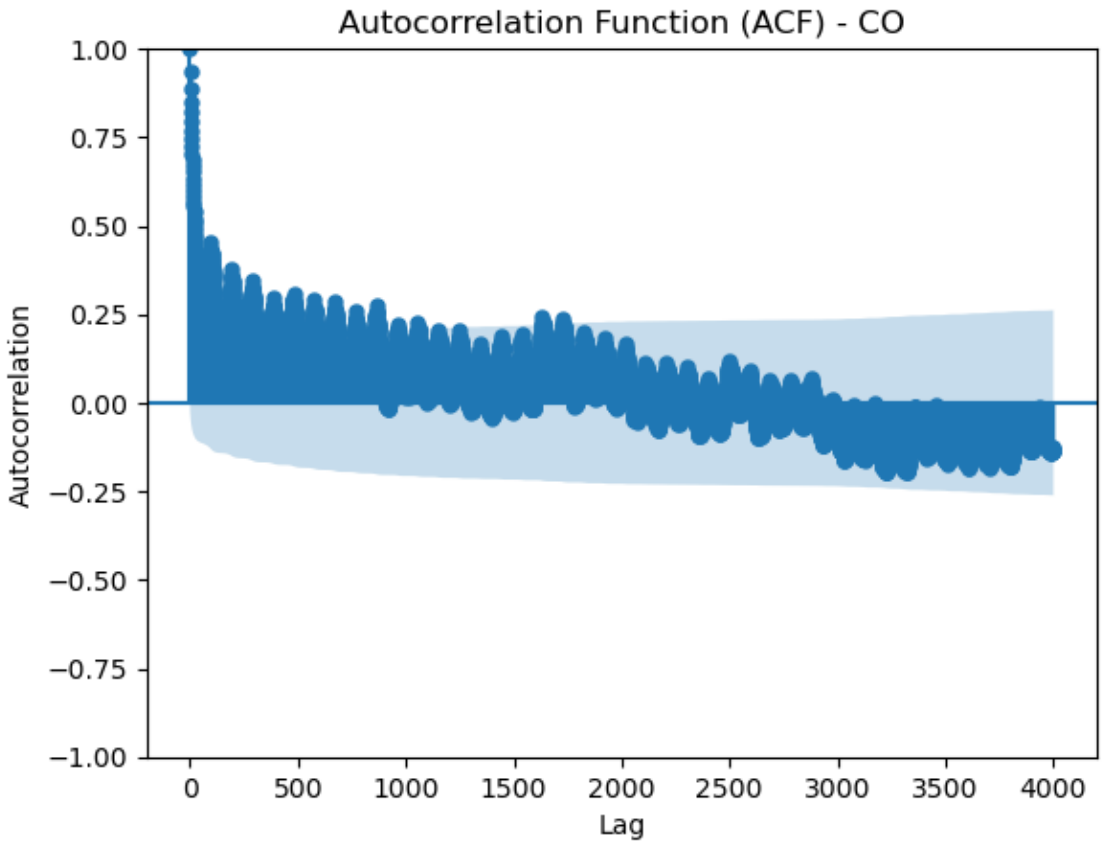



now we have to check ARIMA(p,d,q) finding p,d,q values by analysing ACF and PACF versue lagK

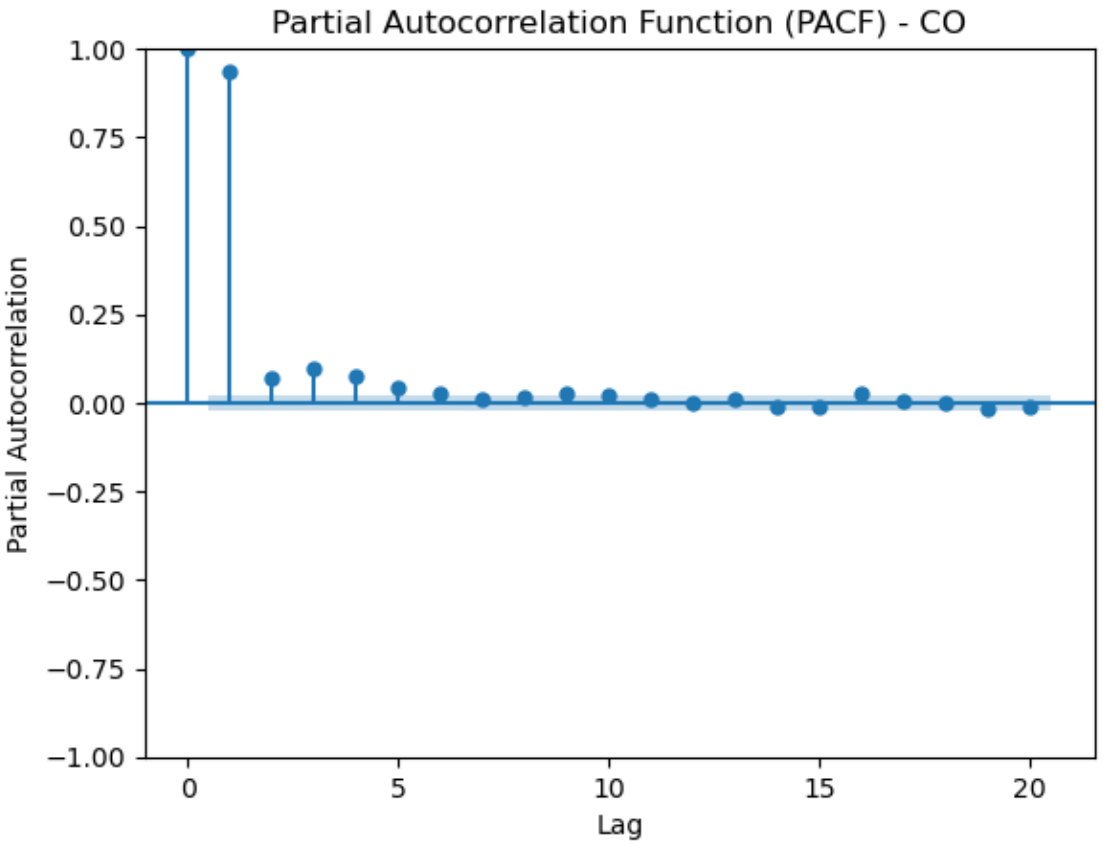
[463]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

columns_of_interest = ['CO', 'NO2', 'Ozone', 'PM10', 'PM2.5', 'SO2', 'NOX']

for column in columns_of_interest:
    plt.figure(figsize=(12, 4))
    plot_acf(df[column], lags=4000, alpha=0.05)
    plt.xlabel('Lag')
    plt.ylabel('Autocorrelation')
    plt.title(f'Autocorrelation Function (ACF) - {column}')
    plt.show()
```

```
    # Plot PACF
    plt.figure(figsize=(12, 4))
    plot_pacf(df[column], lags=20, alpha=0.05, method='ols')
    plt.xlabel('Lag')
    plt.ylabel('Partial Autocorrelation')
    plt.title(f'Partial Autocorrelation Function (PACF) - {column}')
    plt.show()
```

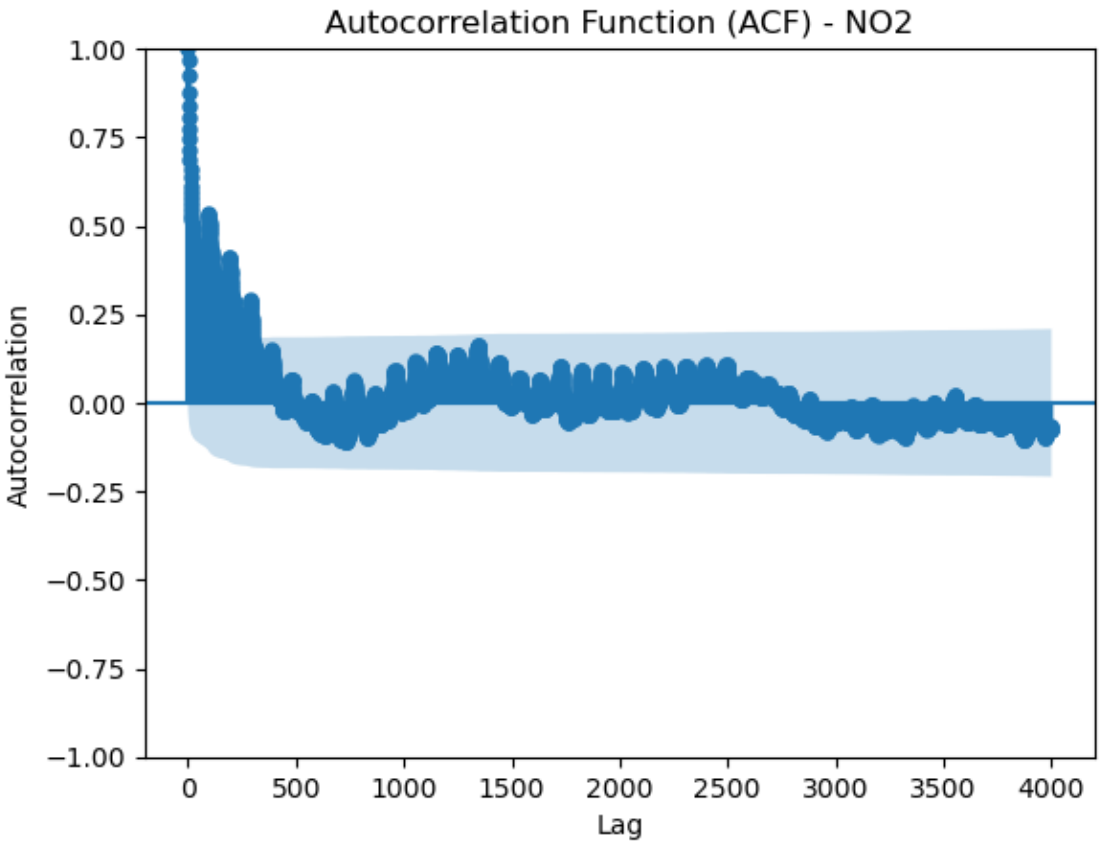
```
<Figure size 1200x400 with 0 Axes>
```



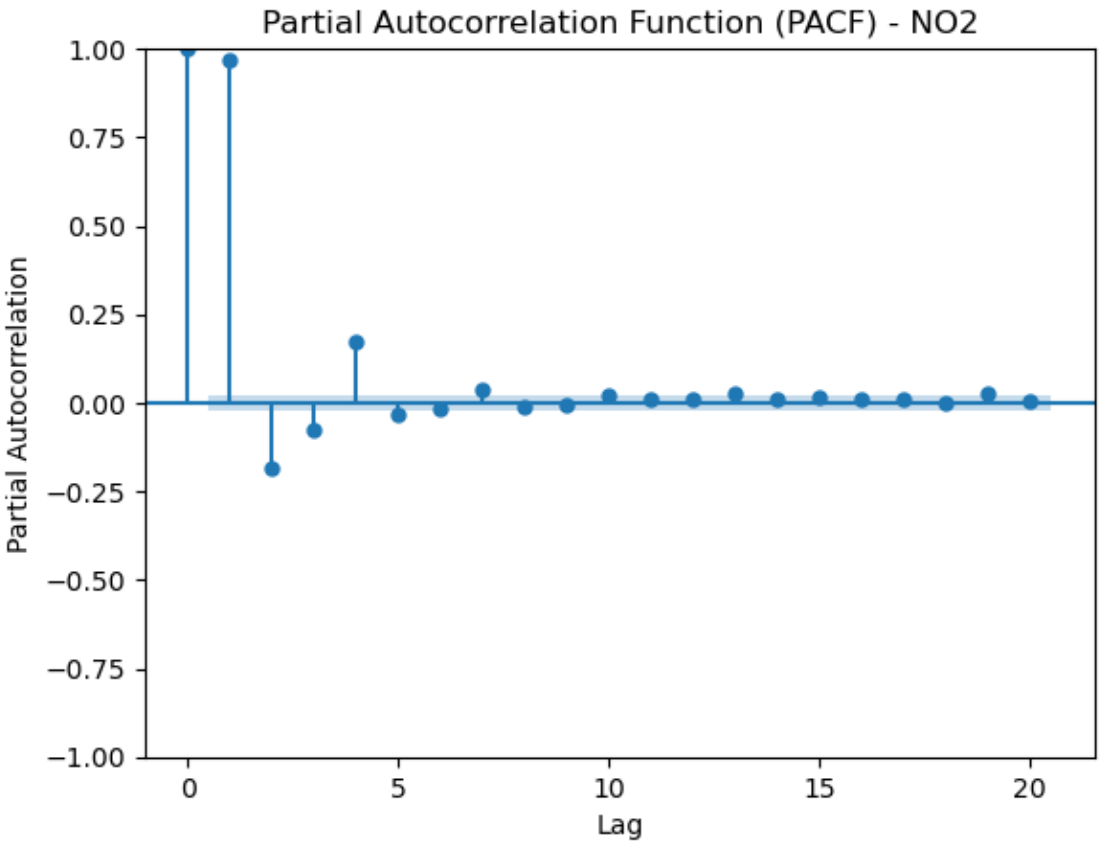

```
<Figure size 1200x400 with 0 Axes>
```



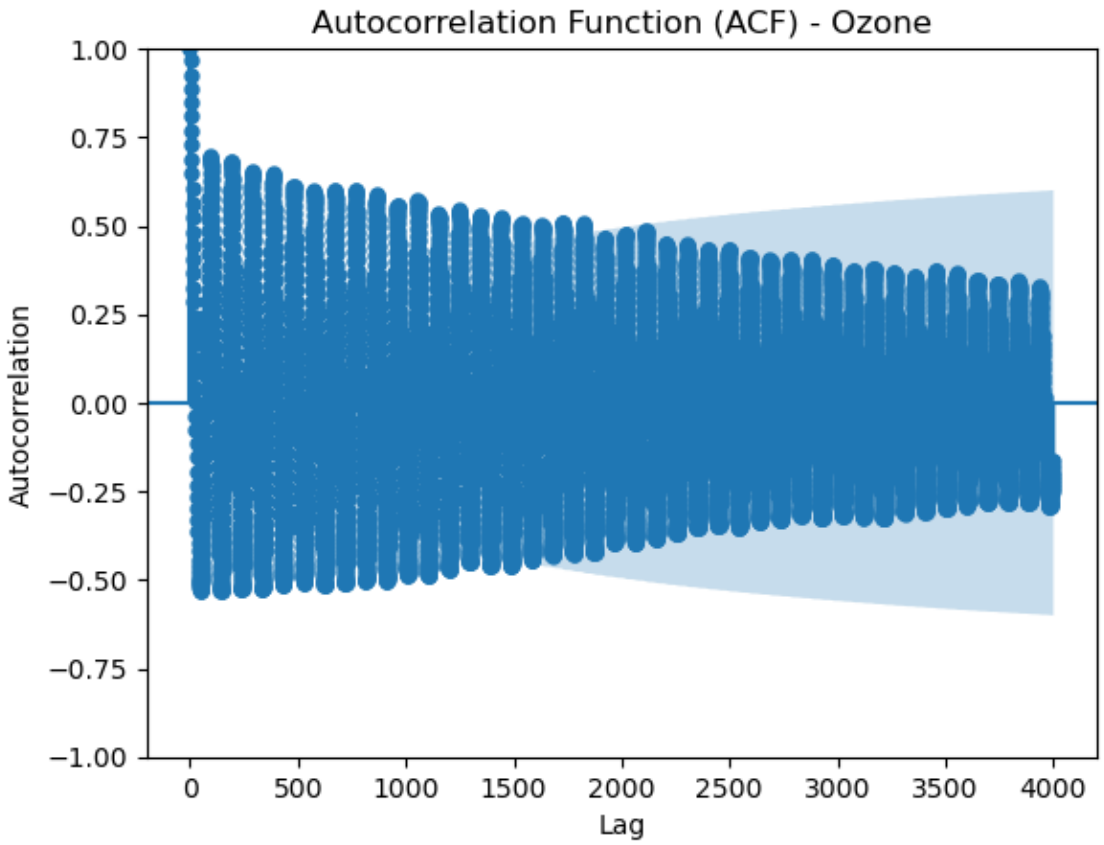

```
<Figure size 1200x400 with 0 Axes>
```



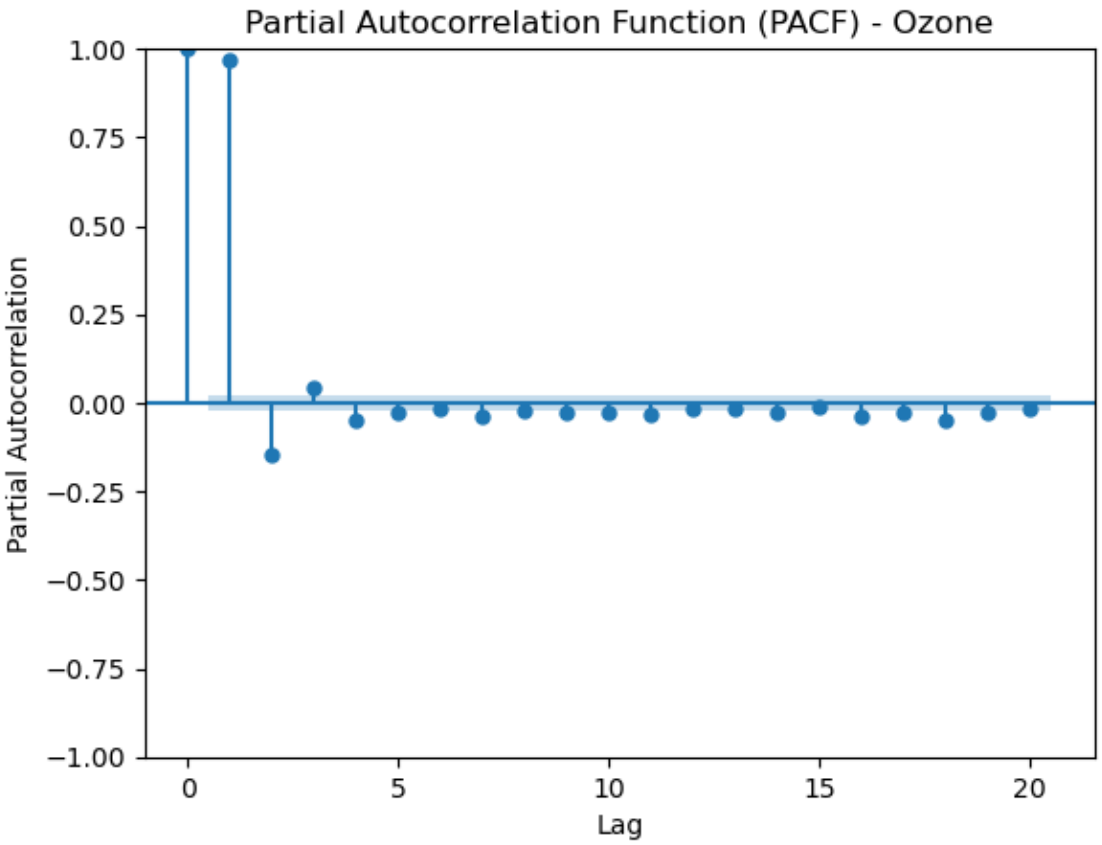

```
<Figure size 1200x400 with 0 Axes>
```



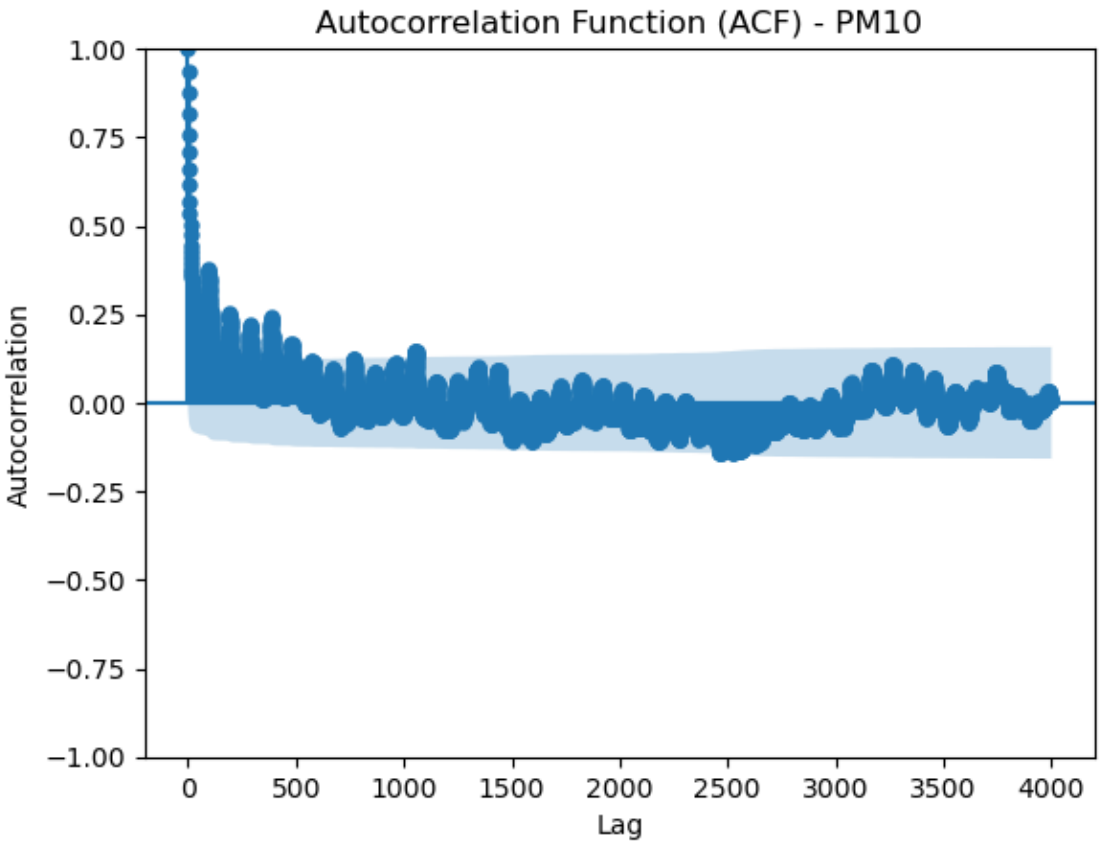

```
<Figure size 1200x400 with 0 Axes>
```



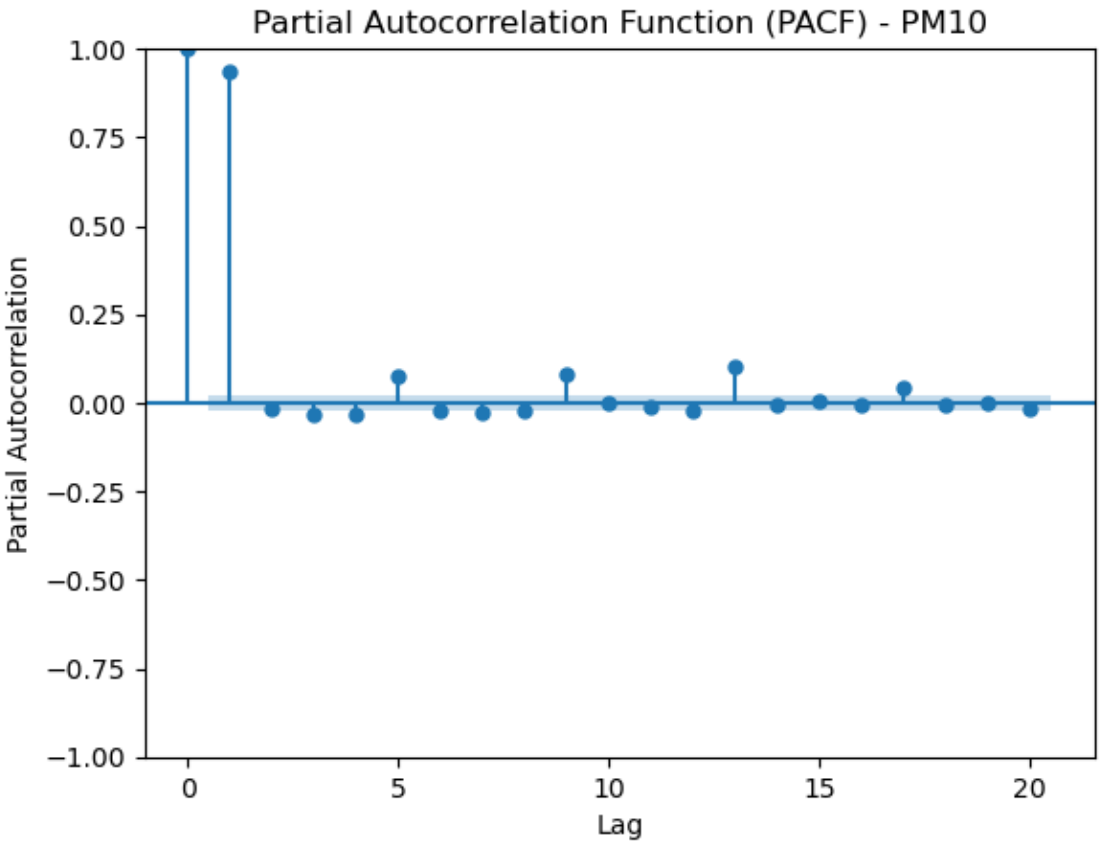

```
<Figure size 1200x400 with 0 Axes>
```



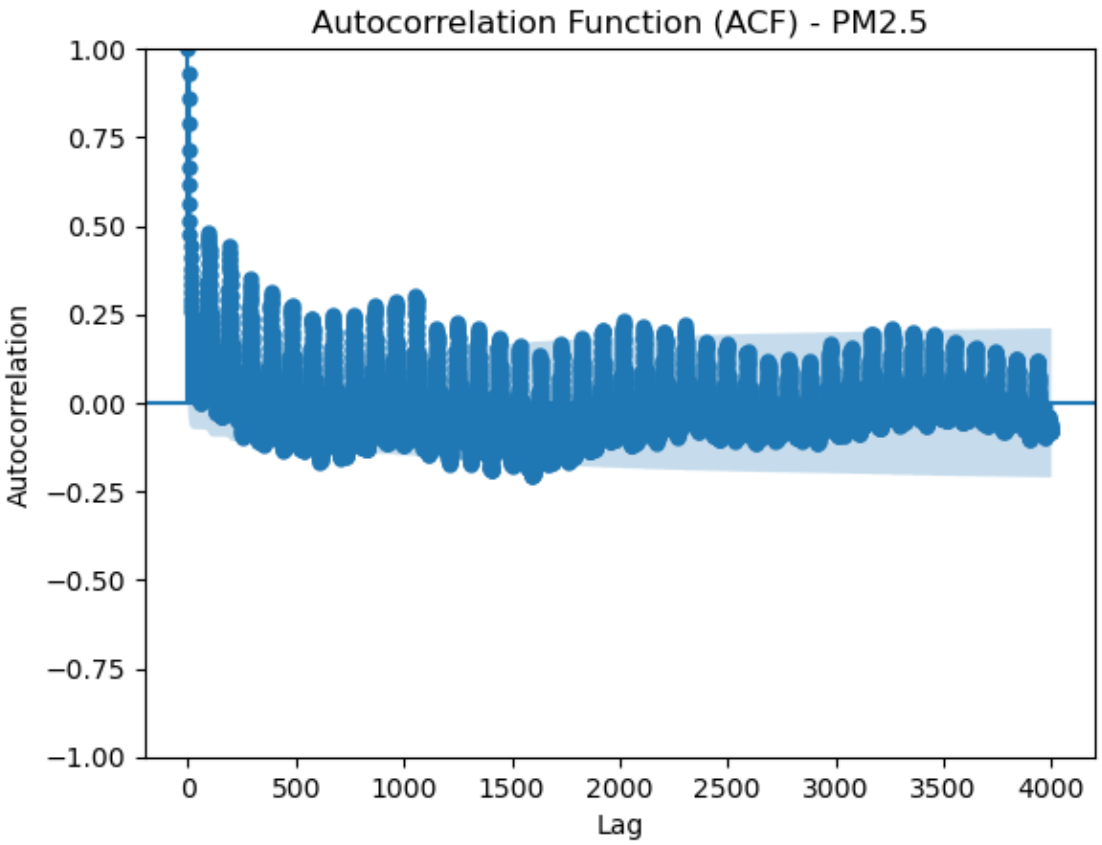

```
<Figure size 1200x400 with 0 Axes>
```



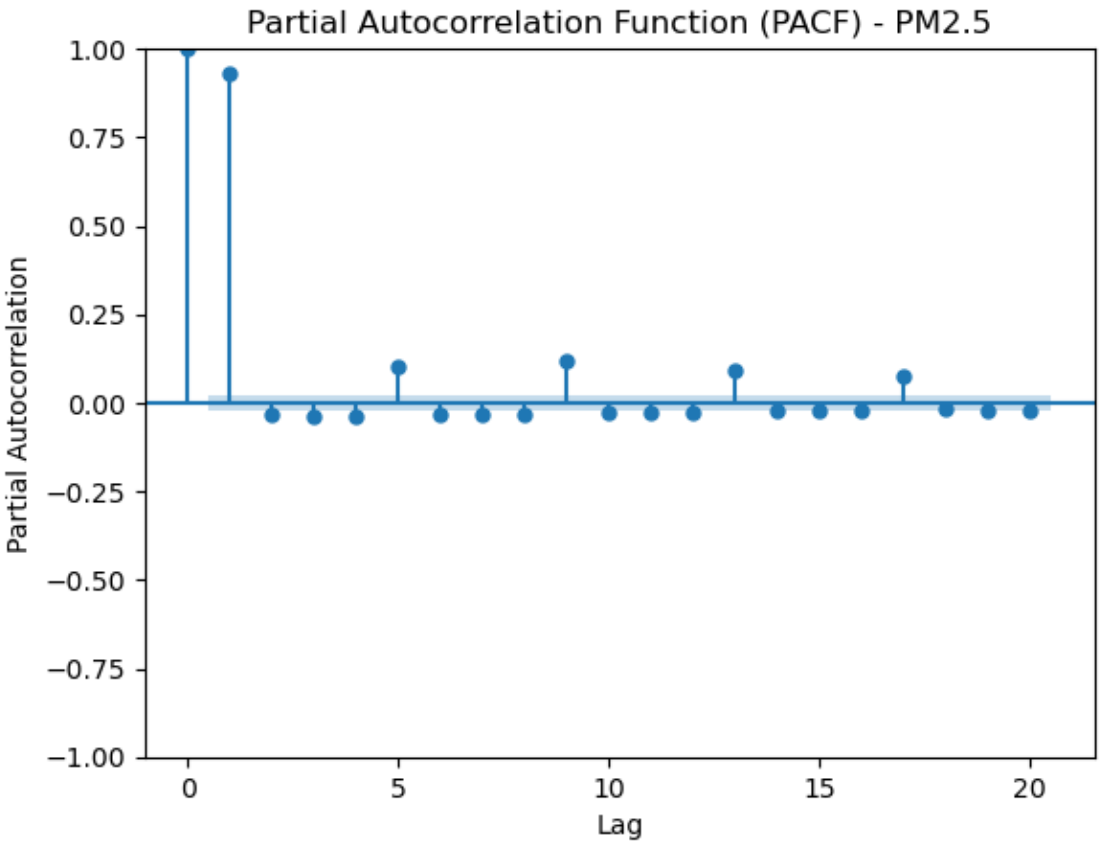

<Figure size 1200x400 with 0 Axes>



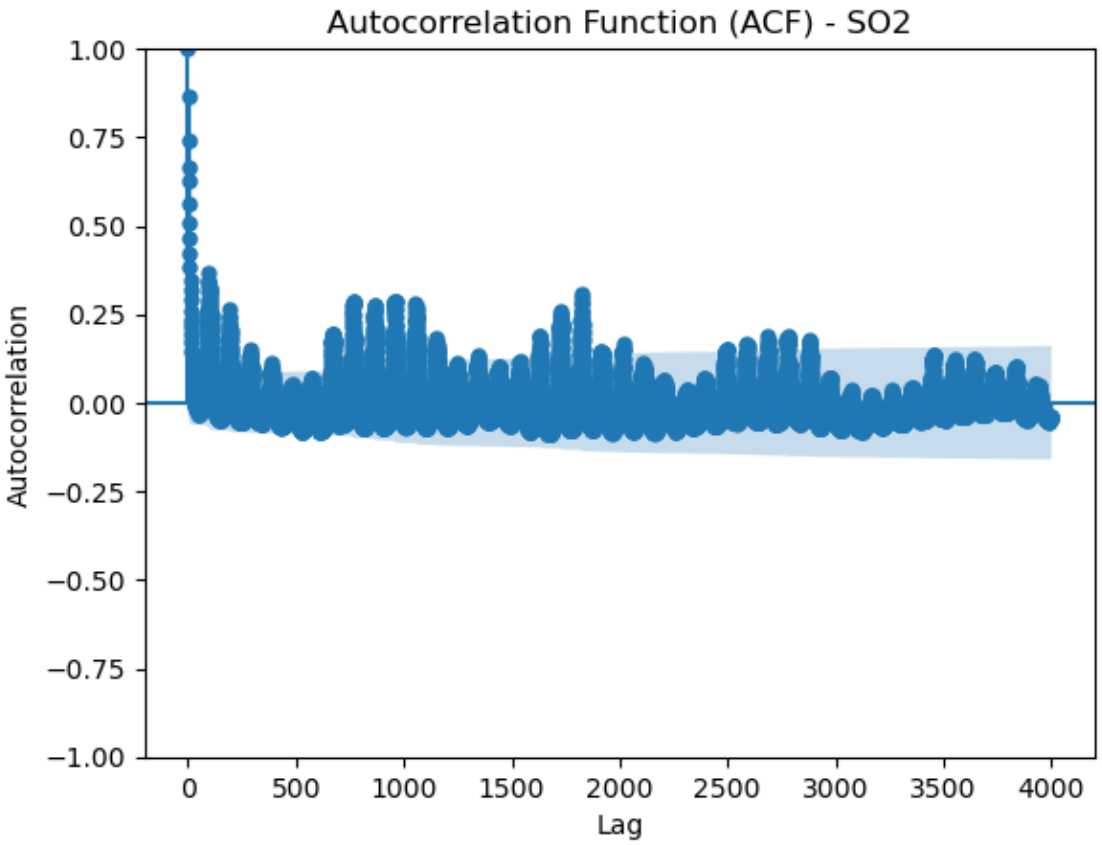

```
<Figure size 1200x400 with 0 Axes>
```



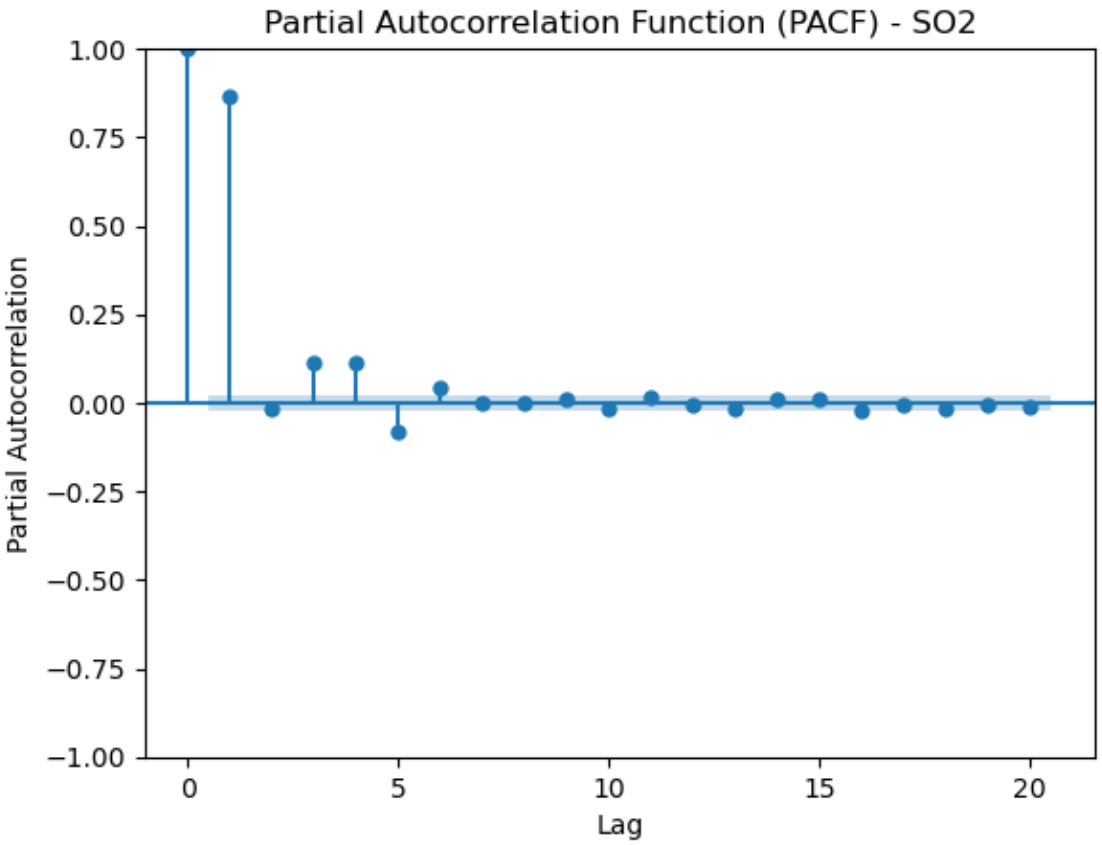

<Figure size 1200x400 with 0 Axes>



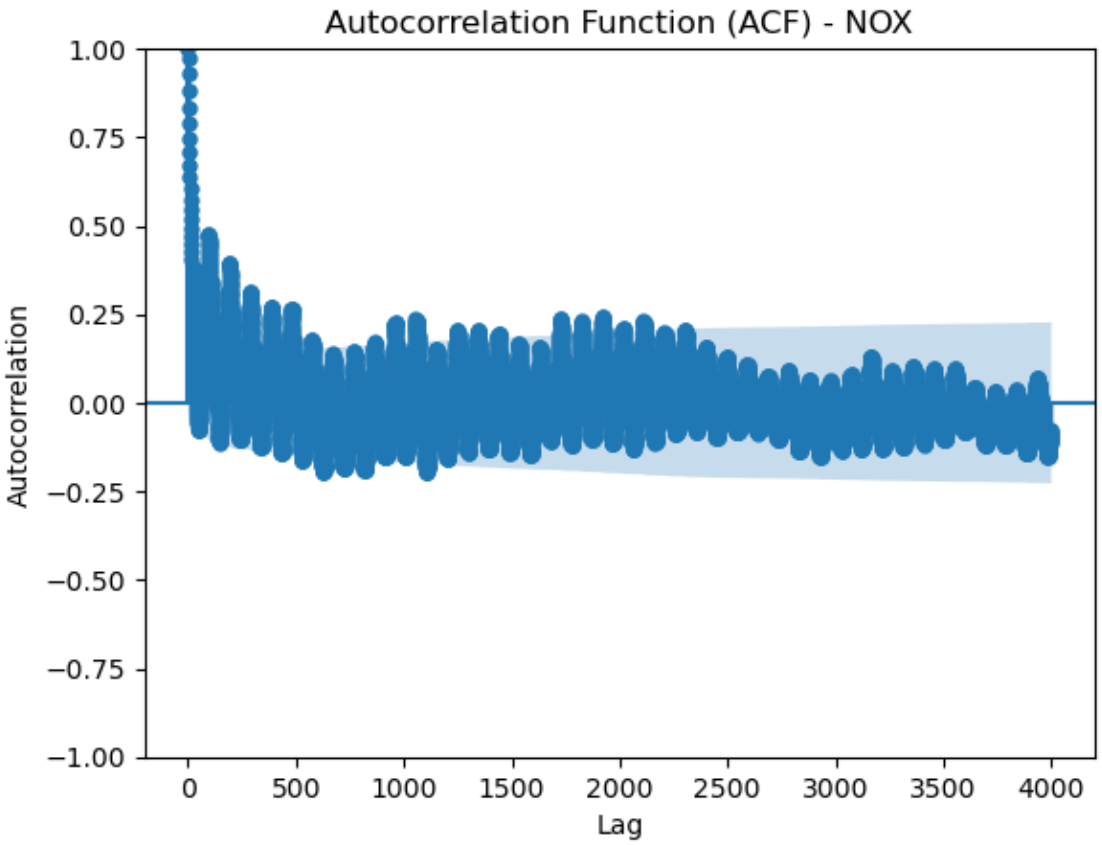

```
<Figure size 1200x400 with 0 Axes>
```



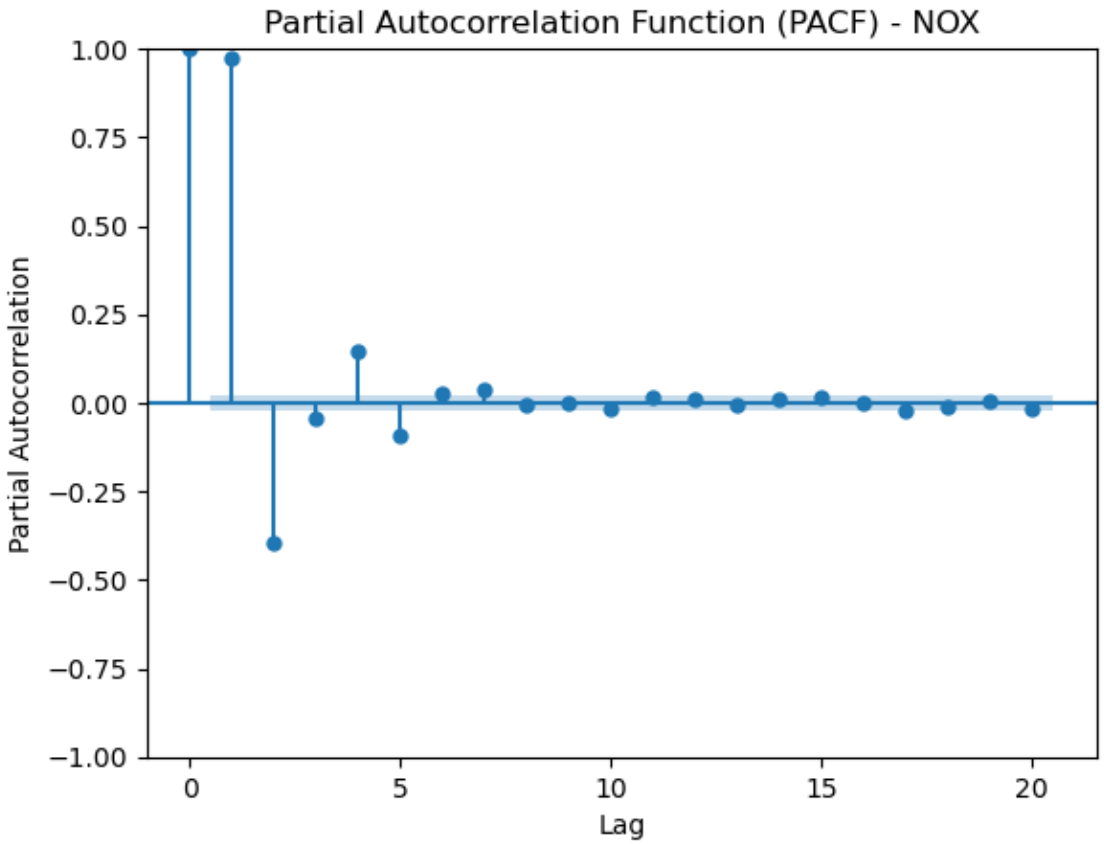

<Figure size 1200x400 with 0 Axes>




<Figure size 1200x400 with 0 Axes>



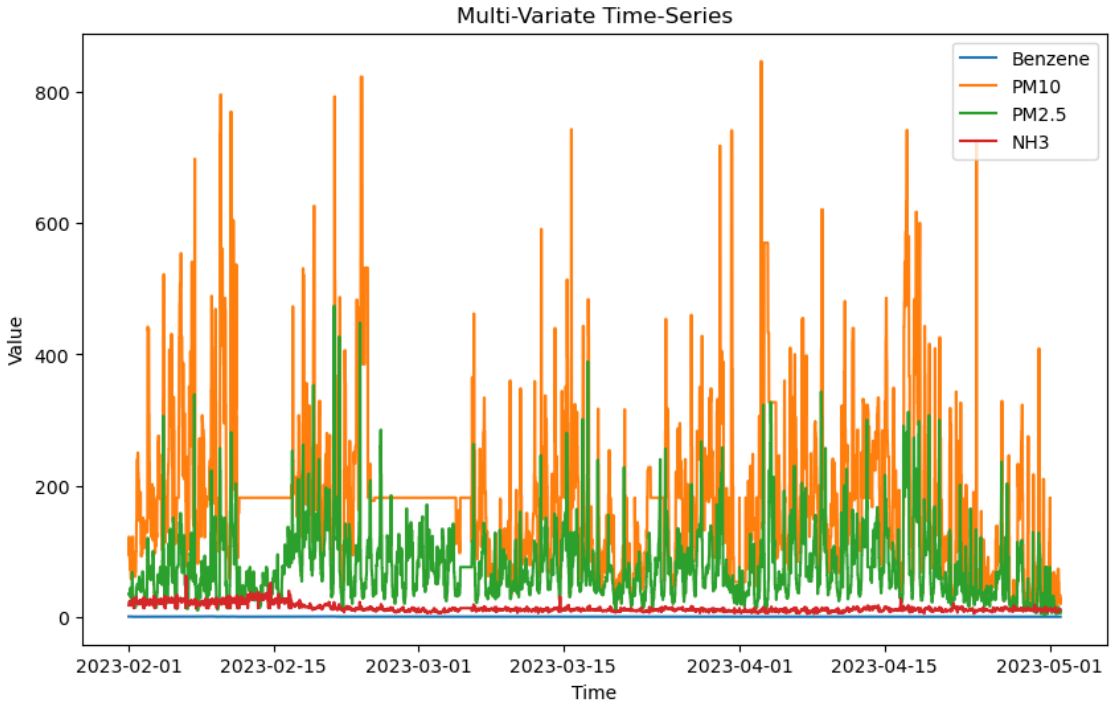

<Figure size 1200x400 with 0 Axes>




As you can see there is clearly p=1 and q=0

we can ignore benzene and NH3 as we can see it has very less amount as compare to other pollutants

[437]:
```
columns_of_interest = ['Benzene','PM10', 'PM2.5','NH3']
plt.figure(figsize=(10, 6))


for col in columns_of_interest:
    plt.plot(df['end'], df[col], label=col)
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Multi-Variate Time-Series')
plt.legend()
plt.show()
```

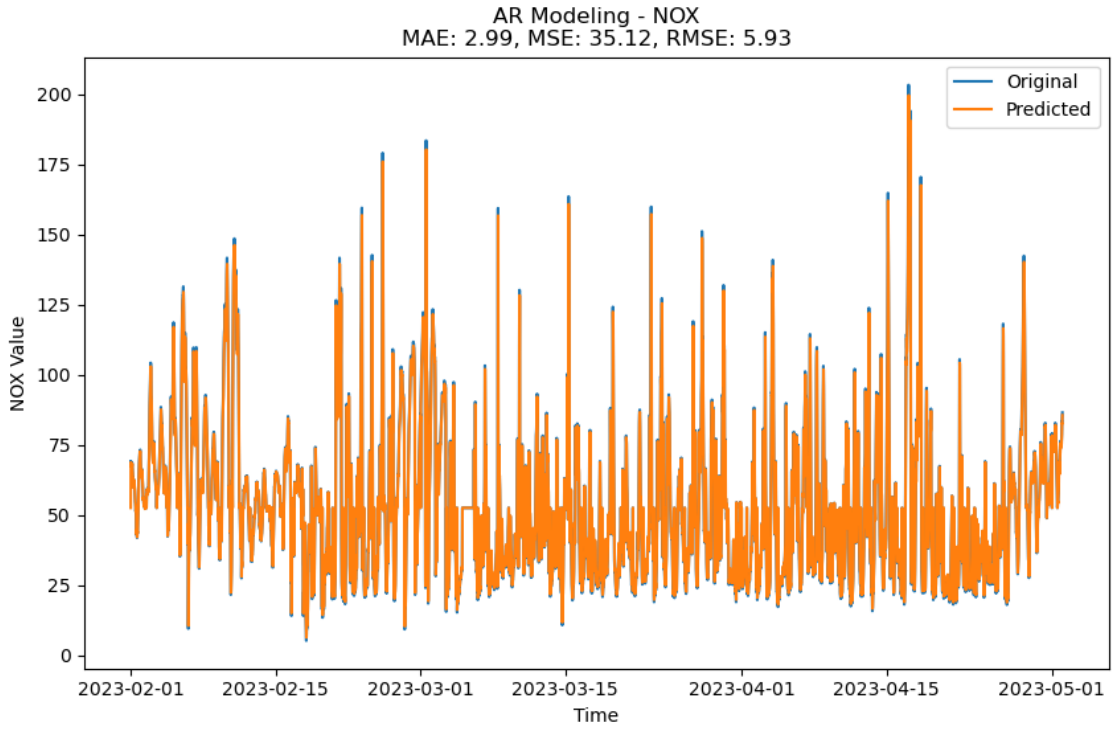



Below ARIMA model applied on major air pollutant

[438]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
columns_of_interest = [ 'NOX', 'SO2', 'PM10', 'PM2.5','CO', 'Ozone', 'NO']
for col in columns_of_interest:
    data = df[col]

    # Fit the AR model
    model = ARIMA(data, order=(1,0 , 0))  # Set AR order to 1
    model_fit = model.fit()

    # Make predictions using the fitted model
    predictions = model_fit.predict(start=0, end=len(data)-1)

    # Calculate evaluation metrics
    mae = mean_absolute_error(data, predictions)
    mse = mean_squared_error(data, predictions)
    rmse = np.sqrt(mse)

    # Plot the original data and predicted values
    plt.figure(figsize=(10, 6))
    plt.plot(df['end'], data, label='Original')
    plt.plot(df['end'], predictions, label='Predicted')
    plt.xlabel('Time')
```

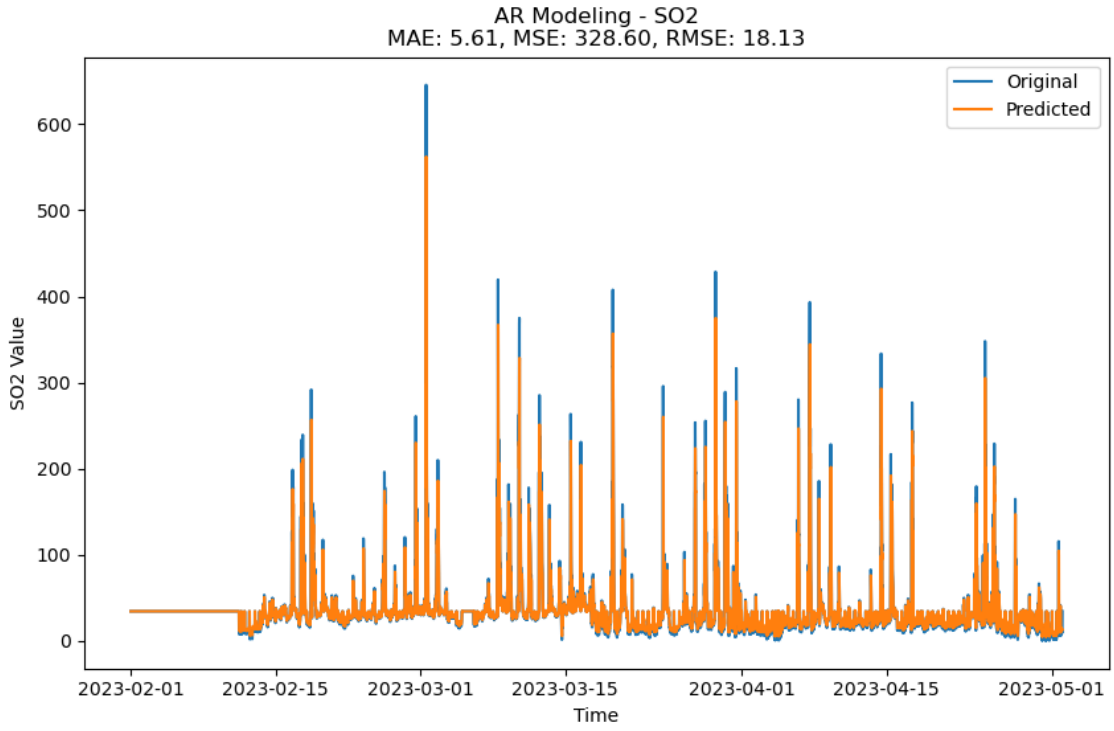
```
    plt.ylabel(f'{col} Value')
    plt.title(f'AR Modeling - {col}\nMAE: {mae:.2f}, MSE: {mse:.2f}, RMSE:␣
↪{rmse:.2f}')
    plt.legend()
    plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```

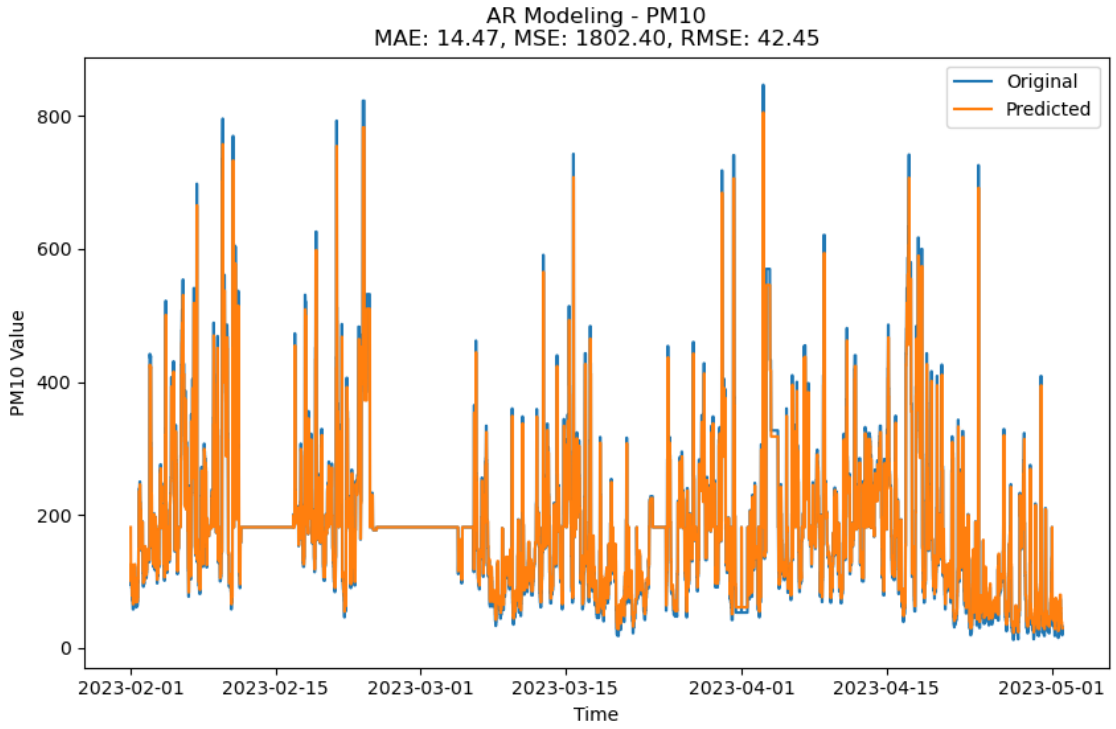



```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
```

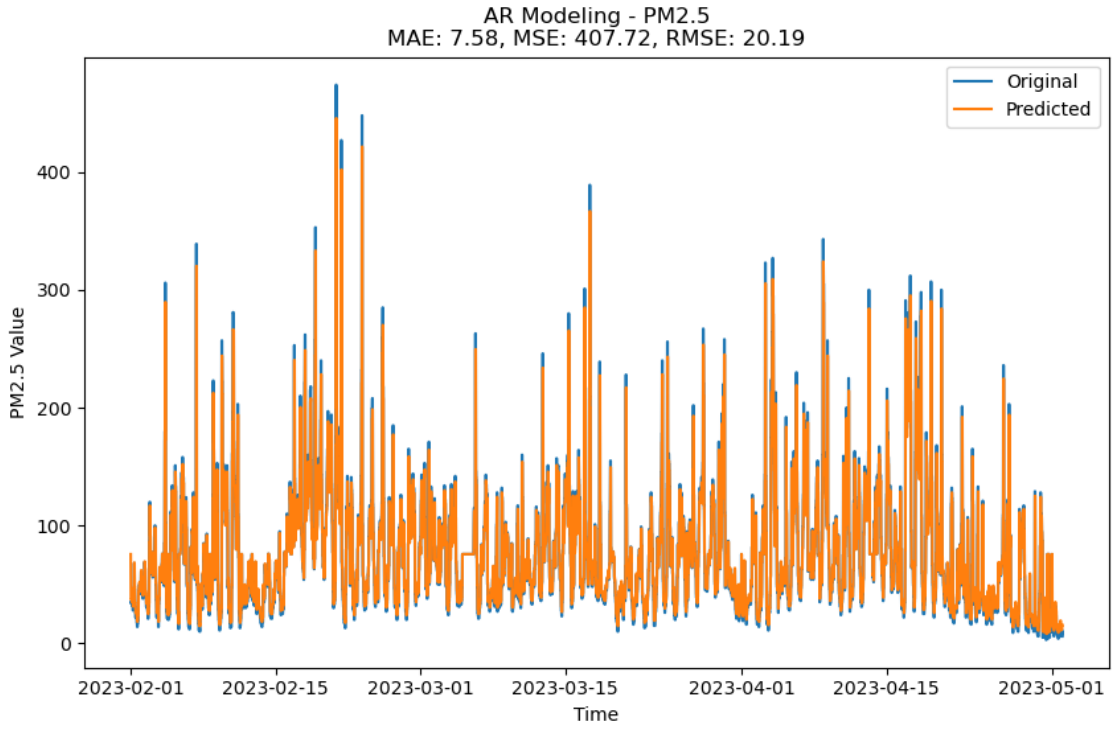
```
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```



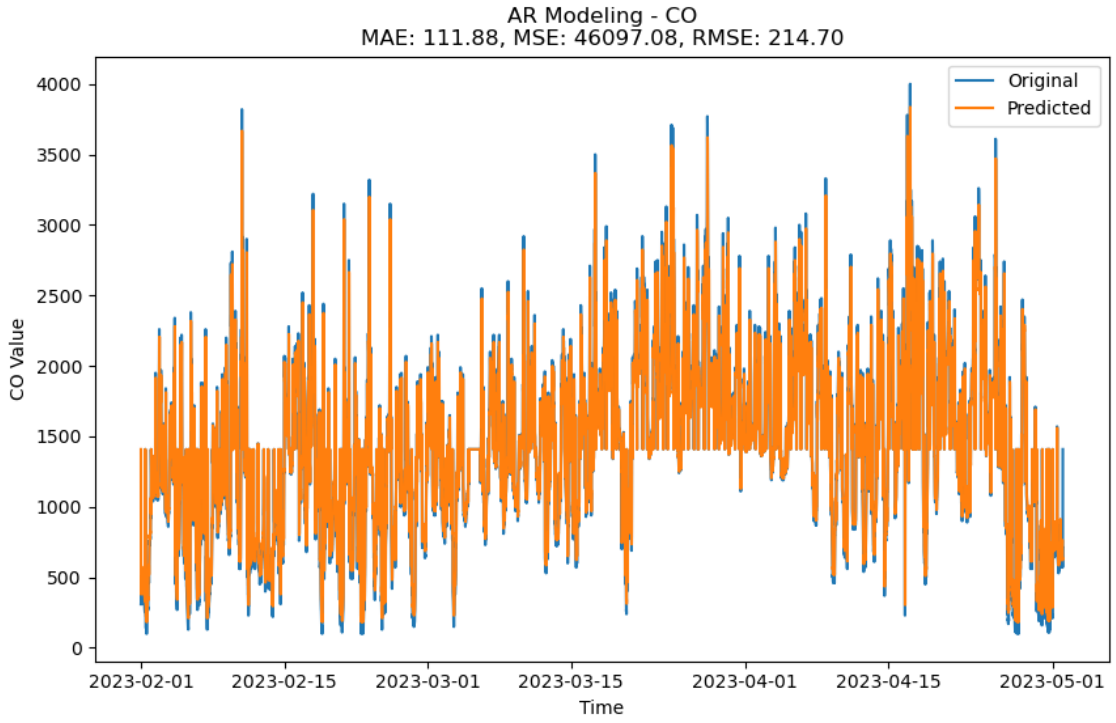

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```



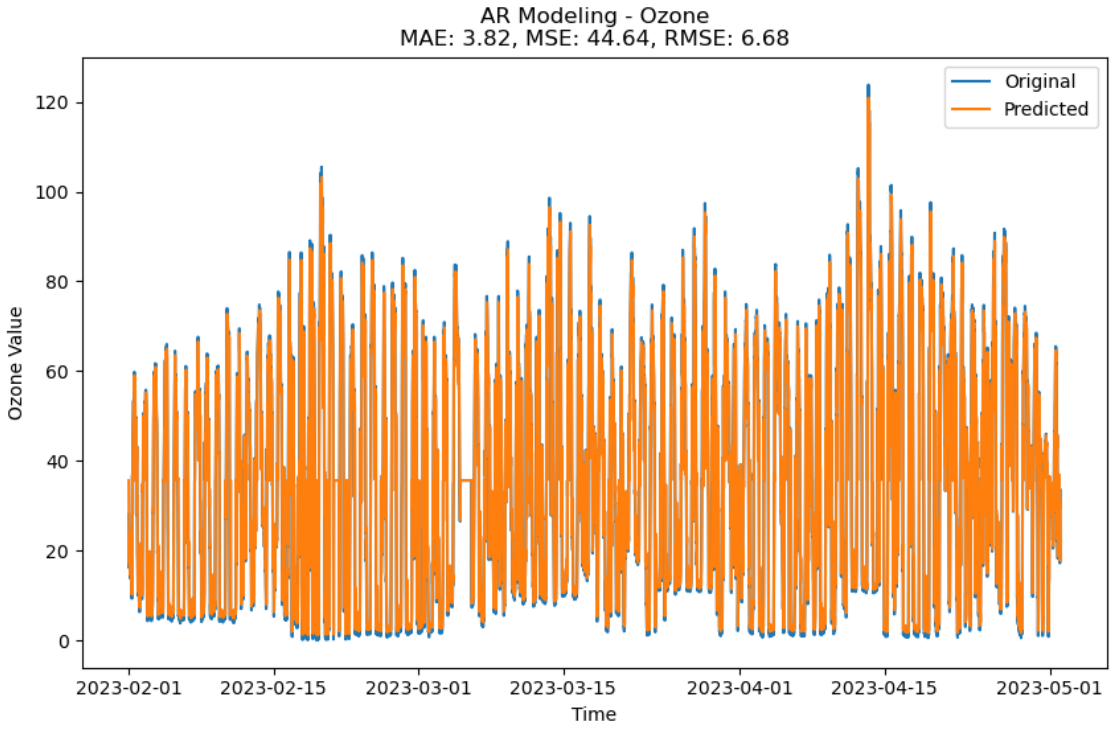

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```



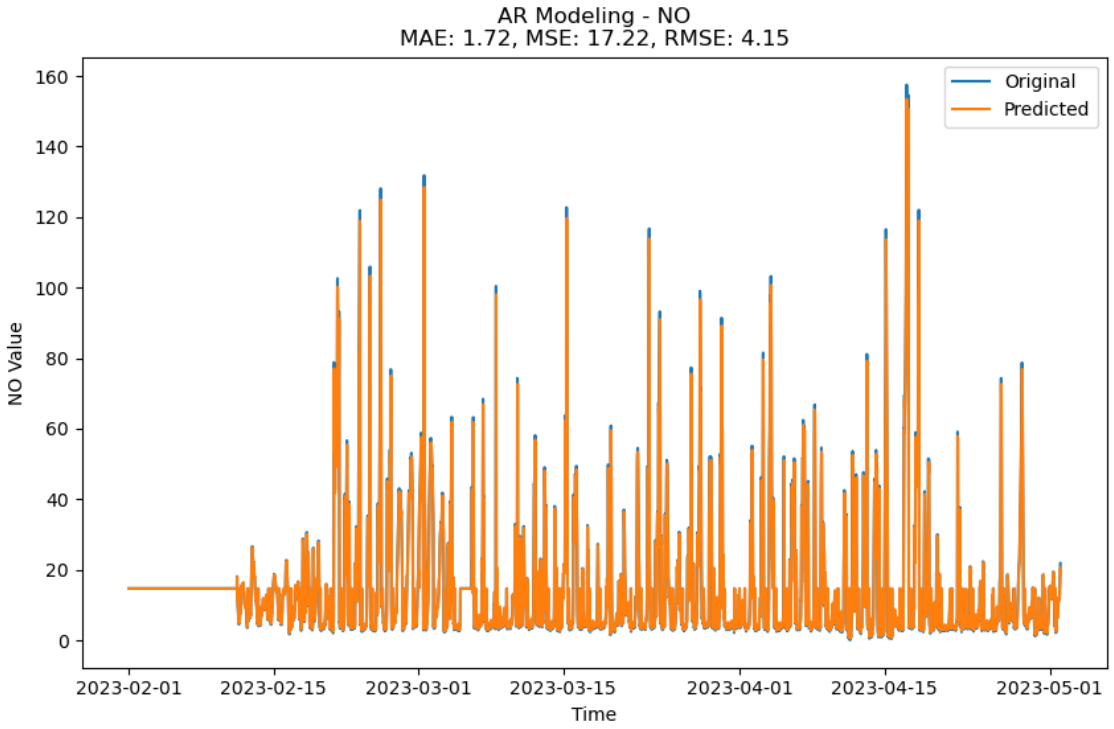

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```




```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```




```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```




as you can see we can do better than the just filling missing values with linear interpolation we can appply linear or cubic interpolation technique

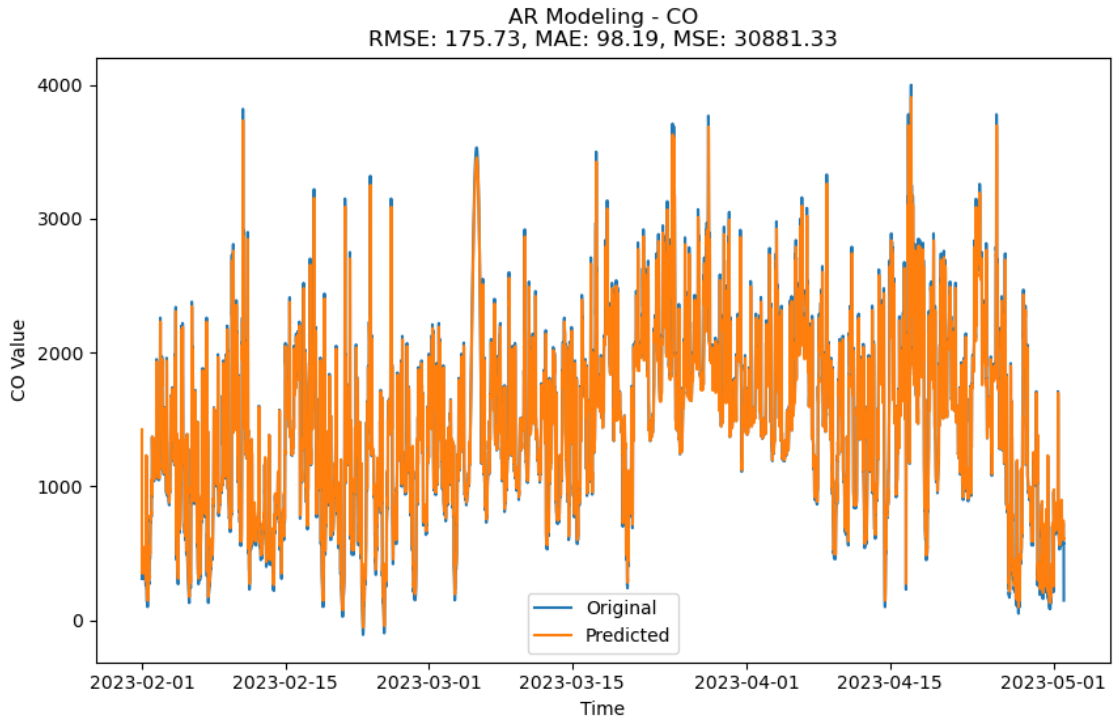
we can do better than this with the help of spline interpolation

[439]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from scipy.interpolate import CubicSpline

# Read the CSV file
df = pd.read_csv('Open pit blasting 01-02-2023 000000 To 01-05-2023 235959.
 ↪csv', na_values='NA')

# Preprocess the data and handle missing values
df['s'] = pd.to_datetime(df['From'], format='%Y-%m-%d %H:%M:%S',␣
 ↪errors='coerce')
df['end'] = pd.to_datetime(df['To (Interval: 15M)'], format='%Y-%m-%d %H:%M:
 ↪%S', errors='coerce')
df = df.drop(['From', '#', 'To (Interval: 15M)'], axis=1)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM10 (µg/m3)':␣
 ↪'PM10'}, inplace=True)
```

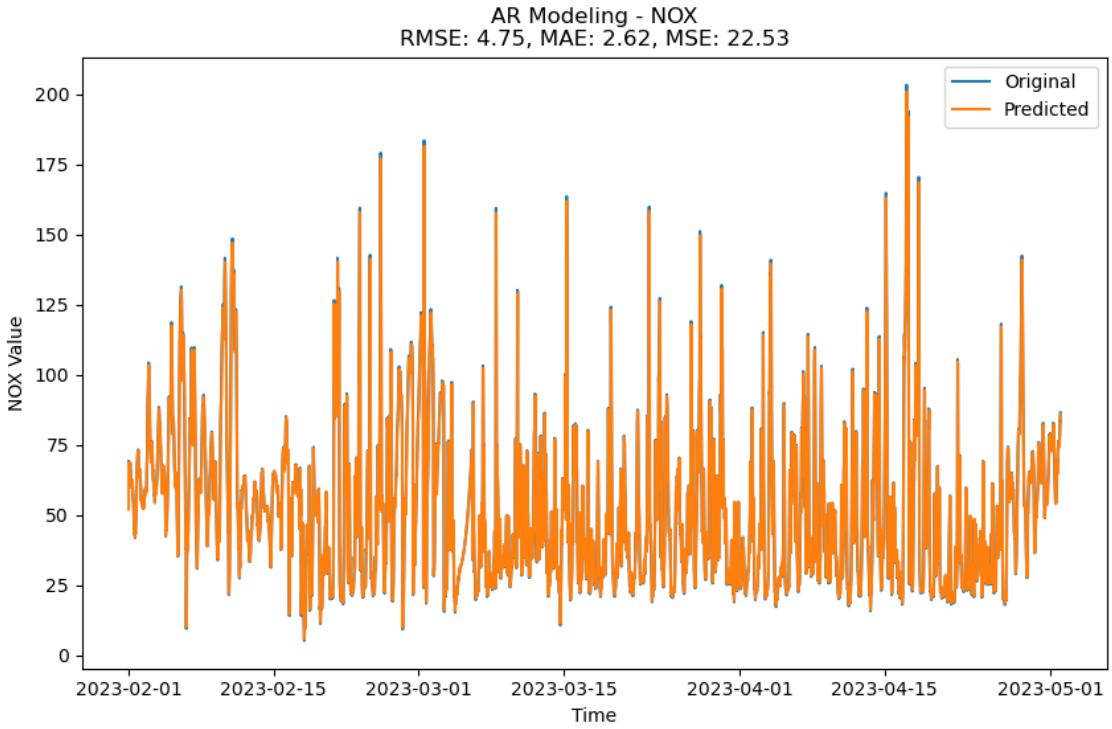
```
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM2.5 (µg/m3)':␣
 ↪'PM2.5'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO2 (µg/m3)':␣
 ↪'NO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO (µg/m3)':␣
 ↪'NO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  CO (mg/m3)':␣
 ↪'CO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  SO2 (µg/m3)':␣
 ↪'SO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NH3 (µg/m3)':␣
 ↪'NH3'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Ozone (µg/m3)':␣
 ↪'Ozone'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Benzene (µg/m3)':␣
 ↪'Benzene'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NOX (ppb)':␣
 ↪'NOX'}, inplace=True)
df.set_index('s', inplace=True)
df = df.iloc[:-3]
columns_of_interest = ['CO', 'NOX', 'SO2', 'PM10', 'PM2.5', 'Ozone', 'NO']
df['CO']=df['CO']*1000
df['NOX'] = df['NOX'] * 1.23
# Handle missing values using spline interpolation
for col in columns_of_interest:
    df[col].interpolate(method='spline', order=3, inplace=True)
    df[col].fillna(method='ffill', inplace=True)
    df[col].fillna(method='bfill', inplace=True)

# Perform ARIMA modeling and plot the results
for col in columns_of_interest:
    data = df[col]

    # Fit the ARIMA model
    model = ARIMA(data, order=(1, 0, 0))  # Set AR order to 2
    model_fit = model.fit()

    # Make predictions
    predictions = model_fit.predict(start=0, end=len(data)-1)

    # Calculate accuracy metrics
    residuals = predictions - data
    rmse = np.sqrt(np.mean(residuals**2))
    mse = np.mean(residuals**2)  # Calculate MSE
    mae = np.mean(np.abs(residuals))
```

```
    # Plot the original data and predicted values
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, data, label='Original')
    plt.plot(df.index, predictions, label='Predicted')
    plt.xlabel('Time')
    plt.ylabel(f'{col} Value')
    plt.title(f'AR Modeling - {col}\nRMSE: {rmse:.2f}, MAE: {mae:.2f}, MSE:␣
↪{mse:.2f}')
    plt.legend()
    plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```

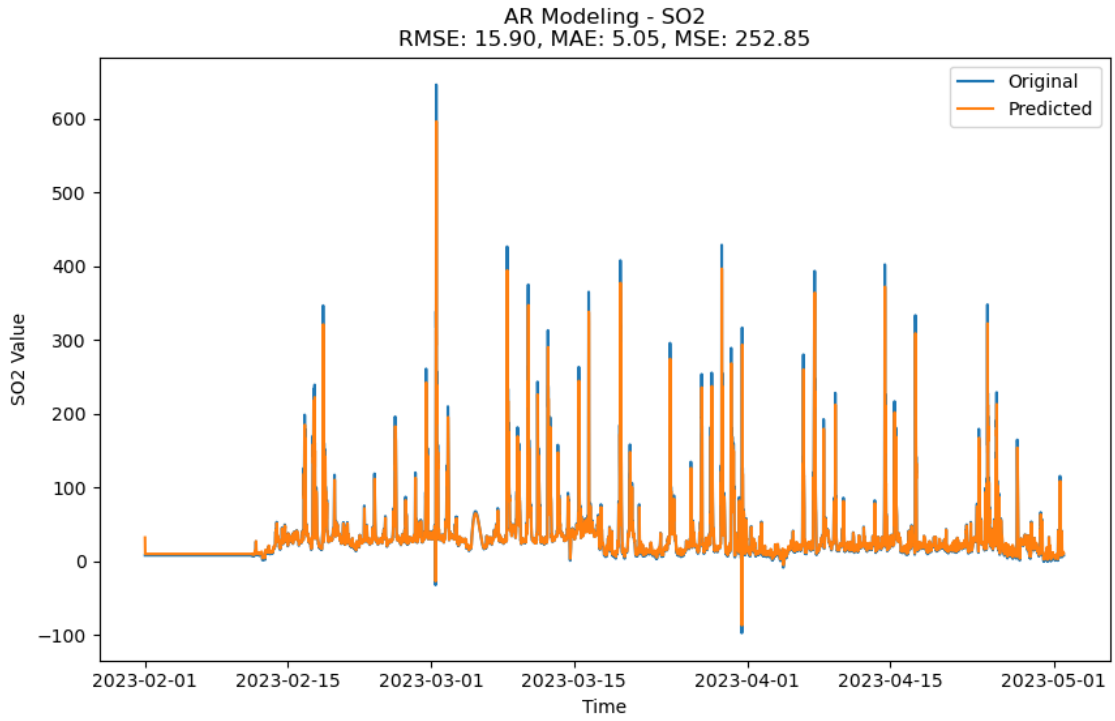


```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```

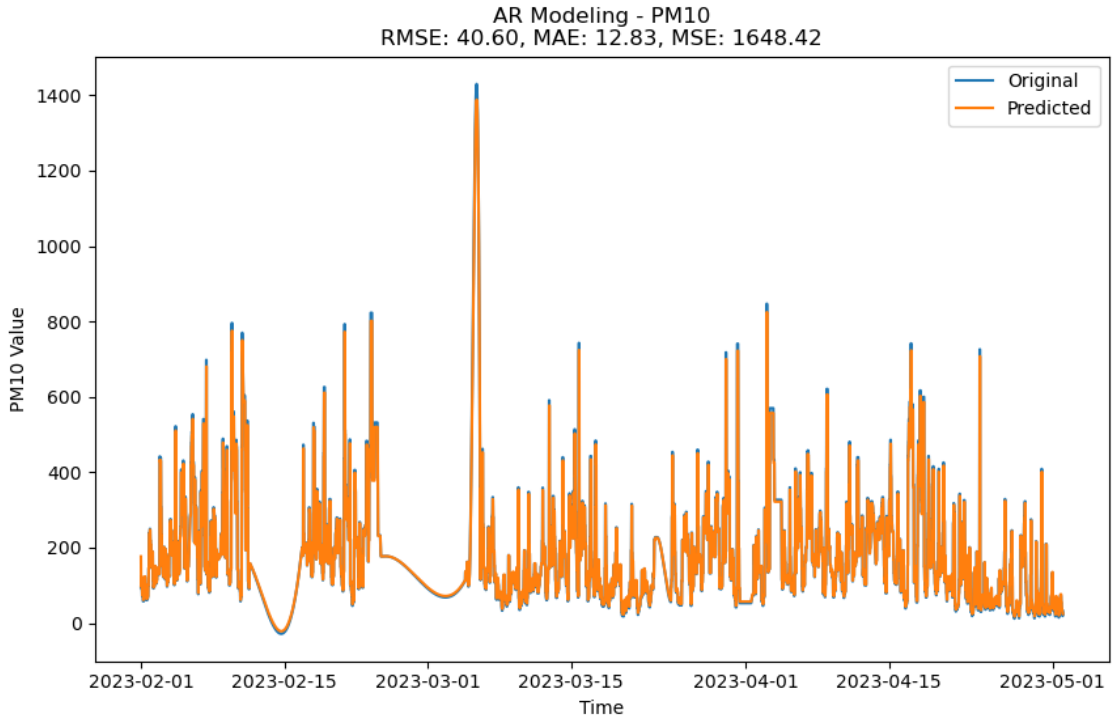



```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
```

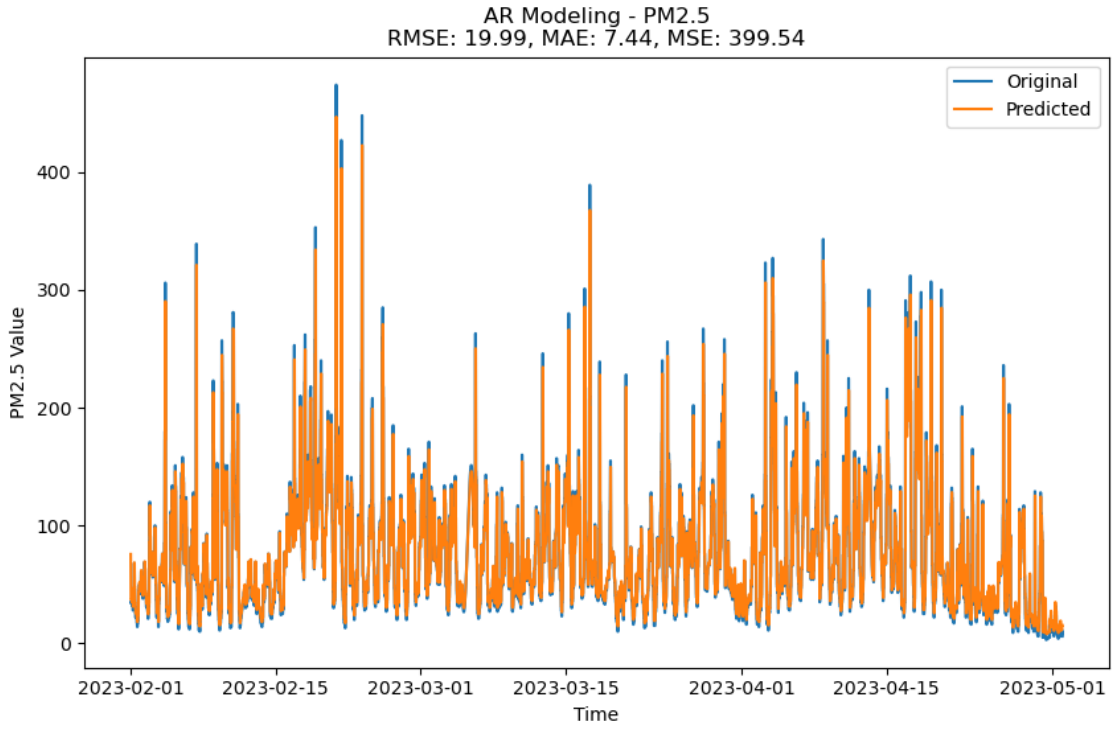
```
  self._init_dates(dates, freq)
```



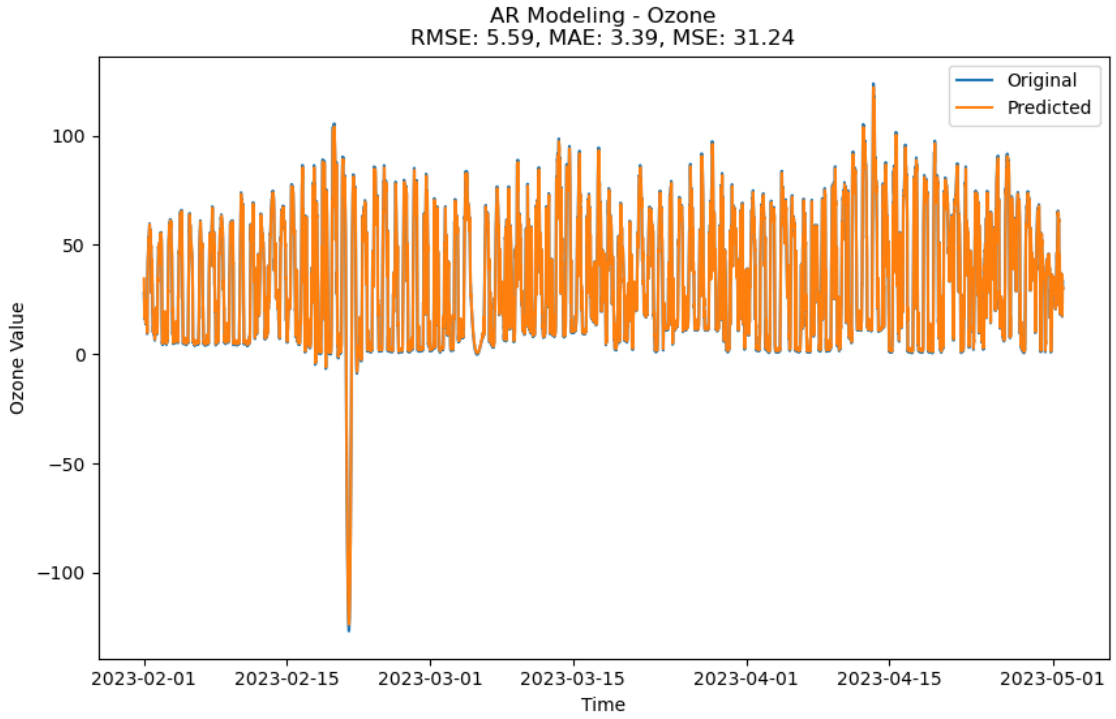

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```



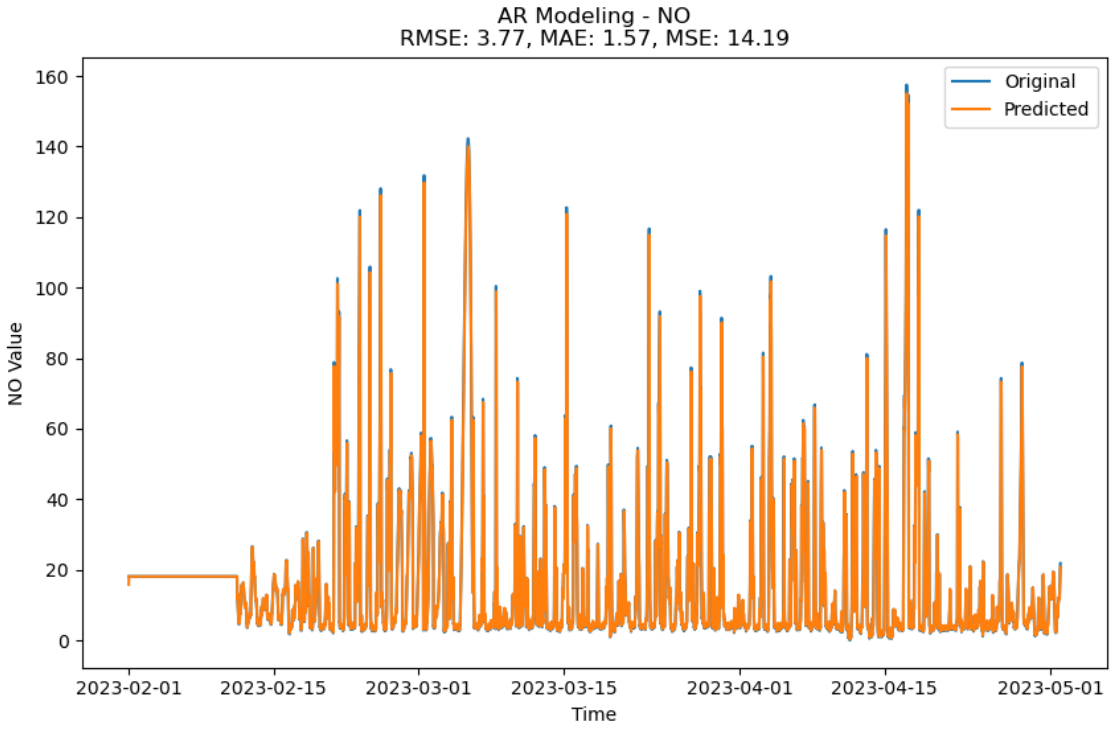

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```




```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```




```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```



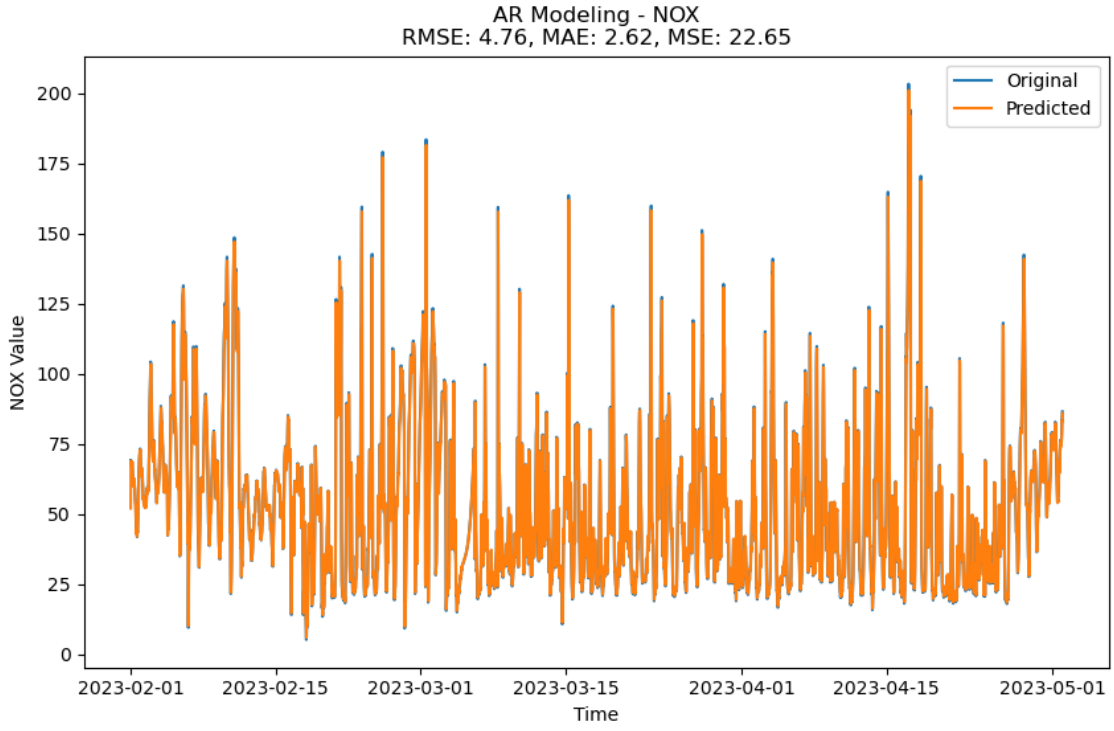

we can do better than this with the help of cubic interpolation

[440]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
import seaborn as sns
from scipy import stats

# Read the CSV file
df = pd.read_csv('Open pit blasting 01-02-2023 000000 To 01-05-2023 235959.
 ↪csv', na_values='NA')

# Preprocess the data and handle missing values
df['s'] = pd.to_datetime(df['From'], format='%Y-%m-%d %H:%M:%S',␣
 ↪errors='coerce')
df['end'] = pd.to_datetime(df['To (Interval: 15M)'], format='%Y-%m-%d %H:%M:
 ↪%S', errors='coerce')
df = df.drop(['From', '#', 'To (Interval: 15M)'], axis=1)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM10 (µg/m3)':␣
 ↪'PM10'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM2.5 (µg/m3)':␣
 ↪'PM2.5'}, inplace=True)
```

```
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO2 (µg/m3)':␣
 ↪'NO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO (µg/m3)':␣
 ↪'NO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  CO (mg/m3)':␣
 ↪'CO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  SO2 (µg/m3)':␣
 ↪'SO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NH3 (µg/m3)':␣
 ↪'NH3'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Ozone (µg/m3)':␣
 ↪'Ozone'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Benzene (µg/m3)':␣
 ↪'Benzene'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NOX (ppb)':␣
 ↪'NOX'}, inplace=True)
df.set_index('s', inplace=True)
df = df.iloc[:-3]
columns_of_interest = [ 'NOX', 'SO2', 'PM10', 'PM2.5','CO', 'Ozone', 'NO']
df['CO'] = df['CO'] * 1000
df['NOX'] = df['NOX'] * 1.23
# Handle missing values using interpolation
for col in columns_of_interest:
    df[col].interpolate(method='cubic', inplace=True)
    df[col].fillna(method='ffill', inplace=True)
    df[col].fillna(method='bfill', inplace=True)


# Perform ARIMA modeling and plot the results
for col in columns_of_interest:
    data = df[col]

    # Fit the ARIMA model
    model = ARIMA(data, order=(1, 0, 0))  # Set AR order to 2
    model_fit = model.fit()

    # Make predictions
    predictions = model_fit.predict(start=0, end=len(data)-1)

    # Calculate accuracy metrics
    residuals = predictions - data
    rmse = np.sqrt(np.mean(residuals**2))
    mse = np.mean(residuals**2)  # Calculate MSE
    mae = np.mean(np.abs(residuals))

    # Plot the original data and predicted values
```

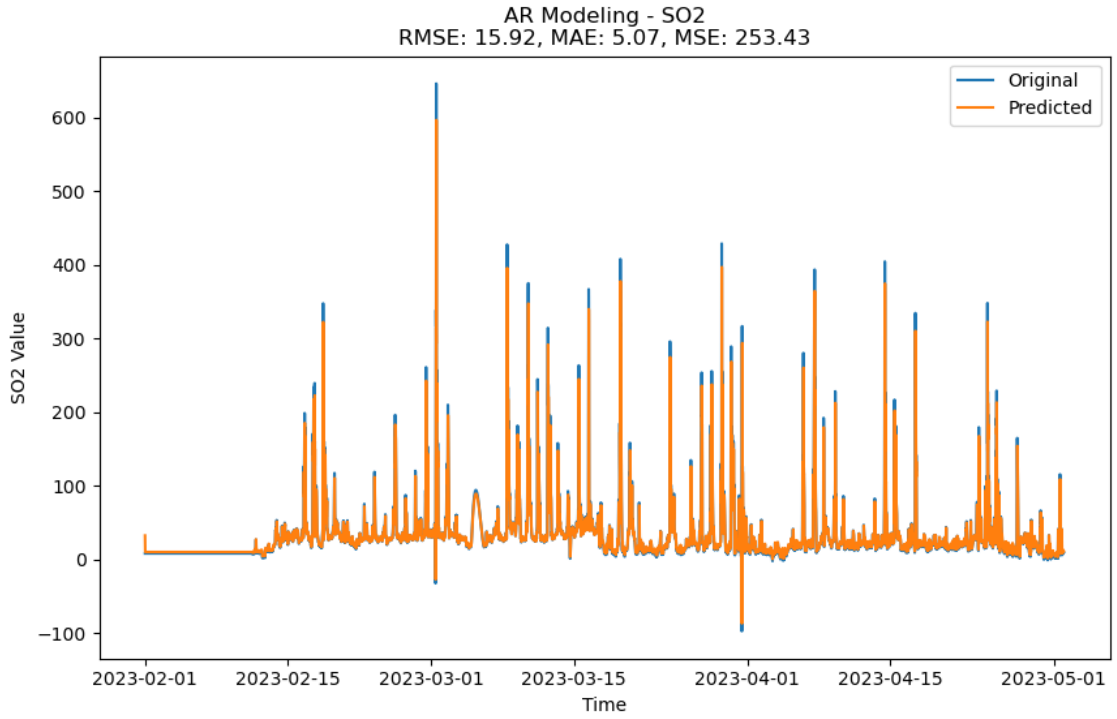
```
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, data, label='Original')
    plt.plot(df.index, predictions, label='Predicted')
    plt.xlabel('Time')
    plt.ylabel(f'{col} Value')
    plt.title(f'AR Modeling - {col}\nRMSE: {rmse:.2f}, MAE: {mae:.2f}, MSE:␣
 ↪{mse:.2f}')
    plt.legend()
    plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```

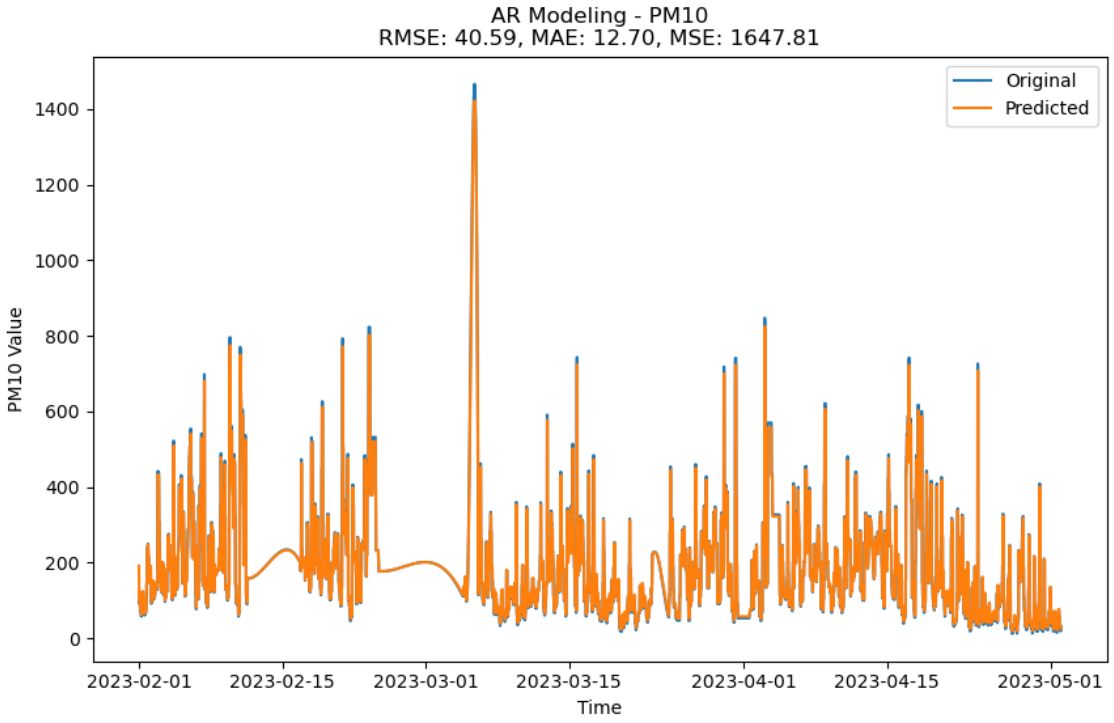



```
/opt/anaconda3/lib/python3.9/site-
```

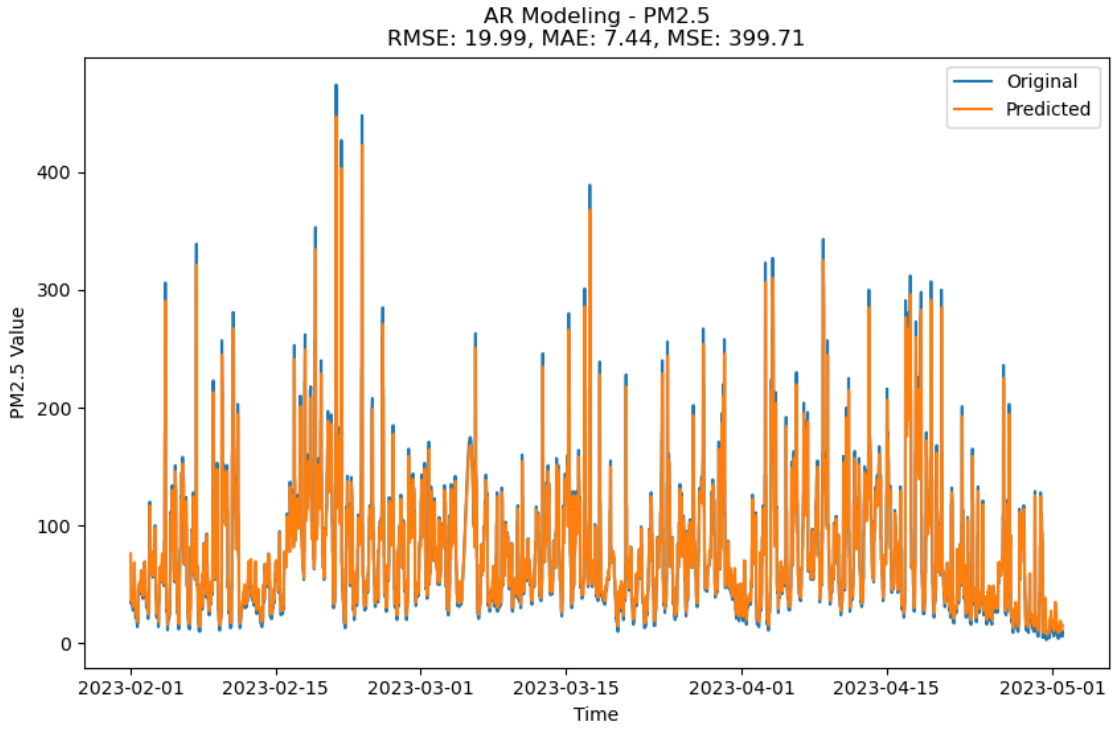
```
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```



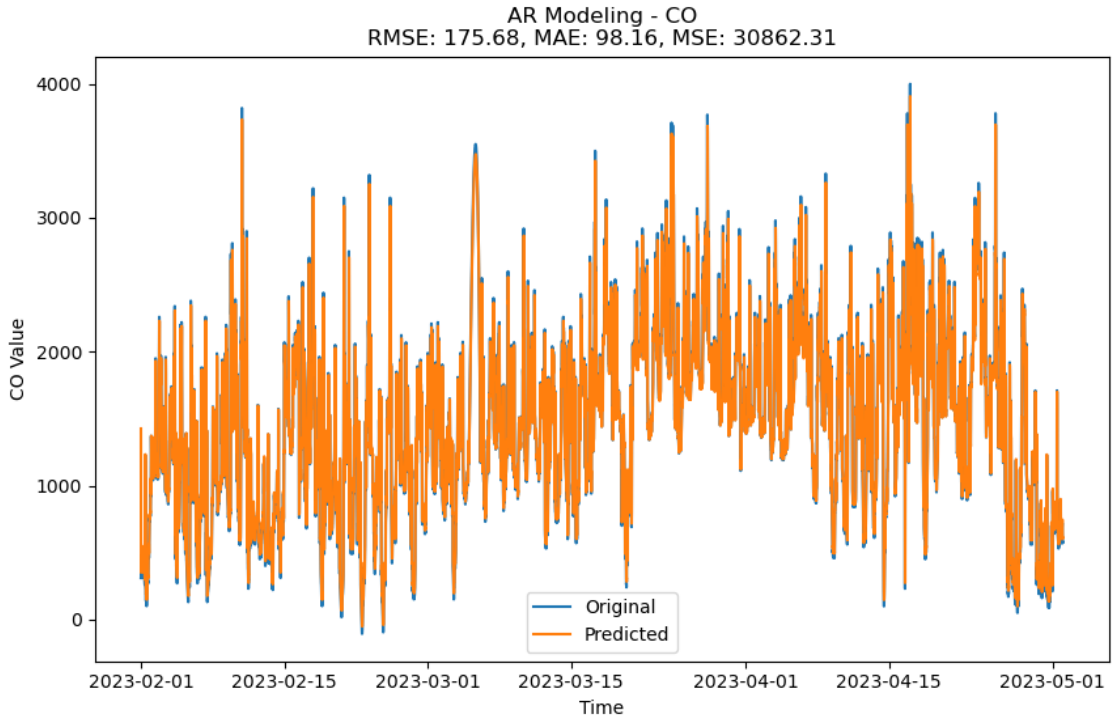

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```



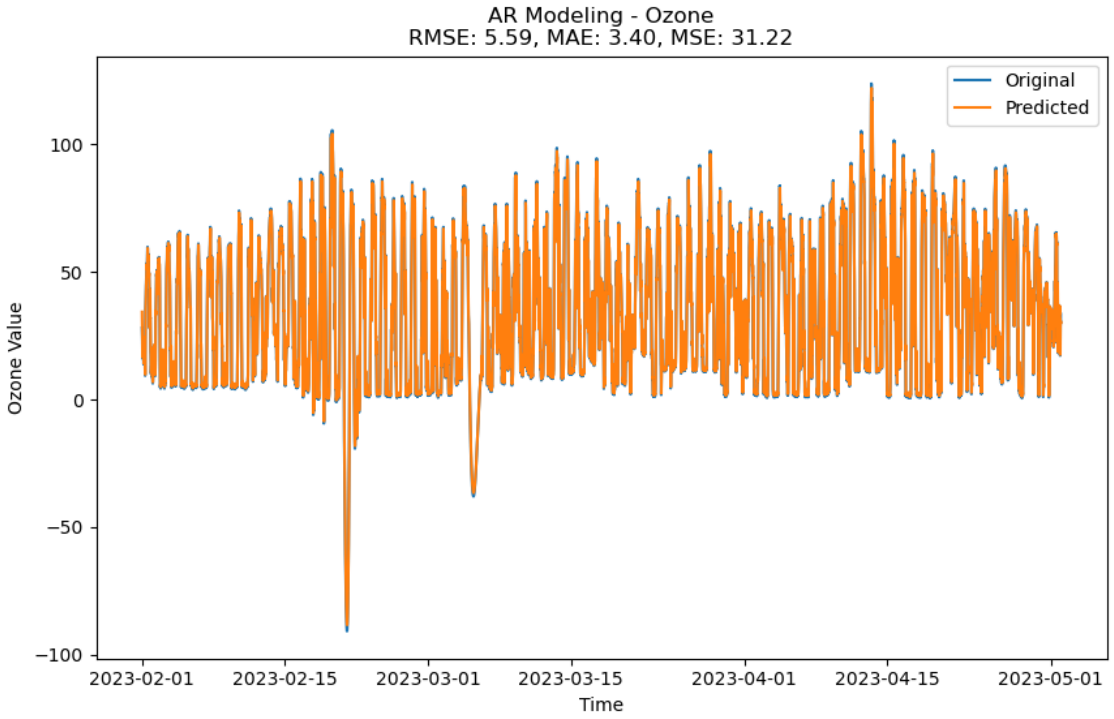

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```



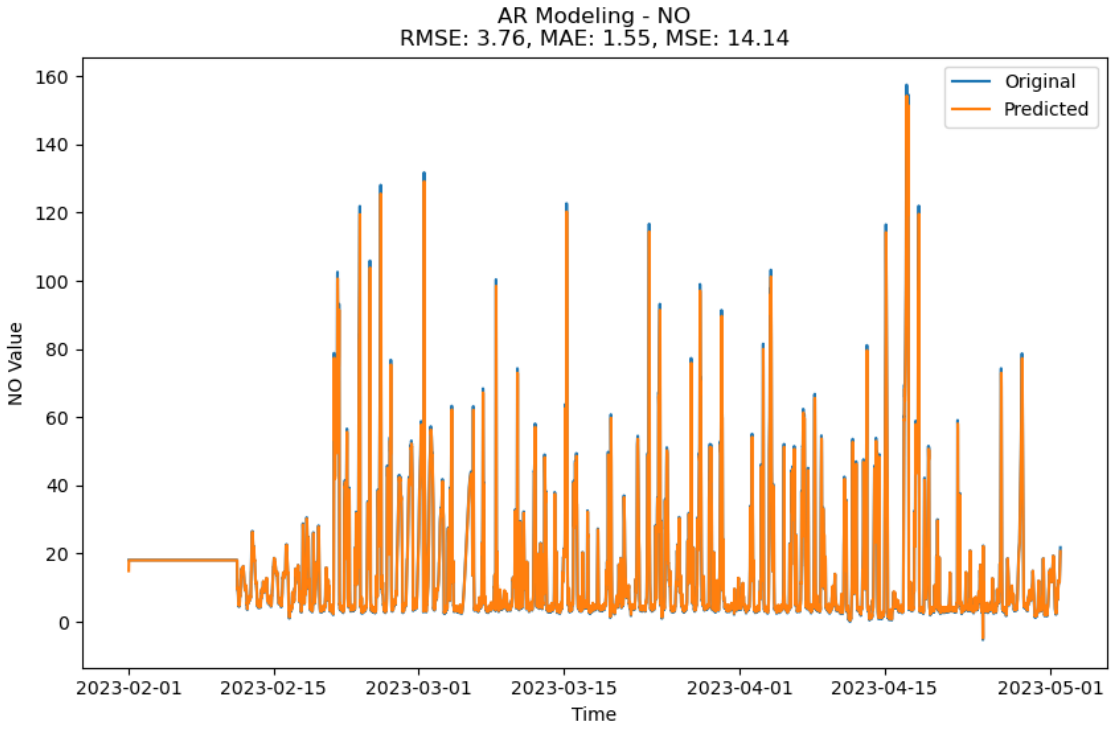

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```




```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```




```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```

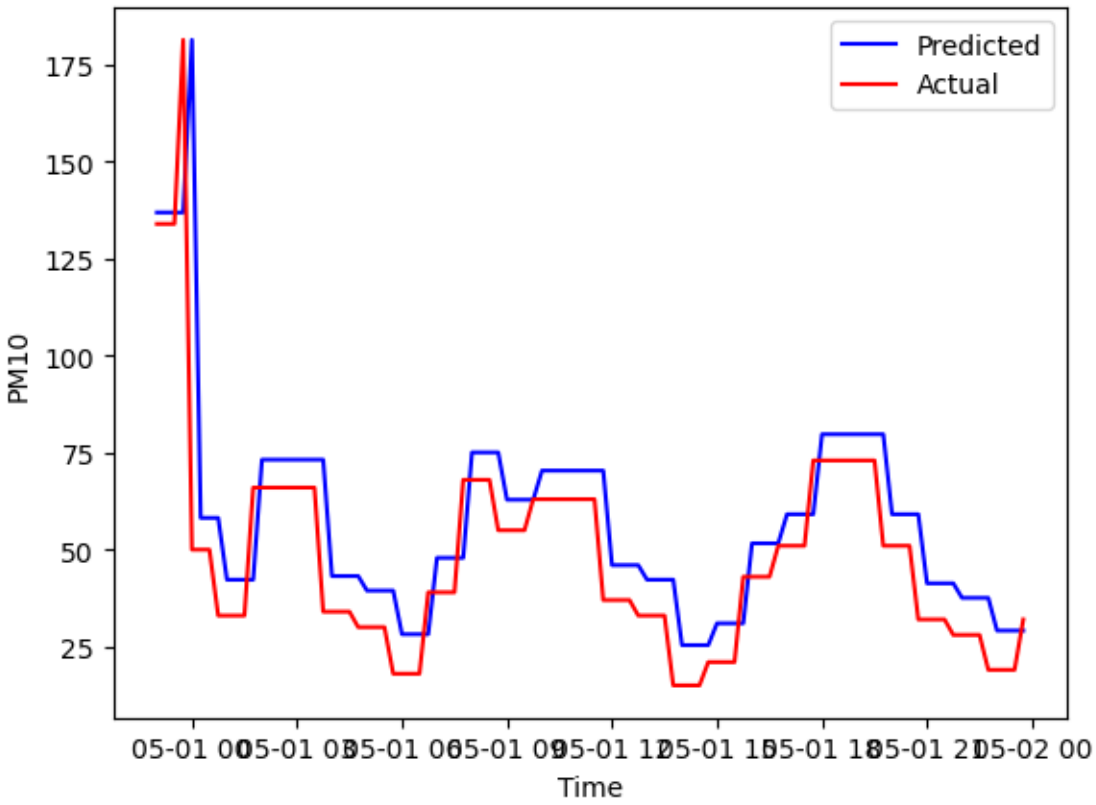


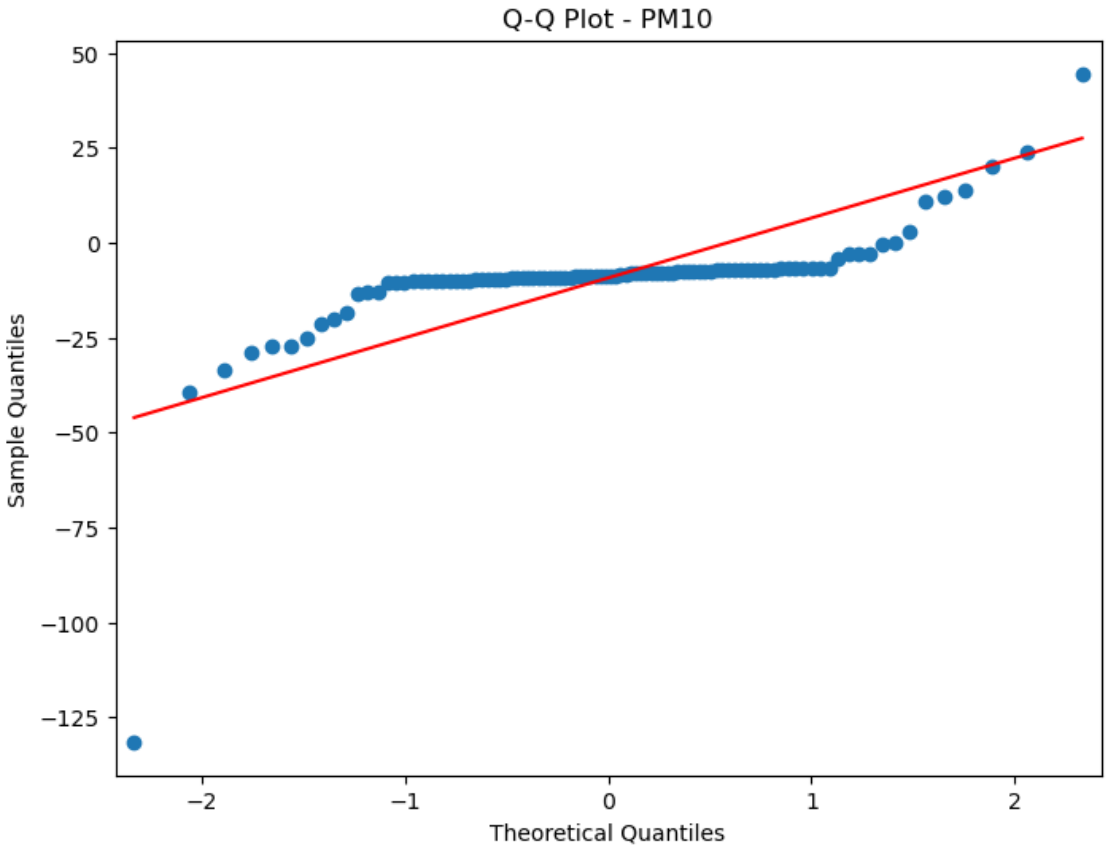

[466]:
```
lags=[1,1,3,4,4,4,4,5,2,2]
```

[467]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
import math

# Assuming you have defined the `column_names` list and `lags` list

i = 0
for column in columns_of_interest[3:]:
    train_data = df[column][:-100]
    test_data = df[column][-100:]
    ar_model = AutoReg(df[column], lags=lags[i]).fit()
    print(ar_model.summary())

    pred = ar_model.predict(start=len(train_data), end=len(df)-1, dynamic=False)

    plt.plot(pred, color="blue", label='Predicted')
    plt.plot(test_data, color="red", label='Actual')
    plt.xlabel('Time')
```

```
    plt.ylabel(column)
    plt.legend()
    plt.show()

    ar_residuals = test_data - pred

    fig, ax = plt.subplots(figsize=(8, 6))
    sm.qqplot(ar_residuals, line='s', ax=ax)
    ax.set_title(f'Q-Q Plot - {column}')
    plt.show()

    rmse = math.sqrt(mean_squared_error(pred, test_data))
    mean = df[column].mean()
    print("Mean: %f" % mean)
    print("Root Mean Squared Error: %f" % rmse)

    i += 1
```

```
                            AutoReg Model Results
==============================================================================
Dep. Variable:                   PM10   No. Observations:                 8640
Model:                   AutoReg(1)   Log Likelihood              -44639.378
Method:               Conditional MLE   S.D. of innovations             42.447
Date:                Tue, 27 Jun 2023   AIC                         89284.755
Time:                        20:19:59   BIC                         89305.948
Sample:                    02-01-2023   HQIC                        89291.981
                         - 05-01-2023
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         11.3065      0.818     13.819      0.000       9.703      12.910
PM10.L1        0.9376      0.004    250.603      0.000       0.930       0.945
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0665           +0.0000j            1.0665            0.0000
-----------------------------------------------------------------------------

/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```


Predicted
Actual
PM10
175
150
125
100
75
50
25
Time




Mean: 181.422546
Root Mean Squared Error: 18.303049

AutoReg Model Results

| | | | |
|---|---|---|---|
| Dep. Variable: | PM2.5 | No. Observations: | 8640 |
| Model: | AutoReg(1) | Log Likelihood | -38219.371 |
| Method: | Conditional MLE | S.D. of innovations | 20.188 |
| Date: | Tue, 27 Jun 2023 | AIC | 76444.741 |
| Time: | 20:19:59 | BIC | 76465.933 |
| Sample: | 02-01-2023 | HQIC | 76451.967 |
| | - 05-01-2023 | | |

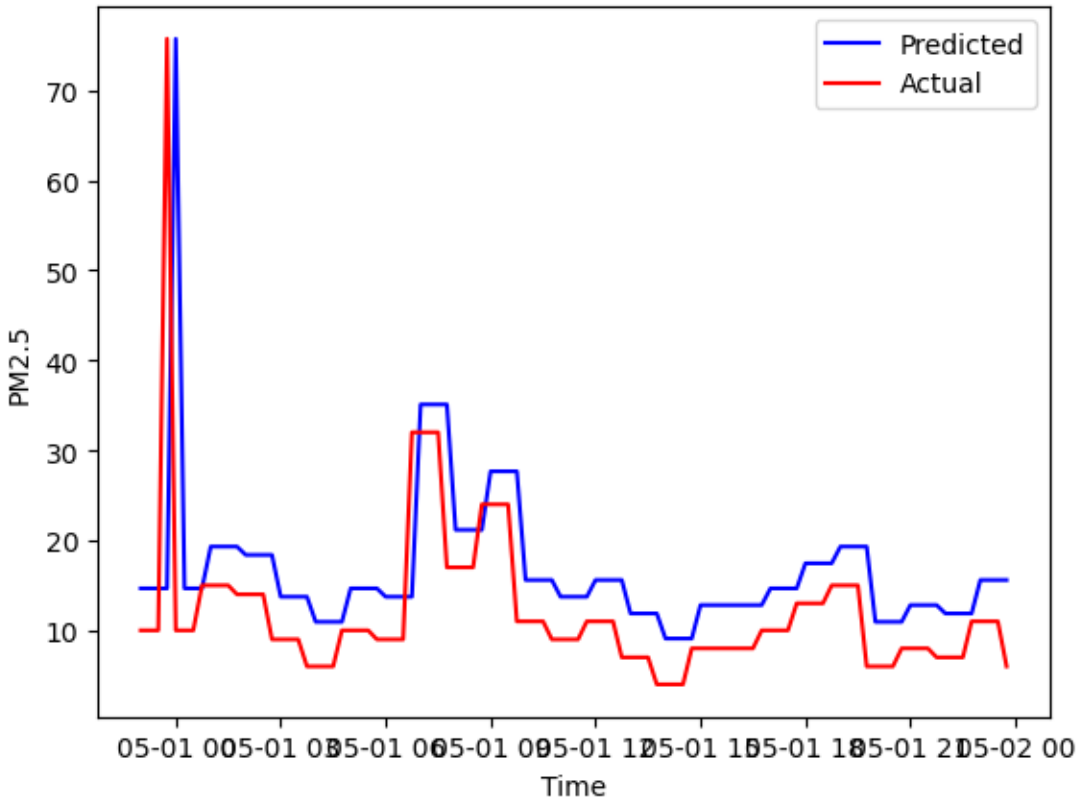
| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| const | 5.3742 | 0.372 | 14.459 | 0.000 | 4.646 | 6.103 |
| PM2.5.L1 | 0.9290 | 0.004 | 233.144 | 0.000 | 0.921 | 0.937 |

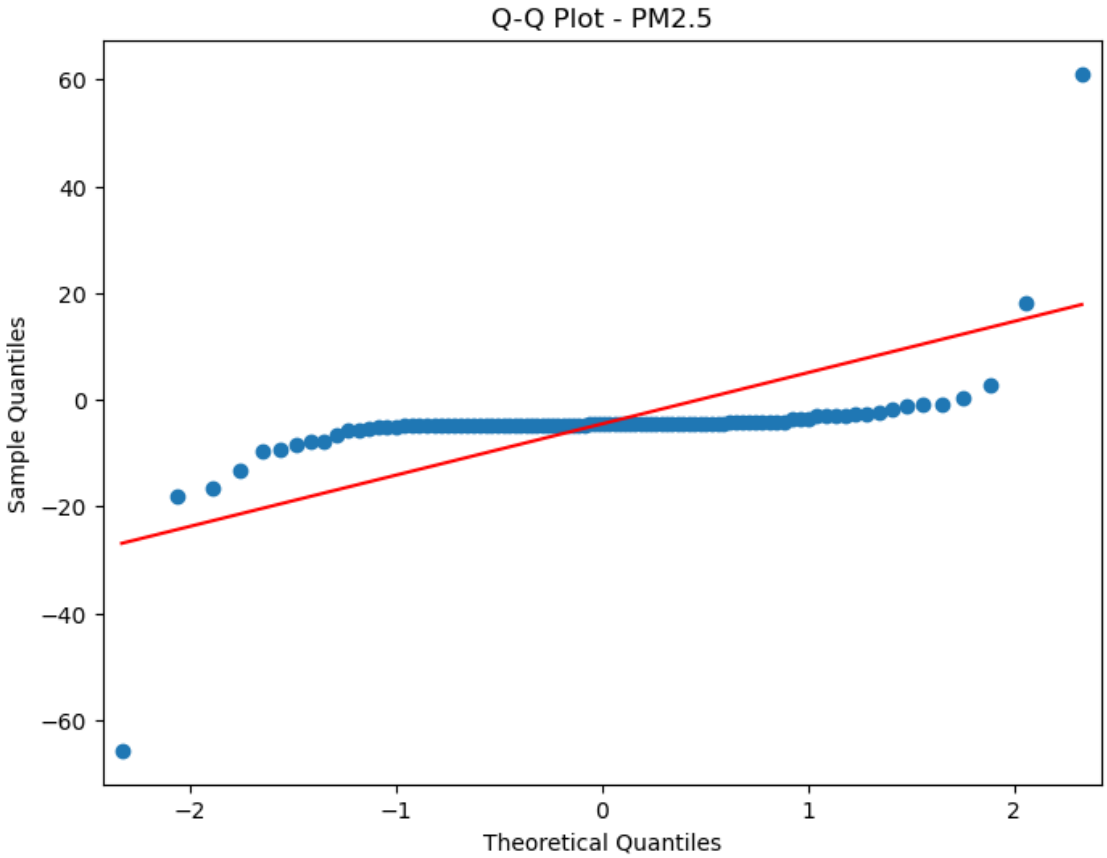
Roots

| Real | Imaginary | Modulus | Frequency |
|---|---|---|---|

```
AR.1            1.0765           +0.0000j            1.0765            0.0000
-----------------------------------------------------------------------------
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```






Mean: 75.691409
Root Mean Squared Error: 10.630435

AutoReg Model Results

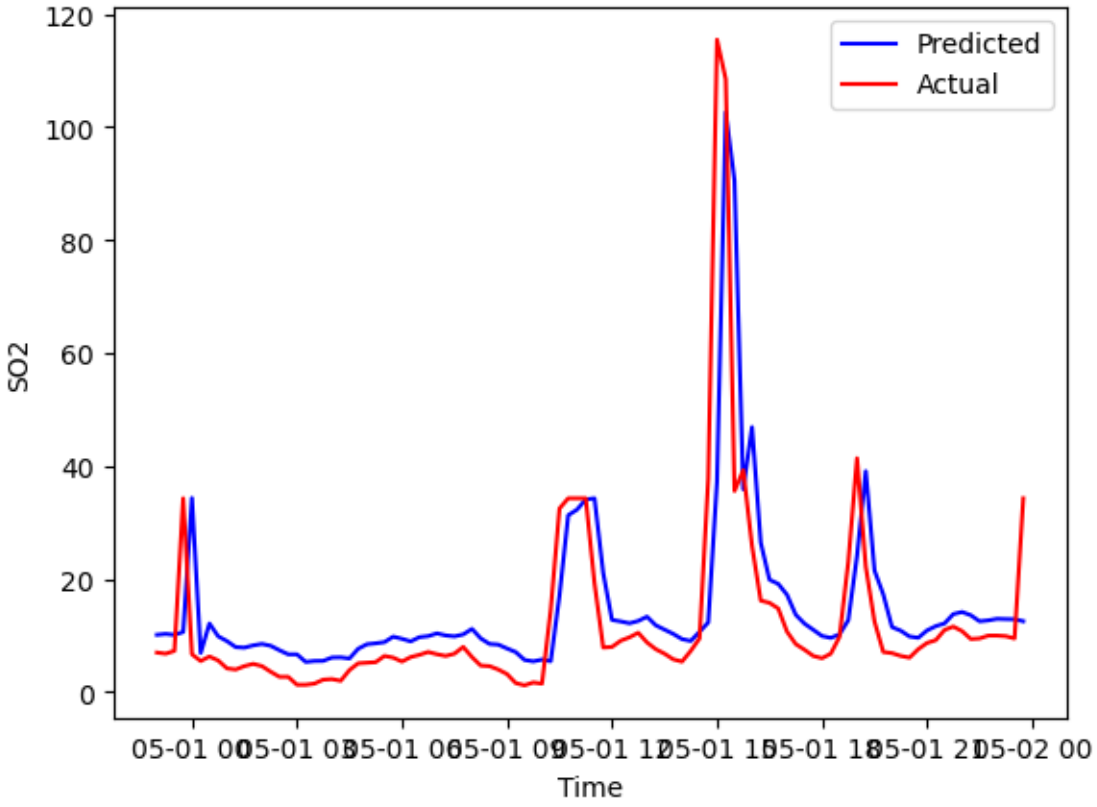
| | | | |
|---|---|---|---|
| Dep. Variable: | SO2 | No. Observations: | 8640 |
| Model: | AutoReg(3) | Log Likelihood | -37225.267 |
| Method: | Conditional MLE | S.D. of innovations | 18.012 |
| Date: | Tue, 27 Jun 2023 | AIC | 74460.533 |
| Time: | 20:20:00 | BIC | 74495.852 |
| Sample: | 02-01-2023 | HQIC | 74472.576 |
| | - 05-01-2023 | | |

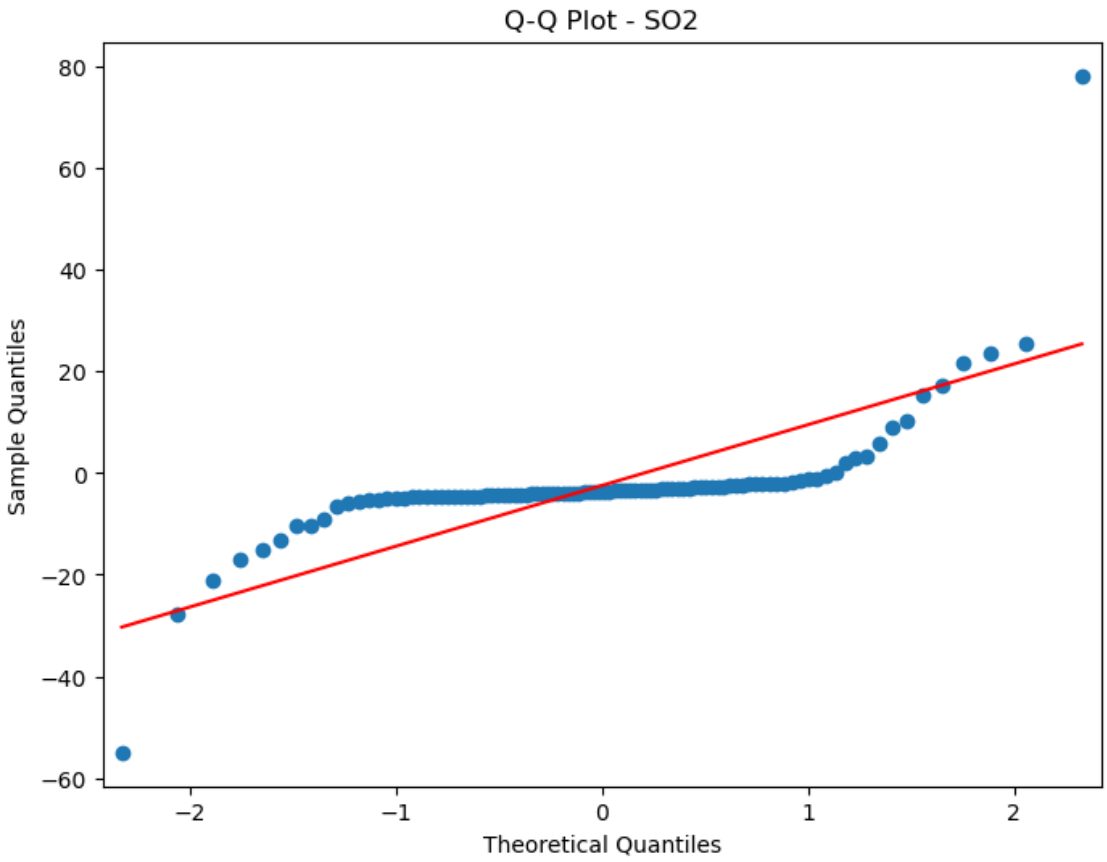
| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| const | 4.2106 | 0.277 | 15.210 | 0.000 | 3.668 | 4.753 |
| SO2.L1 | 0.8813 | 0.011 | 82.424 | 0.000 | 0.860 | 0.902 |
| SO2.L2 | -0.1170 | 0.014 | -8.220 | 0.000 | -0.145 | -0.089 |
| SO2.L3 | 0.1127 | 0.011 | 10.544 | 0.000 | 0.092 | 0.134 |

Roots

```
                Real          Imaginary           Modulus         Frequency
------------------------------------------------------------------------------
AR.1            1.1213           -0.0000j            1.1213           -0.0000
AR.2           -0.0417           -2.8122j            2.8125           -0.2524
AR.3           -0.0417           +2.8122j            2.8125            0.2524
------------------------------------------------------------------------------
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```



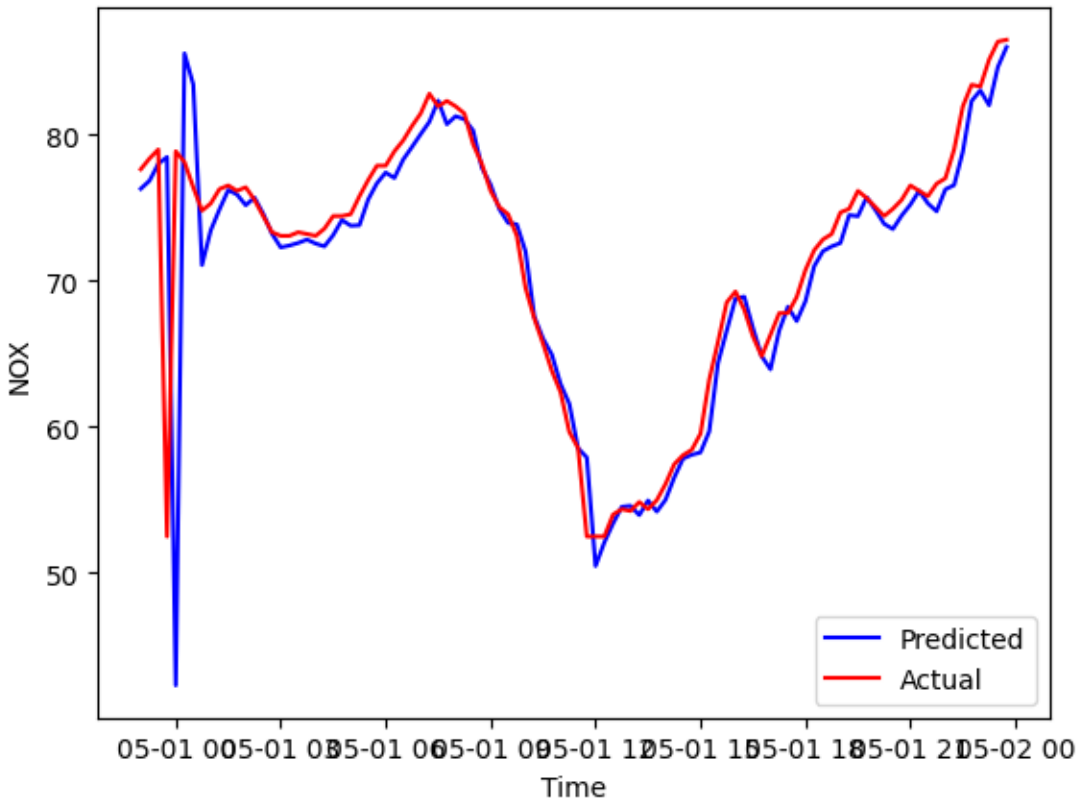


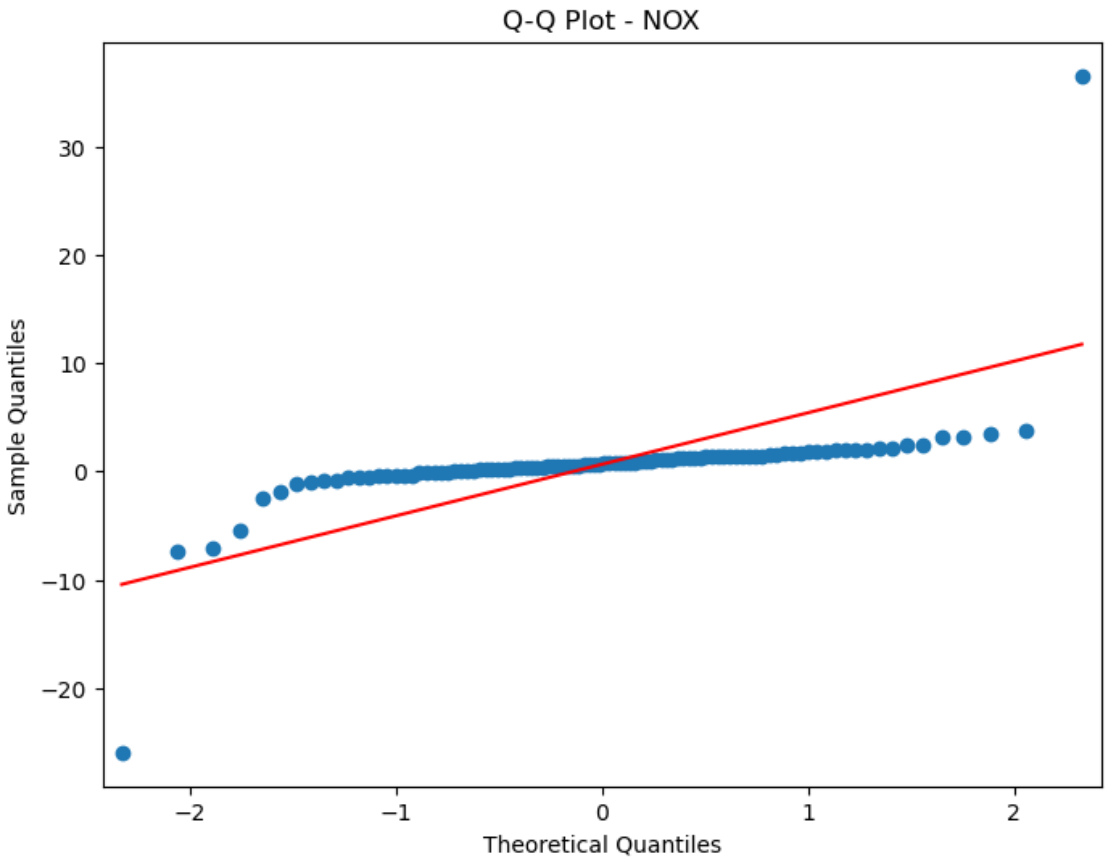

```
Mean: 34.246209
Root Mean Squared Error: 12.209654

/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)

                            AutoReg Model Results
==============================================================================
Dep. Variable:                    NOX   No. Observations:                 8640
Model:                   AutoReg(4)   Log Likelihood              -26774.184
Method:               Conditional MLE   S.D. of innovations              5.373
Date:                Tue, 27 Jun 2023   AIC                          53560.368
Time:                        20:20:00   BIC                          53602.750
Sample:                  02-01-2023   HQIC                         53574.819
                       - 05-01-2023
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.6002      0.130     12.286      0.000       1.345       1.855
```

| | | | | | | |
|---|---|---|---|---|---|---|
| NOX.L1 | 1.3524 | 0.011 | 127.053 | 0.000 | 1.332 | 1.373 |
| NOX.L2 | -0.2884 | 0.018 | -16.164 | 0.000 | -0.323 | -0.253 |
| NOX.L3 | -0.2411 | 0.018 | -13.513 | 0.000 | -0.276 | -0.206 |
| NOX.L4 | 0.1466 | 0.011 | 13.776 | 0.000 | 0.126 | 0.168 |

Roots

| | Real | Imaginary | Modulus | Frequency |
|---|---|---|---|---|
| AR.1 | -2.1044 | -0.0000j | 2.1044 | -0.5000 |
| AR.2 | 1.0481 | -0.0000j | 1.0481 | -0.0000 |
| AR.3 | 1.3502 | -1.1262j | 1.7583 | -0.1106 |
| AR.4 | 1.3502 | +1.1262j | 1.7583 | 0.1106 |





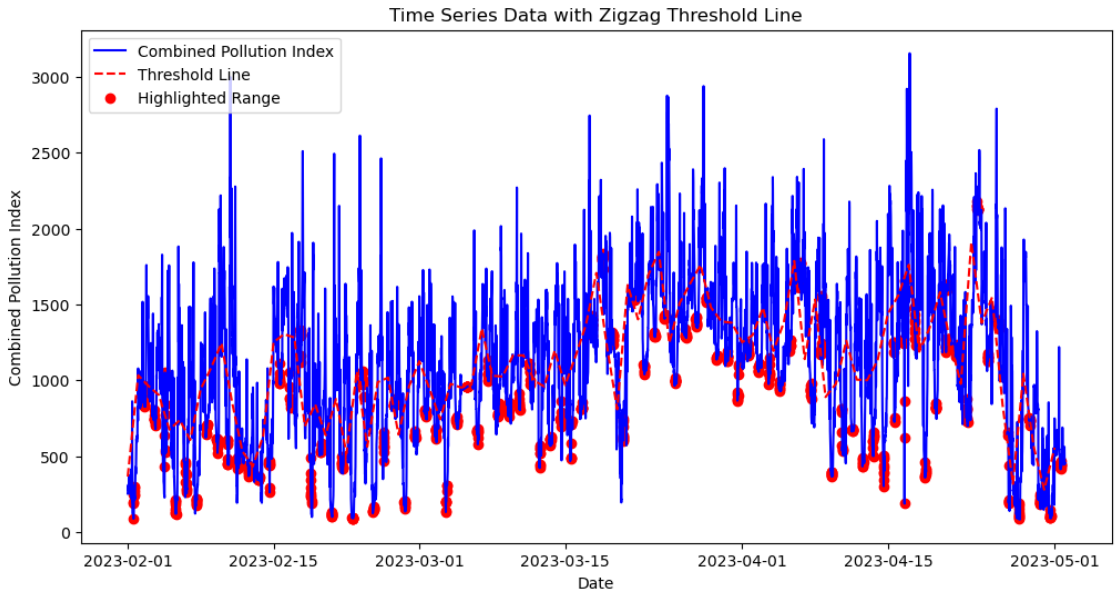

```
Mean: 52.487433
Root Mean Squared Error: 4.800257
```

as you see data not following Normal ditributuion

[507]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV file and preprocess the data
df = pd.read_csv('Open pit blasting 01-02-2023 000000 To 01-05-2023 235959.
 ↪csv', na_values='NA')
df['s'] = pd.to_datetime(df['From'], format='%Y-%m-%d %H:%M:%S',␣
 ↪errors='coerce')
df = df.drop(['From', '#'], axis=1)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM10 (µg/m3)':␣
 ↪'PM10'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  PM2.5 (µg/m3)':␣
 ↪'PM2.5'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO2 (µg/m3)':␣
 ↪'NO2'}, inplace=True)
```

```
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NO (µg/m3)':␣
 ↪'NO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  CO (mg/m3)':␣
 ↪'CO'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  SO2 (µg/m3)':␣
 ↪'SO2'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NH3 (µg/m3)':␣
 ↪'NH3'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Ozone (µg/m3)':␣
 ↪'Ozone'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  Benzene (µg/m3)':␣
 ↪'Benzene'}, inplace=True)
df.rename(columns={'Singrauli, Surya Kiran Bhawan Dudhichua  NOX (ppb)':␣
 ↪'NOX'}, inplace=True)
df.set_index('s', inplace=True)
df['CO'] = df['CO'] * 1000
df['NOX'] = df['NOX'] * 1.23

columns_of_interest = ['CO', 'NOX', 'SO2', 'PM10', 'PM2.5', 'Ozone', 'NO']
df = df.iloc[:-3]
for col in columns_of_interest:
    df[col].interpolate(method='linear', order=3, inplace=True)
    df[col].fillna(method='ffill', inplace=True)
    df[col].fillna(method='bfill', inplace=True)
# Calculate the combined pollution index
co_avg = 1410
nox_avg = 80
pm10_avg = 181.39
pm25_avg = 75.96
so2_avg = 34.3
ozone_avg = 35.63
no_avg = 14.65

sum_avg = co_avg + nox_avg + pm10_avg + pm25_avg + so2_avg + ozone_avg + no_avg

df['Combined Pollution Index'] = (
    (df['CO'] * co_avg / sum_avg) +
    (df['NOX'] * nox_avg / sum_avg) +
    (df['PM10'] * pm10_avg / sum_avg) +
    (df['PM2.5'] * pm25_avg / sum_avg) +
    (df['SO2'] * so2_avg / sum_avg) +
    (df['Ozone'] * ozone_avg / sum_avg) +
    (df['NO'] * no_avg / sum_avg)
)
# Resample the data to daily intervals and calculate the mean pollution value␣
 ↪for each day
```

```
df_resampled = df.resample('1D').mean()
mean_pollution = df_resampled['Combined Pollution Index']
mean_pollution_time = df_resampled.index

# Create zigzag threshold line by connecting mean values
threshold_line_x = mean_pollution.index.repeat(2)[1:-1]
threshold_line_y = mean_pollution.repeat(2)[1:-1]

# Plot the resampled time series data with the zigzag threshold line
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Combined Pollution Index'], color='blue',␣
 ↪label='Combined Pollution Index')
plt.plot(threshold_line_x, threshold_line_y, color='red', linestyle='--',␣
 ↪label='Threshold Line')
plt.xlabel('Date')
plt.ylabel('Combined Pollution Index')
plt.title('Time Series Data with Zigzag Threshold Line')

# Highlight the time range from 1:45 to 2:45
start_time = pd.to_datetime('13:00', format='%H:%M').time()
end_time = pd.to_datetime('15:00', format='%H:%M').time()

# Convert index to datetime column for comparison
df['datetime'] = df.index

highlighted_data = df[(df['datetime'].dt.time >= start_time) & (df['datetime'].
 ↪dt.time <= end_time)]
plt.scatter(highlighted_data['datetime'], highlighted_data['Combined Pollution␣
 ↪Index'], color='red', label='Highlighted Range')

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




as you can zig zag line is passing throught mean of pollution of each data and most of the red dots are way below the threshold line so we can say that blast don't occurs between 1.45 to 2.45 is more

[483]:
```
# Extract hour from the datetime index
df['Hour'] = df.index.hour

# Group the data by date and hour, and calculate the maximum pollution value␣
 ↪for each hour
max_pollution_per_hour = df.groupby([df.index.date, 'Hour'])['Combined␣
 ↪Pollution Index'].max()

# Find the hour(s) with the maximum pollution value for each date
dates_with_max_pollution = max_pollution_per_hour.groupby(level=0).idxmax()

# Iterate over the dates and print the hour(s) with maximum pollution
for date, hour in dates_with_max_pollution:
    print(f"On {date}, the hour(s) with maximum pollution is/are: {hour}")
```

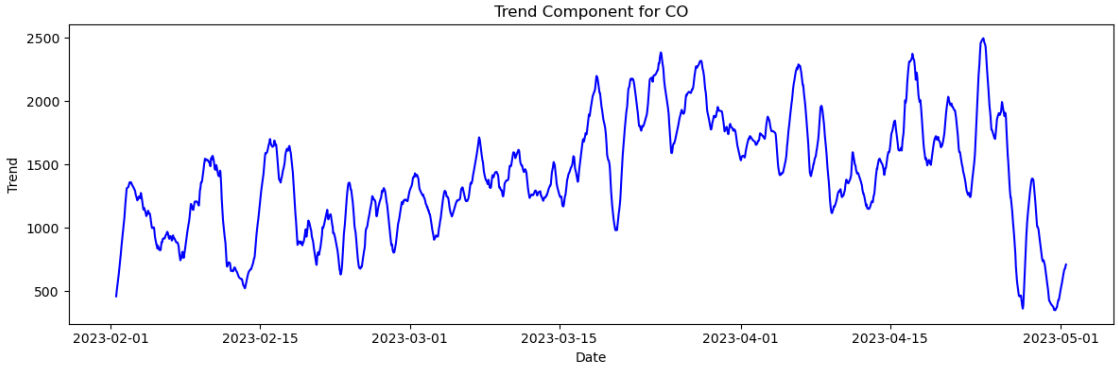
```
On 2023-02-01, the hour(s) with maximum pollution is/are: 23
On 2023-02-02, the hour(s) with maximum pollution is/are: 19
On 2023-02-03, the hour(s) with maximum pollution is/are: 10
On 2023-02-04, the hour(s) with maximum pollution is/are: 7
On 2023-02-05, the hour(s) with maximum pollution is/are: 20
On 2023-02-06, the hour(s) with maximum pollution is/are: 21
On 2023-02-07, the hour(s) with maximum pollution is/are: 7
On 2023-02-08, the hour(s) with maximum pollution is/are: 22
On 2023-02-09, the hour(s) with maximum pollution is/are: 22
On 2023-02-10, the hour(s) with maximum pollution is/are: 20
```

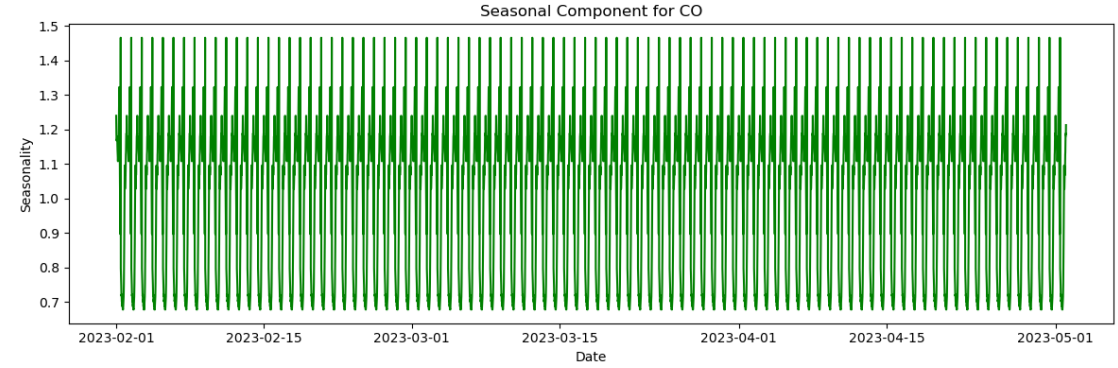
```
On 2023-02-11, the hour(s) with maximum pollution is/are: 7
On 2023-02-12, the hour(s) with maximum pollution is/are: 10
On 2023-02-13, the hour(s) with maximum pollution is/are: 10
On 2023-02-14, the hour(s) with maximum pollution is/are: 23
On 2023-02-15, the hour(s) with maximum pollution is/are: 10
On 2023-02-16, the hour(s) with maximum pollution is/are: 18
On 2023-02-17, the hour(s) with maximum pollution is/are: 19
On 2023-02-18, the hour(s) with maximum pollution is/are: 19
On 2023-02-19, the hour(s) with maximum pollution is/are: 23
On 2023-02-20, the hour(s) with maximum pollution is/are: 19
On 2023-02-21, the hour(s) with maximum pollution is/are: 7
On 2023-02-22, the hour(s) with maximum pollution is/are: 0
On 2023-02-23, the hour(s) with maximum pollution is/are: 7
On 2023-02-24, the hour(s) with maximum pollution is/are: 7
On 2023-02-25, the hour(s) with maximum pollution is/are: 7
On 2023-02-26, the hour(s) with maximum pollution is/are: 21
On 2023-02-27, the hour(s) with maximum pollution is/are: 1
On 2023-02-28, the hour(s) with maximum pollution is/are: 7
On 2023-03-01, the hour(s) with maximum pollution is/are: 23
On 2023-03-02, the hour(s) with maximum pollution is/are: 2
On 2023-03-03, the hour(s) with maximum pollution is/are: 22
On 2023-03-04, the hour(s) with maximum pollution is/are: 4
On 2023-03-05, the hour(s) with maximum pollution is/are: 23
On 2023-03-06, the hour(s) with maximum pollution is/are: 6
On 2023-03-07, the hour(s) with maximum pollution is/are: 10
On 2023-03-08, the hour(s) with maximum pollution is/are: 20
On 2023-03-09, the hour(s) with maximum pollution is/are: 4
On 2023-03-10, the hour(s) with maximum pollution is/are: 8
On 2023-03-11, the hour(s) with maximum pollution is/are: 10
On 2023-03-12, the hour(s) with maximum pollution is/are: 10
On 2023-03-13, the hour(s) with maximum pollution is/are: 5
On 2023-03-14, the hour(s) with maximum pollution is/are: 5
On 2023-03-15, the hour(s) with maximum pollution is/are: 22
On 2023-03-16, the hour(s) with maximum pollution is/are: 19
On 2023-03-17, the hour(s) with maximum pollution is/are: 8
On 2023-03-18, the hour(s) with maximum pollution is/are: 10
On 2023-03-19, the hour(s) with maximum pollution is/are: 7
On 2023-03-20, the hour(s) with maximum pollution is/are: 19
On 2023-03-21, the hour(s) with maximum pollution is/are: 22
On 2023-03-22, the hour(s) with maximum pollution is/are: 2
On 2023-03-23, the hour(s) with maximum pollution is/are: 20
On 2023-03-24, the hour(s) with maximum pollution is/are: 19
On 2023-03-25, the hour(s) with maximum pollution is/are: 0
On 2023-03-26, the hour(s) with maximum pollution is/are: 0
On 2023-03-27, the hour(s) with maximum pollution is/are: 6
On 2023-03-28, the hour(s) with maximum pollution is/are: 7
On 2023-03-29, the hour(s) with maximum pollution is/are: 19
On 2023-03-30, the hour(s) with maximum pollution is/are: 7
```

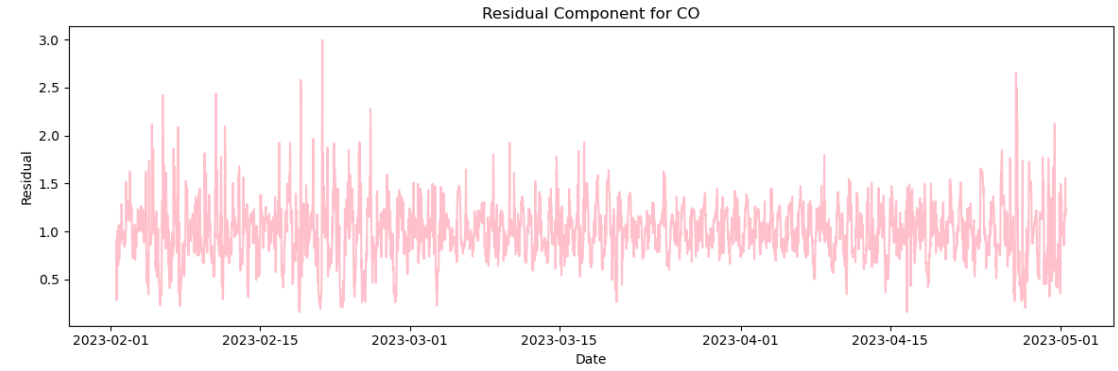
```
On 2023-03-31, the hour(s) with maximum pollution is/are: 10
On 2023-04-01, the hour(s) with maximum pollution is/are: 10
On 2023-04-02, the hour(s) with maximum pollution is/are: 8
On 2023-04-03, the hour(s) with maximum pollution is/are: 22
On 2023-04-04, the hour(s) with maximum pollution is/are: 0
On 2023-04-05, the hour(s) with maximum pollution is/are: 20
On 2023-04-06, the hour(s) with maximum pollution is/are: 21
On 2023-04-07, the hour(s) with maximum pollution is/are: 1
On 2023-04-08, the hour(s) with maximum pollution is/are: 20
On 2023-04-09, the hour(s) with maximum pollution is/are: 0
On 2023-04-10, the hour(s) with maximum pollution is/are: 2
On 2023-04-11, the hour(s) with maximum pollution is/are: 7
On 2023-04-12, the hour(s) with maximum pollution is/are: 22
On 2023-04-13, the hour(s) with maximum pollution is/are: 22
On 2023-04-14, the hour(s) with maximum pollution is/are: 23
On 2023-04-15, the hour(s) with maximum pollution is/are: 3
On 2023-04-16, the hour(s) with maximum pollution is/are: 19
On 2023-04-17, the hour(s) with maximum pollution is/are: 2
On 2023-04-18, the hour(s) with maximum pollution is/are: 5
On 2023-04-19, the hour(s) with maximum pollution is/are: 6
On 2023-04-20, the hour(s) with maximum pollution is/are: 6
On 2023-04-21, the hour(s) with maximum pollution is/are: 10
On 2023-04-22, the hour(s) with maximum pollution is/are: 10
On 2023-04-23, the hour(s) with maximum pollution is/are: 18
On 2023-04-24, the hour(s) with maximum pollution is/are: 10
On 2023-04-25, the hour(s) with maximum pollution is/are: 10
On 2023-04-26, the hour(s) with maximum pollution is/are: 6
On 2023-04-27, the hour(s) with maximum pollution is/are: 23
On 2023-04-28, the hour(s) with maximum pollution is/are: 1
On 2023-04-29, the hour(s) with maximum pollution is/are: 7
On 2023-04-30, the hour(s) with maximum pollution is/are: 10
On 2023-05-01, the hour(s) with maximum pollution is/are: 10
```

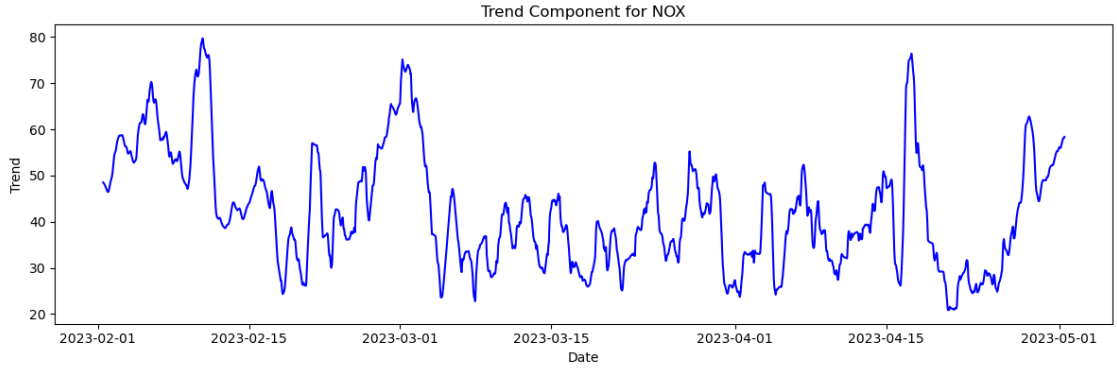
[499]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

column_names_of_interest = ['CO', 'NOX', 'SO2', 'PM10', 'PM2.5', 'Ozone', 'NO']

for column in column_names_of_interest:
    decompose_result_mult = seasonal_decompose(df[column], period=96,␣
 ↪model="multiplicative")
    trend = decompose_result_mult.trend
    seasonal = decompose_result_mult.seasonal
    residual = decompose_result_mult.resid

    plt.figure(figsize=(12, 12))

    # Plotting trend component
```

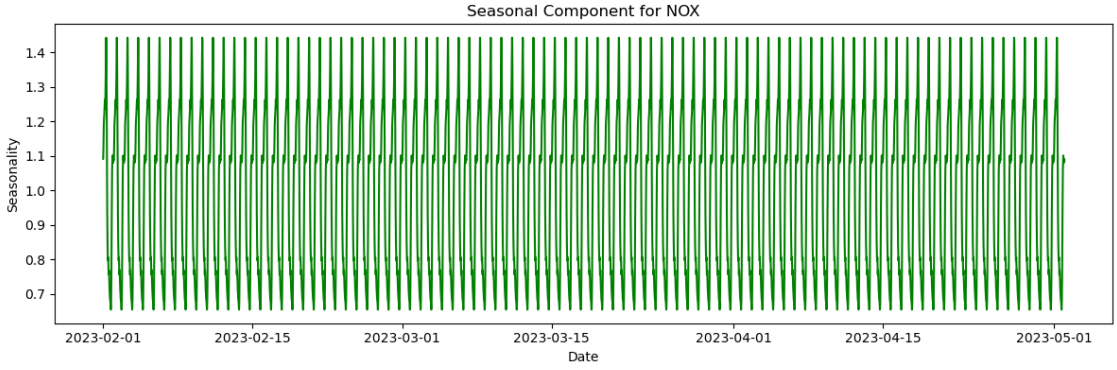
```
    plt.subplot(3, 1, 1)
    plt.plot(trend.index, trend.values, color='blue')
    plt.xlabel('Date')
    plt.ylabel('Trend')
    plt.title('Trend Component for {}'.format(column))

    # Plotting seasonal component
    plt.subplot(3, 1, 2)
    plt.plot(seasonal.index, seasonal.values, color='green')
    plt.xlabel('Date')
    plt.ylabel('Seasonality')
    plt.title('Seasonal Component for {}'.format(column))

    # Plotting residual component
    plt.subplot(3, 1, 3)
    plt.plot(residual.index, residual.values, color='pink')
    plt.xlabel('Date')
    plt.ylabel('Residual')
    plt.title('Residual Component for {}'.format(column))

    plt.tight_layout()
    plt.show()
```

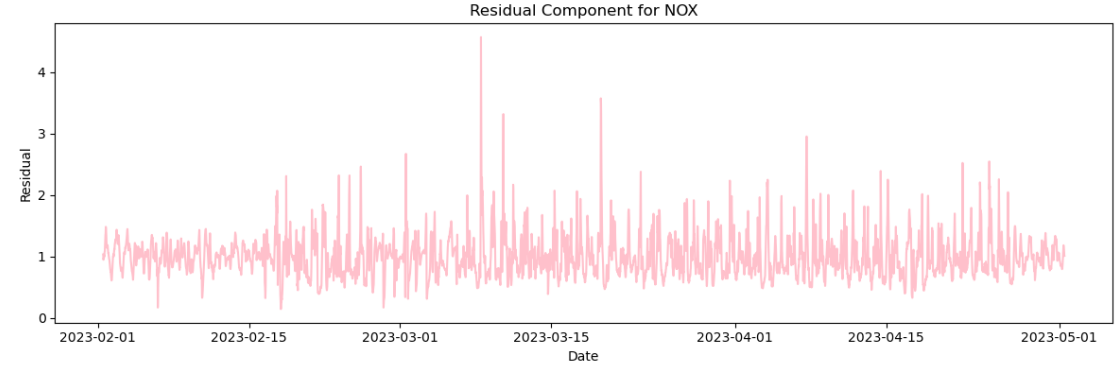

Trend Component for CO
Trend
2500
2000
1500
1000
500
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date

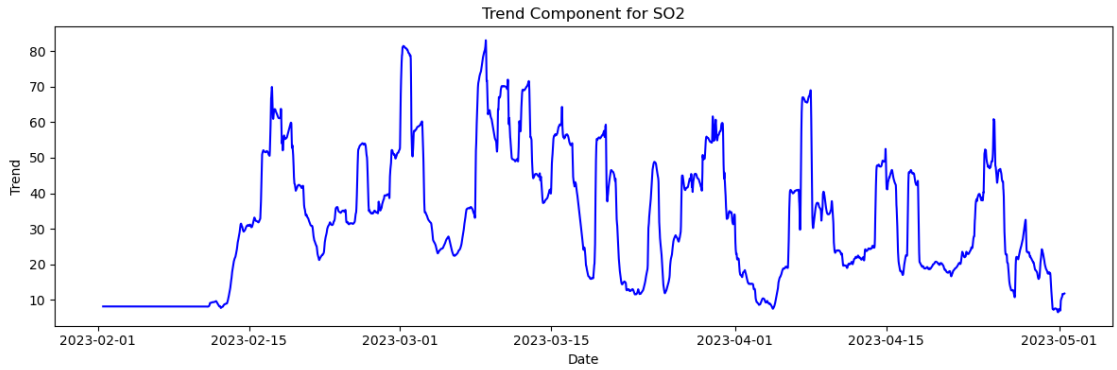

Seasonal Component for CO
Seasonality
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date

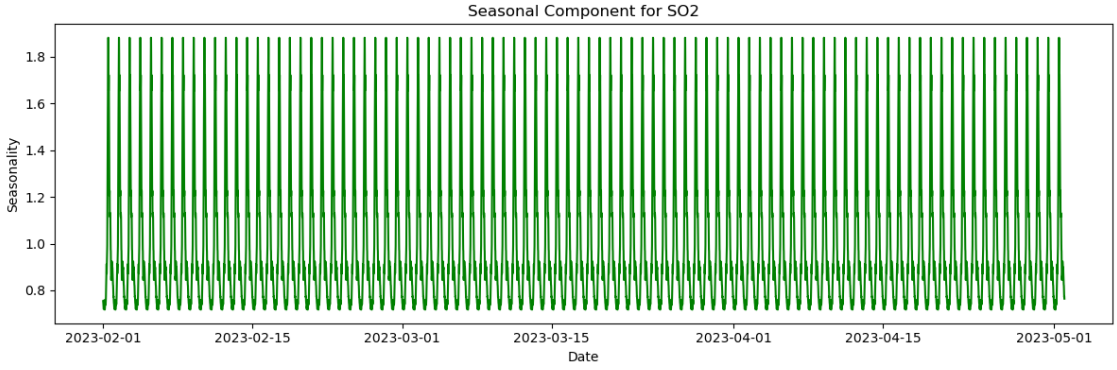

Residual Component for CO
Residual
3.0
2.5
2.0
1.5
1.0
0.5
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date

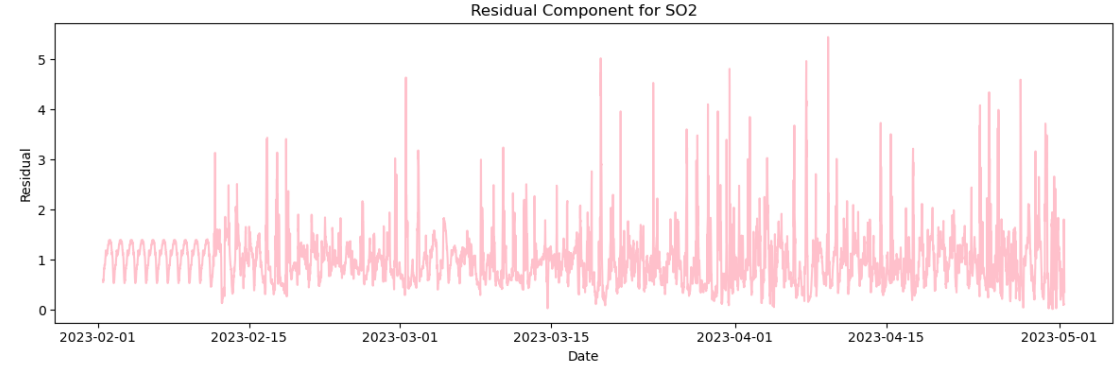

Trend Component for NOX
Trend
Date
80
70
60
50
40
30
20
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01

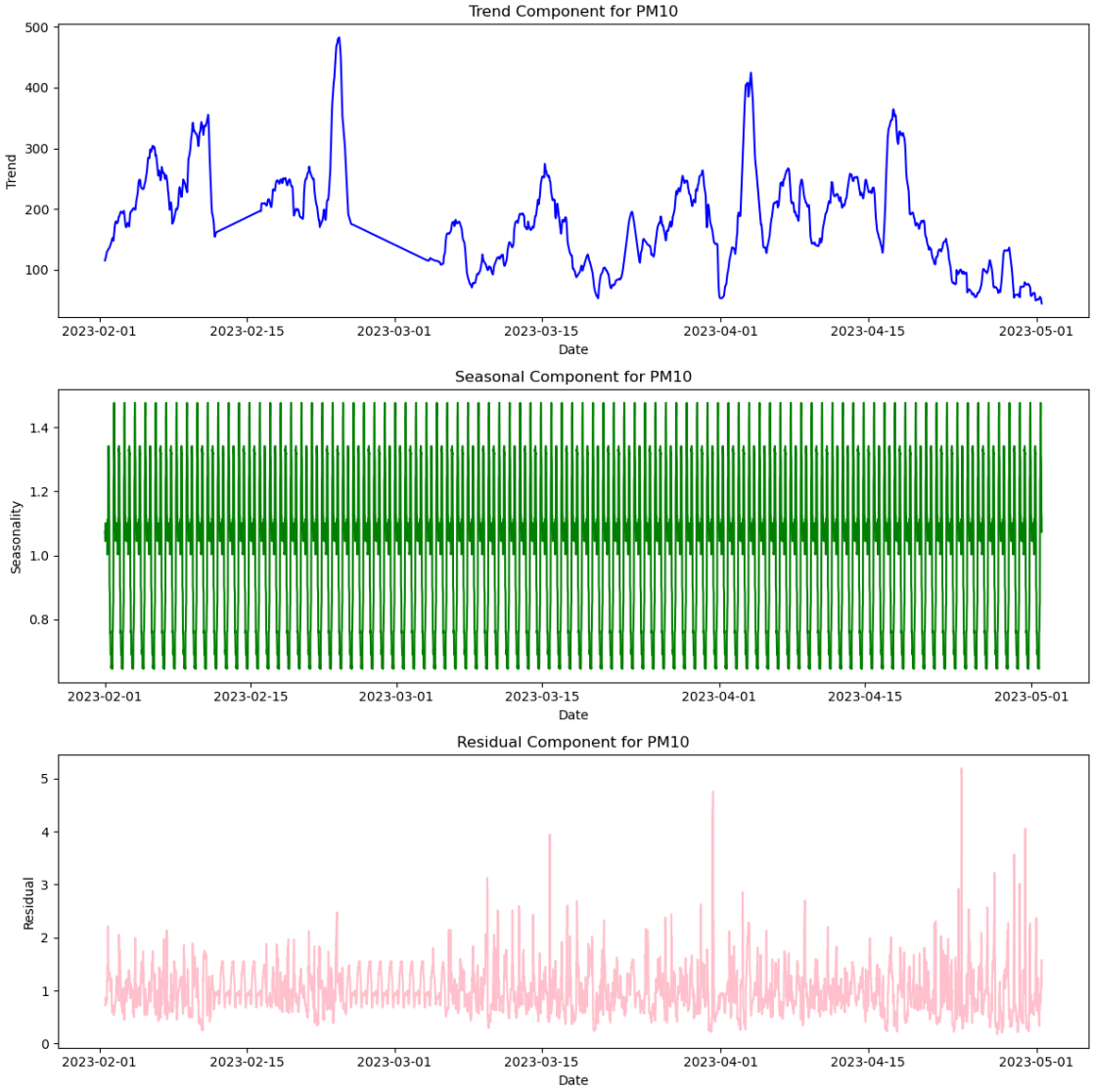

Seasonal Component for NOX
Seasonality
Date
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01

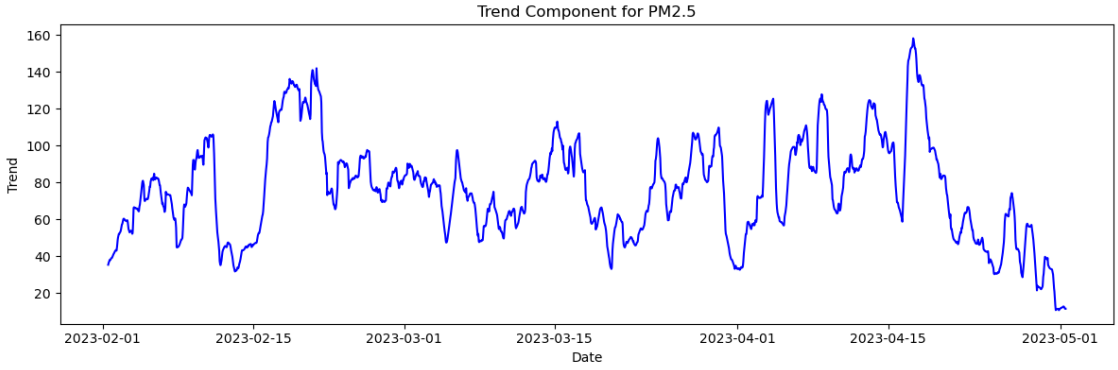

Residual Component for NOX
Residual
Date
4
3
2
1
0
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01

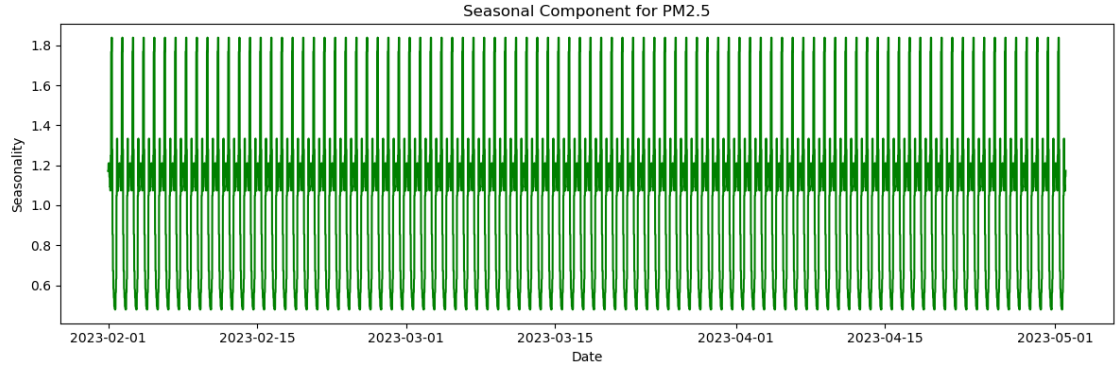

Trend Component for SO2
Trend
Date
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01

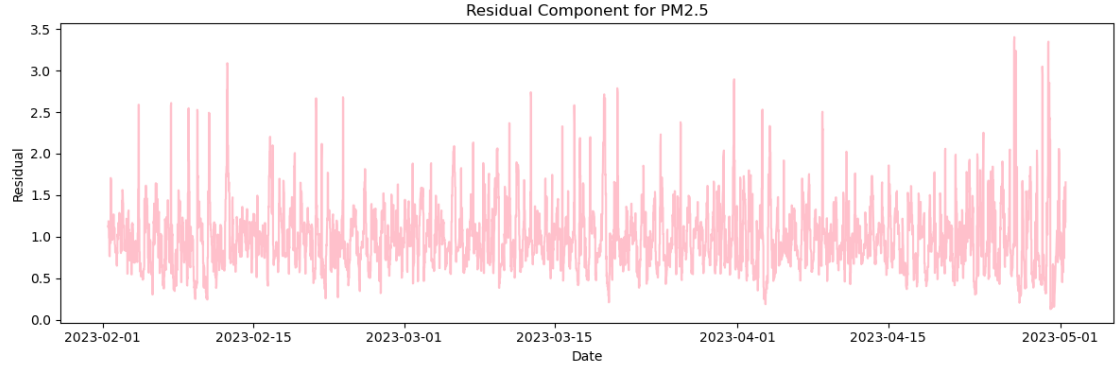

Seasonal Component for SO2
Seasonality
Date
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01

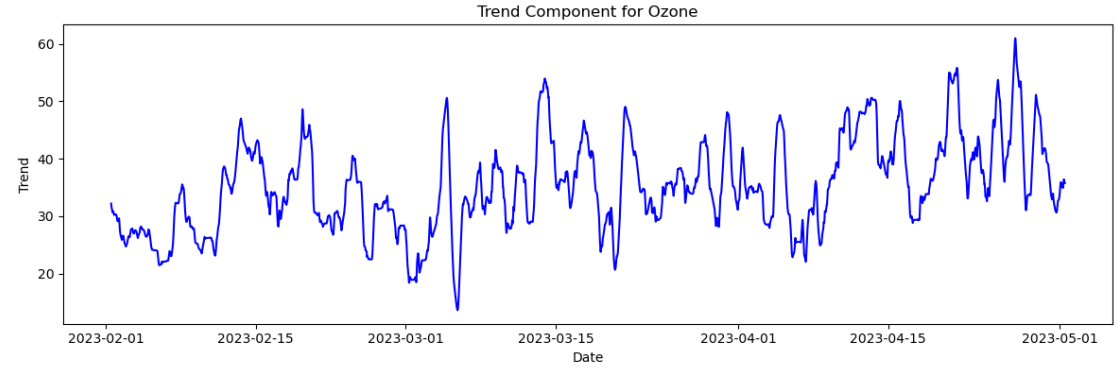

Residual Component for SO2
Residual
Date
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01

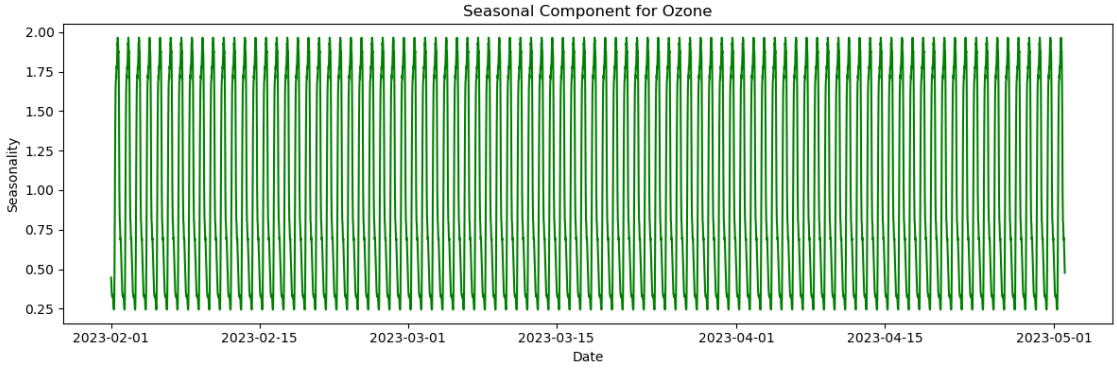

Trend Component for PM10
Trend
500
400
300
200
100
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date
Seasonal Component for PM10
Seasonality
1.4
1.2
1.0
0.8
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date
Residual Component for PM10
Residual
5
4
3
2
1
0
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date

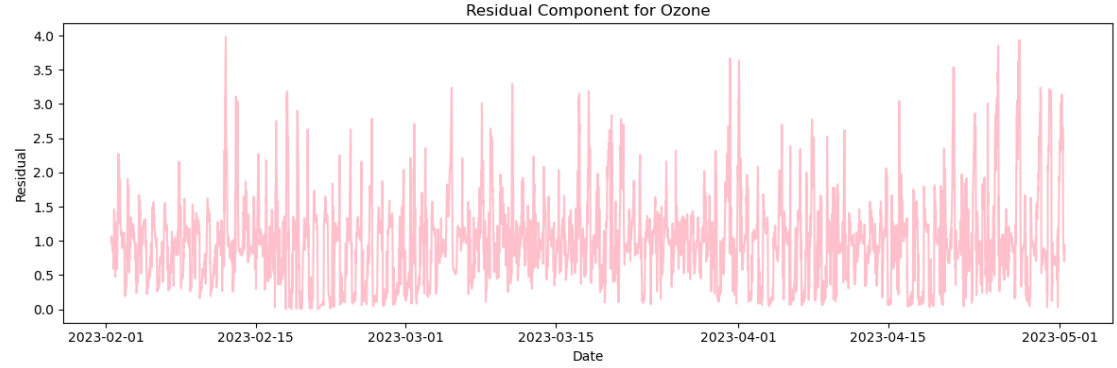

Trend Component for PM2.5
Trend
160
140
120
100
80
60
40
20
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date


Seasonal Component for PM2.5
Seasonality
1.8
1.6
1.4
1.2
1.0
0.8
0.6
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date


Residual Component for PM2.5
Residual
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date


Trend Component for Ozone
Trend
60
50
40
30
20
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date


Seasonal Component for Ozone
Seasonality
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date


Residual Component for Ozone
Residual
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01
2023-04-15
2023-05-01
Date


The trend appears to be random but the seasonality graph exhibits certian patterns, which means that the time series exhibit recurring patterns or cycles at regular intervals. While it may be challenging to make long-term predictions based on the random trend, the presence of strong seasonality allows for short-term forecasting. Seasonal patterns provide valuable information for forecasting future values within each seasonal cycle. It is useful to decompose the time series into its individual components, such as trend, seasonality, and residual, to better understand their contributions. The random trend suggests focusing on modeling the seasonality component explicitly.

NOW we can forecast the value for short period

[500]:
```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error,␣
↪mean_absolute_percentage_error, mean_squared_error
```

```
# Define the columns of interest for forecasting
columns_of_interest = ['CO', 'NOX', 'SO2', 'PM10', 'PM2.5', 'Ozone', 'NO']

# Number of entries for training
train_entries = 8300

# Iterate over the columns of interest
for column in columns_of_interest:
    # Split the data into training and testing sets
    train_data = df[column].iloc[:train_entries]
    test_data = df[column].iloc[train_entries:]

    # Fit the ARIMA model
    model = ARIMA(train_data, order=(1, 0, 0))
    model_fit = model.fit()

    # Make predictions for the test data
    forecast = model_fit.predict(start=len(train_data), end=len(df)-1)

    # Calculate the evaluation metrics
    mae = mean_absolute_error(test_data, forecast)
    mape = mean_absolute_percentage_error(test_data, forecast)
    rmse = np.sqrt(mean_squared_error(test_data, forecast))

    # Print the evaluation metrics
    print(f'Evaluation metrics for {column}:')
    print(f'MAE: {mae}')
    print(f'MAPE: {mape}')
    print(f'RMSE: {rmse}')
    print()

    # Add the forecasted values to the DataFrame
    df[column + '_forecast'] = np.nan
    df[column + '_forecast'].iloc[train_entries:] = forecast

# Plot the actual data and the ARIMA forecasts
for column in columns_of_interest:
    data_to_plot = df[[column, column + '_forecast']]
    data_to_plot.plot()
    plt.title(f'ARIMA Forecast for {column}')
    plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
```

```
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)

Evaluation metrics for CO:
MAE: 785.4716943735745
MAPE: 2.338137589174748
RMSE: 856.4211440622438


/var/folders/h1/m9dw6pns1tgclwkz5gxqn9hc0000gn/T/ipykernel_62106/1464582754.py:3
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column + '_forecast'].iloc[train_entries:] = forecast
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)

Evaluation metrics for NOX:
MAE: 11.48143217670949
MAPE: 0.2236226930184814
RMSE: 13.492534248800984


/var/folders/h1/m9dw6pns1tgclwkz5gxqn9hc0000gn/T/ipykernel_62106/1464582754.py:3
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column + '_forecast'].iloc[train_entries:] = forecast
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
```

```
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)

Evaluation metrics for SO2:
MAE: 21.282659562303518
MAPE: 6.386339322474773
RMSE: 23.055766812284016


/var/folders/h1/m9dw6pns1tgclwkz5gxqn9hc0000gn/T/ipykernel_62106/1464582754.py:3
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column + '_forecast'].iloc[train_entries:] = forecast
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)

Evaluation metrics for PM10:
MAE: 127.94704239832673
MAPE: 4.0155551269741085
RMSE: 133.65396296154023


/var/folders/h1/m9dw6pns1tgclwkz5gxqn9hc0000gn/T/ipykernel_62106/1464582754.py:3
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column + '_forecast'].iloc[train_entries:] = forecast
/opt/anaconda3/lib/python3.9/site-
```

```
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```

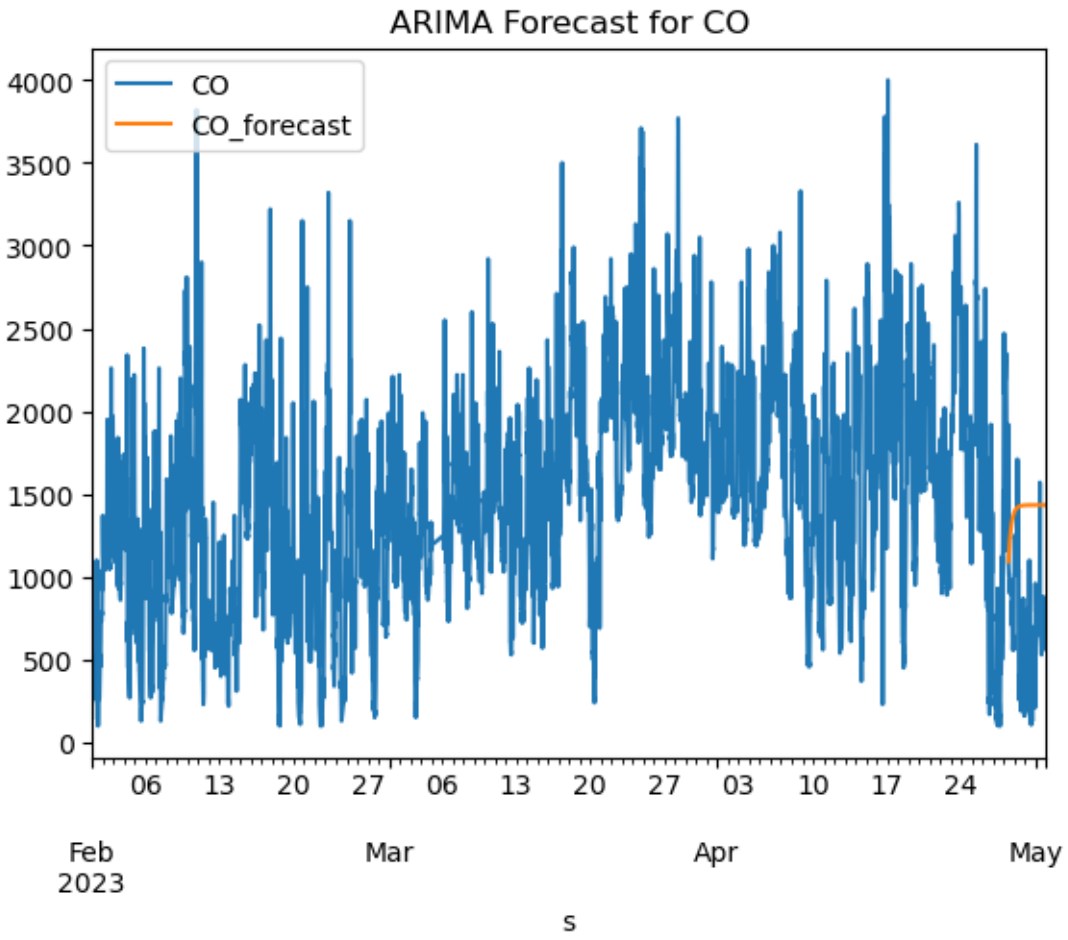
```
Evaluation metrics for PM2.5:
MAE: 58.56318025334853
MAPE: 6.613458088696565
RMSE: 60.879954125620685
```

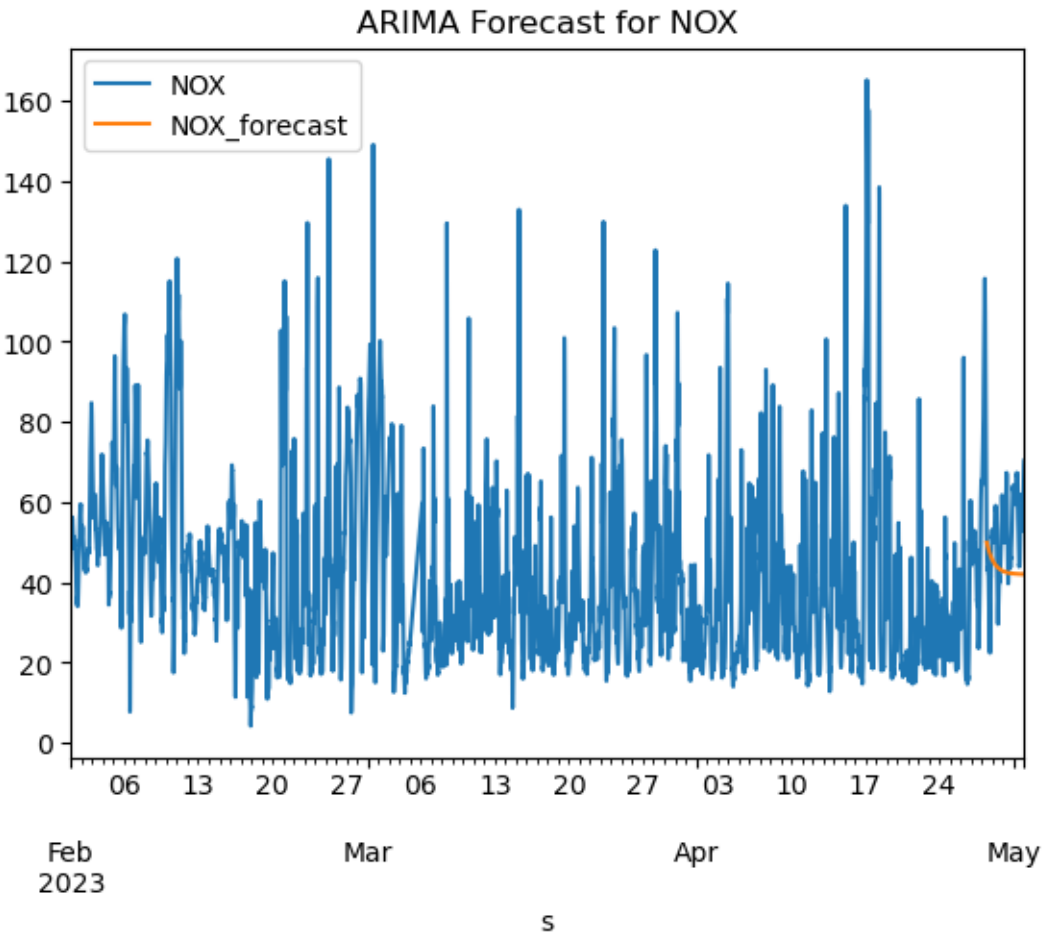
```
/var/folders/h1/m9dw6pns1tgclwkz5gxqn9hc0000gn/T/ipykernel_62106/1464582754.py:3
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column + '_forecast'].iloc[train_entries:] = forecast
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
```

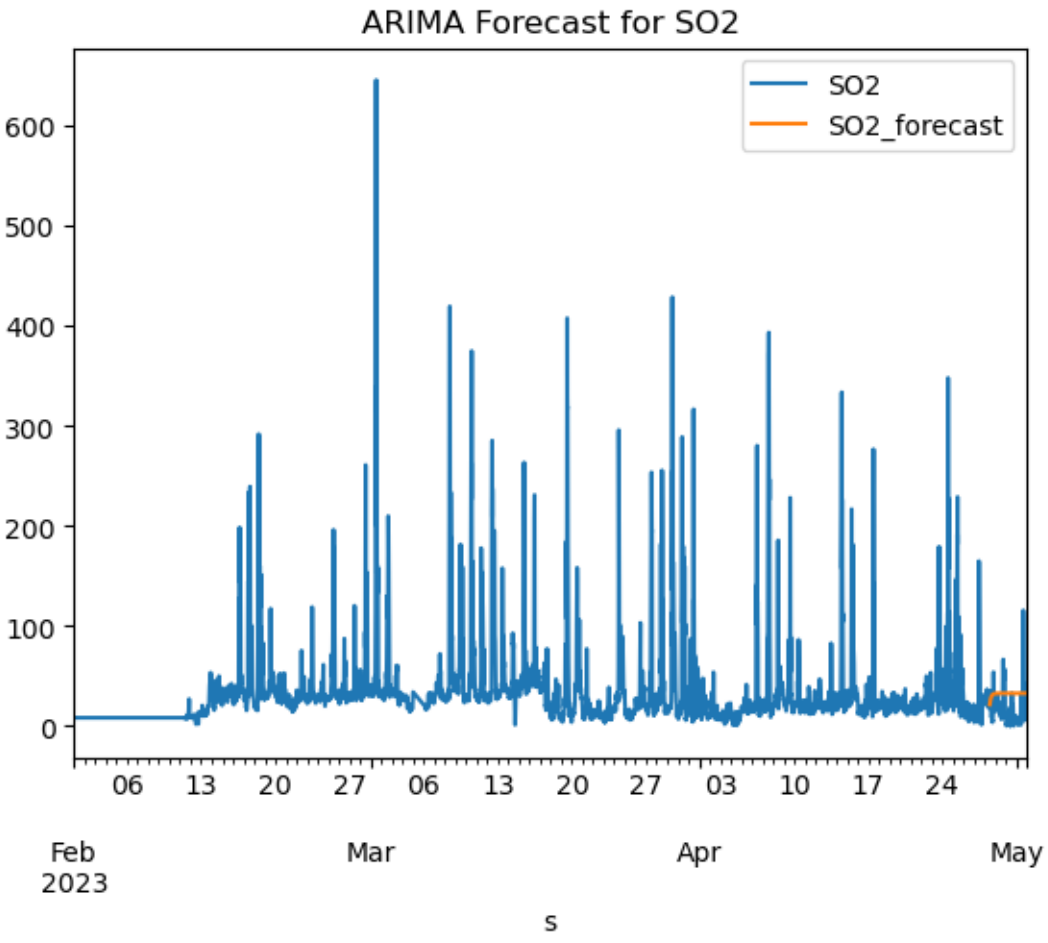
```
Evaluation metrics for Ozone:
MAE: 10.957468314843684
MAPE: 0.8989928426507389
RMSE: 14.33328345894768
```

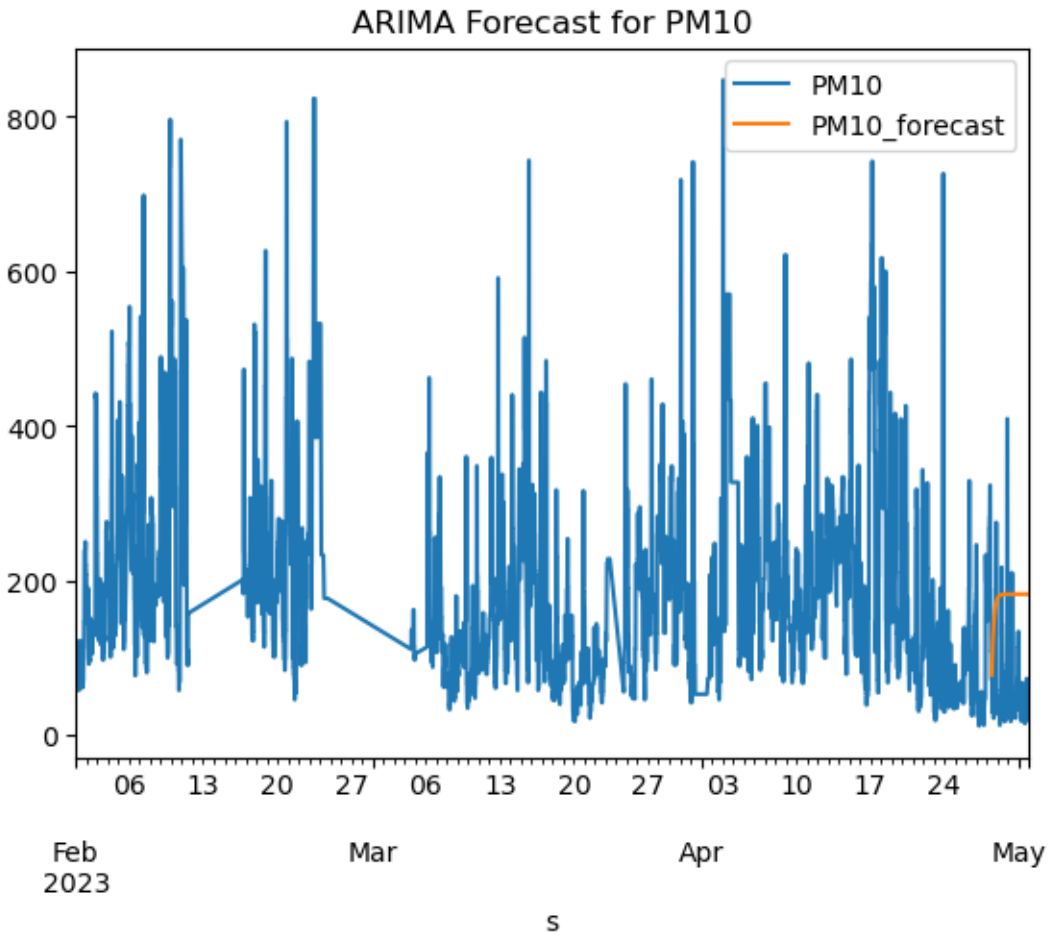
```
/var/folders/h1/m9dw6pns1tgclwkz5gxqn9hc0000gn/T/ipykernel_62106/1464582754.py:3
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

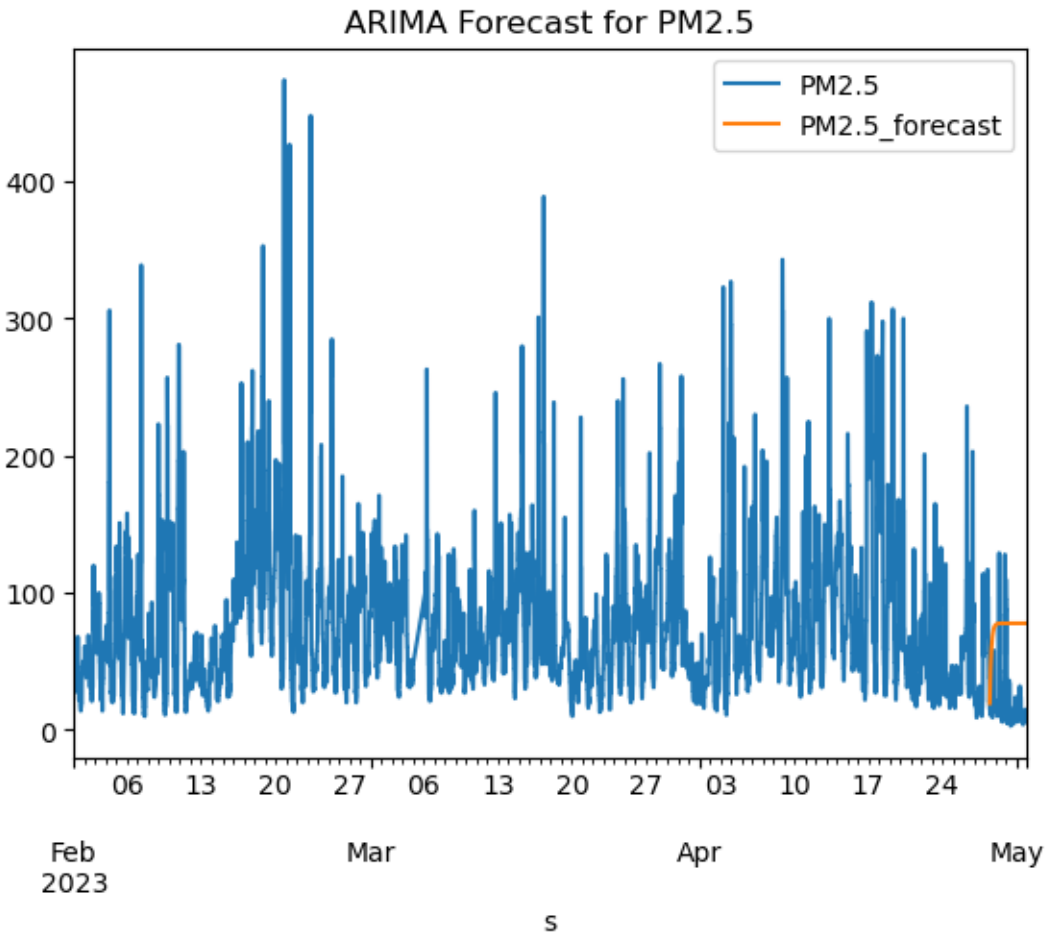
```
  df[column + '_forecast'].iloc[train_entries:] = forecast
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency 15T will be used.
  self._init_dates(dates, freq)

Evaluation metrics for NO:
MAE: 6.644349254428599
MAPE: 1.8175392830260781
RMSE: 7.802545437235278


/var/folders/h1/m9dw6pns1tgclwkz5gxqn9hc0000gn/T/ipykernel_62106/1464582754.py:3
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column + '_forecast'].iloc[train_entries:] = forecast
```

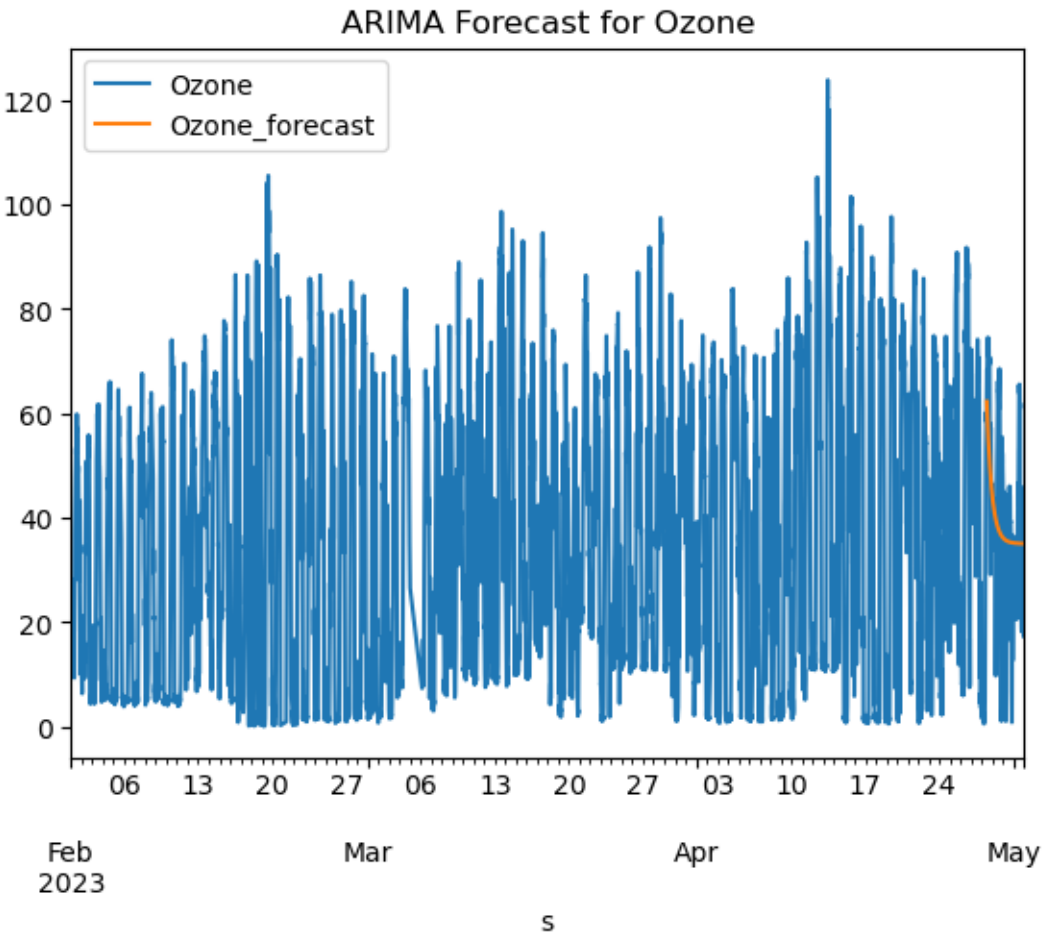

ARIMA Forecast for CO
CO
CO_forecast
4000
3500
3000
2500
2000
1500
1000
500
0
06
13
20
27
06
13
20
27
03
10
17
24
Feb
2023
Mar
Apr
May
s

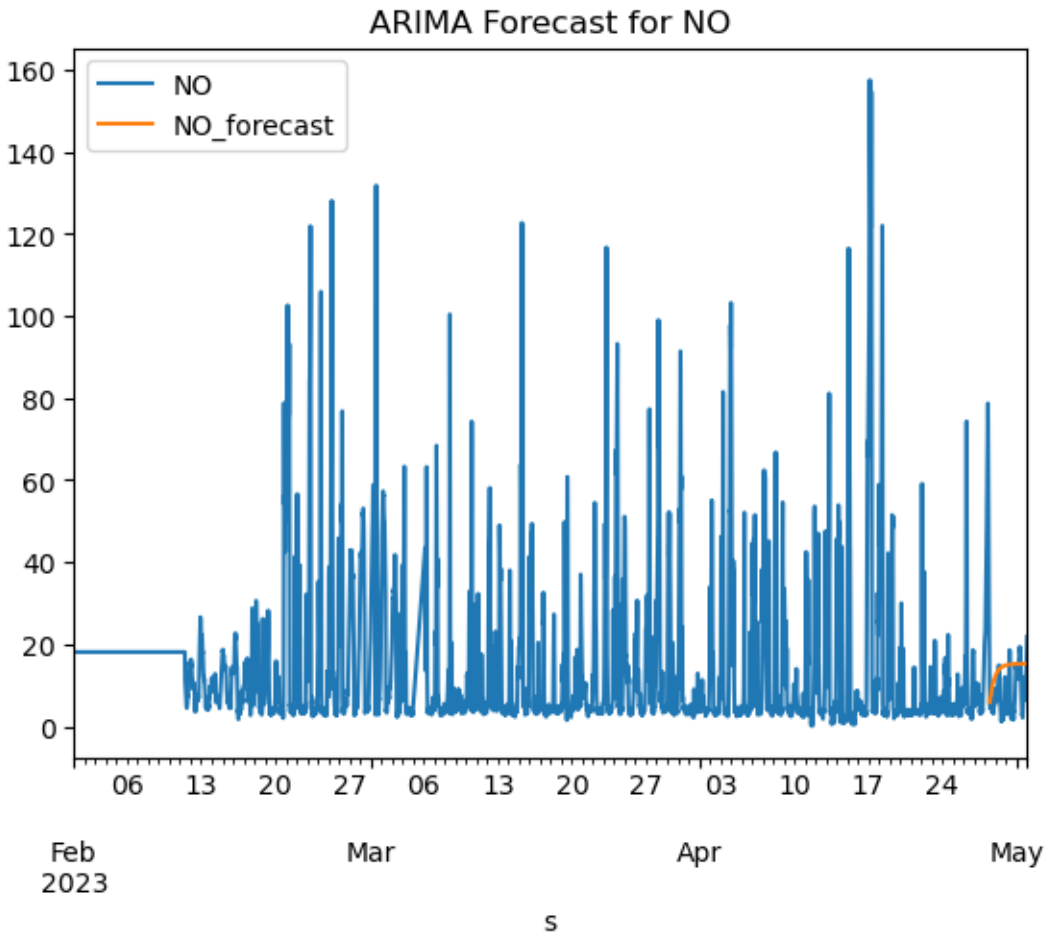

ARIMA Forecast for NOX
NOX
NOX_forecast
160
140
120
100
80
60
40
20
0
06
13
20
27
06
13
20
27
03
10
17
24
Feb
2023
Mar
Apr
May
s


ARIMA Forecast for SO2
SO2
SO2_forecast
600
500
400
300
200
100
0
06
13
20
27
06
13
20
27
03
10
17
24
Feb
2023
Mar
Apr
May
s


ARIMA Forecast for PM10
PM10
PM10_forecast
800
600
400
200
0
06
13
20
27
06
13
20
27
03
10
17
24
Feb
2023
Mar
Apr
May
s


ARIMA Forecast for PM2.5
PM2.5
PM2.5_forecast
400
300
200
100
0
06
13
20
27
06
13
20
27
03
10
17
24
Feb
2023
Mar
Apr
May
s


ARIMA Forecast for Ozone
Ozone
Ozone_forecast
120
100
80
60
40
20
0
06
13
20
27
06
13
20
27
03
10
17
24
Feb
2023
Mar
Apr
May
s




as you can see it is succefully forecasting for shorter period

[ ]: