EE798_Assignment_1

June 27, 2023

Name: Omkar Deepak Chavan

Roll No.: 210280

0.1 Problem Statement:

Analysing the time series data of pollutants' concentrations of Singrauli Coalfield to find blasting time and trends among the data.

The air pollution data set was obtained from the Singrauli Coalfield Pollution Control Board for coal India's (Singrauli Coalfield). The pollution was monitored during open-pit blasting. There are 13 columns overall in the air pollution data collection of pollutants that are available at intervals of 15 minutes.

The project is divided into following parts:

1) Data Exploratory Analysis(Exploring the data)
2) Forecasting (Prediction of data from historical data using methods like ARIMA)
3) Finding Combined Weighted Combination of air polluting factors
4) Finding blasting time and relevant analysis
5) Curve Fitting

[1]:
```
!pip install pmdarima
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pmdarima in c:\programdata\anaconda3\lib\site-
packages (2.0.3)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.29.35)
Requirement already satisfied: urllib3 in c:\programdata\anaconda3\lib\site-
packages (from pmdarima) (1.26.14)
Requirement already satisfied: numpy>=1.21.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.23.5)
Requirement already satisfied: scikit-learn>=0.22 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.2.1)
Requirement already satisfied: joblib>=0.11 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.1.1)
```

```
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (65.6.3)
Requirement already satisfied: pandas>=0.19 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.5.3)
Requirement already satisfied: scipy>=1.3.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.10.0)
Requirement already satisfied: statsmodels>=0.13.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.13.5)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2022.7)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.22->pmdarima)
(2.2.0)
Requirement already satisfied: packaging>=21.3 in
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(22.0)
Requirement already satisfied: patsy>=0.5.2 in
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(0.5.3)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages
(from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
```

[2]:
```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import pandas.plotting
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
%matplotlib inline
```

[3]:
```
file_path = 'C:/Users/Omkar/Desktop/EE798Q/Open pit blasting 01-02-2023 000000␣
↪To 01-05-2023 235959.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path , index_col=0)
```

[4]:
```
df.head()
```

[4]:
```
                   From   To (Interval: 15M)  \
#
1  2023-02-01 00:00:00  2023-02-01 00:15:00
2  2023-02-01 00:15:00  2023-02-01 00:30:00
3  2023-02-01 00:30:00  2023-02-01 00:45:00
```

```
4  2023-02-01 00:45:00  2023-02-01 01:00:00
5  2023-02-01 01:00:00  2023-02-01 01:15:00

   Singrauli, Surya Kiran Bhawan Dudhichua  PM10 (µg/m3)  \
#
1                                                   95.0
2                                                   95.0
3                                                   95.0
4                                                  122.0
5                                                  122.0

   Singrauli, Surya Kiran Bhawan Dudhichua  PM2.5 (µg/m3)  \
#
1                                                   35.0
2                                                   35.0
3                                                   35.0
4                                                   34.0
5                                                   34.0

   Singrauli, Surya Kiran Bhawan Dudhichua  NO (µg/m3)  \
#
1                                                    NaN
2                                                    NaN
3                                                    NaN
4                                                    NaN
5                                                    NaN

   Singrauli, Surya Kiran Bhawan Dudhichua  NO2 (µg/m3)  \
#
1                                                   90.1
2                                                   88.0
3                                                   87.7
4                                                   88.9
5                                                   90.0

   Singrauli, Surya Kiran Bhawan Dudhichua  NOX (ppb)  \
#
1                                                   56.2
2                                                   55.1
3                                                   55.2
4                                                   55.7
5                                                   55.8

   Singrauli, Surya Kiran Bhawan Dudhichua  CO (mg/m3)  \
#
1                                                   0.31
2                                                   0.33
```

```
3                                               0.38
4                                               0.38
5                                               0.38

   Singrauli, Surya Kiran Bhawan Dudhichua  SO2 (µg/m3)  \
#
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN
5                                                NaN

   Singrauli, Surya Kiran Bhawan Dudhichua  NH3 (µg/m3)  \
#
1                                               17.7
2                                               18.3
3                                               19.7
4                                               21.3
5                                               22.3

   Singrauli, Surya Kiran Bhawan Dudhichua  Ozone (µg/m3)  \
#
1                                               28.1
2                                               27.1
3                                               24.9
4                                               21.9
5                                               16.7

   Singrauli, Surya Kiran Bhawan Dudhichua  Benzene (µg/m3)
#
1                                                0.4
2                                                0.4
3                                                0.4
4                                                0.4
5                                                0.4
```

[5]:
```
df.info()
```

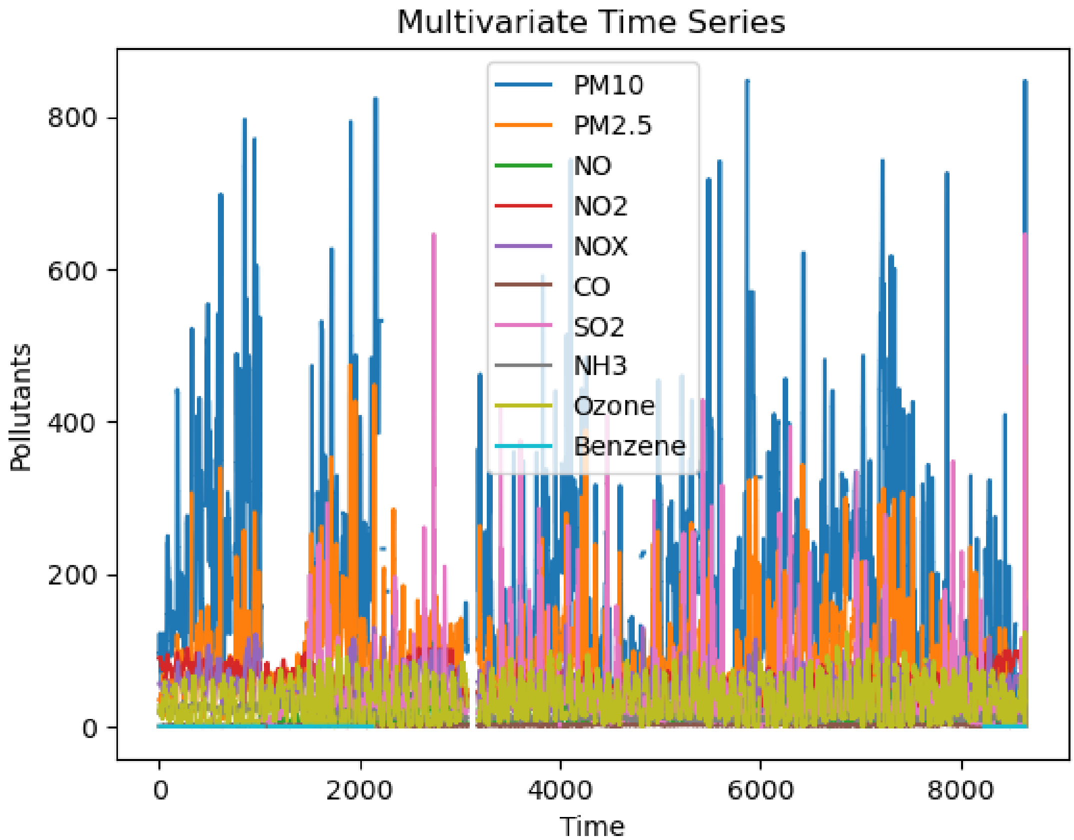
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8643 entries, 1 to 8643
Data columns (total 12 columns):
 #   Column                                                          Non-Null Count
Dtype
---  ------                                                          --------------
-----
 0   From                                                            8643 non-null
object
 1   To (Interval: 15M)                                              8640 non-null
```

```
object
 2   Singrauli, Surya Kiran Bhawan Dudhichua  PM10 (µg/m3)     6962 non-null
float64
 3   Singrauli, Surya Kiran Bhawan Dudhichua  PM2.5 (µg/m3)    8417 non-null
float64
 4   Singrauli, Surya Kiran Bhawan Dudhichua  NO (µg/m3)       7274 non-null
float64
 5   Singrauli, Surya Kiran Bhawan Dudhichua  NO2 (µg/m3)      8227 non-null
float64
 6   Singrauli, Surya Kiran Bhawan Dudhichua  NOX (ppb)        8228 non-null
float64
 7   Singrauli, Surya Kiran Bhawan Dudhichua  CO (mg/m3)       8147 non-null
float64
 8   Singrauli, Surya Kiran Bhawan Dudhichua  SO2 (µg/m3)      7192 non-null
float64
 9   Singrauli, Surya Kiran Bhawan Dudhichua  NH3 (µg/m3)      8317 non-null
float64
 10  Singrauli, Surya Kiran Bhawan Dudhichua  Ozone (µg/m3)    8190 non-null
float64
 11  Singrauli, Surya Kiran Bhawan Dudhichua  Benzene (µg/m3)  2448 non-null
float64
dtypes: float64(10), object(2)
memory usage: 877.8+ KB
```

[6]:
```
# Simplify column names
df.columns = ['from', 'to', 'PM10', 'PM2.5',␣
↪'NO','NO2','NOX','CO','SO2','NH3','Ozone','Benzene']
```

[7]:
```
df.plot()
plt.xlabel('Time')
plt.ylabel('Pollutants')
plt.title('Multivariate Time Series')
```

[7]:
```
Text(0.5, 1.0, 'Multivariate Time Series')
```



[8]:
```
# deleting to column as we need only one timestamp column for to be index and
↪we choose it to be from column
df = df.drop('to', axis=1)
```

[9]:
```
# removing last 3 rows as they contaion max , min , avg data instead of actual
↪observations
df = df.iloc[:-3]
df.tail()
```

[9]:
```
                    from  PM10  PM2.5    NO    NO2   NOX    CO   SO2   NH3  \
#
8636  2023-05-01 22:45:00  19.0   11.0  17.9  100.0  67.8  0.63  10.0  10.7
8637  2023-05-01 23:00:00  19.0   11.0  17.9  100.0  67.7  0.57  10.0  10.4
8638  2023-05-01 23:15:00  19.0   11.0  19.6  100.2  69.2  0.58   9.9  10.5
8639  2023-05-01 23:30:00  19.0   11.0  20.8  100.2  70.2  0.58   9.5  10.8
8640  2023-05-01 23:45:00  32.0    6.0  21.8   98.8  70.3   NaN   NaN  11.0

      Ozone  Benzene
#
8636   26.1      0.1
```

```
8637   30.9      0.1
8638   29.6      0.1
8639   30.0      0.1
8640   33.5      0.1
```

[10]:
```python
# conveting timestamp as a string object into a datetime numerical
date_format = '%Y-%m-%d %H:%M:%S'

# Convert the 'from' column to numerical datetime representation
df['from'] = pd.to_datetime(df['from'], format=date_format)
```

[11]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8640 entries, 1 to 8640
Data columns (total 11 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   from     8640 non-null   datetime64[ns]
 1   PM10     6959 non-null   float64
 2   PM2.5    8414 non-null   float64
 3   NO       7271 non-null   float64
 4   NO2      8224 non-null   float64
 5   NOX      8225 non-null   float64
 6   CO       8144 non-null   float64
 7   SO2      7189 non-null   float64
 8   NH3      8314 non-null   float64
 9   Ozone    8187 non-null   float64
 10  Benzene  2445 non-null   float64
dtypes: datetime64[ns](1), float64(10)
memory usage: 810.0 KB
```

[12]:
```python
# set datetime "from" column as an index column
df.set_index('from', inplace=True)
df.head()
```

[12]:
```
                      PM10  PM2.5  NO   NO2   NOX    CO  SO2   NH3  Ozone  \
from
2023-02-01 00:00:00   95.0   35.0 NaN  90.1  56.2  0.31  NaN  17.7   28.1
2023-02-01 00:15:00   95.0   35.0 NaN  88.0  55.1  0.33  NaN  18.3   27.1
2023-02-01 00:30:00   95.0   35.0 NaN  87.7  55.2  0.38  NaN  19.7   24.9
2023-02-01 00:45:00  122.0   34.0 NaN  88.9  55.7  0.38  NaN  21.3   21.9
2023-02-01 01:00:00  122.0   34.0 NaN  90.0  55.8  0.38  NaN  22.3   16.7

                     Benzene
from
2023-02-01 00:00:00      0.4
2023-02-01 00:15:00      0.4
```

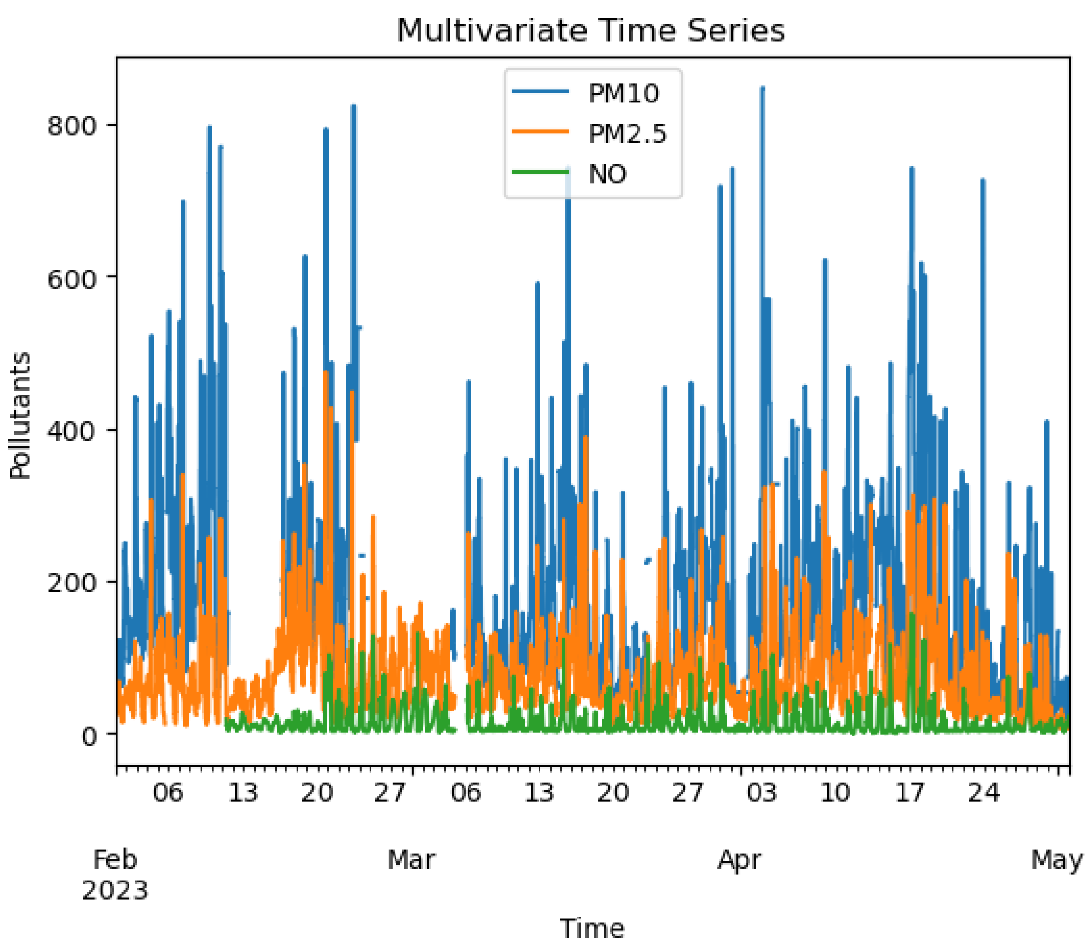
```
2023-02-01 00:30:00      0.4
2023-02-01 00:45:00      0.4
2023-02-01 01:00:00      0.4
```

1 Part 1: Exploring the data

In this part of analysis, we look at all the variables(pollutants here) all at a time.

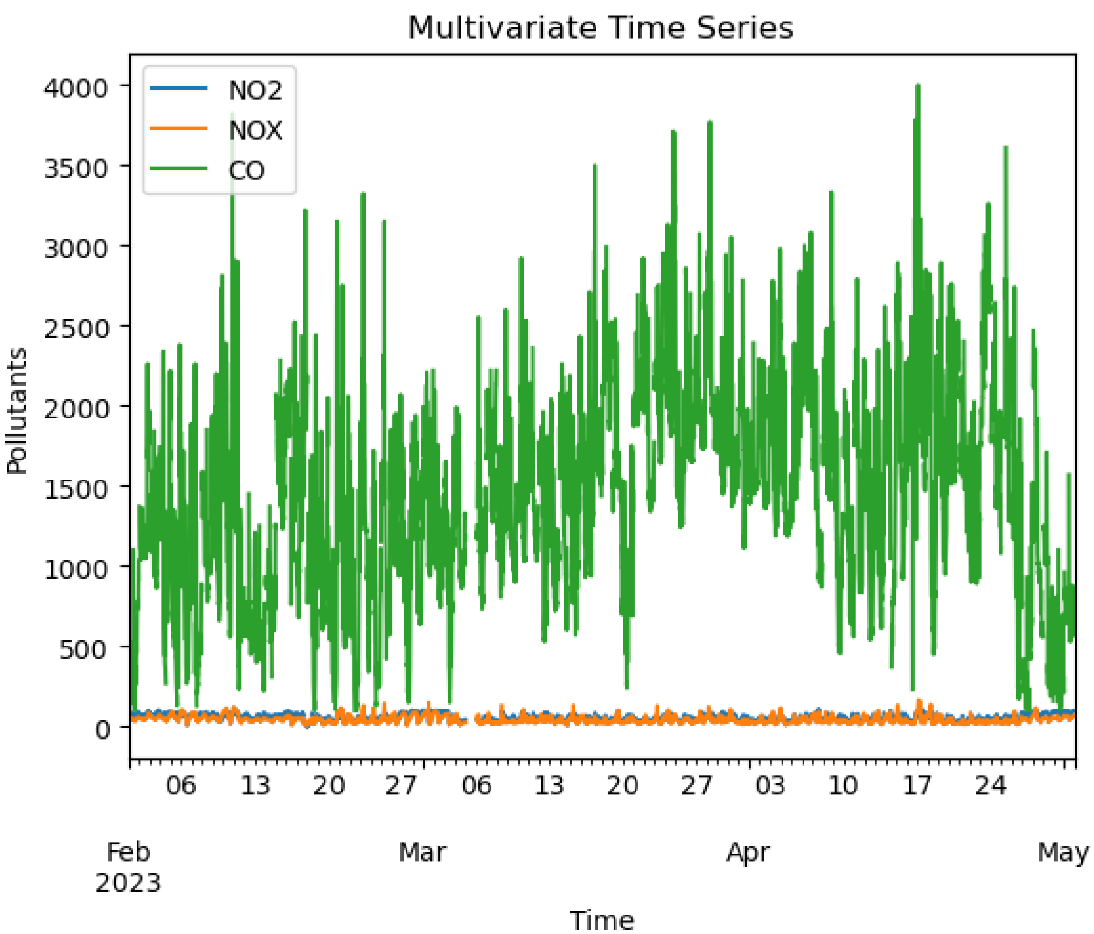
[13]:
```python
# 1. Plotting several columns and finding out which ones affect more.
# code to plot several columns at a time

columns_to_plot=['PM10','PM2.5', 'NO']

data_to_plot = df[columns_to_plot]
data_to_plot.plot()
plt.xlabel('Time')
plt.ylabel('Pollutants')
plt.title('Multivariate Time Series')
```

[13]:
```
Text(0.5, 1.0, 'Multivariate Time Series')
```



[14]:
```
columns_to_plot=['NO2','NOX','CO']
df['CO']*=1000
data_to_plot = df[columns_to_plot]
data_to_plot.plot()
plt.xlabel('Time')
plt.ylabel('Pollutants')
plt.title('Multivariate Time Series')
```

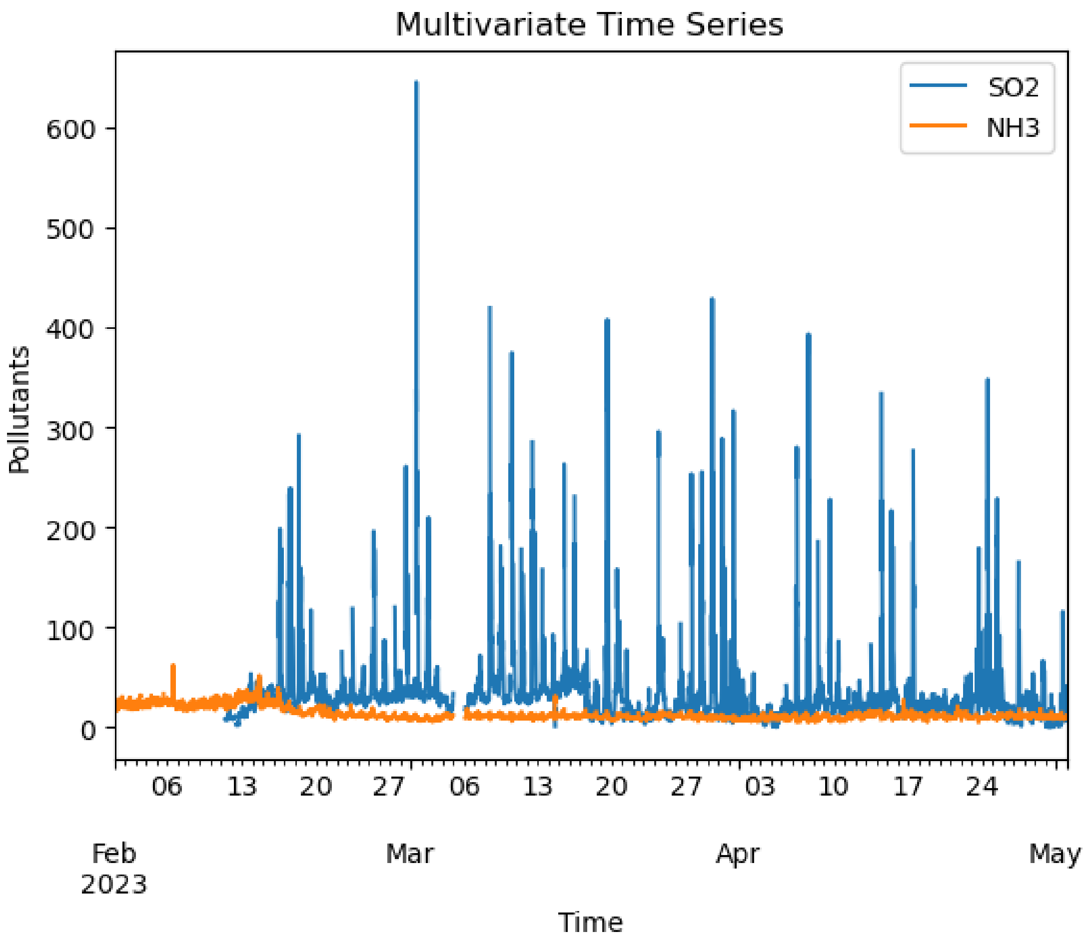
[14]:
```
Text(0.5, 1.0, 'Multivariate Time Series')
```



From Multivariate plot we can observe that some of the variables have significant concentration in the pollutants whereas some pollutants have comparatoively less concentration (such as Benzene and NH3).

[15]:
```
columns_to_plot=['SO2','NH3']

data_to_plot = df[columns_to_plot]
data_to_plot.plot()
plt.xlabel('Time')
plt.ylabel('Pollutants')
plt.title('Multivariate Time Series')
```

[15]: Text(0.5, 1.0, 'Multivariate Time Series')

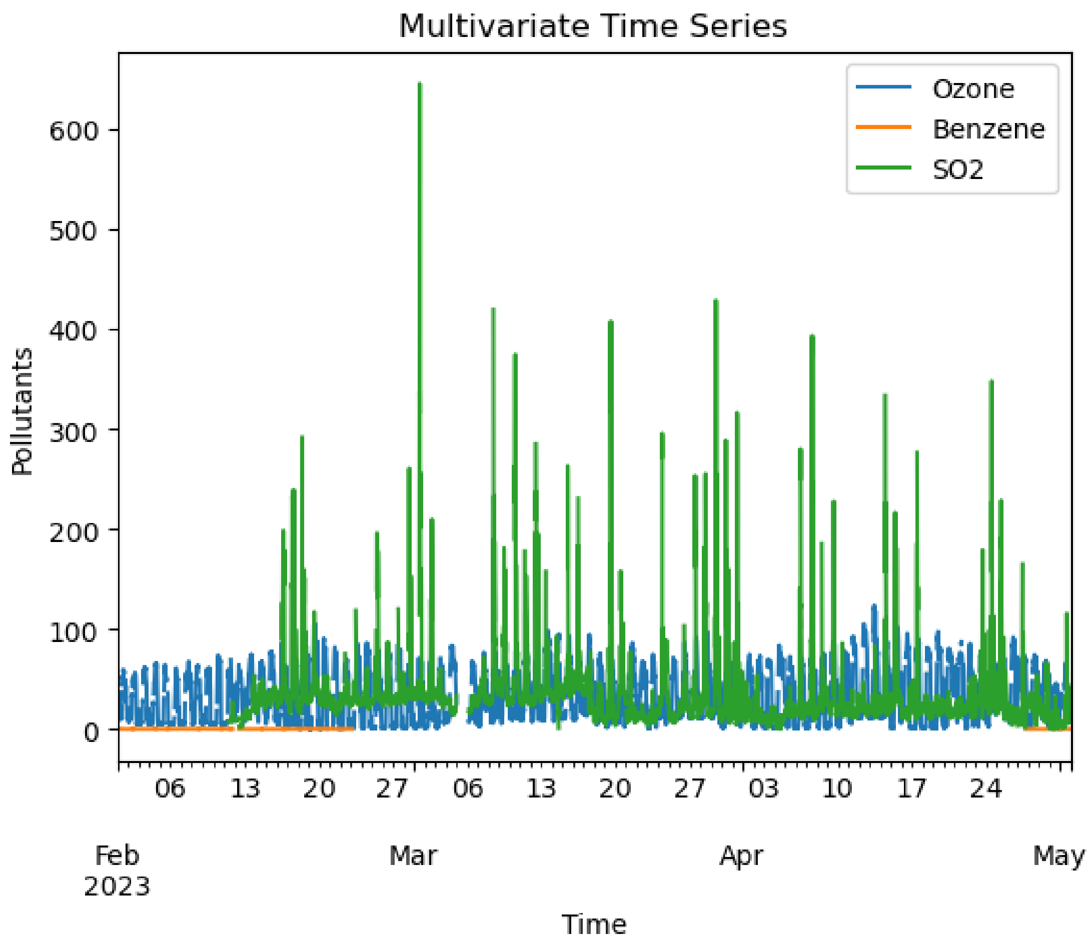


We can observe that NH3 has very less concentration as compared to that of SO2.

[16]:
```
columns_to_plot=['Ozone','Benzene','SO2']

data_to_plot = df[columns_to_plot]
data_to_plot.plot()
plt.xlabel('Time')
plt.ylabel('Pollutants')
```

```python
plt.title('Multivariate Time Series')
```

[16]: Text(0.5, 1.0, 'Multivariate Time Series')



We can observe that Benzene has very less concentration as compared to that of Ozone and benzene.

2 Data Preprocessing

As we can see, the dataset has some null values. We will have to clean them by replacing them by nan values and filling them afterwards.So now we have to fix missing values.

Handling missing values 3 ways 1) Replacing with zeroes 2) Replacing with mean 3) Interpolation

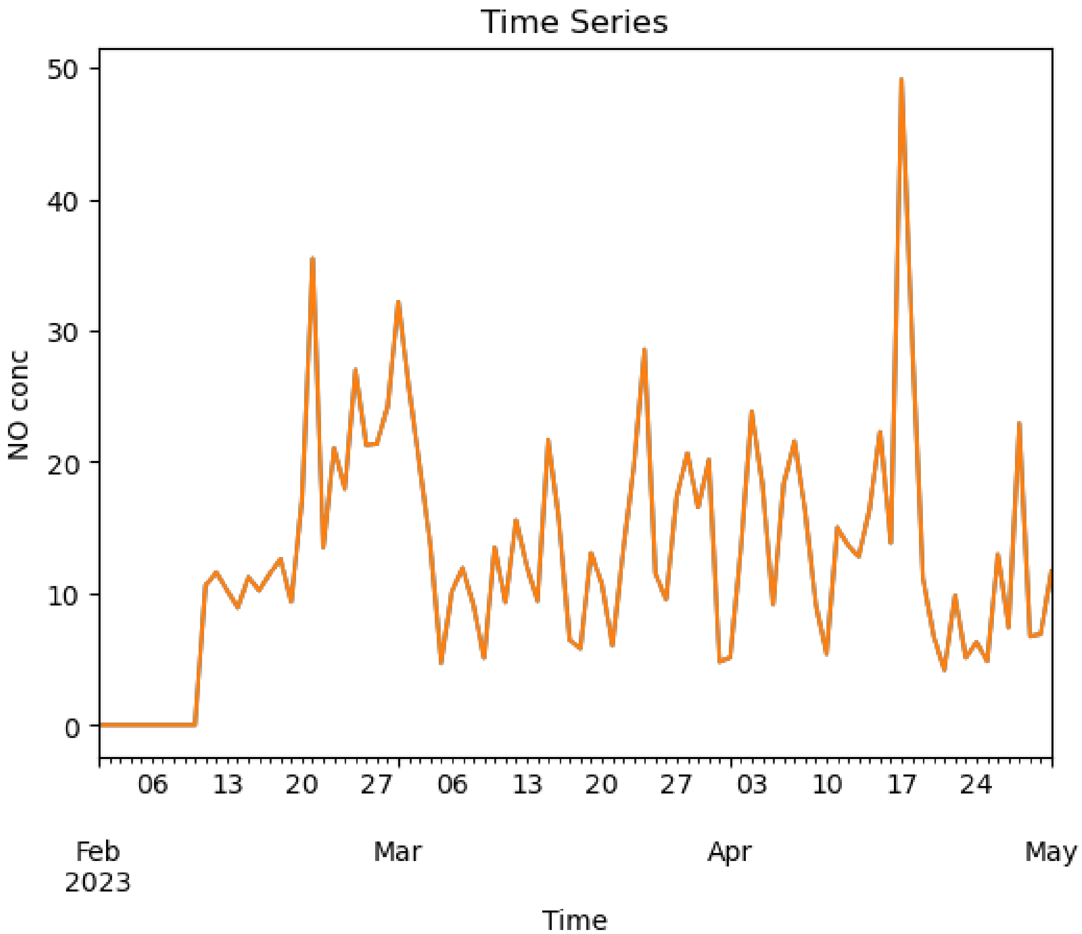
We will test the effect of replacing missing values with each of zero, mean , interpolated data by taking the example of NO dataset.

3 NO Analysis

[17]:
```python
s1= df['NO'].copy()
```

3.1 1) Replacing missing values with zeroes

[18]:
```python
# Resample the time series to a different frequency (e.g., from hourly to daily)



# resampling
s1=s1.resample('D').mean()
s1.fillna(0, inplace=True)
s1.plot()
# s will be dataseries

s1.plot()
plt.xlabel('Time')
plt.ylabel('NO conc')
plt.title('Time Series')
plt.show()
s1.head()
```



[18]:
```
from
2023-02-01    0.0
2023-02-02    0.0
2023-02-03    0.0
2023-02-04    0.0
2023-02-05    0.0
Freq: D, Name: NO, dtype: float64
```

[19]:
```
s1
```

[19]:
```
from
2023-02-01     0.000000
2023-02-02     0.000000
2023-02-03     0.000000
2023-02-04     0.000000
2023-02-05     0.000000
                  …
```

```
2023-04-27     7.424731
2023-04-28    22.978495
2023-04-29     6.743011
2023-04-30     6.916304
2023-05-01    11.701075
Freq: D, Name: NO, Length: 90, dtype: float64
```

Conducting ADF test to check for stationarity of time series data.

[20]:
```python
 from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(s1)
print(f'p-value: {adf_test[1]}')
```

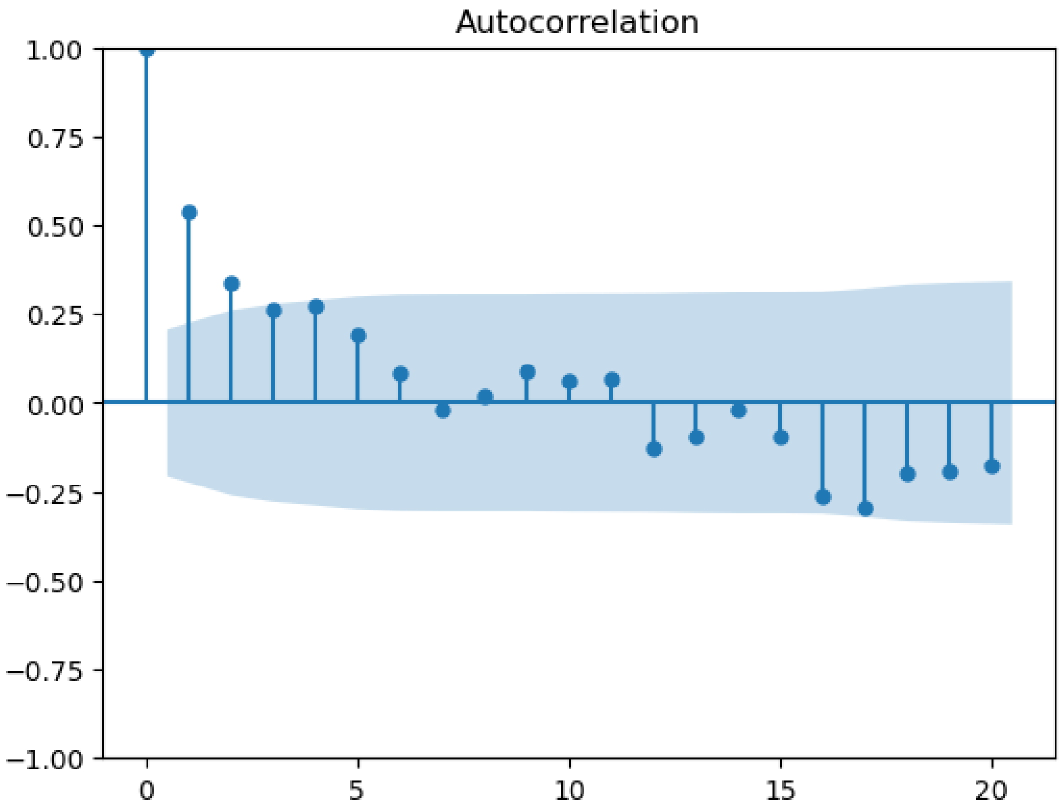
```
p-value: 9.42588901782776e-06
```

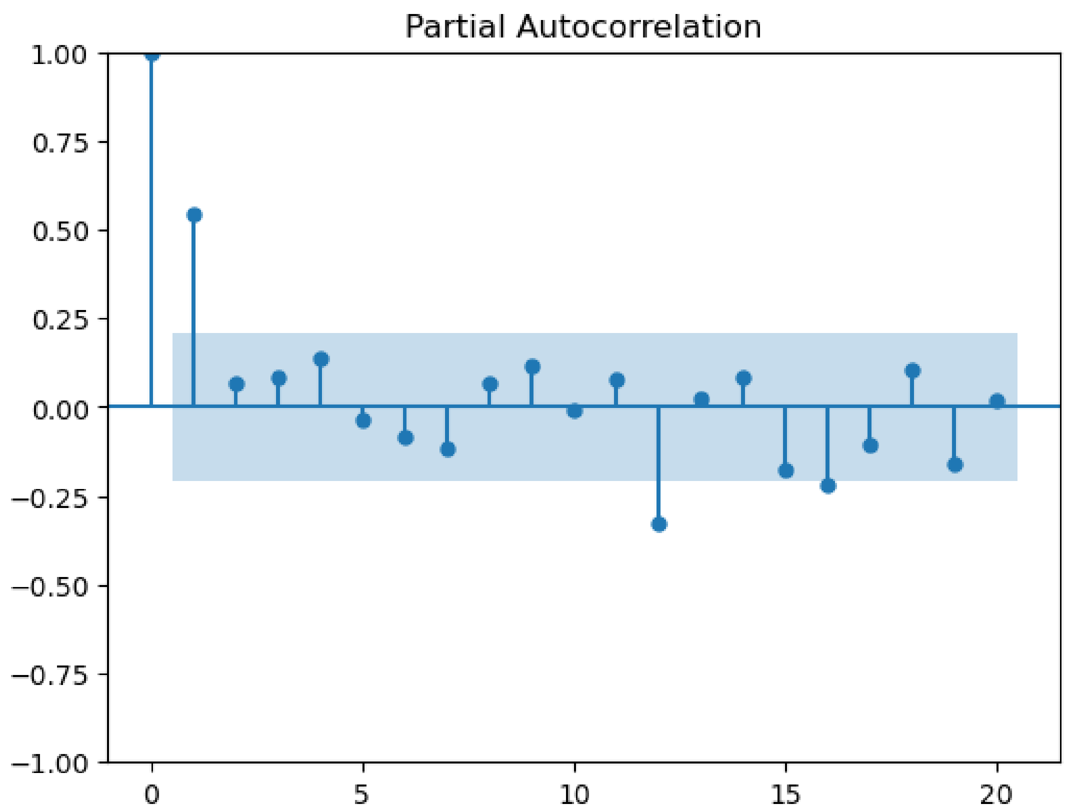
A very low p value implies that it is indeed stationary.

Now we try to attempt ARIMA model on this zero-replaced data.

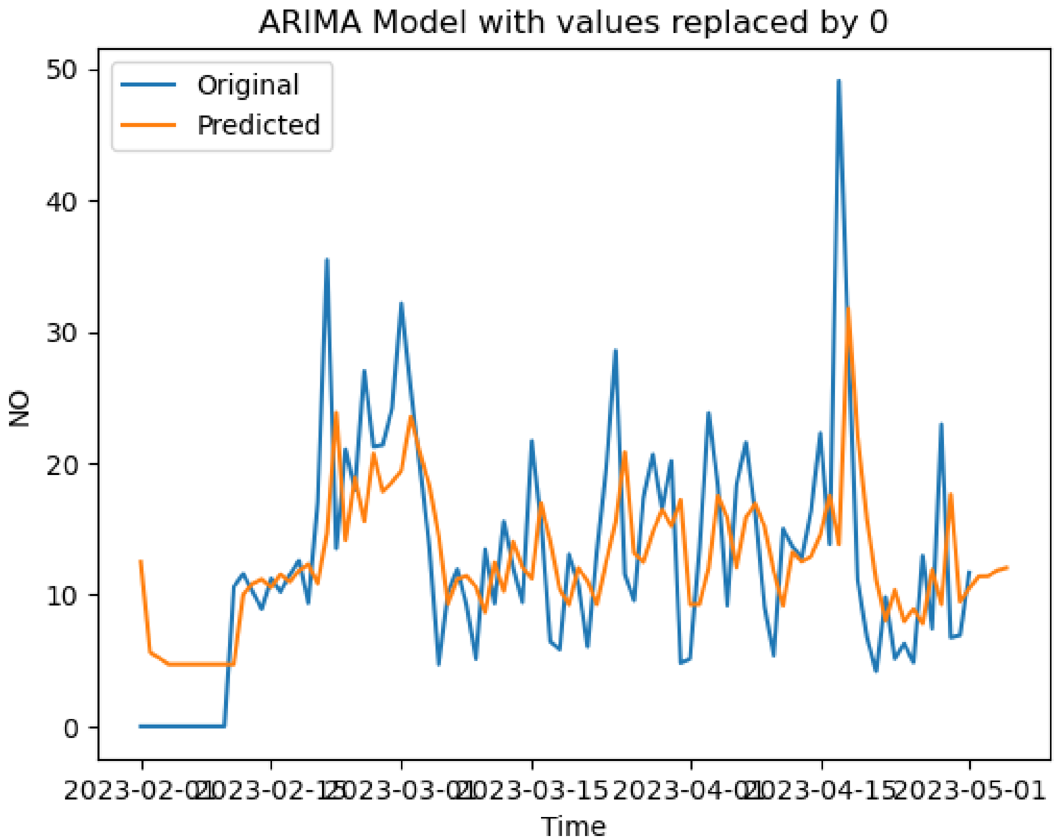
[21]:
```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

acf_original = plot_acf(s1)

pacf_original = plot_pacf(s1)
```

```
C:\ProgramData\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20



[22]:
```
%matplotlib inline
data_series = s1
data_series = pd.Series(data_series)
model = sm.tsa.ARIMA(data_series, order=(3, 0, 0))
result = model.fit()
predictions = result.predict(start='2023-02-01', end='2023-05-05')
plt.plot(data_series, label='Original')
# label indicates color and corresponding value
plt.plot(predictions, label='Predicted')
plt.xlabel('Time')
plt.ylabel('NO')
plt.title('ARIMA Model with values replaced by 0')
plt.legend()
plt.show()
```



We can observe that prediction by ARIMA deviates largely at points where NA values are replaced by zeroes. So, it seems naive to replace missing values with zeroes.

3.2 2) Replacing missing values with mean.

[23]:
```
s2= df['NO'].copy()
```

[24]:
```
# Calculate the mean of non-null values
# Replace missing values with the mean
s2=s2.resample('D').mean()
s2.fillna(14.65, inplace=True)
# 14.65 is the mean of non NA values
s2
```

[24]:
```
from
2023-02-01    14.650000
2023-02-02    14.650000
2023-02-03    14.650000
2023-02-04    14.650000
2023-02-05    14.650000
```

```
                   …
2023-04-27     7.424731
2023-04-28    22.978495
2023-04-29     6.743011
2023-04-30     6.916304
2023-05-01    11.701075
Freq: D, Name: NO, Length: 90, dtype: float64
```

Conducting ADF test to check for stationarity of time series data.

[25]:
```python
# just checking code   ADF test to check for stationarity
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(s2)
print(f'p-value: {adf_test[1]}')
```

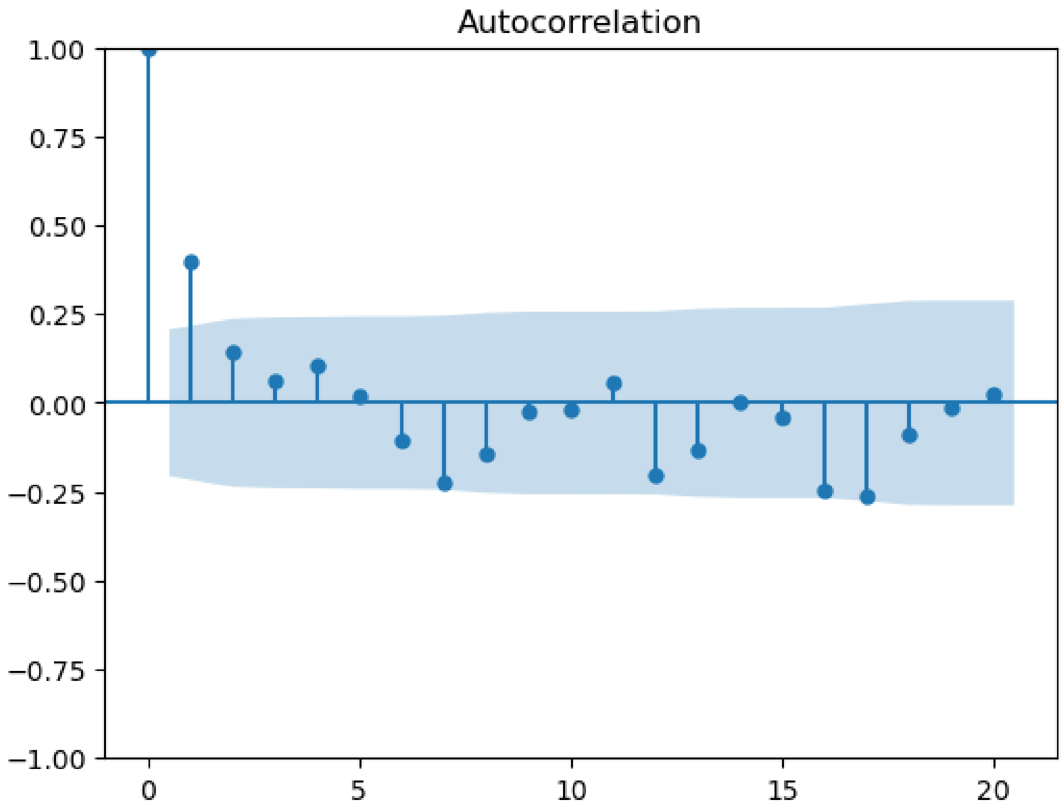
```
p-value: 8.460261214114683e-08
```

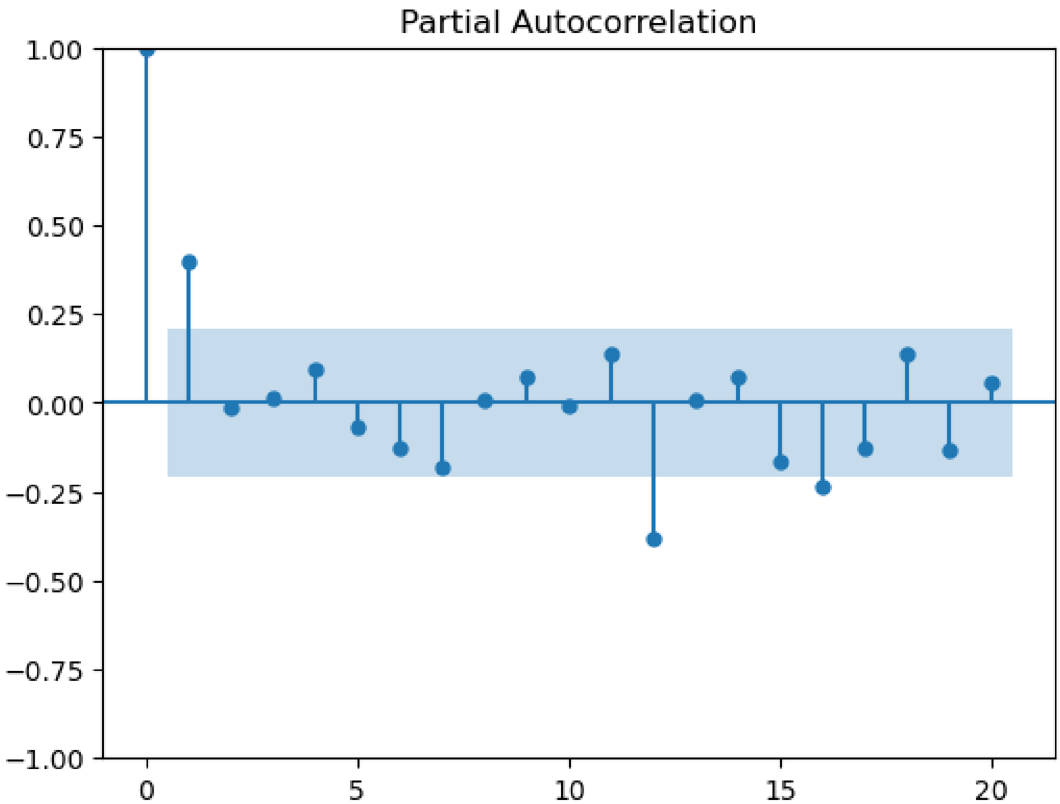
A very low p value implies that it is indeed stationary.

Now we try to attempt ARIMA model on this mean-replaced data.

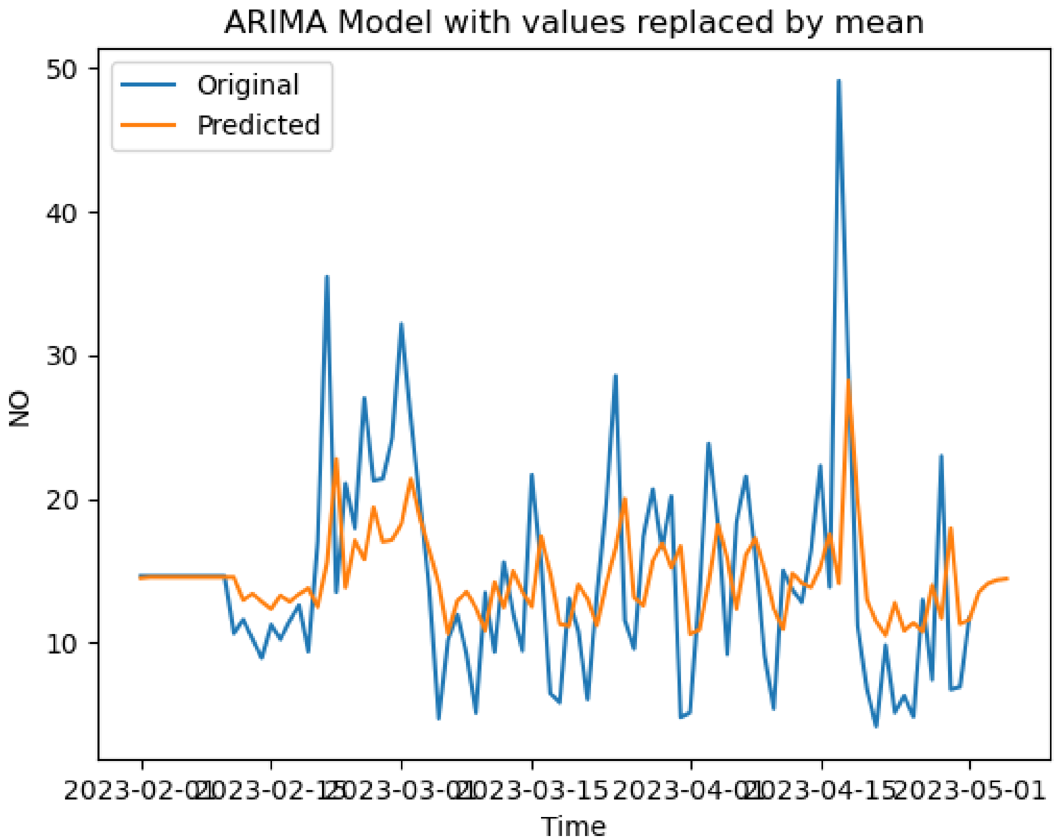
[26]:
```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

acf_original = plot_acf(s2)

pacf_original = plot_pacf(s2)
```

```
C:\ProgramData\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20



[27]:
```
%matplotlib inline
data_series = s2
data_series = pd.Series(data_series)
model = sm.tsa.ARIMA(data_series, order=(2, 0, 0))
result = model.fit()
predictions = result.predict(start='2023-02-01', end='2023-05-05')
plt.plot(data_series, label='Original')
# label indicates color and corresponding value
plt.plot(predictions, label='Predicted')
plt.xlabel('Time')
plt.ylabel('NO')
plt.title('ARIMA Model with values replaced by mean')
plt.legend()
plt.show()
```



We can observe that though it is better to replace with mean as compared to that of zeroes , but there are still abnormalities in the plot. Filling NaNs with the mean value is also not sufficient and naive, and doesn't seem to be a good option.

4 3) Interpolation

Interpolation is used when the intended time(t) falls between the greatest and smallest of the time. There are 3 ways to interpolate the data to replace missing values:

(a) Linear Interpolation: This is basically like connecting two points in a dataset by drawing a line between them.

(b) Cubical Interpolation: It offers true continuity between the segments. As such it requires more than just the two endpoints of the segment but also the two points on either side of them.

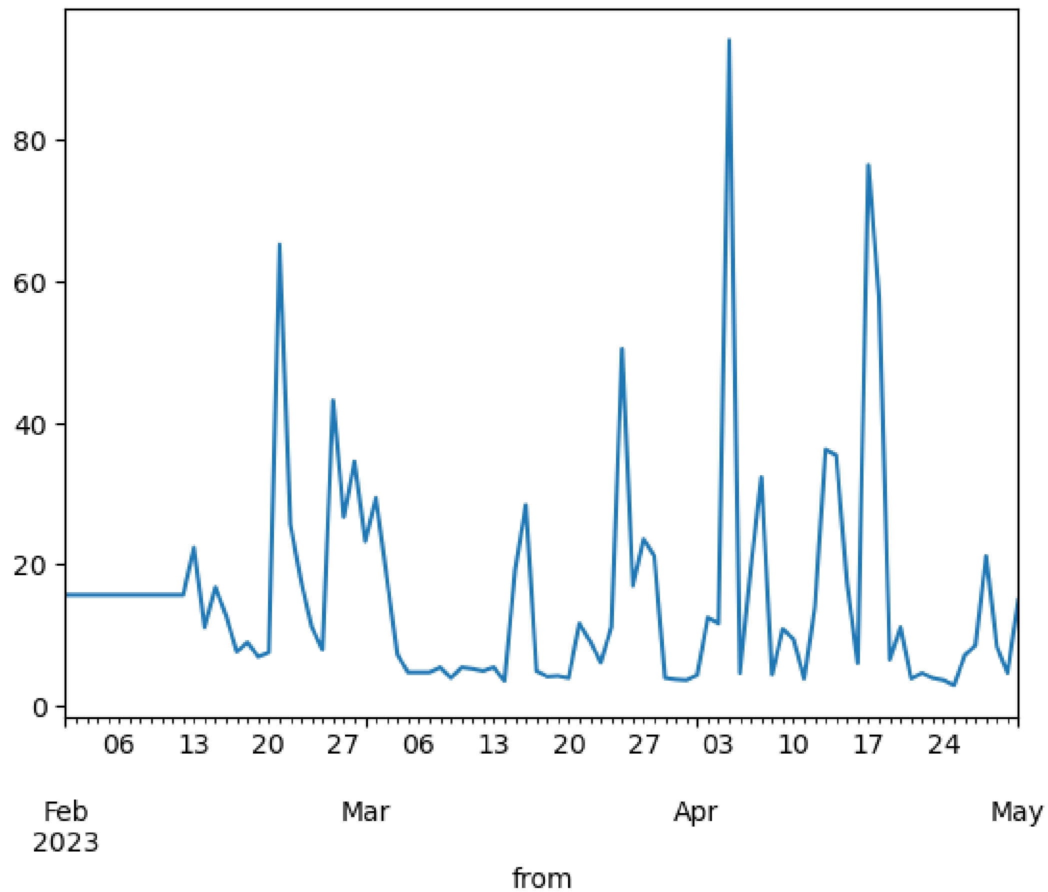
(c) Spline Interpolation: Low-degree polynomials are used in each of the intervals in spline interpolation, which is similar to polynomial interpolation in that it selects the polynomial parts to fit together smoothly. The outcome is a function known as a spline.

Here we will use .interpolate() function to interpolate the data.

4.0.1 1) Linear interpolation

[28]:
```
s3=df['NO'].copy()

# Resample the time series to a different frequency (e.g., from hourly to daily)
s3= s3.resample('D')
s3 = s3.interpolate()
s3.fillna(method='ffill', inplace=True)  # Fill missing values forward
s3.fillna(method='bfill', inplace=True)  # Fill missing values backward
s3.plot()
```

[28]: <Axes: xlabel='from'>

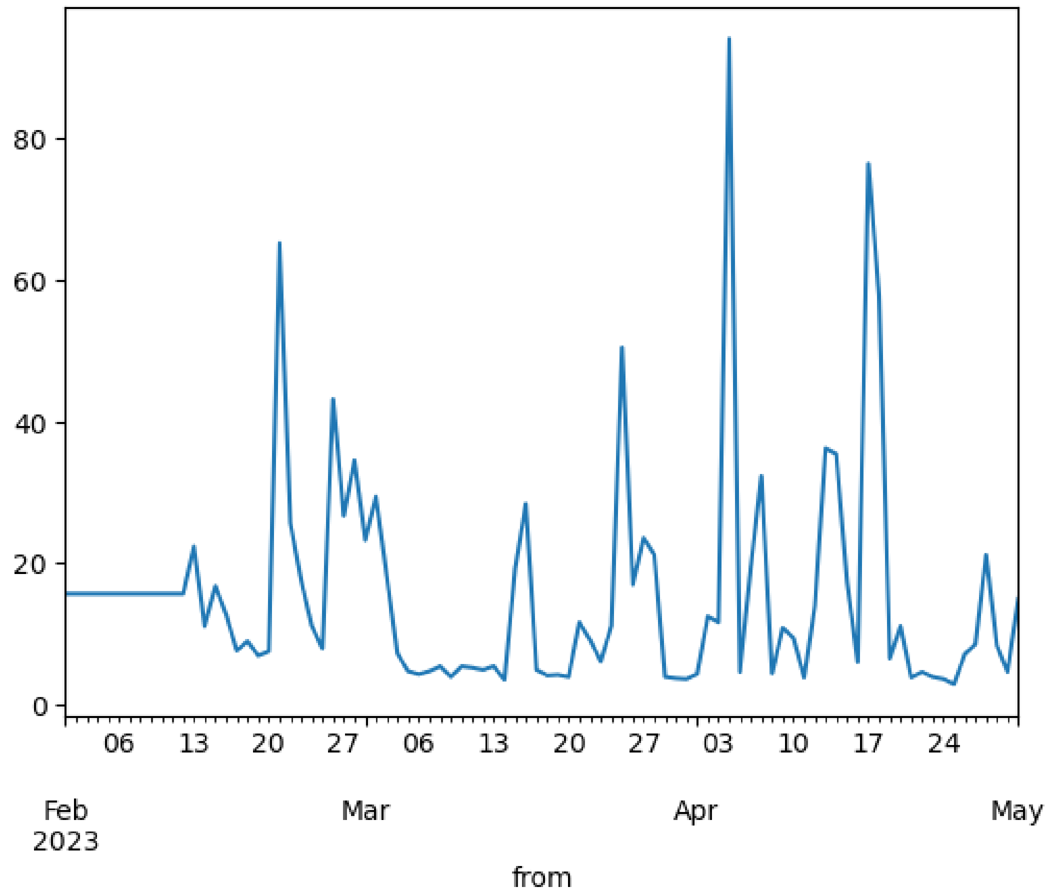


4.0.2 2) Cubical interpolation

[29]:
```
#  cubic interpolation
s4=df['NO'].copy()
```

```
# Resample the time series to a different frequency (e.g., from hourly to daily)
s4= s4.resample('D')
s4 = s4.interpolate(method='cubic')
s4.fillna(method='ffill', inplace=True)  # Fill missing values forward
s4.fillna(method='bfill', inplace=True)  # Fill missing values backward
s4.plot()
```

[29]: <Axes: xlabel='from'>

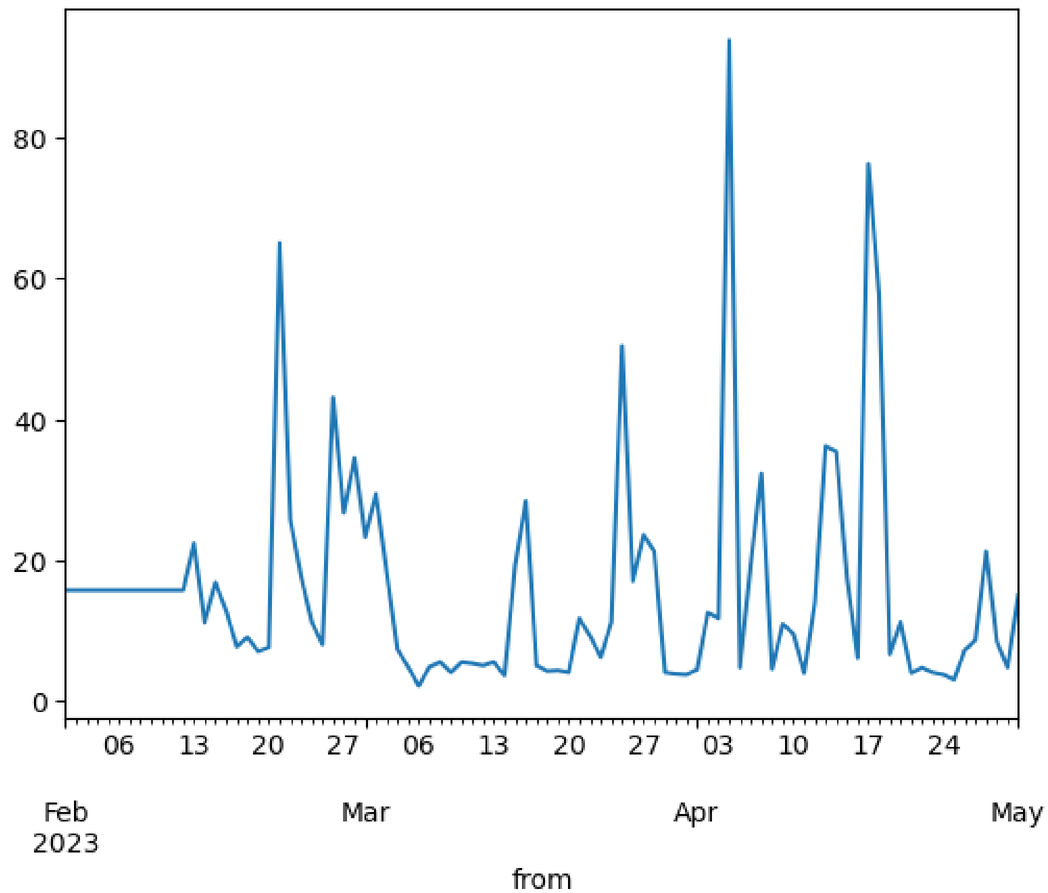


4.0.3 3) Spline Interpolation

[30]:
```
# i) spline interpolation
s5=df['NO'].copy()

# Resample the time series to a different frequency (e.g., from hourly to daily)
s5= s5.resample('D')
s5 = s5.interpolate(method='spline', order=3)
s5.fillna(method='ffill', inplace=True)  # Fill missing values forward
```

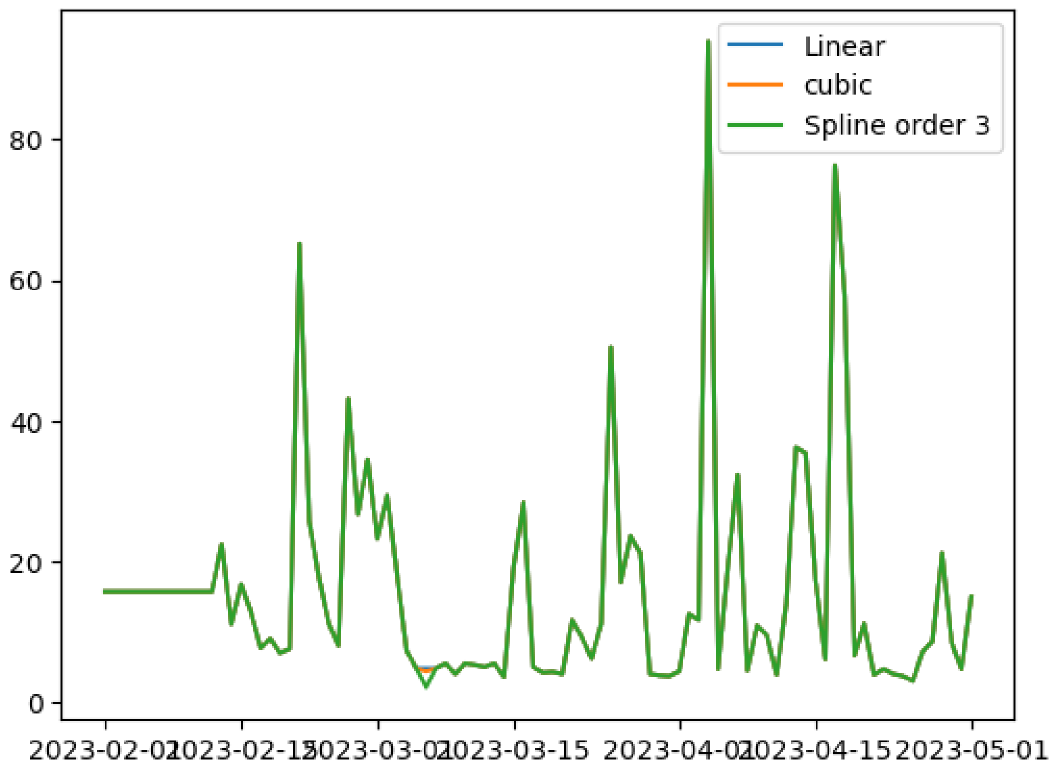
```
s5.fillna(method='bfill', inplace=True)  # Fill missing values backward

s5.plot()
```

[30]: <Axes: xlabel='from'>



Now we will plot all 3 - linear, cubical, spline of order 3 together to see which one is the best option.

[31]:
```
# s3= s3.resample('D')
# s4= s4.resample('D')
# s5= s5.resample('D')
plt.plot(s3, label='Linear')
plt.plot(s4, label='cubic')
plt.plot(s5, label='Spline order 3')
plt.legend()
plt.show()
```



We can observe that spline interpolation of order 3 provides us with smoothest out of all 3 options. So, lets interpolate the missing values with spline interpolation for further analysis.

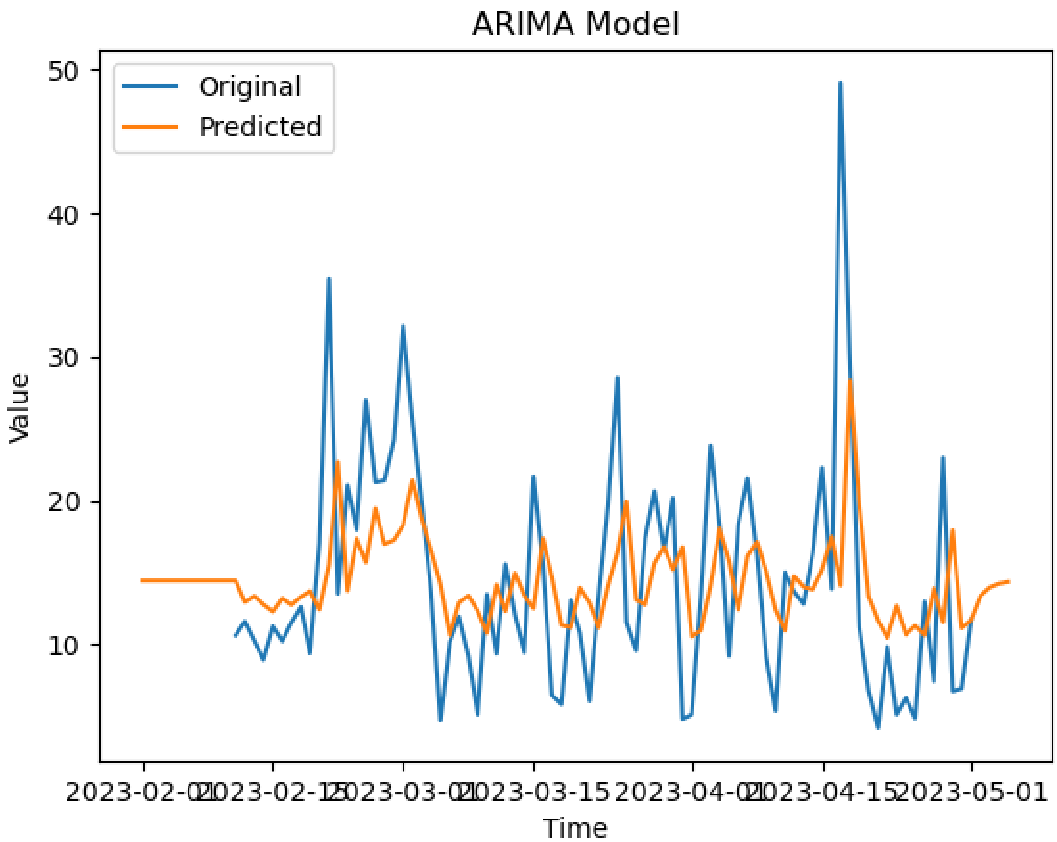
4.1 Interpolation, ARIMA Comparison

First we will attmept ARIMA without interpolation.

[32]:
```
#ARIMA on original
s6=df['NO'].copy()
# Resample the time series to a different frequency (e.g., from hourly to daily)
s6= s6.resample('D').mean()
```

[33]:
```
%matplotlib inline
data_series = s6
data_series = pd.Series(data_series)
model = sm.tsa.ARIMA(data_series, order=(3, 0, 0))
result = model.fit()
predictions = result.predict(start='2023-02-01', end='2023-05-05')
plt.plot(data_series, label='Original')
# label indicates color and corresponding value
plt.plot(predictions, label='Predicted')
# plt.plot(s5, label='spline interpolated')
plt.xlabel('Time')
```

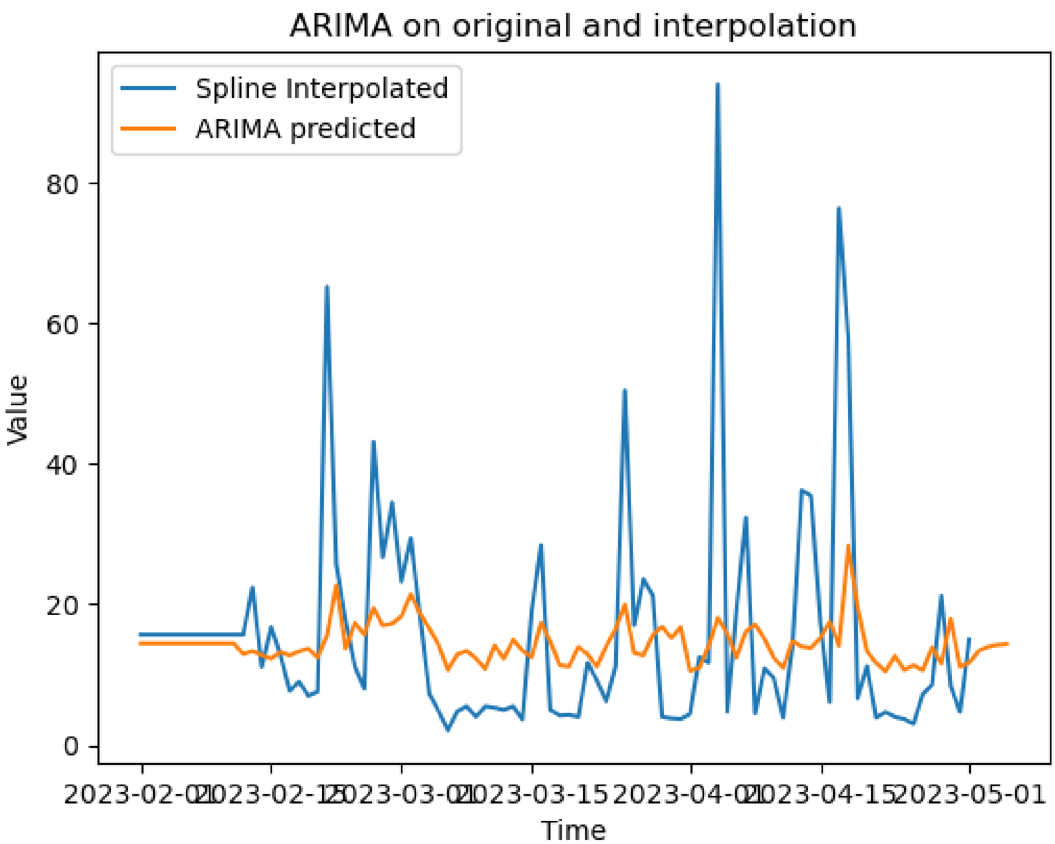
```
plt.ylabel('Value')
plt.title('ARIMA Model')
plt.legend()
plt.show()
```



As we can see we cant apply ARIMA on missing values containing data. We first have to replace missing data with appropriate values.(here we will use interpolation(spline 3rd order))

[34]:
```
#plot s5 and s6 together
plt.plot(s5, label='Spline Interpolated')
plt.plot(predictions, label='ARIMA predicted')



# Set x and y labels, and plot title
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('ARIMA on original and interpolation')

# Add a legend
plt.legend()
```

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

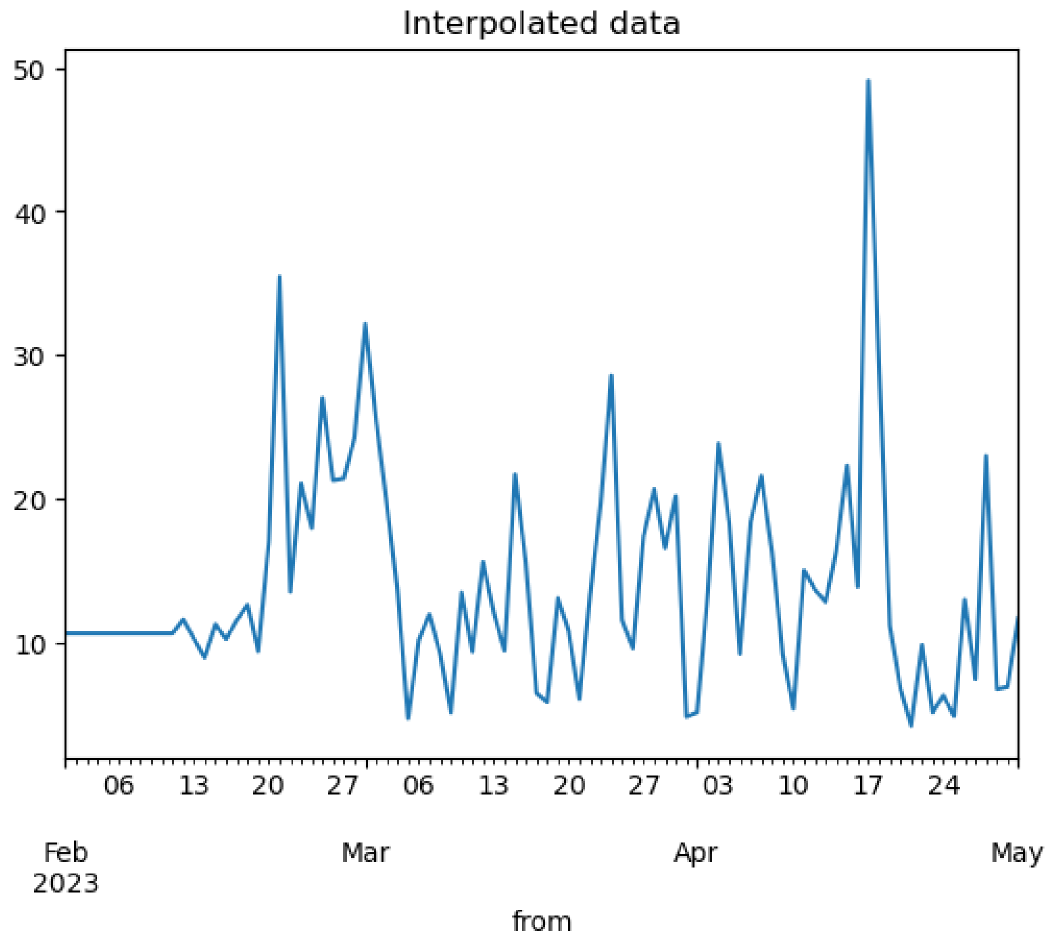


Now we will do ARIMA on spline interpolated data

[35]:
```
#Now doing ARIMA on interpolated data
s7=df['NO'].copy()
s8=df['NO'].copy()
s8= s8.resample('D').mean()
```

[36]:
```
# interpolation first
# interpolating
s7= s7.resample('D').mean()
s7 = s7.interpolate(method='spline', order=3)
s7.fillna(method='ffill', inplace=True)  # Fill missing values forward
s7.fillna(method='bfill', inplace=True)  # Fill missing values backward
# Resample the time series to a different frequency (e.g., from hourly to daily)

s7.plot()
```

```python
plt.title('Interpolated data')
plt.show()
```



[37]:
```python
s7
```

[37]:
```
from
2023-02-01    10.622222
2023-02-02    10.622222
2023-02-03    10.622222
2023-02-04    10.622222
2023-02-05    10.622222
                 …
2023-04-27     7.424731
2023-04-28    22.978495
2023-04-29     6.743011
2023-04-30     6.916304
2023-05-01    11.701075
```

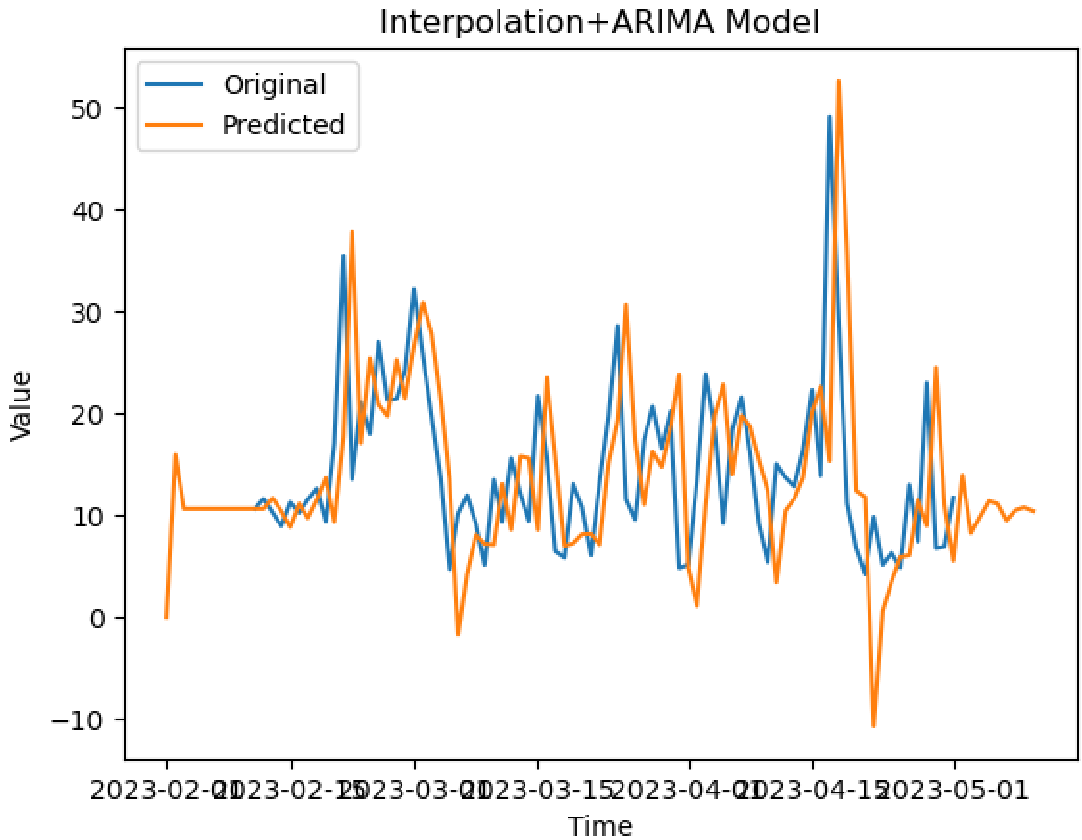
```
Freq: D, Name: NO, Length: 90, dtype: float64
```

[38]:
```
# ADF test to check for stationarity
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(s7)
print(f'p-value: {adf_test[1]}')
```

```
p-value: 1.4711835715040855e-07
```

Such a low p-value implies stationarity.

[39]:
```
# apply arima on interpolated s7
%matplotlib inline
data_series = s7
data_series = pd.Series(data_series)
model = sm.tsa.ARIMA(data_series, order=(3, 2, 0))
result = model.fit()
predictions = result.predict(start='2023-02-01', end='2023-05-10')
# plt.plot(data_series, label='Interpolated')
plt.plot(s8, label='Original')
# label indicates color and corresponding value
plt.plot(predictions, label='Predicted')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Interpolation+ARIMA Model')
plt.legend()
plt.show()
```



We can observe that interpolation+ARIMA model works much better than that of ARIMA without interpolation or ARIMA with missing values replaced by zeroes and mean. Hence interpolation+ARIMA works better than ARIMA or interpolation individually

4.2 Smoothing data / Resampling

Resampling can provide additional information on the data.Resampling helps in smoothening the curve.

There are two types of resampling:

Upsampling is when the frequency of samples is increased (e.g. days to hours)

Downsampling is when the frequency of samples is decreased (e.g. days to weeks)

In this modelling and for all other upcoming models, we will do some downsampling with the .resample() function from 15 min interval to days and also we will use spline cubic interpolation for filling missing data.

5 Part 2: Forecasting

5.1 Prediction Analysis for 'NO' data

Here we will apply ARIMA modelling to predict the future data for NO concentration.

[40]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8640 entries, 2023-02-01 00:00:00 to 2023-05-01 23:45:00
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   PM10     6959 non-null   float64
 1   PM2.5    8414 non-null   float64
 2   NO       7271 non-null   float64
 3   NO2      8224 non-null   float64
 4   NOX      8225 non-null   float64
 5   CO       8144 non-null   float64
 6   SO2      7189 non-null   float64
 7   NH3      8314 non-null   float64
 8   Ozone    8187 non-null   float64
 9   Benzene  2445 non-null   float64
dtypes: float64(10)
memory usage: 1000.5 KB
```

[41]:
```
df2=df.copy()
```

[42]:
```
# interpolating
NO = df2['NO']
df2['NO'] = df2['NO'].interpolate(method='spline',order=3)
df2['NO'].fillna(method='ffill', inplace=True)  # Fill missing values forward
df2['NO'].fillna(method='bfill', inplace=True)  # Fill missing values backward
```

Here we will take some part of our data as training set for ARIMA modelling while the remaing part will be predicted by the model. Then we will compare the actual data and the predicted data.

[43]:
```
t=df2.index[8000]
# msk = (df.index <= pd.to_datetime(t, format='%y-%m-%d %H:%M:%S'))
msk=(df2.index<=t)

df_train = df2['NO'][msk].copy()
df_test = df2['NO'][~msk].copy()
```

[44]:
```
t
```

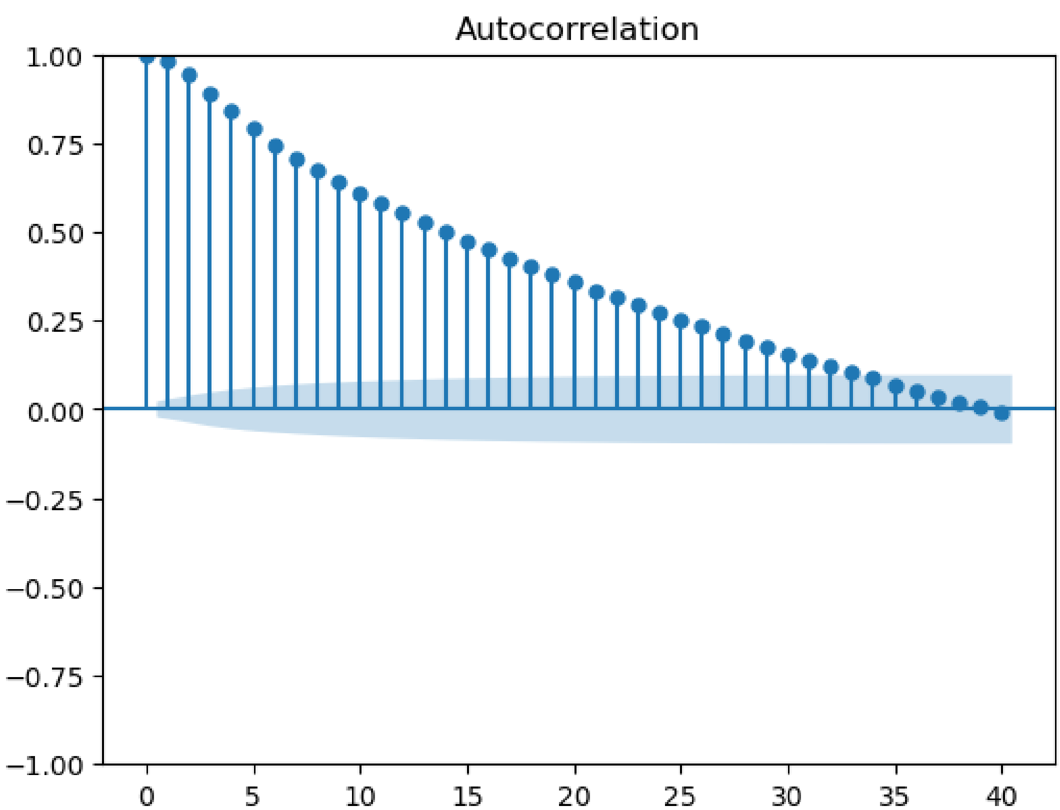
[44]:
```
Timestamp('2023-04-25 08:00:00')
```

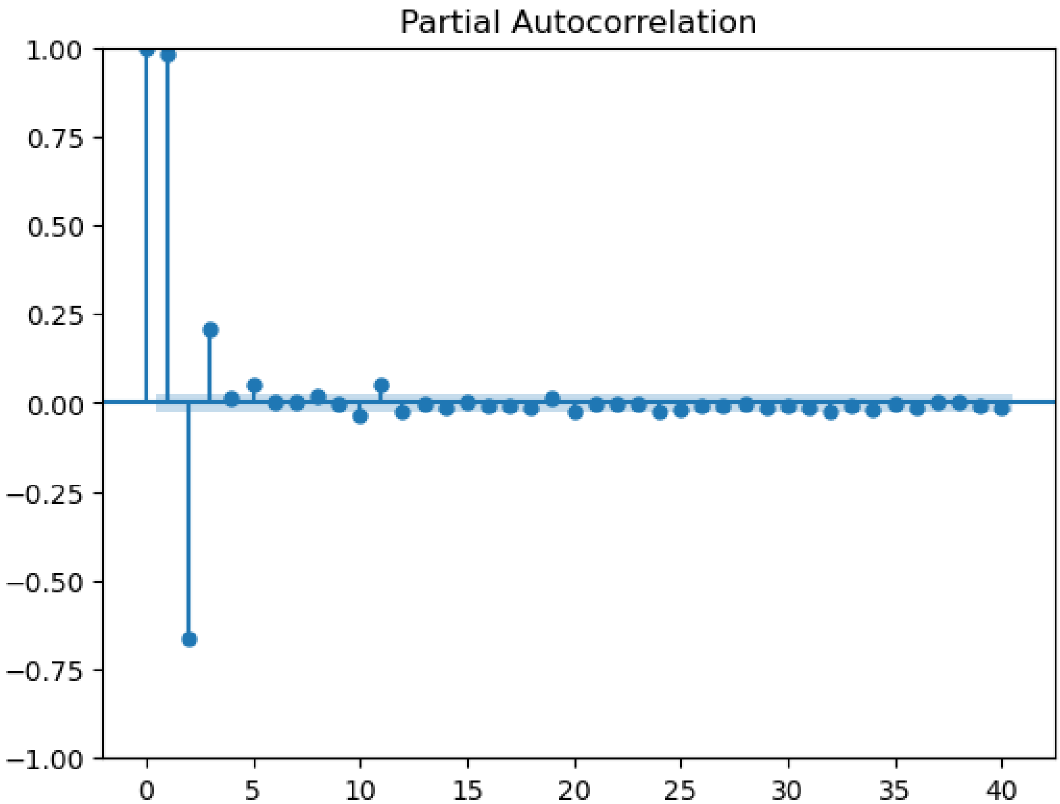
[45]:
```
# just checking code   ADF test to check for stationarity
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(df2['NO'])
print(f'p-value: {adf_test[1]}')
```

```
p-value: 3.58739703711905e-24
```

[46]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

acf_original = plot_acf(df_train)

pacf_original = plot_pacf(df_train)
```

```
C:\ProgramData\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```





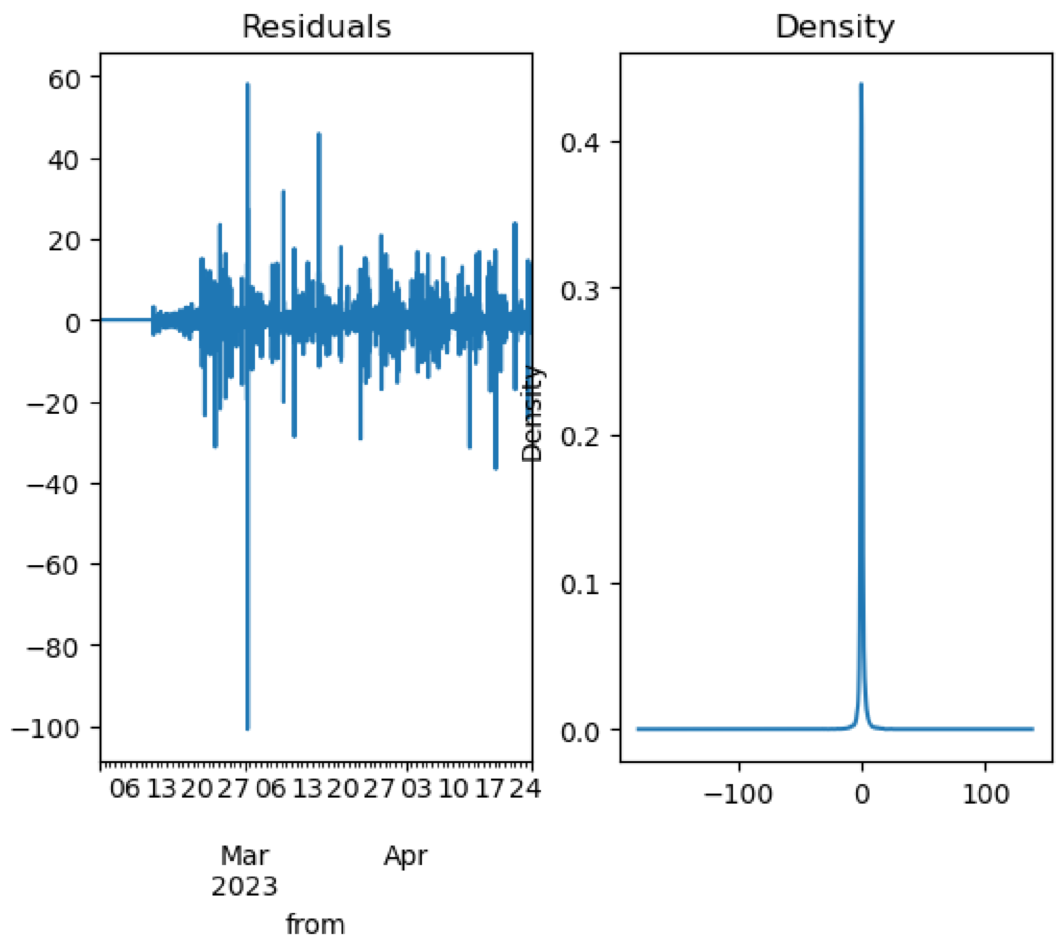
[47]:
```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df_train, order=(3,0,1))
model_fit = model.fit()
print(model_fit.summary())
```

```
C:\ProgramData\anaconda3\lib\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)
C:\ProgramData\anaconda3\lib\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)
C:\ProgramData\anaconda3\lib\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)
                               SARIMAX Results
==============================================================================
Dep. Variable:                     NO   No. Observations:                 8001
Model:                 ARIMA(3, 0, 1)   Log Likelihood              -19718.727
```

```
Date:                Tue, 27 Jun 2023   AIC                         39449.454
Time:                        17:03:52   BIC                         39491.378
Sample:                    02-01-2023   HQIC                        39463.804
                         - 04-25-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         16.1851      2.291      7.066      0.000      11.696      20.675
ar.L1          1.9229      0.031     61.133      0.000       1.861       1.985
ar.L2         -1.2467      0.052    -23.984      0.000      -1.349      -1.145
ar.L3          0.3050      0.022     14.115      0.000       0.263       0.347
ma.L1         -0.1661      0.033     -5.102      0.000      -0.230      -0.102
sigma2         8.0899      0.024    338.101      0.000       8.043       8.137
================================================================================
===
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):
20681931.74
Prob(Q):                              0.82   Prob(JB):
0.00
Heteroskedasticity (H):               2.00   Skew:
-4.41
Prob(H) (two-sided):                  0.00   Kurtosis:
251.92
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

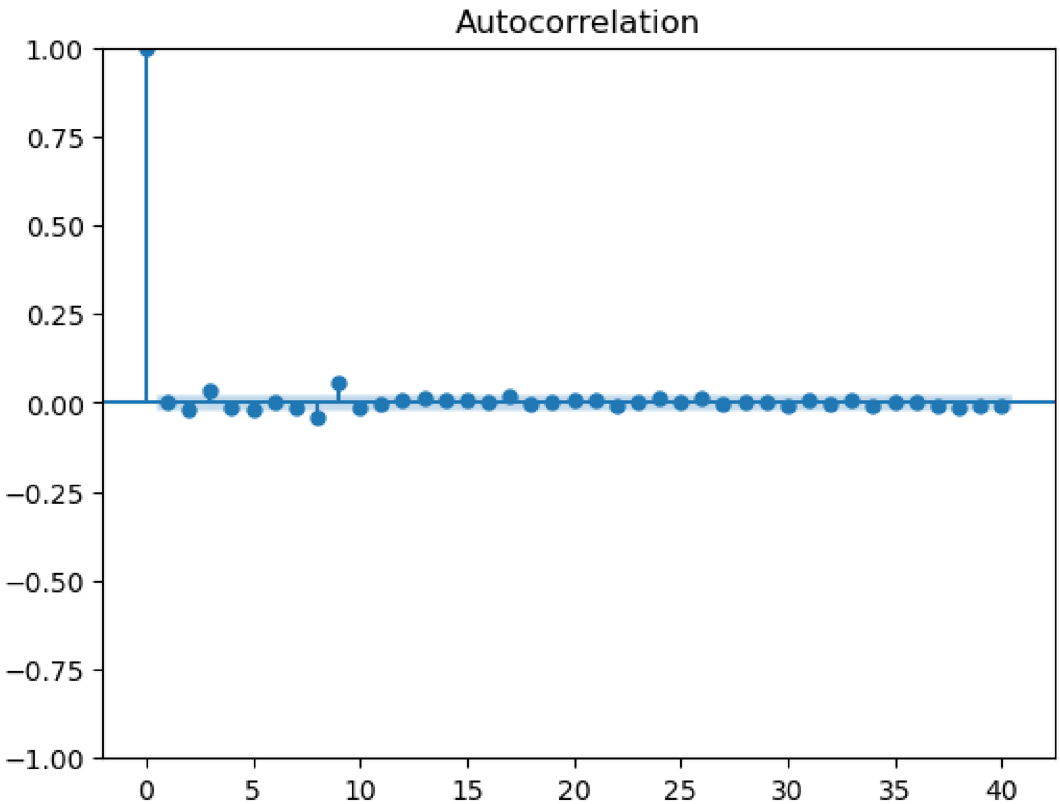
[48]:

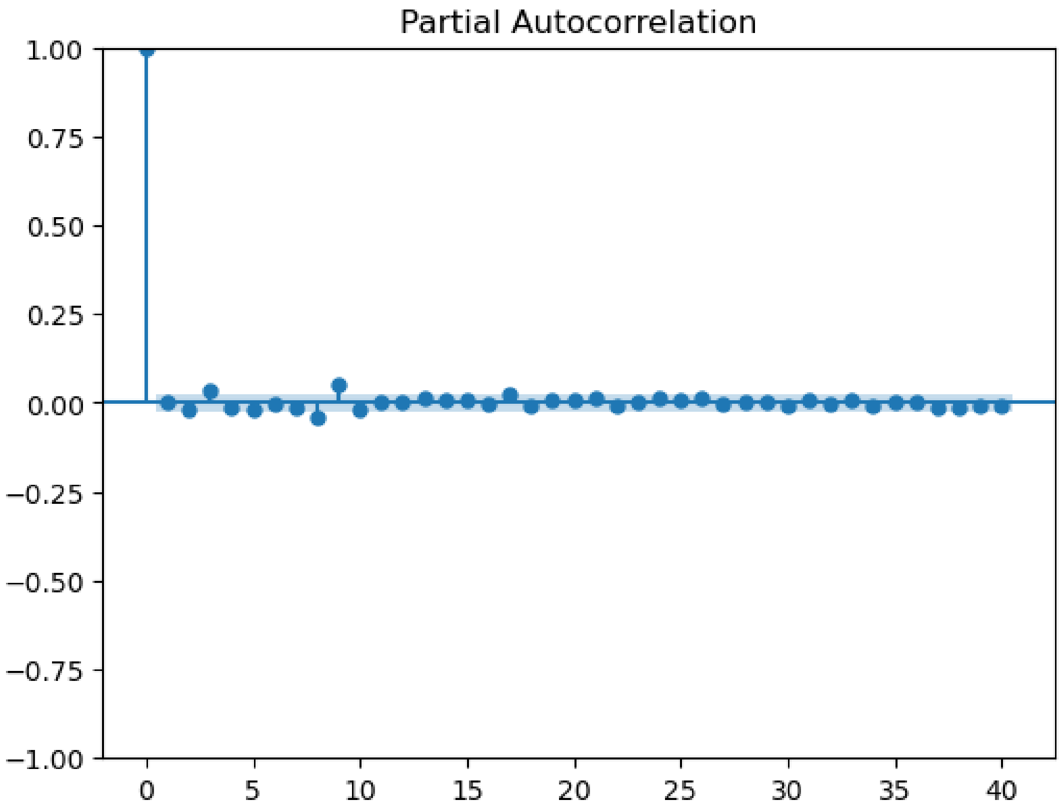
```python
import matplotlib.pyplot as plt
residuals = model_fit.resid[1:]
fig, ax = plt.subplots(1,2)
residuals.plot(title='Residuals', ax=ax[0])
residuals.plot(title='Density', kind='kde', ax=ax[1])
plt.show()
```



[49]:
```
acf_res = plot_acf(residuals)

pacf_res = plot_pacf(residuals)
```

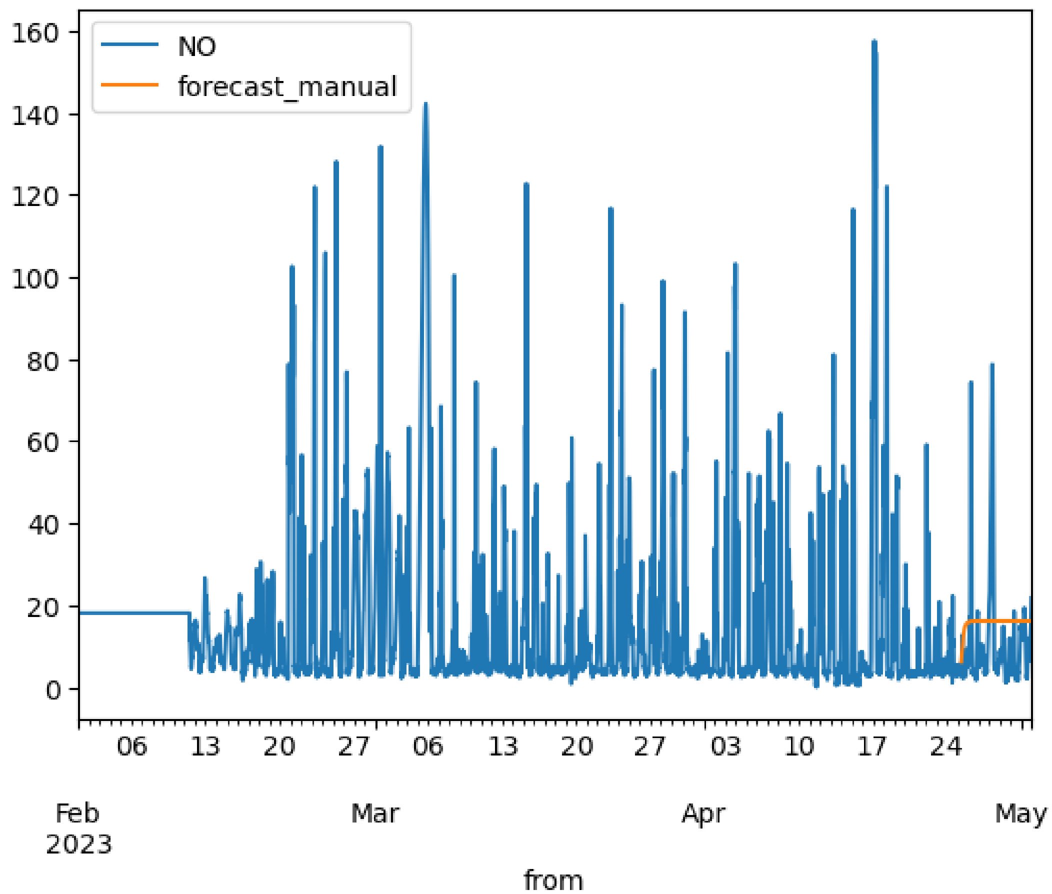
```
C:\ProgramData\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40



[50]:
```python
forecast_test = model_fit.forecast(len(df_test))

df2['forecast_manual'] = [None]*len(df_train) + list(forecast_test)

columns_to_plot = ['NO', 'forecast_manual']
data_to_plot = df2[columns_to_plot]
data_to_plot.plot()
```

[50]: <Axes: xlabel='from'>



6 Trying with resampled data

[51]:
```
df3=df.copy()
```

[52]:
```
p2=df3['NO']
```

[53]:
```
# resample
df3 = df3.resample('D').mean()
# interpolating
NO = df3['NO']
df3 = df3.interpolate(method='spline',order=3)
df3.fillna(method='ffill', inplace=True)  # Fill missing values forward
df3.fillna(method='bfill', inplace=True)  # Fill missing values backward
```

[54]:
```
df3.head()
```

[54]:
```
                  PM10      PM2.5         NO        NO2        NOX  \
from
2023-02-01  114.739583  35.145833  10.622222  79.180645  48.821505
2023-02-02  177.458333  52.020833  10.622222  79.286957  53.986957
2023-02-03  171.270833  52.916667  10.622222  82.408602  56.936559
2023-02-04  222.552941  74.651685  10.622222  76.781319  53.721978
2023-02-05  271.354430  86.987952  10.622222  74.712048  66.445783

                     CO        SO2        NH3      Ozone   Benzene
from
2023-02-01   441.182796  10.244681  22.080851  30.076744  0.232292
2023-02-02  1303.152174  10.244681  22.266667  25.331183  0.120000
2023-02-03  1211.075269  10.244681  23.105319  27.535106  0.165625
2023-02-04  1146.222222  10.244681  25.094565  26.340659  0.184444
2023-02-05   890.357143  10.244681  25.571429  21.824706  0.214286
```

[55]:
```
t=df3.index[70]
# msk = (df.index <= pd.to_datetime(t, format='%y-%m-%d %H:%M:%S'))
msk=(df3.index<=t)
df_train =df3['NO'][msk].copy()
df_test = df3['NO'][~msk].copy()
```

[56]:
```
t
```

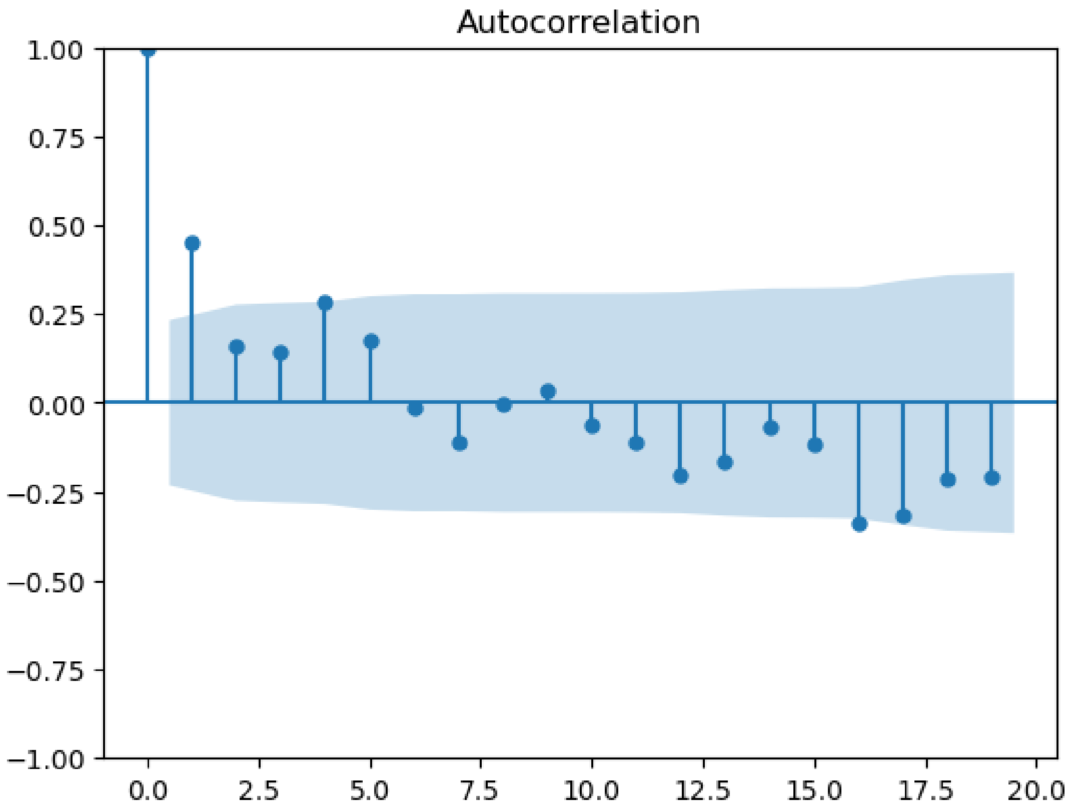
[56]:
```
Timestamp('2023-04-12 00:00:00', freq='D')
```

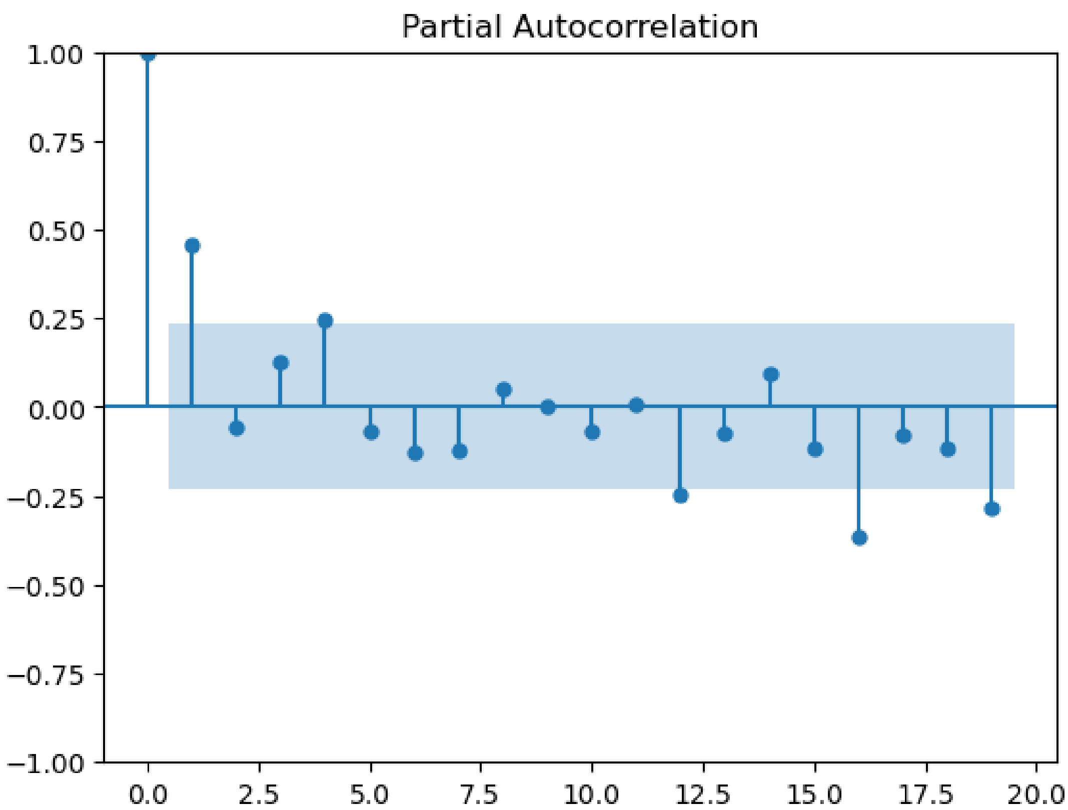
[57]:
```
# just checking code   ADF test to check for stationarity
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(df3['NO'])
print(f'p-value: {adf_test[1]}')
```

```
p-value: 1.4711835715040855e-07
```

[58]:
```
%matplotlib inline
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

acf_original = plot_acf(df_train)

pacf_original = plot_pacf(df_train)
```

```
C:\ProgramData\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



[59]:
```python
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df_train, order=(5,0,1))
model_fit = model.fit()
print(model_fit.summary())
```

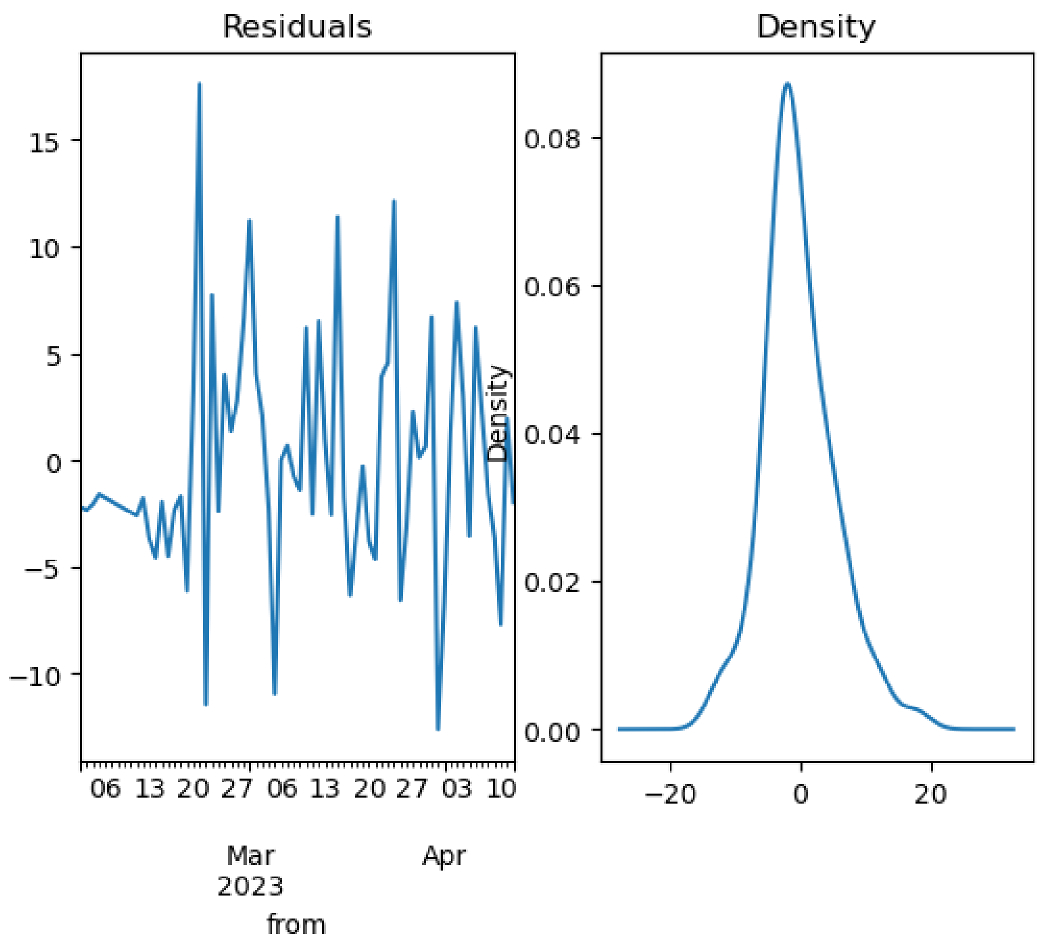
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                     NO   No. Observations:                   71
Model:                 ARIMA(5, 0, 1)   Log Likelihood                -220.242
Date:                Tue, 27 Jun 2023   AIC                            456.485
Time:                        17:03:55   BIC                            474.586
Sample:                    02-01-2023   HQIC                           463.683
                         - 04-12-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         14.4992      0.512     28.329      0.000      13.496      15.502
ar.L1          1.3650      0.129     10.561      0.000       1.112       1.618
ar.L2         -0.4885      0.189     -2.583      0.010      -0.859      -0.118
ar.L3          0.0775      0.233      0.333      0.739      -0.379       0.534
```

```
ar.L4          0.2650      0.246      1.079      0.281      -0.217       0.746
ar.L5         -0.3044      0.158     -1.930      0.054      -0.614       0.005
ma.L1         -0.9992      5.488     -0.182      0.856     -11.755       9.756
sigma2        27.7369    151.283      0.183      0.855    -268.772     324.246
================================================================================
===
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
8.59
Prob(Q):                              0.93   Prob(JB):
0.01
Heteroskedasticity (H):               1.10   Skew:
0.62
Prob(H) (two-sided):                  0.82   Kurtosis:
4.17
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

```
C:\ProgramData\anaconda3\lib\site-packages\statsmodels\base\model.py:604:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

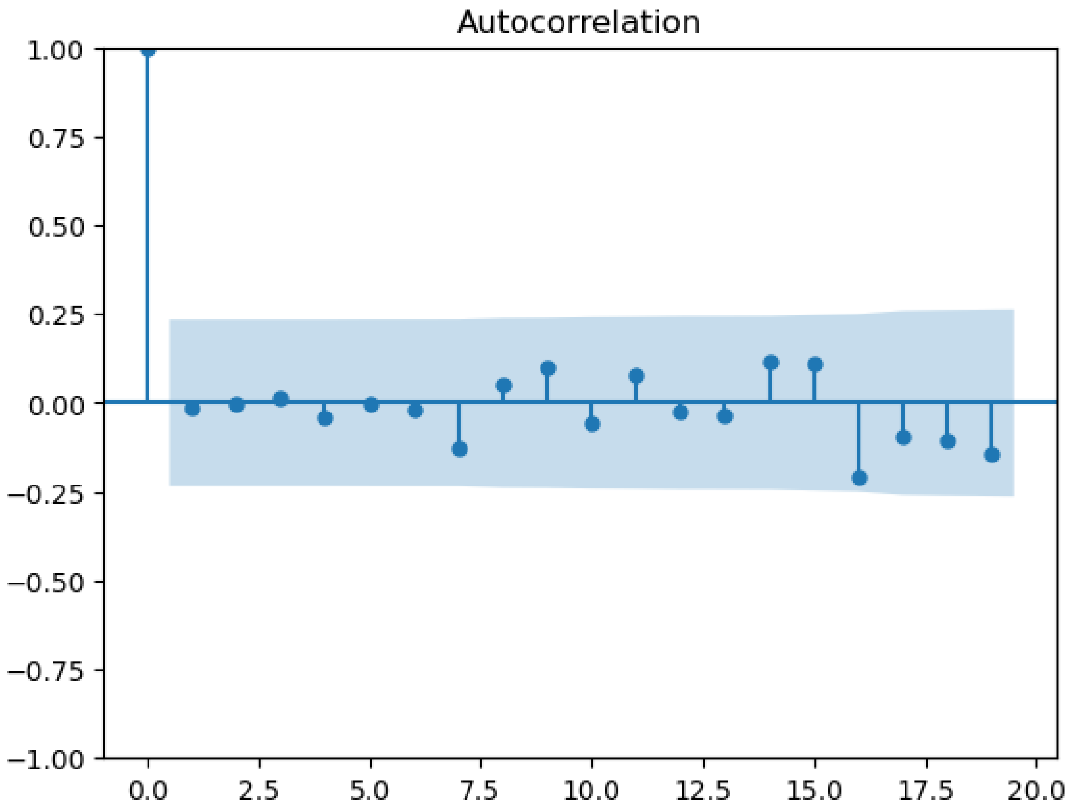
[60]:

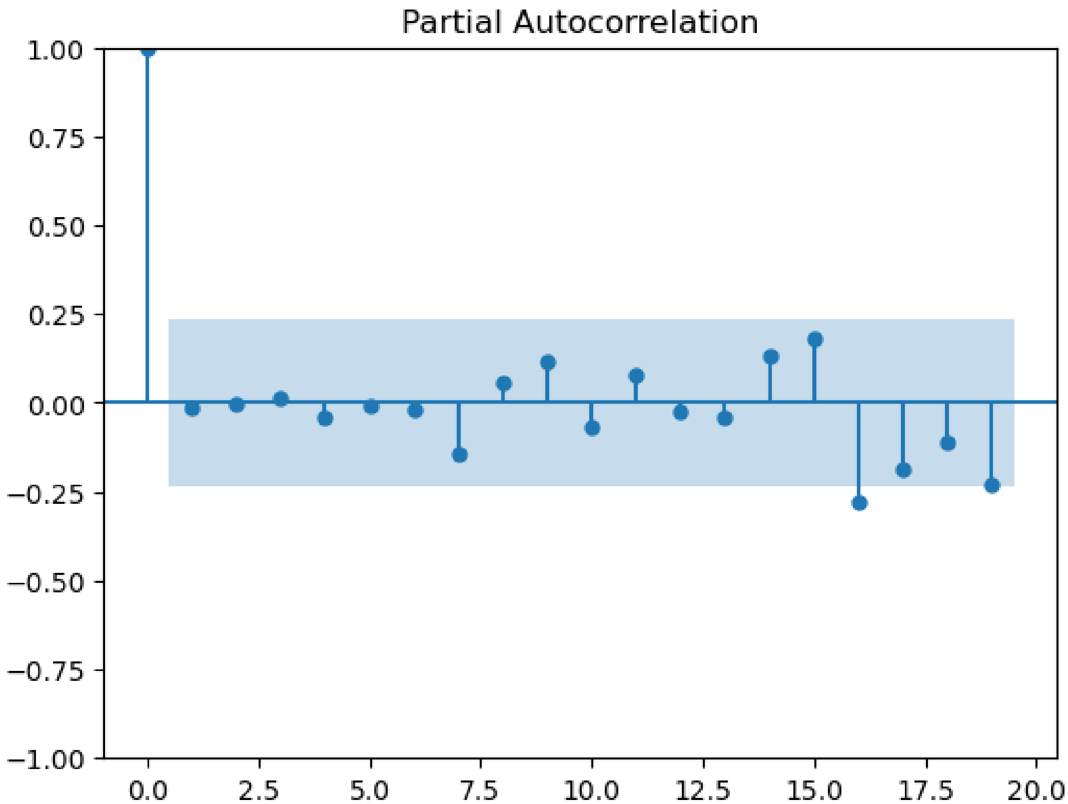
```
import matplotlib.pyplot as plt
%matplotlib inline
residuals = model_fit.resid[1:]
fig, ax = plt.subplots(1,2)
residuals.plot(title='Residuals', ax=ax[0])
residuals.plot(title='Density', kind='kde', ax=ax[1])
plt.show()
```



[61]:
```
acf_res = plot_acf(residuals)

pacf_res = plot_pacf(residuals)
```

```
C:\ProgramData\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

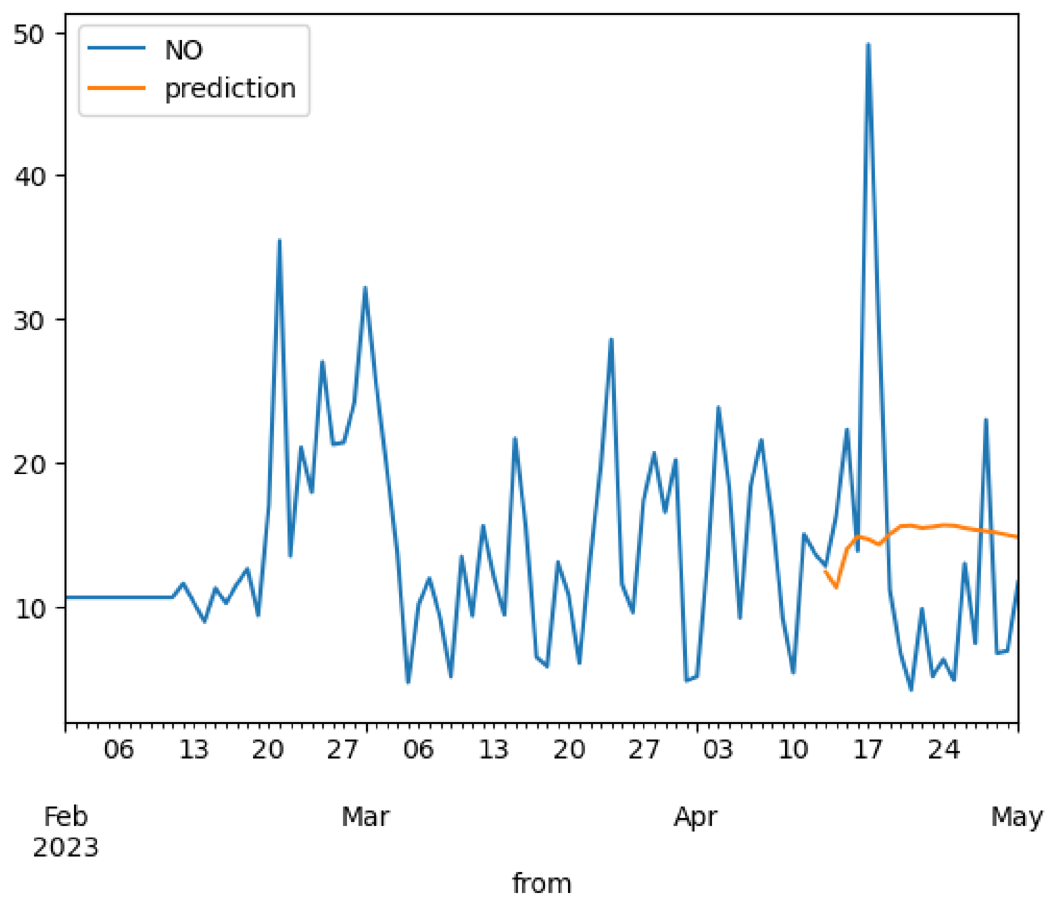


[62]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 90 entries, 2023-02-01 to 2023-05-01
Freq: D
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   PM10     90 non-null     float64
 1   PM2.5    90 non-null     float64
 2   NO       90 non-null     float64
 3   NO2      90 non-null     float64
 4   NOX      90 non-null     float64
 5   CO       90 non-null     float64
 6   SO2      90 non-null     float64
 7   NH3      90 non-null     float64
 8   Ozone    90 non-null     float64
 9   Benzene  90 non-null     float64
dtypes: float64(10)
memory usage: 7.7 KB
```

[63]:
```
forecast_test = model_fit.forecast(len(df_test))

df3['prediction'] = [None]*len(df_train) + list(forecast_test)
%matplotlib inline
columns_to_plot = ['NO', 'prediction']

data_to_plot = df3[columns_to_plot]

data_to_plot.plot()
```

[63]: <Axes: xlabel='from'>



Here we can observe that resampled data provides better forecasting than one without it.

Finally we will calculate error of prediction and actual data.

[64]:
```
from sklearn.metrics import mean_absolute_error,␣
↪mean_absolute_percentage_error, mean_squared_error
```

```
mae = mean_absolute_error(df_test, forecast_test)
mape = mean_absolute_percentage_error(df_test, forecast_test)
rmse = np.sqrt(mean_squared_error(df_test, forecast_test))

print(f'mae - manual: {mae}')
print(f'mape - manual: {mape}')
print(f'rmse - manual: {rmse}')
```

```
mae - manual: 8.524822710545166
mape - manual: 0.8918506939382178
rmse - manual: 11.10568140395216
```

7 Using Auto arima

[65]:
```
import pmdarima as pm
auto_arima = pm.auto_arima(df_train, stepwise=False, seasonal=False)
auto_arima
```

[65]:
```
ARIMA(order=(4, 0, 0), scoring_args={}, suppress_warnings=True,
      with_intercept=False)
```

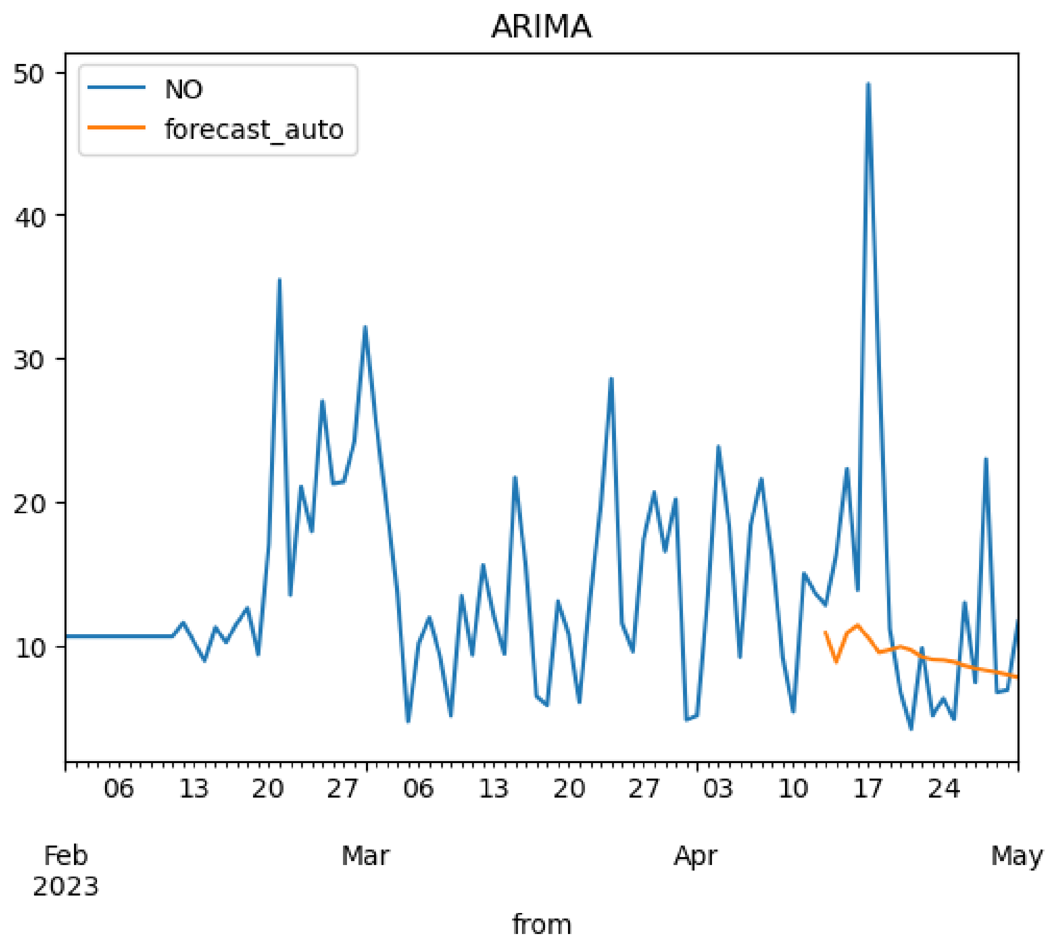
[66]:
```
auto_arima.summary()
```

[66]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   71
Model:               SARIMAX(4, 0, 0)   Log Likelihood                -227.127
Date:                Tue, 27 Jun 2023   AIC                            464.253
Time:                        17:03:57   BIC                            475.567
Sample:                    02-01-2023   HQIC                           468.752
                         - 04-12-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.5619      0.091      6.191      0.000       0.384       0.740
ar.L2         -0.0210      0.128     -0.165      0.869      -0.271       0.229
ar.L3          0.0798      0.150      0.533      0.594      -0.213       0.373
ar.L4          0.3364      0.121      2.776      0.005       0.099       0.574
sigma2        34.0082      4.278      7.949      0.000      25.623      42.393
================================================================================
===
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):
13.12
Prob(Q):                              0.87   Prob(JB):
```

```
0.00
Heteroskedasticity (H):               1.16   Skew:
0.33
Prob(H) (two-sided):                  0.73   Kurtosis:
5.00
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

[67]:

```
forecast_test_auto = auto_arima.predict(n_periods=len(df_test))
df3['forecast_auto'] = [None]*len(df_train) + list(forecast_test_auto)
%matplotlib inline
columns_to_plot = ['NO', 'forecast_auto']

data_to_plot = df3[columns_to_plot]

data_to_plot.plot()
plt.title('ARIMA')
plt.show()
```



[68]:
```
mae = mean_absolute_error(df_test, forecast_test_auto)
mape = mean_absolute_percentage_error(df_test, forecast_test_auto)
rmse = np.sqrt(mean_squared_error(df_test, forecast_test_auto))

print(f'mae - auto: {mae}')
print(f'mape - auto: {mape}')
print(f'rmse - auto: {rmse}')
```

```
mae - auto: 6.807528250602216
mape - auto: 0.4498355910554497
rmse - auto: 11.258926464904992
```

Forecasting for PM10

June 25, 2023

0.1 Prediction Analysis for 'PM10' data

Here we will apply ARIMA modelling to predict the future data for PM10 concentration.

[1]:
```
!pip install pmdarima
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pmdarima in c:\programdata\anaconda3\lib\site-
packages (2.0.3)
Requirement already satisfied: statsmodels>=0.13.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.13.5)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (65.6.3)
Requirement already satisfied: scikit-learn>=0.22 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.2.1)
Requirement already satisfied: pandas>=0.19 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.5.3)
Requirement already satisfied: urllib3 in c:\programdata\anaconda3\lib\site-
packages (from pmdarima) (1.26.14)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.29.35)
Requirement already satisfied: joblib>=0.11 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.1.1)
Requirement already satisfied: scipy>=1.3.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.10.0)
Requirement already satisfied: numpy>=1.21.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.23.5)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2022.7)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.22->pmdarima)
(2.2.0)
Requirement already satisfied: packaging>=21.3 in
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(22.0)
Requirement already satisfied: patsy>=0.5.2 in
```

```
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(0.5.3)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages
(from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
```

[2]:
```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import pandas.plotting
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
%matplotlib inline
```

[3]:
```
file_path = 'C:/Users/Omkar/Desktop/EE798Q/Open pit blasting 01-02-2023 000000␣
↪To 01-05-2023 235959.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path , index_col=0)
```

[4]:
```
# Simplify column names
df.columns = ['from', 'to', 'PM10', 'PM2.5',␣
↪'NO','NO2','NOX','CO','SO2','NH3','Ozone','Benzene']
# deleting to column as we need only one timestamp column for to be index and␣
↪we choose it to be from column
df = df.drop('to', axis=1)
# removing last 3 rows as they contaion max , min , avg data instead of actual␣
↪observations
df = df.iloc[:-3]
df.tail()
```

[4]:
```
                     from  PM10  PM2.5    NO    NO2   NOX    CO   SO2   NH3  \
#
8636  2023-05-01 22:45:00  19.0   11.0  17.9  100.0  67.8  0.63  10.0  10.7
8637  2023-05-01 23:00:00  19.0   11.0  17.9  100.0  67.7  0.57  10.0  10.4
8638  2023-05-01 23:15:00  19.0   11.0  19.6  100.2  69.2  0.58   9.9  10.5
8639  2023-05-01 23:30:00  19.0   11.0  20.8  100.2  70.2  0.58   9.5  10.8
8640  2023-05-01 23:45:00  32.0    6.0  21.8   98.8  70.3   NaN   NaN  11.0

      Ozone  Benzene
#
8636   26.1      0.1
8637   30.9      0.1
8638   29.6      0.1
8639   30.0      0.1
8640   33.5      0.1
```

[5]:
```
# conveting timestamp as a string object into a datetime numerical
date_format = '%Y-%m-%d %H:%M:%S'

# Convert the 'from' column to numerical datetime representation
df['from'] = pd.to_datetime(df['from'], format=date_format)
```

[6]:
```
# set datetime "from" column as an index column
df.set_index('from', inplace=True)
df.head()
```

[6]:
```
                      PM10  PM2.5  NO   NO2   NOX    CO  SO2   NH3  Ozone  \
from
2023-02-01 00:00:00   95.0   35.0 NaN  90.1  56.2  0.31  NaN  17.7   28.1
2023-02-01 00:15:00   95.0   35.0 NaN  88.0  55.1  0.33  NaN  18.3   27.1
2023-02-01 00:30:00   95.0   35.0 NaN  87.7  55.2  0.38  NaN  19.7   24.9
2023-02-01 00:45:00  122.0   34.0 NaN  88.9  55.7  0.38  NaN  21.3   21.9
2023-02-01 01:00:00  122.0   34.0 NaN  90.0  55.8  0.38  NaN  22.3   16.7

                     Benzene
from
2023-02-01 00:00:00      0.4
2023-02-01 00:15:00      0.4
2023-02-01 00:30:00      0.4
2023-02-01 00:45:00      0.4
2023-02-01 01:00:00      0.4
```

[7]:
```
df3=df.copy()
```

[8]:
```
p2=df3['PM10']
```

[9]:
```
# resample
df3 = df3.resample('D').mean()
# interpolating
PM10 = df3['PM10']
df3 = df3.interpolate(method='spline',order=3)
df3.fillna(method='ffill', inplace=True)  # Fill missing values forward
df3.fillna(method='bfill', inplace=True)  # Fill missing values backward
```

[10]:
```
df3.head()
```

[10]:
```
                  PM10      PM2.5         NO        NO2        NOX        CO  \
from
2023-02-01  114.739583  35.145833  10.622222  79.180645  48.821505  0.441183
2023-02-02  177.458333  52.020833  10.622222  79.286957  53.986957  1.303152
2023-02-03  171.270833  52.916667  10.622222  82.408602  56.936559  1.211075
2023-02-04  222.552941  74.651685  10.622222  76.781319  53.721978  1.146222
2023-02-05  271.354430  86.987952  10.622222  74.712048  66.445783  0.890357
```

```
                 SO2        NH3      Ozone   Benzene
from
2023-02-01  10.244681  22.080851  30.076744  0.232292
2023-02-02  10.244681  22.266667  25.331183  0.120000
2023-02-03  10.244681  23.105319  27.535106  0.165625
2023-02-04  10.244681  25.094565  26.340659  0.184444
2023-02-05  10.244681  25.571429  21.824706  0.214286
```

Here we will take some part of our data as training set for ARIMA modelling while the remaing part will be predicted by the model. Then we will compare the actual data and the predicted data.

[11]:
```
t=df3.index[70]
# msk = (df.index <= pd.to_datetime(t, format='%y-%m-%d %H:%M:%S'))
msk=(df3.index<=t)
df_train =df3['PM10'][msk].copy()
df_test = df3['PM10'][~msk].copy()
```

[12]:
```
t
```

[12]:
```
Timestamp('2023-04-12 00:00:00', freq='D')
```

[13]:
```
# just checking code   ADF test to check for stationarity
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(df3['PM10'])
print(f'p-value: {adf_test[1]}')
```

```
p-value: 0.004522349467992982
```

[14]:
```
import pmdarima as pm
auto_arima = pm.auto_arima(df_train, stepwise=False, seasonal=False)
auto_arima
```

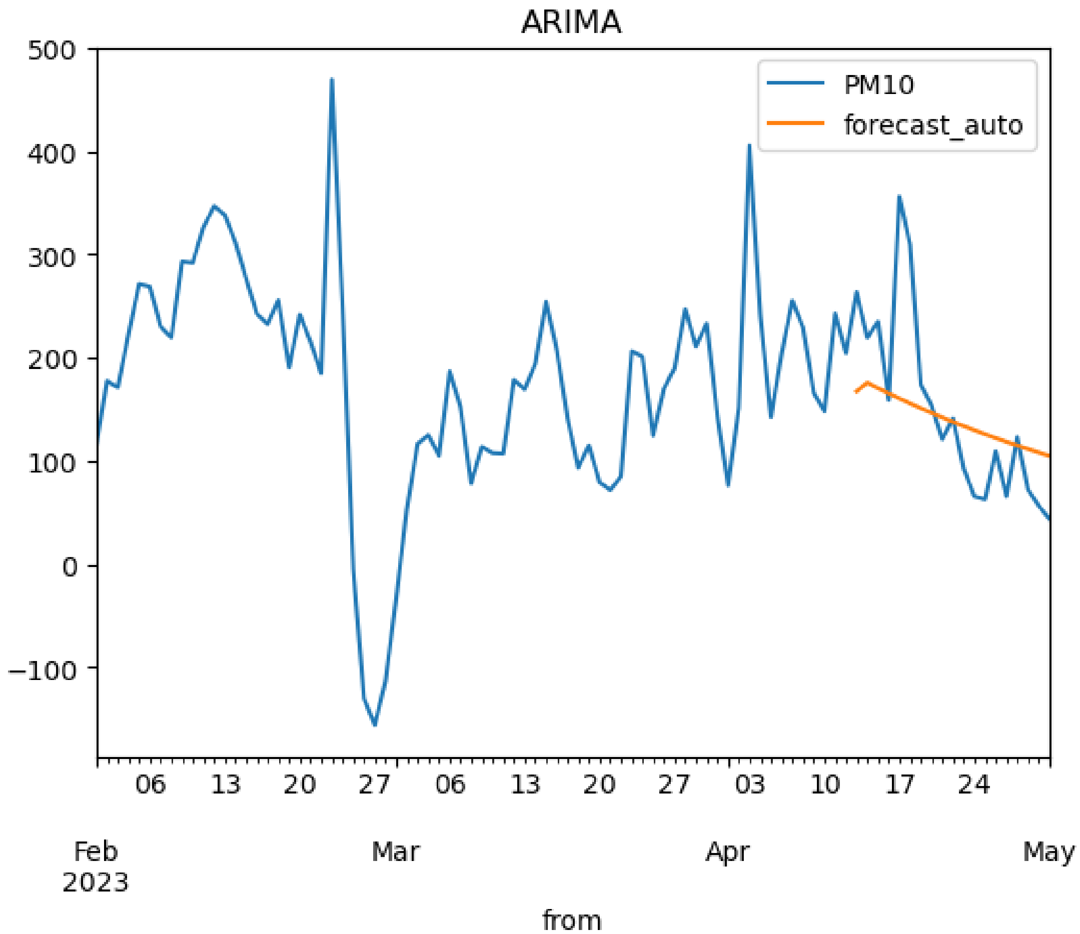
[14]:
```
ARIMA(order=(1, 0, 2), scoring_args={}, suppress_warnings=True,
      with_intercept=False)
```

[15]:
```
auto_arima.summary()
```

[15]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   71
Model:               SARIMAX(1, 0, 2)   Log Likelihood                -409.341
Date:                Sun, 25 Jun 2023   AIC                            826.682
Time:                        19:44:37   BIC                            835.733
Sample:                    02-01-2023   HQIC                           830.282
                         - 04-12-2023
Covariance Type:                  opg
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9702      0.029     33.392      0.000       0.913       1.027
ma.L1         -0.0254      0.089     -0.285      0.776      -0.200       0.149
ma.L2         -0.3680      0.095     -3.882      0.000      -0.554      -0.182
sigma2      5778.4197    628.213      9.198      0.000    4547.144    7009.695
================================================================================
===
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):
27.84
Prob(Q):                              0.77   Prob(JB):
0.00
Heteroskedasticity (H):               0.84   Skew:
0.21
Prob(H) (two-sided):                  0.67   Kurtosis:
6.04
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

[16]:
```
forecast_test_auto = auto_arima.predict(n_periods=len(df_test))
df3['forecast_auto'] = [None]*len(df_train) + list(forecast_test_auto)

columns_to_plot = ['PM10', 'forecast_auto']
%matplotlib inline
data_to_plot = df3[columns_to_plot]

data_to_plot.plot()
plt.title('ARIMA')
plt.show()
```



In the above plot forecast_auto is the prediction by ARIMA model. Here we can observe that resampled data provides better forecasting than one without it.

Finally we will calculate error of prediction and actual data.

[17]:
```
from sklearn.metrics import mean_absolute_error,␣
↪mean_absolute_percentage_error, mean_squared_error

mae = mean_absolute_error(df_test, forecast_test_auto)
mape = mean_absolute_percentage_error(df_test, forecast_test_auto)
rmse = np.sqrt(mean_squared_error(df_test, forecast_test_auto))

print(f'mae - auto: {mae}')
print(f'mape - auto: {mape}')
print(f'rmse - auto: {rmse}')
```

```
mae - auto: 53.009429041707186
mape - auto: 0.44952142020881874
rmse - auto: 72.07877893064887
```

```
[ ]:
```

For combined weighted mean-Copy1

June 27, 2023

1 Part 3: Finding Combined Weighted Mean

1.0.1 A single time- series to capture effects of all pollutants

Here we will calculate combined weighted combination of air polluting factors to obtain a single time-series data in new column- "tot_pol".

We will use means of individual columns to obtain the weights used to form column "tot_col".

For example, for the weight corresponding to column PM10,

We know that mean of all PM10 readings is 181.41, while the sum of means of all pollutant data readings is 484.87.

Hence weight value for PM10 column will be 181.41/484.87= 0.398

Same process is followed to get weights for remaining columns.

Giving different weights for different columns will result in giving more importance to pollutants with higher concentration and less priority to the remaining.

[1]:
```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib notebook
import pandas as pd
import pandas.plotting
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


import seaborn as sns
from scipy.stats import norm
from statsmodels.graphics.gofplots import qqplot
```

[2]:
```
file_path = 'C:/Users/Omkar/Desktop/EE798Q/Open pit blasting 01-02-2023 000000␣
↪To 01-05-2023 235959.csv'

# Read the CSV file into a DataFrame
```

```
df = pd.read_csv(file_path , index_col=0)
```

[3]:
```
# Simplify column names
df.columns = ['from', 'to', 'PM10', 'PM2.5',␣
 ↪'NO','NO2','NOX','CO','SO2','NH3','Ozone','Benzene']
# deleting to column as we need only one timestamp column for to be index and␣
 ↪we choose it to be from column
df = df.drop('to', axis=1)
# removing last 3 rows as they contaion max , min , avg data instead of actual␣
 ↪observations
df = df.iloc[:-3]
df.tail()
```

[3]:
```
                     from  PM10  PM2.5    NO    NO2   NOX    CO   SO2   NH3  \
#
8636  2023-05-01 22:45:00  19.0   11.0  17.9  100.0  67.8  0.63  10.0  10.7
8637  2023-05-01 23:00:00  19.0   11.0  17.9  100.0  67.7  0.57  10.0  10.4
8638  2023-05-01 23:15:00  19.0   11.0  19.6  100.2  69.2  0.58   9.9  10.5
8639  2023-05-01 23:30:00  19.0   11.0  20.8  100.2  70.2  0.58   9.5  10.8
8640  2023-05-01 23:45:00  32.0    6.0  21.8   98.8  70.3   NaN   NaN  11.0

      Ozone  Benzene
#
8636   26.1      0.1
8637   30.9      0.1
8638   29.6      0.1
8639   30.0      0.1
8640   33.5      0.1
```

[4]:
```
# conveting timestamp as a string object into a datetime numerical
date_format = '%Y-%m-%d %H:%M:%S'

# Convert the 'from' column to numerical datetime representation
df['from'] = pd.to_datetime(df['from'], format=date_format)
```

[5]:
```
# set datetime "from" column as an index column
df.set_index('from', inplace=True)
df.head()
```

[5]:
```
                      PM10  PM2.5  NO   NO2   NOX    CO  SO2   NH3  Ozone  \
from
2023-02-01 00:00:00   95.0   35.0 NaN  90.1  56.2  0.31  NaN  17.7   28.1
2023-02-01 00:15:00   95.0   35.0 NaN  88.0  55.1  0.33  NaN  18.3   27.1
2023-02-01 00:30:00   95.0   35.0 NaN  87.7  55.2  0.38  NaN  19.7   24.9
2023-02-01 00:45:00  122.0   34.0 NaN  88.9  55.7  0.38  NaN  21.3   21.9
2023-02-01 01:00:00  122.0   34.0 NaN  90.0  55.8  0.38  NaN  22.3   16.7
```

```
                     Benzene
from
2023-02-01 00:00:00      0.4
2023-02-01 00:15:00      0.4
2023-02-01 00:30:00      0.4
2023-02-01 00:45:00      0.4
2023-02-01 01:00:00      0.4
```

[6]:
```
df3=df.copy()
```

[7]:
```
df3
```

[7]:
```
                      PM10  PM2.5    NO    NO2   NOX    CO   SO2   NH3  Ozone  \
from
2023-02-01 00:00:00   95.0   35.0   NaN   90.1  56.2  0.31   NaN  17.7   28.1
2023-02-01 00:15:00   95.0   35.0   NaN   88.0  55.1  0.33   NaN  18.3   27.1
2023-02-01 00:30:00   95.0   35.0   NaN   87.7  55.2  0.38   NaN  19.7   24.9
2023-02-01 00:45:00  122.0   34.0   NaN   88.9  55.7  0.38   NaN  21.3   21.9
2023-02-01 01:00:00  122.0   34.0   NaN   90.0  55.8  0.38   NaN  22.3   16.7
…                      …     …     …     …     …     …     …     …
2023-05-01 22:45:00   19.0   11.0  17.9  100.0  67.8  0.63  10.0  10.7   26.1
2023-05-01 23:00:00   19.0   11.0  17.9  100.0  67.7  0.57  10.0  10.4   30.9
2023-05-01 23:15:00   19.0   11.0  19.6  100.2  69.2  0.58   9.9  10.5   29.6
2023-05-01 23:30:00   19.0   11.0  20.8  100.2  70.2  0.58   9.5  10.8   30.0
2023-05-01 23:45:00   32.0    6.0  21.8   98.8  70.3   NaN   NaN  11.0   33.5

                     Benzene
from
2023-02-01 00:00:00      0.4
2023-02-01 00:15:00      0.4
2023-02-01 00:30:00      0.4
2023-02-01 00:45:00      0.4
2023-02-01 01:00:00      0.4
…                       …
2023-05-01 22:45:00      0.1
2023-05-01 23:00:00      0.1
2023-05-01 23:15:00      0.1
2023-05-01 23:30:00      0.1
2023-05-01 23:45:00      0.1

[8640 rows x 10 columns]
```

[8]:
```
# resample
# df3 = df3.resample('D').mean()
# interpolating
PM10 = df['PM10']
df = df.interpolate(method='spline',order=3)
```

```
df.fillna(method='ffill', inplace=True)  # Fill missing values forward
df.fillna(method='bfill', inplace=True)  # Fill missing values backward
```

[9]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8640 entries, 2023-02-01 00:00:00 to 2023-05-01 23:45:00
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   PM10     6959 non-null   float64
 1   PM2.5    8414 non-null   float64
 2   NO       7271 non-null   float64
 3   NO2      8224 non-null   float64
 4   NOX      8225 non-null   float64
 5   CO       8144 non-null   float64
 6   SO2      7189 non-null   float64
 7   NH3      8314 non-null   float64
 8   Ozone    8187 non-null   float64
 9   Benzene  2445 non-null   float64
dtypes: float64(10)
memory usage: 742.5 KB
```

[10]:
```
weights=[0.398,0.166,0.032,0.123,0.094,3,0.075,0.029,0.078,0.0004]
# Define the weights for each column
```

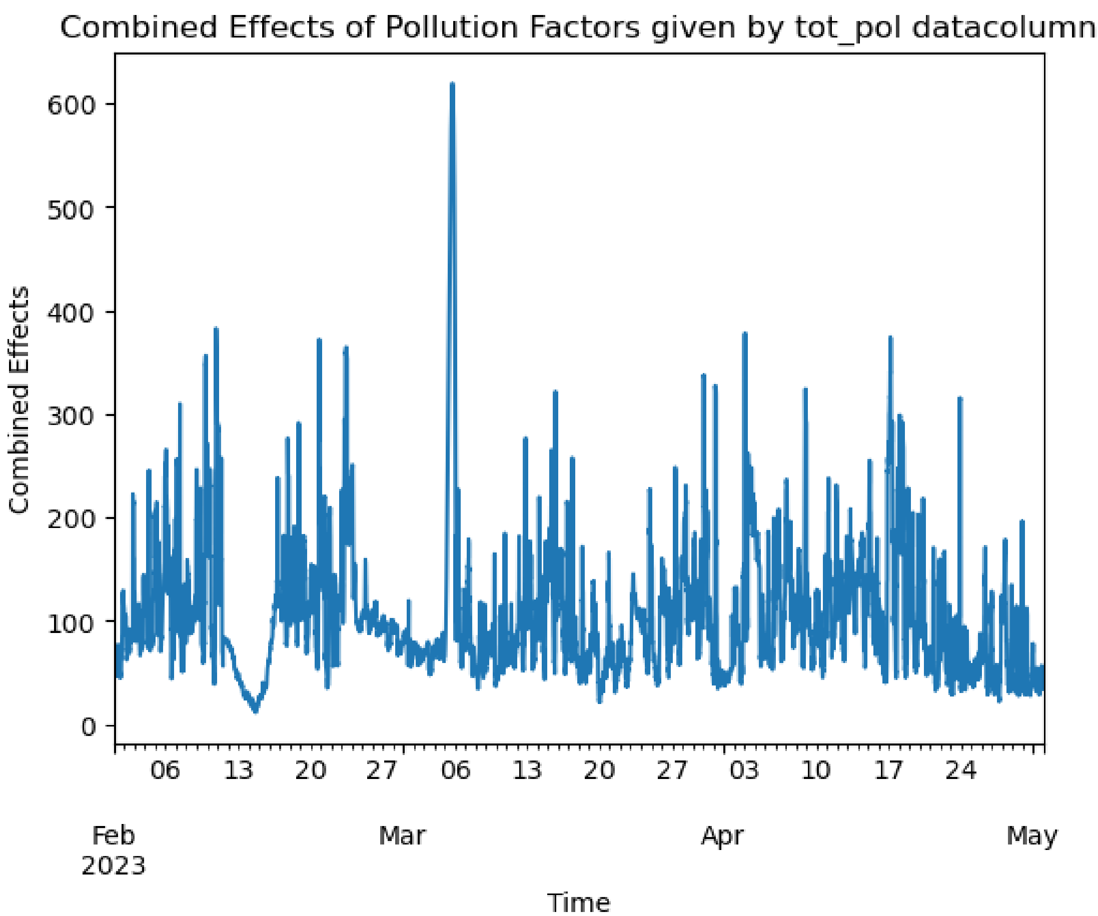
[11]:
```
# Calculate the weighted mean across the columns
df['tot_pol'] = (df.iloc[:, :10] * weights).sum(axis=1)
```

[12]:
```
df
```

[12]:
```
                      PM10  PM2.5    NO    NO2   NOX        CO        SO2  \
from
2023-02-01 00:00:00   95.0   35.0  18.1   90.1  56.2  0.310000   8.200000
2023-02-01 00:15:00   95.0   35.0  18.1   88.0  55.1  0.330000   8.200000
2023-02-01 00:30:00   95.0   35.0  18.1   87.7  55.2  0.380000   8.200000
2023-02-01 00:45:00  122.0   34.0  18.1   88.9  55.7  0.380000   8.200000
2023-02-01 01:00:00  122.0   34.0  18.1   90.0  55.8  0.380000   8.200000
…                      …      …     …      …     …         …          …
2023-05-01 22:45:00   19.0   11.0  17.9  100.0  67.8  0.630000  10.000000
2023-05-01 23:00:00   19.0   11.0  17.9  100.0  67.7  0.570000  10.000000
2023-05-01 23:15:00   19.0   11.0  19.6  100.2  69.2  0.580000   9.900000
2023-05-01 23:30:00   19.0   11.0  20.8  100.2  70.2  0.580000   9.500000
2023-05-01 23:45:00   32.0    6.0  21.8   98.8  70.3  0.828505   8.403109

                      NH3  Ozone  Benzene    tot_pol
from
2023-02-01 00:00:00  17.7   28.1      0.4  64.814560
```

```
2023-02-01 00:15:00  18.3   27.1       0.4  64.452260
2023-02-01 00:30:00  19.7   24.9       0.4  64.443760
2023-02-01 00:45:00  21.3   21.9       0.4  75.030760
2023-02-01 01:00:00  22.3   16.7       0.4  74.798860
…                      …     …          …       …
2023-05-01 22:45:00  10.7   26.1       0.1  33.620140
2023-05-01 23:00:00  10.4   30.9       0.1  33.796440
2023-05-01 23:15:00  10.5   29.6       0.1  33.940440
2023-05-01 23:30:00  10.8   30.0       0.1  34.082740
2023-05-01 23:45:00  11.0   33.5       0.1  39.237987

[8640 rows x 11 columns]
```

[13]:
```
df['tot_pol'].plot()
plt.xlabel('Time')
plt.ylabel('Combined Effects')
plt.title('Combined Effects of Pollution Factors given by tot_pol datacolumn')
plt.show()
```



Checking for stationarity in the new column data.

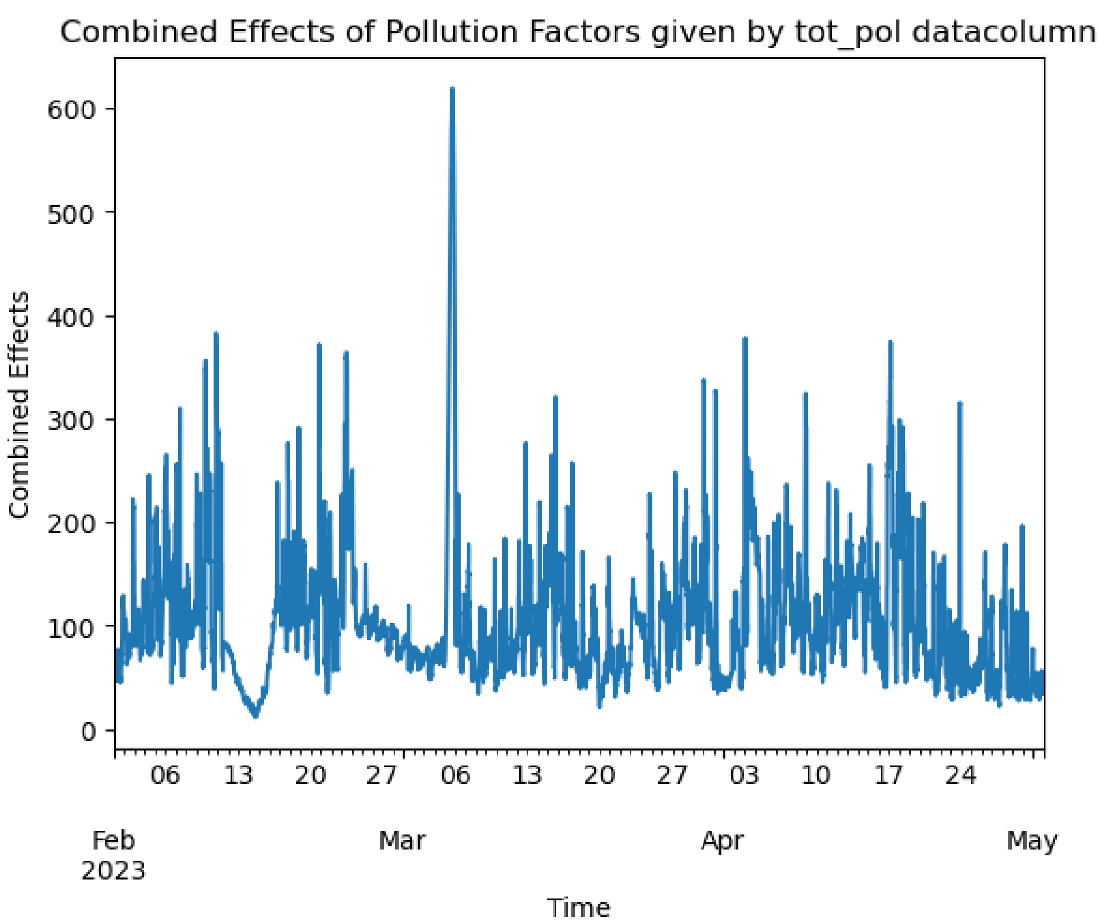
[14]:
```
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(df['tot_pol'])
print(f'p-value: {adf_test[1]}')
```

```
p-value: 5.741498169692062e-13
```

Such a low p-value implies that tot_pol(combined effective data) also follows time-series stationarity.

[15]:
```
# df = df.resample('D').mean()
```

[16]:
```
df['tot_pol'].plot()
plt.xlabel('Time')
plt.ylabel('Combined Effects')
plt.title('Combined Effects of Pollution Factors given by tot_pol datacolumn')
plt.show()
```



Finding Blasting Time

June 27, 2023

1 Part 4: Finding Blasting Time

Air quality monitoring is regularly carried out at both dust generating and non-generating locations in the vicinity in order to evaluate the particulate pollution in and around the opencast mining projects of the Singrauli coalfield.

Air pollution measurements available via multi-sensory system are PM10, PM2.5, SO2, NO2, NOx, CO, NH3, O3 and BENZENE.

[1]:
```
!pip install pmdarima
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pmdarima in c:\programdata\anaconda3\lib\site-
packages (2.0.3)
Requirement already satisfied: urllib3 in c:\programdata\anaconda3\lib\site-
packages (from pmdarima) (1.26.14)
Requirement already satisfied: statsmodels>=0.13.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.13.5)
Requirement already satisfied: joblib>=0.11 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.1.1)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (65.6.3)
Requirement already satisfied: numpy>=1.21.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.10.0)
Requirement already satisfied: pandas>=0.19 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.5.3)
Requirement already satisfied: scikit-learn>=0.22 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.2.1)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.29.35)
Requirement already satisfied: pytz>=2020.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2022.7)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.22->pmdarima)
```

```
(2.2.0)
Requirement already satisfied: packaging>=21.3 in
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(22.0)
Requirement already satisfied: patsy>=0.5.2 in
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(0.5.3)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages
(from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
```

During blasting, pollution levels reach their maximum.To detect these blasting times on each day, we will find out outliers in the givenb data and then find out corresponding timestamps. From those timestamps one with maximum frequency of outliers(i.e. blasting) will be the blasting time.

Here to find outliers, we have first replaced missing values with spline interpolation of order 3. After that a combined weighted combination "tot_col" is used to capture effective pattern in the concentration of pollutants across time.

[2]:
```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib notebook
import pandas as pd
import pandas.plotting
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from scipy.stats import norm
from statsmodels.graphics.gofplots import qqplot
```

[3]:
```
file_path = 'C:/Users/Omkar/Desktop/EE798Q/Open pit blasting 01-02-2023 000000␣
↪To 01-05-2023 235959.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path , index_col=0)
```

[4]:
```
# Simplify column names
df.columns = ['from', 'to', 'PM10', 'PM2.5',␣
↪'NO','NO2','NOX','CO','SO2','NH3','Ozone','Benzene']
# deleting to column as we need only one timestamp column for to be index and␣
↪we choose it to be from column
df = df.drop('to', axis=1)
# removing last 3 rows as they contaion max , min , avg data instead of actual␣
↪observations
df = df.iloc[:-3]
df.tail()
```

[4]:
```
                      from  PM10  PM2.5    NO    NO2   NOX    CO   SO2   NH3  \
#
8636  2023-05-01 22:45:00  19.0   11.0  17.9  100.0  67.8  0.63  10.0  10.7
8637  2023-05-01 23:00:00  19.0   11.0  17.9  100.0  67.7  0.57  10.0  10.4
8638  2023-05-01 23:15:00  19.0   11.0  19.6  100.2  69.2  0.58   9.9  10.5
8639  2023-05-01 23:30:00  19.0   11.0  20.8  100.2  70.2  0.58   9.5  10.8
8640  2023-05-01 23:45:00  32.0    6.0  21.8   98.8  70.3   NaN   NaN  11.0

      Ozone  Benzene
#
8636   26.1      0.1
8637   30.9      0.1
8638   29.6      0.1
8639   30.0      0.1
8640   33.5      0.1
```

[5]:
```python
# conveting timestamp as a string object into a datetime numerical
date_format = '%Y-%m-%d %H:%M:%S'

# Convert the 'from' column to numerical datetime representation
df['from'] = pd.to_datetime(df['from'], format=date_format)
```

[6]:
```python
# set datetime "from" column as an index column
df.set_index('from', inplace=True)
df.head()
```

[6]:
```
                      PM10  PM2.5  NO   NO2   NOX    CO  SO2   NH3  Ozone  \
from
2023-02-01 00:00:00   95.0   35.0 NaN  90.1  56.2  0.31  NaN  17.7   28.1
2023-02-01 00:15:00   95.0   35.0 NaN  88.0  55.1  0.33  NaN  18.3   27.1
2023-02-01 00:30:00   95.0   35.0 NaN  87.7  55.2  0.38  NaN  19.7   24.9
2023-02-01 00:45:00  122.0   34.0 NaN  88.9  55.7  0.38  NaN  21.3   21.9
2023-02-01 01:00:00  122.0   34.0 NaN  90.0  55.8  0.38  NaN  22.3   16.7

                     Benzene
from
2023-02-01 00:00:00      0.4
2023-02-01 00:15:00      0.4
2023-02-01 00:30:00      0.4
2023-02-01 00:45:00      0.4
2023-02-01 01:00:00      0.4
```

[7]:
```python
# # resample
# # df = df.resample('D').mean()
# interpolating
df = df.interpolate(method='spline',order=3)
df.fillna(method='ffill', inplace=True)  # Fill missing values forward
```

```
df.fillna(method='bfill', inplace=True)  # Fill missing values backward
# df = df.fillna(0)
```

[8]:
```
# t=df.index[4500]
# msk=(df.index<=t)
# df = df[~msk].copy()
```

[9]:
```
df
```

[9]:
```
                     PM10  PM2.5    NO    NO2   NOX        CO        SO2  \
from
2023-02-01 00:00:00   95.0   35.0  18.1   90.1  56.2  0.310000   8.200000
2023-02-01 00:15:00   95.0   35.0  18.1   88.0  55.1  0.330000   8.200000
2023-02-01 00:30:00   95.0   35.0  18.1   87.7  55.2  0.380000   8.200000
2023-02-01 00:45:00  122.0   34.0  18.1   88.9  55.7  0.380000   8.200000
2023-02-01 01:00:00  122.0   34.0  18.1   90.0  55.8  0.380000   8.200000
…                      …     …     …      …     …       …          …
2023-05-01 22:45:00   19.0   11.0  17.9  100.0  67.8  0.630000  10.000000
2023-05-01 23:00:00   19.0   11.0  17.9  100.0  67.7  0.570000  10.000000
2023-05-01 23:15:00   19.0   11.0  19.6  100.2  69.2  0.580000   9.900000
2023-05-01 23:30:00   19.0   11.0  20.8  100.2  70.2  0.580000   9.500000
2023-05-01 23:45:00   32.0    6.0  21.8   98.8  70.3  0.828505   8.403109

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

[8640 rows x 10 columns]
```

The weights are decided by the same way as before in section "Finding combined weighted mean." (i.e. wt= mean of column/sum of means of all columns)

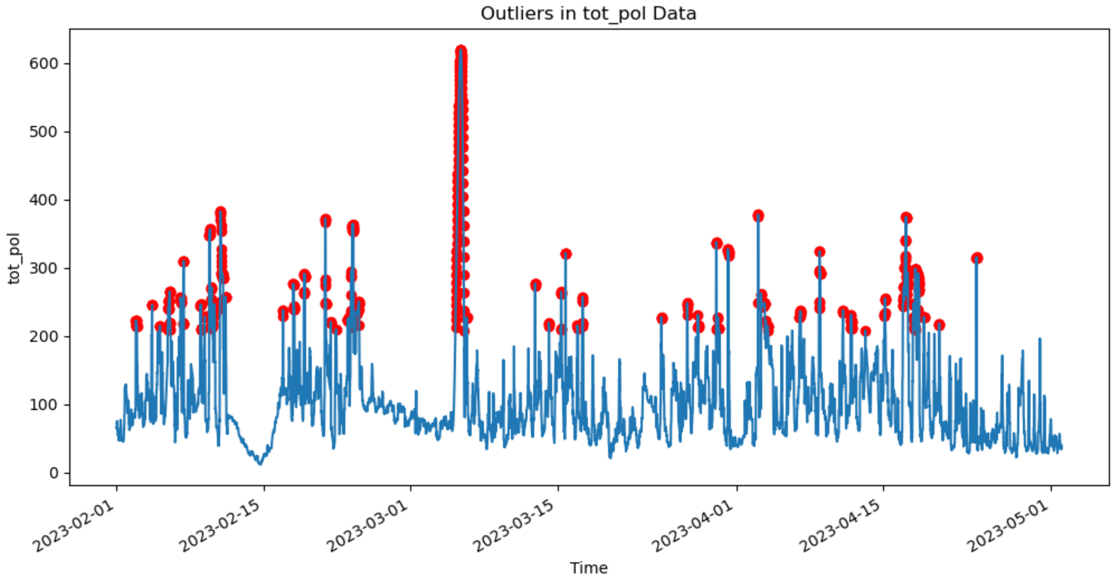
[10]:
```
weights=[0.398,0.166,0.032,0.123,0.094,3,0.075,0.029,0.078,0.0004]
# Define the weights for each column
```

[11]:
```
# Calculate the weighted mean across the columns
df['tot_pol'] = (df.iloc[:, :10] * weights).sum(axis=1)
```

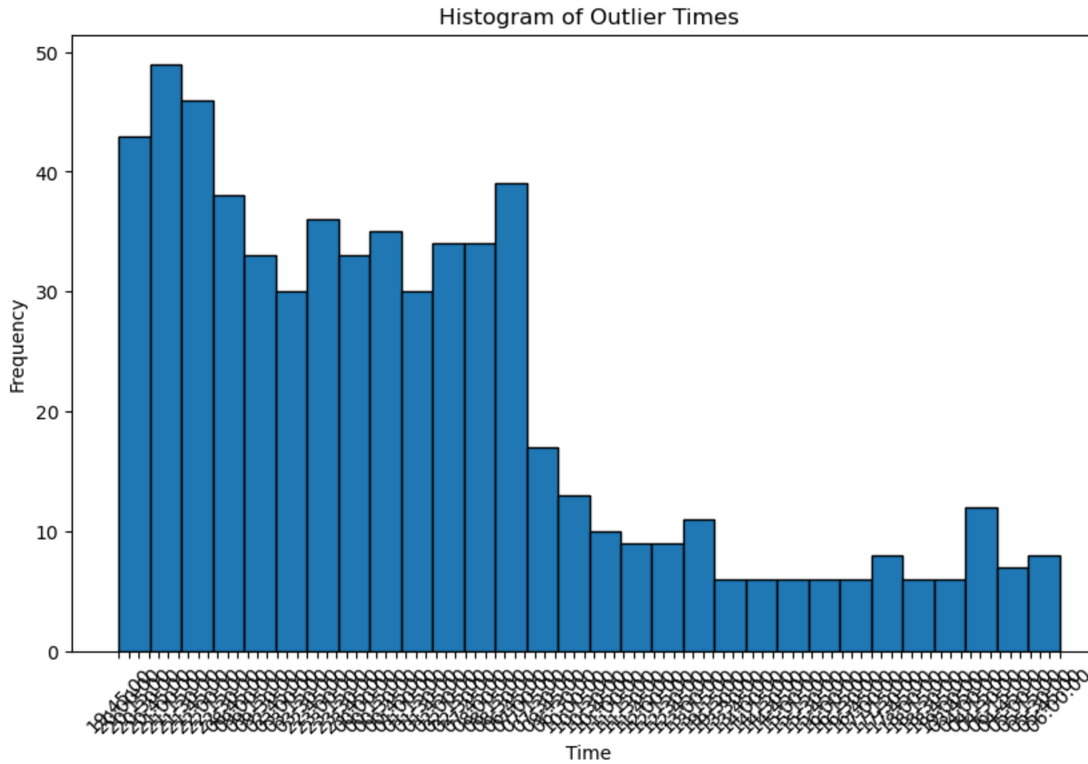
[12]:
```python
import pandas as pd

# Assuming your DataFrame is named 'df'

# Calculate mean and standard deviation
mean = df['tot_pol'].mean()
std = df['tot_pol'].std()

# Filter outliers based on the condition (greater than mean + 2 * standard␣
↪deviation)
outliers = df[df['tot_pol'] > (mean + 1.5* std)]



# Count the frequency of outliers for each time
outliers_count = outliers.groupby(outliers.index.time).size()

# Find the time with the most frequent outlier occurrences
most_frequent_time = outliers_count.idxmax()

# Print the most frequent time and its frequency
print("Most Frequent Blasting Time:")
print(f"Time: {most_frequent_time}")
```

```
Most Frequent Blasting Time:
Time: 20:45:00
```

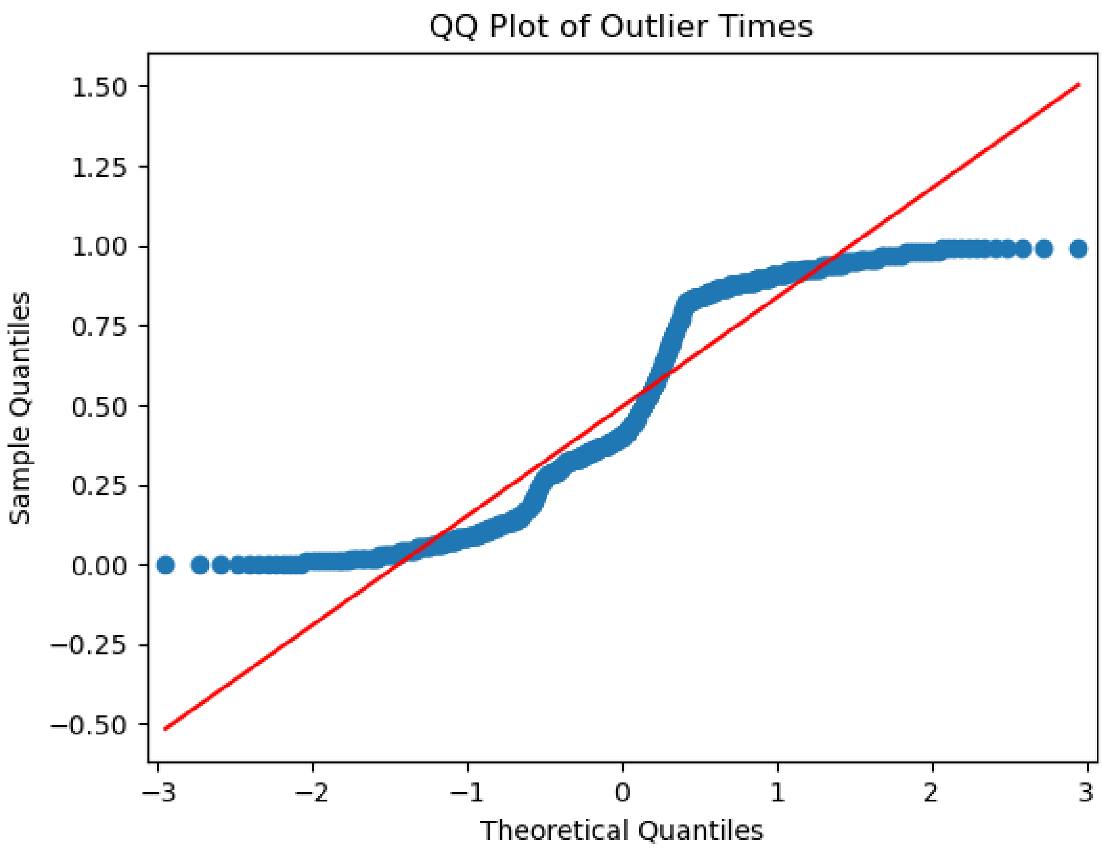
[13]:
```python
# Plot the outliers in a new graph
outliers.plot(y='tot_pol', style='ro', figsize=(12, 6), legend=False)
df['tot_pol'].plot()
plt.xlabel('Time')
plt.ylabel('tot_pol')
plt.title('Outliers in tot_pol Data')
plt.show()
```



[14]:
```
import matplotlib.pyplot as plt
%matplotlib inline
outlier_times = outliers.index.strftime('%H:%M:%S')  # Convert datetime to␣
↪string format

plt.figure(figsize=(10, 6))
plt.hist(outlier_times, bins=30, edgecolor='black')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.title('Histogram of Outlier Times')
plt.xticks(rotation=45)
plt.show()
```




```
[15]: import statsmodels.api as sm
%matplotlib inline
# Perform a QQ plot to infer the distribution (Normal or not)
# Convert outlier times to fractional values of a day
outlier_times_numeric = pd.to_timedelta(outlier_times).total_seconds() / (60 *
 ↪60 * 24)

# Perform a QQ plot using the outlier times
sm.qqplot(outlier_times_numeric, line='s')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.title('QQ Plot of Outlier Times')
plt.show()
```



As the curve is not lying along the red line, this indicates that data is not following normal distribution.

[16]:
```
# Find the probability of outliers happening during 14:15 to 14:30
start_time = pd.to_datetime('14:15:00').time()
end_time = pd.to_datetime('14:30:00').time()

outliers_between_interval = outliers[(outliers.index.time >= start_time) &
    (outliers.index.time <= end_time)]
probability = len(outliers_between_interval) / len(outliers)

print(f"Probability of blasting between {start_time} and {end_time}:
    {probability}")
```

```
Probability of blasting between 14:15:00 and 14:30:00: 0.006389776357827476
```

Hence the probability of open-pit blast happening during 14:15 to 14:30 is 0.5%.

[ ]:
```
```

Curve Fitting in Time Series Data

June 27, 2023

1 Part 5: Curve Fitting

Curve fitting is a technique used in the analysis of pollution data to study the relationships between variables and make predictions or extrapolations.It helps in:

Data Smoothing: Curve fitting can be used to smooth out noisy or erratic pollution data. By fitting a curve to the data, we can remove the random fluctuations and obtain a smoother representation of the pollution levels, making it easier to identify underlying patterns or trends.

Comparative Analysis: Curve fitting allows to compare the pollution levels of different pollutants on the same scale. By fitting curves for multiple pollutants on a single plot, we can visually compare their concentrations and observe any similarities or differences in their trends over time.

Overall, curve fitting helps in understanding the patterns, trends, and relationships within the pollution data, enabling better analysis, visualization, and interpretation of the data.

Here we have plotted the air pollution data along a curve to study the relationships of variables within the data.

Also non-parametric curve fitting or fitting data via parametric distributions is explored via methods like spline interpolation, lowess regression and also fitting data using methods like polynomial curve fitting.

[1]:
```
!pip install pmdarima
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pmdarima in c:\programdata\anaconda3\lib\site-
packages (2.0.3)
Requirement already satisfied: scikit-learn>=0.22 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.2.1)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.29.35)
Requirement already satisfied: pandas>=0.19 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.5.3)
Requirement already satisfied: scipy>=1.3.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.10.0)
Requirement already satisfied: statsmodels>=0.13.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.13.5)
Requirement already satisfied: urllib3 in c:\programdata\anaconda3\lib\site-
packages (from pmdarima) (1.26.14)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
```

```
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (65.6.3)
Requirement already satisfied: joblib>=0.11 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.1.1)
Requirement already satisfied: numpy>=1.21.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.23.5)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2022.7)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.22->pmdarima)
(2.2.0)
Requirement already satisfied: patsy>=0.5.2 in
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(0.5.3)
Requirement already satisfied: packaging>=21.3 in
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(22.0)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages
(from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
```

[2]:
```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib notebook
import pandas as pd
import pandas.plotting
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


import seaborn as sns
from scipy.stats import norm
from statsmodels.graphics.gofplots import qqplot
```

[3]:
```
file_path = 'C:/Users/Omkar/Desktop/EE798Q/Open pit blasting 01-02-2023 000000␣
↪To 01-05-2023 235959.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path , index_col=0)
```

[4]:
```
# Simplify column names
df.columns = ['from', 'to', 'PM10', 'PM2.5',␣
↪'NO','NO2','NOX','CO','SO2','NH3','Ozone','Benzene']
```

```
df['CO']*=1000
# deleting to column as we need only one timestamp column for to be index and␣
↪we choose it to be from column
df = df.drop('to', axis=1)
# removing last 3 rows as they contaion max , min , avg data instead of actual␣
↪observations
df = df.iloc[:-3]
df.tail()
```

[4]:

```
                     from  PM10  PM2.5    NO    NO2   NOX     CO   SO2   NH3  \
#
8636  2023-05-01 22:45:00  19.0   11.0  17.9  100.0  67.8  630.0  10.0  10.7
8637  2023-05-01 23:00:00  19.0   11.0  17.9  100.0  67.7  570.0  10.0  10.4
8638  2023-05-01 23:15:00  19.0   11.0  19.6  100.2  69.2  580.0   9.9  10.5
8639  2023-05-01 23:30:00  19.0   11.0  20.8  100.2  70.2  580.0   9.5  10.8
8640  2023-05-01 23:45:00  32.0    6.0  21.8   98.8  70.3    NaN   NaN  11.0

      Ozone  Benzene
#
8636   26.1      0.1
8637   30.9      0.1
8638   29.6      0.1
8639   30.0      0.1
8640   33.5      0.1
```

[5]:

```
# conveting timestamp as a string object into a datetime numerical
date_format = '%Y-%m-%d %H:%M:%S'

# Convert the 'from' column to numerical datetime representation
df['from'] = pd.to_datetime(df['from'], format=date_format)
```

[6]:

```
# set datetime "from" column as an index column
df.set_index('from', inplace=True)
df.head()
```

[6]:

```
                      PM10  PM2.5  NO   NO2   NOX     CO  SO2   NH3  Ozone  \
from
2023-02-01 00:00:00   95.0   35.0 NaN  90.1  56.2  310.0  NaN  17.7   28.1
2023-02-01 00:15:00   95.0   35.0 NaN  88.0  55.1  330.0  NaN  18.3   27.1
2023-02-01 00:30:00   95.0   35.0 NaN  87.7  55.2  380.0  NaN  19.7   24.9
2023-02-01 00:45:00  122.0   34.0 NaN  88.9  55.7  380.0  NaN  21.3   21.9
2023-02-01 01:00:00  122.0   34.0 NaN  90.0  55.8  380.0  NaN  22.3   16.7

                     Benzene
from
2023-02-01 00:00:00      0.4
2023-02-01 00:15:00      0.4
```

```
2023-02-01 00:30:00      0.4
2023-02-01 00:45:00      0.4
2023-02-01 01:00:00      0.4
```

[7]:

```python
# resample
# df = df.resample('D').mean()
# interpolating
df = df.interpolate(method='spline',order=3)
df.fillna(method='ffill', inplace=True)  # Fill missing values forward
df.fillna(method='bfill', inplace=True)  # Fill missing values backward
```

[8]:

```python
df
```

[8]:

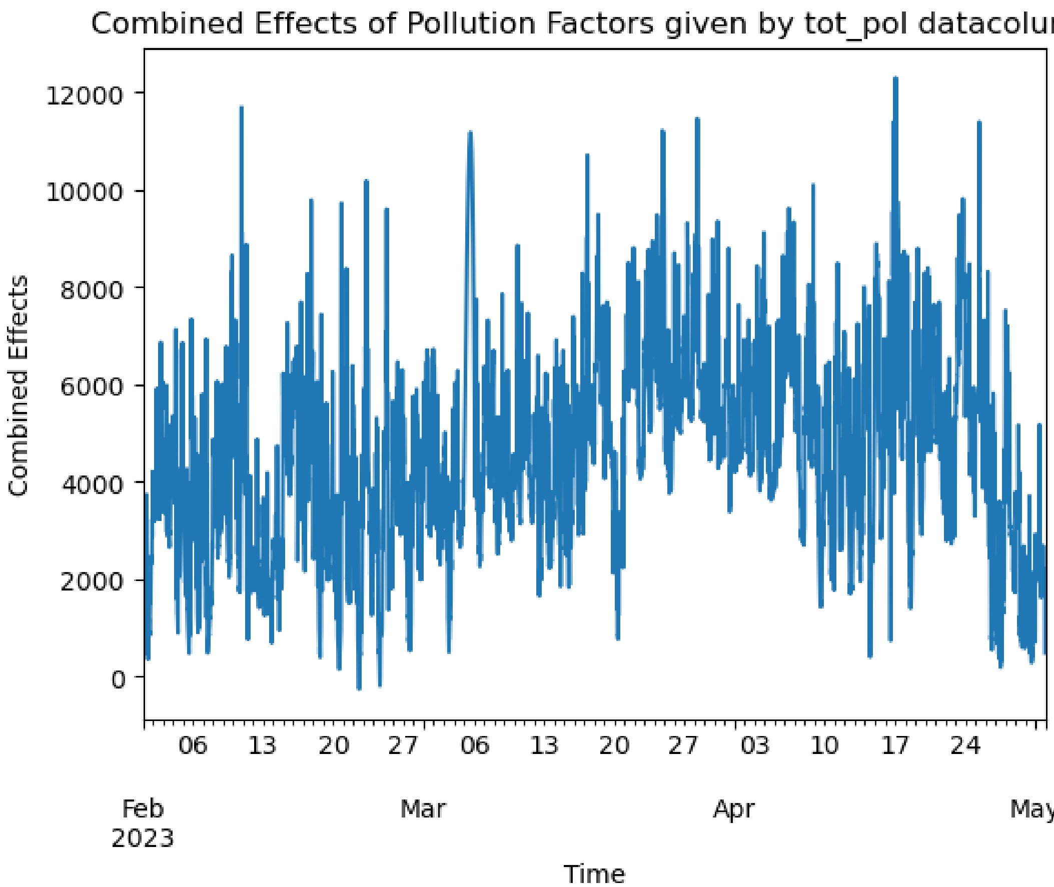
```
                      PM10  PM2.5    NO    NO2   NOX          CO        SO2  \
from
2023-02-01 00:00:00   95.0   35.0  18.1   90.1  56.2  310.000000   8.200000
2023-02-01 00:15:00   95.0   35.0  18.1   88.0  55.1  330.000000   8.200000
2023-02-01 00:30:00   95.0   35.0  18.1   87.7  55.2  380.000000   8.200000
2023-02-01 00:45:00  122.0   34.0  18.1   88.9  55.7  380.000000   8.200000
2023-02-01 01:00:00  122.0   34.0  18.1   90.0  55.8  380.000000   8.200000
…                      …      …     …      …     …           …          …
2023-05-01 22:45:00   19.0   11.0  17.9  100.0  67.8  630.000000  10.000000
2023-05-01 23:00:00   19.0   11.0  17.9  100.0  67.7  570.000000  10.000000
2023-05-01 23:15:00   19.0   11.0  19.6  100.2  69.2  580.000000   9.900000
2023-05-01 23:30:00   19.0   11.0  20.8  100.2  70.2  580.000000   9.500000
2023-05-01 23:45:00   32.0    6.0  21.8   98.8  70.3  147.845344   8.403109

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

[8640 rows x 10 columns]
```

[9]:

```python
weights=[0.398,0.166,0.032,0.123,0.094,3,0.075,0.029,0.078,0.0004]
# Define the weights for each column
```

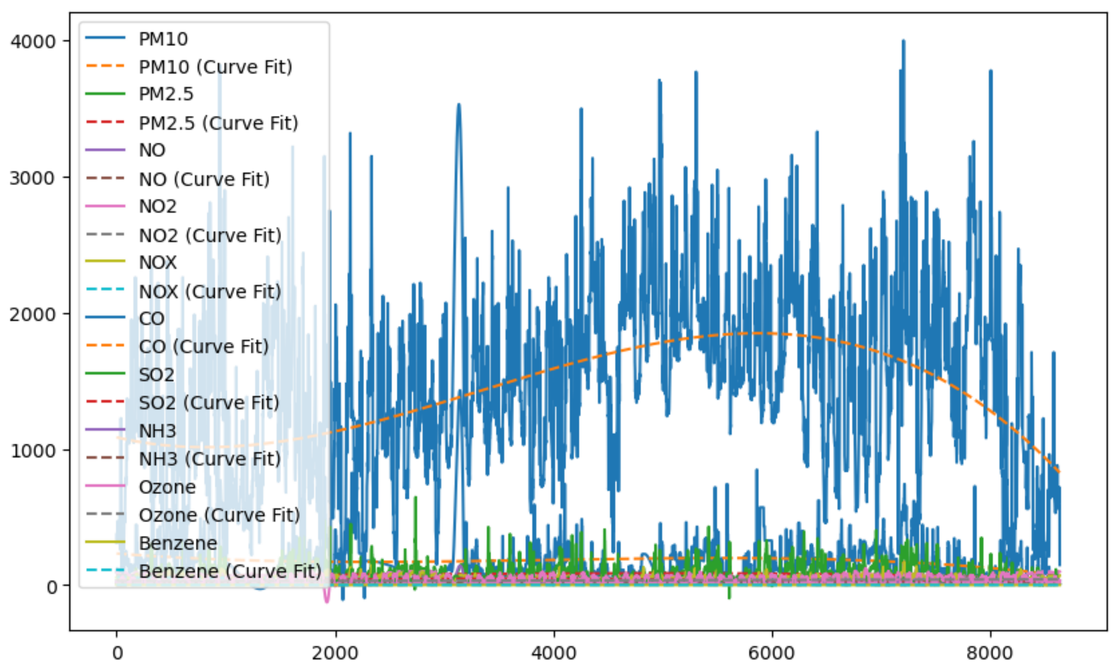
[10]:
```
# Calculate the weighted mean across the columns
df['tot_pol'] = (df.iloc[:, :10] * weights).sum(axis=1)
```

[11]:
```
df['tot_pol'].plot()
plt.xlabel('Time')
plt.ylabel('Combined Effects')
plt.title('Combined Effects of Pollution Factors given by tot_pol datacolumn')
plt.show()
```



[12]:
```
%matplotlib inline
from scipy.optimize import curve_fit
import matplotlib.dates as mdates

# Define the pollutants of interest
pollutants = ['PM10', 'PM2.5', 'NO', 'NO2', 'NOX', 'CO', 'SO2', 'NH3', 'Ozone',␣
 ↪'Benzene']

# Fit a polynomial curve for each pollutant
```

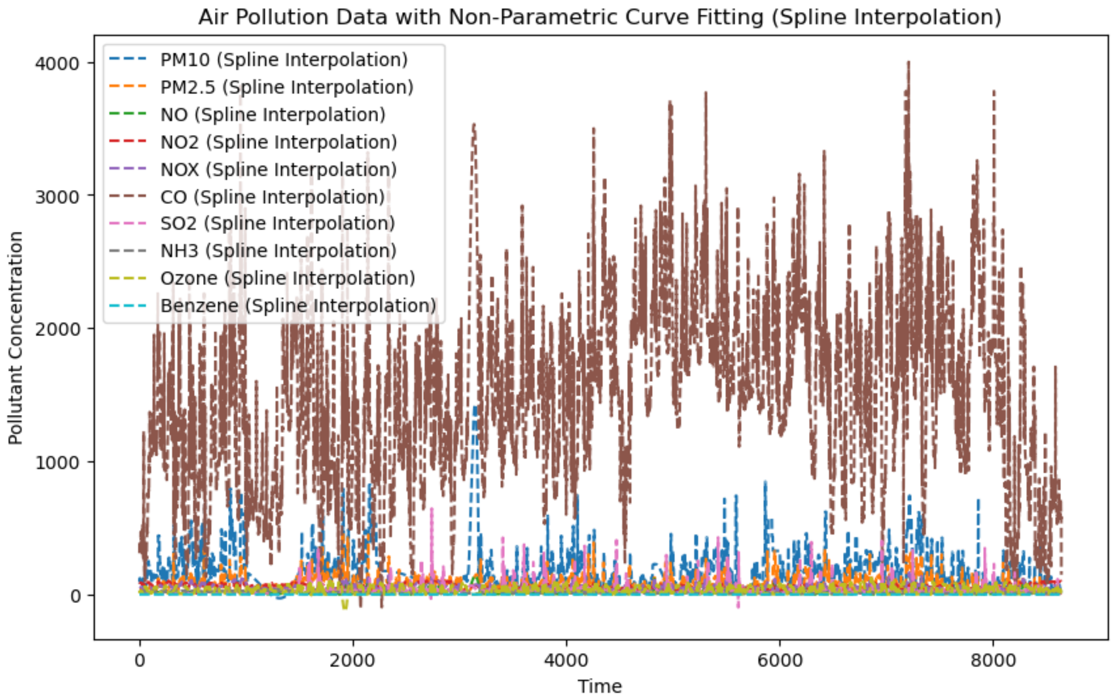
```python
curve_params = {}
for pollutant in pollutants:
    x = np.arange(len(df))
    y = df[pollutant].values
    # Adjust the degree of the polynomial as needed
    degree = 3
    # Perform curve fitting
    params = np.polyfit(x, y, degree)
    curve_params[pollutant] = params

# Plot the air pollution data along the curve
fig, ax = plt.subplots(figsize=(10, 6))
for pollutant in pollutants:
    x = np.arange(len(df))
    y = df[pollutant].values
    params = curve_params[pollutant]
    y_fit = np.polyval(params, x)
    ax.plot(x, y, label=pollutant)
    ax.plot(x, y_fit, '--', label=pollutant + ' (Curve Fit)')
    plt.legend()
```



In this example, a polynomial curve of degree 3 is fitted to the pollutant data for each variable.

[13]:
```python
from scipy.interpolate import interp1d
# Define the pollutants of interest
```

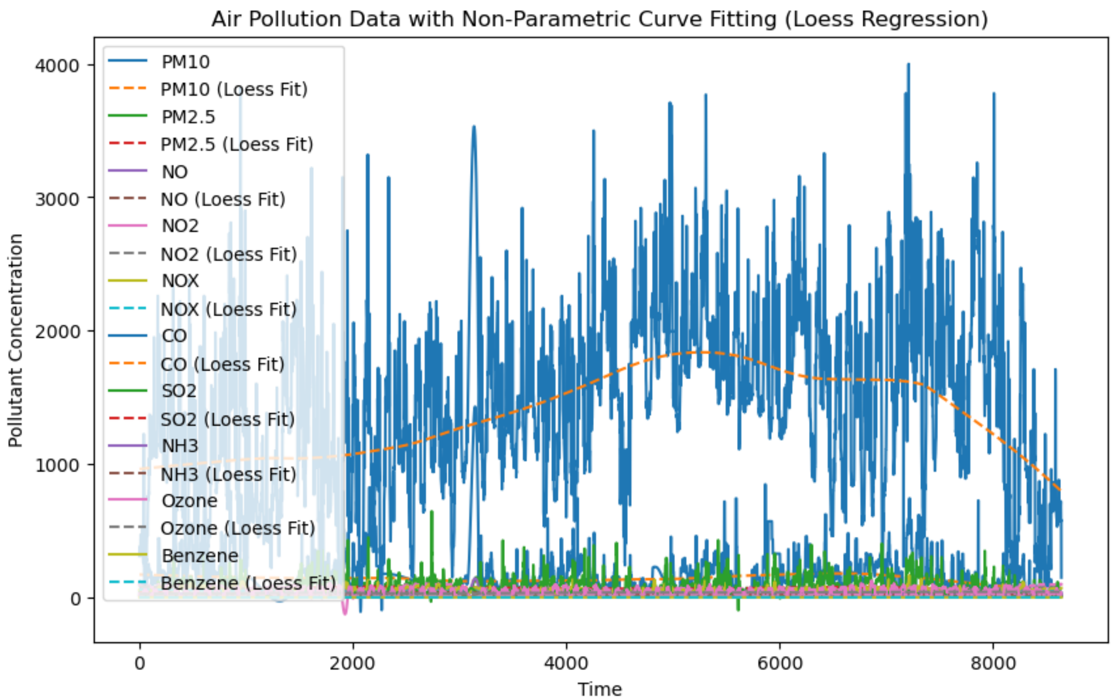
```
pollutants = ['PM10', 'PM2.5', 'NO', 'NO2', 'NOX', 'CO', 'SO2', 'NH3', 'Ozone', 'Benzene']

# Fit a non-parametric curve (spline interpolation) for each pollutant
curve_data = {}
for pollutant in pollutants:
    x = np.arange(len(df))
    y = df[pollutant].values
    # Perform spline interpolation
    spline_fit = interp1d(x, y, kind='cubic')
    curve_data[pollutant] = spline_fit(x)

# Plot the air pollution data with non-parametric curve fitting
fig, ax = plt.subplots(figsize=(10, 6))
for pollutant in pollutants:
    x = np.arange(len(df))
    y = df[pollutant].values
    spline_fit = curve_data[pollutant]
#     ax.plot(x, y, label=pollutant)
    ax.plot(x, spline_fit, '--', label=pollutant + ' (Spline Interpolation)')

plt.xlabel('Time')
plt.ylabel('Pollutant Concentration')
plt.title('Air Pollution Data with Non-Parametric Curve Fitting (Spline Interpolation)')
plt.legend()
plt.show()
```



[14]:

```python
from statsmodels.nonparametric.smoothers_lowess import lowess
# Define the pollutants of interest
pollutants = ['PM10', 'PM2.5', 'NO', 'NO2', 'NOX', 'CO', 'SO2', 'NH3', 'Ozone',␣
 ↪'Benzene']

# Fit a non-parametric curve (loess regression) for each pollutant
curve_data = {}
for pollutant in pollutants:
    x = np.arange(len(df))
    y = df[pollutant].values
    # Set the span parameter for loess regression
    span = 0.3
    # Perform loess regression
    loess_fit = lowess(y, x, frac=span)
    curve_data[pollutant] = loess_fit[:, 1]

# Plot the air pollution data with non-parametric curve fitting
fig, ax = plt.subplots(figsize=(10, 6))
for pollutant in pollutants:
    x = np.arange(len(df))
    y = df[pollutant].values
    loess_fit = curve_data[pollutant]
    ax.plot(x, y, label=pollutant)
    ax.plot(x, loess_fit, '--', label=pollutant + ' (Loess Fit)')
```

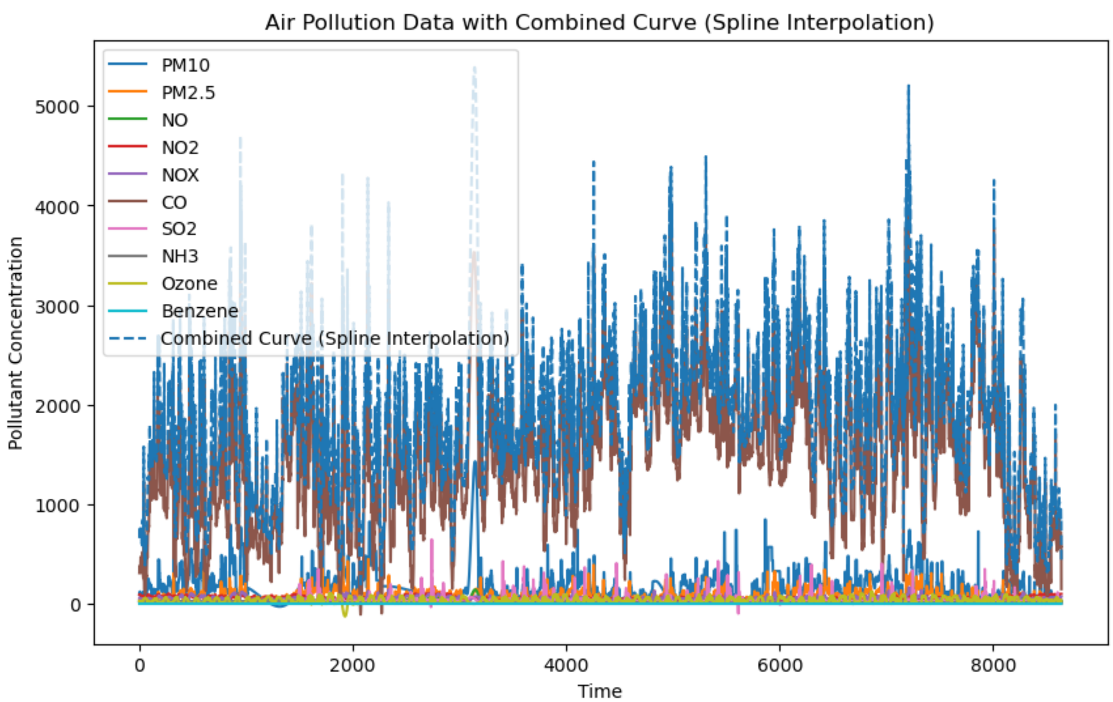
```
plt.xlabel('Time')
plt.ylabel('Pollutant Concentration')
plt.title('Air Pollution Data with Non-Parametric Curve Fitting (Loess␣
↪Regression)')
plt.legend()
plt.show()
```



In this code, the statsmodels.nonparametric.smoothers_lowess module is used to perform loess regression using the lowess() function. The frac parameter specifies the span or fraction of data points to use for each local regression.

[15]:
```
from scipy.interpolate import interp1d
# Define the pollutants of interest
pollutants = ['PM10', 'PM2.5', 'NO', 'NO2', 'NOX', 'CO', 'SO2', 'NH3', 'Ozone',␣
↪'Benzene']

# Concatenate the air pollution data into a single curve
x = np.arange(len(df))
y_combined = np.zeros_like(x, dtype=float)  # Initialize as float array
for pollutant in pollutants:
    y = df[pollutant].values
    y_combined += y
```

```python
# Perform spline interpolation on the combined curve
spline_fit = interp1d(x, y_combined, kind='cubic')

# Plot the air pollution data with combined curve using spline interpolation
fig, ax = plt.subplots(figsize=(10, 6))
for pollutant in pollutants:
    y = df[pollutant].values
    ax.plot(x, y, label=pollutant)

ax.plot(x, spline_fit(x), '--', label='Combined Curve (Spline Interpolation)')

plt.xlabel('Time')
plt.ylabel('Pollutant Concentration')
plt.title('Air Pollution Data with Combined Curve (Spline Interpolation)')
plt.legend()
plt.show()
```



In this code, the scipy.interpolate.interp1d function is used to perform spline interpolation with the kind='cubic' option specifying cubic spline interpolation.

Trends & Seasonality

June 27, 2023

0.1 Part 6: Trends and Seasonality

In this section of report, we will try to analyze trends, seasonality, periodicity, if any present in our data.

[1]:
```
!pip install pmdarima
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pmdarima in c:\programdata\anaconda3\lib\site-
packages (2.0.3)
Requirement already satisfied: numpy>=1.21.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.10.0)
Requirement already satisfied: pandas>=0.19 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.5.3)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (65.6.3)
Requirement already satisfied: joblib>=0.11 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.1.1)
Requirement already satisfied: scikit-learn>=0.22 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (1.2.1)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.29.35)
Requirement already satisfied: urllib3 in c:\programdata\anaconda3\lib\site-
packages (from pmdarima) (1.26.14)
Requirement already satisfied: statsmodels>=0.13.2 in
c:\programdata\anaconda3\lib\site-packages (from pmdarima) (0.13.5)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2022.7)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.22->pmdarima)
(2.2.0)
Requirement already satisfied: patsy>=0.5.2 in
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(0.5.3)
```

```
Requirement already satisfied: packaging>=21.3 in
c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(22.0)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages
(from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
```

[2]:

```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib notebook
import pandas as pd
import pandas.plotting
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
%matplotlib inline
```

[3]:

```
file_path = 'C:/Users/Omkar/Desktop/EE798Q/Open pit blasting 01-02-2023 000000␣
↪To 01-05-2023 235959.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path , index_col=0)
```

[4]:

```
# Simplify column names
df.columns = ['from', 'to', 'PM10', 'PM2.5',␣
↪'NO','NO2','NOX','CO','SO2','NH3','Ozone','Benzene']
# deleting to column as we need only one timestamp column for to be index and␣
↪we choose it to be from column
df = df.drop('to', axis=1)
# removing last 3 rows as they contaion max , min , avg data instead of actual␣
↪observations
df = df.iloc[:-3]
df.tail()
```

[4]:

```
                     from  PM10  PM2.5    NO    NO2   NOX    CO   SO2   NH3  \
#
8636  2023-05-01 22:45:00  19.0   11.0  17.9  100.0  67.8  0.63  10.0  10.7
8637  2023-05-01 23:00:00  19.0   11.0  17.9  100.0  67.7  0.57  10.0  10.4
8638  2023-05-01 23:15:00  19.0   11.0  19.6  100.2  69.2  0.58   9.9  10.5
8639  2023-05-01 23:30:00  19.0   11.0  20.8  100.2  70.2  0.58   9.5  10.8
8640  2023-05-01 23:45:00  32.0    6.0  21.8   98.8  70.3   NaN   NaN  11.0

      Ozone  Benzene
#
8636   26.1      0.1
8637   30.9      0.1
8638   29.6      0.1
8639   30.0      0.1
8640   33.5      0.1
```

[5]:
```
# conveting timestamp as a string object into a datetime numerical
date_format = '%Y-%m-%d %H:%M:%S'

# Convert the 'from' column to numerical datetime representation
df['from'] = pd.to_datetime(df['from'], format=date_format)
```

[6]:
```
# set datetime "from" column as an index column
df.set_index('from', inplace=True)
df.head()
```

[6]:
```
                      PM10  PM2.5  NO   NO2   NOX    CO  SO2   NH3  Ozone  \
from
2023-02-01 00:00:00   95.0   35.0 NaN  90.1  56.2  0.31  NaN  17.7   28.1
2023-02-01 00:15:00   95.0   35.0 NaN  88.0  55.1  0.33  NaN  18.3   27.1
2023-02-01 00:30:00   95.0   35.0 NaN  87.7  55.2  0.38  NaN  19.7   24.9
2023-02-01 00:45:00  122.0   34.0 NaN  88.9  55.7  0.38  NaN  21.3   21.9
2023-02-01 01:00:00  122.0   34.0 NaN  90.0  55.8  0.38  NaN  22.3   16.7

                     Benzene
from
2023-02-01 00:00:00      0.4
2023-02-01 00:15:00      0.4
2023-02-01 00:30:00      0.4
2023-02-01 00:45:00      0.4
2023-02-01 01:00:00      0.4
```

[7]:
```
df=df.copy()
```

[8]:
```
df
```

[8]:
```
                      PM10  PM2.5    NO    NO2   NOX    CO   SO2   NH3  Ozone  \
from
2023-02-01 00:00:00   95.0   35.0   NaN   90.1  56.2  0.31   NaN  17.7   28.1
2023-02-01 00:15:00   95.0   35.0   NaN   88.0  55.1  0.33   NaN  18.3   27.1
2023-02-01 00:30:00   95.0   35.0   NaN   87.7  55.2  0.38   NaN  19.7   24.9
2023-02-01 00:45:00  122.0   34.0   NaN   88.9  55.7  0.38   NaN  21.3   21.9
2023-02-01 01:00:00  122.0   34.0   NaN   90.0  55.8  0.38   NaN  22.3   16.7
…                      …    …     …     …     …     …     …     …      …
2023-05-01 22:45:00   19.0   11.0  17.9  100.0  67.8  0.63  10.0  10.7   26.1
2023-05-01 23:00:00   19.0   11.0  17.9  100.0  67.7  0.57  10.0  10.4   30.9
2023-05-01 23:15:00   19.0   11.0  19.6  100.2  69.2  0.58   9.9  10.5   29.6
2023-05-01 23:30:00   19.0   11.0  20.8  100.2  70.2  0.58   9.5  10.8   30.0
2023-05-01 23:45:00   32.0    6.0  21.8   98.8  70.3   NaN   NaN  11.0   33.5

                     Benzene
from
2023-02-01 00:00:00      0.4
```

```
2023-02-01 00:15:00      0.4
2023-02-01 00:30:00      0.4
2023-02-01 00:45:00      0.4
2023-02-01 01:00:00      0.4
…                       …
2023-05-01 22:45:00      0.1
2023-05-01 23:00:00      0.1
2023-05-01 23:15:00      0.1
2023-05-01 23:30:00      0.1
2023-05-01 23:45:00      0.1

[8640 rows x 10 columns]
```

[9]:
```
# resample
# df3 = df3.resample('D').mean()
# interpolating
PM10 = df['PM10']
df = df.interpolate(method='spline',order=3)
df.fillna(method='ffill', inplace=True)  # Fill missing values forward
df.fillna(method='bfill', inplace=True)  # Fill missing values backward
```

[10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8640 entries, 2023-02-01 00:00:00 to 2023-05-01 23:45:00
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   PM10     8640 non-null   float64
 1   PM2.5    8640 non-null   float64
 2   NO       8640 non-null   float64
 3   NO2      8640 non-null   float64
 4   NOX      8640 non-null   float64
 5   CO       8640 non-null   float64
 6   SO2      8640 non-null   float64
 7   NH3      8640 non-null   float64
 8   Ozone    8640 non-null   float64
 9   Benzene  8640 non-null   float64
dtypes: float64(10)
memory usage: 742.5 KB
```

Just like before we will first form a new data column, "tot_pol" which indicates overall effect of all pollution factors using weighted means.

For weights we will use formula,

weight= mean of a column/ sum of means of all columns

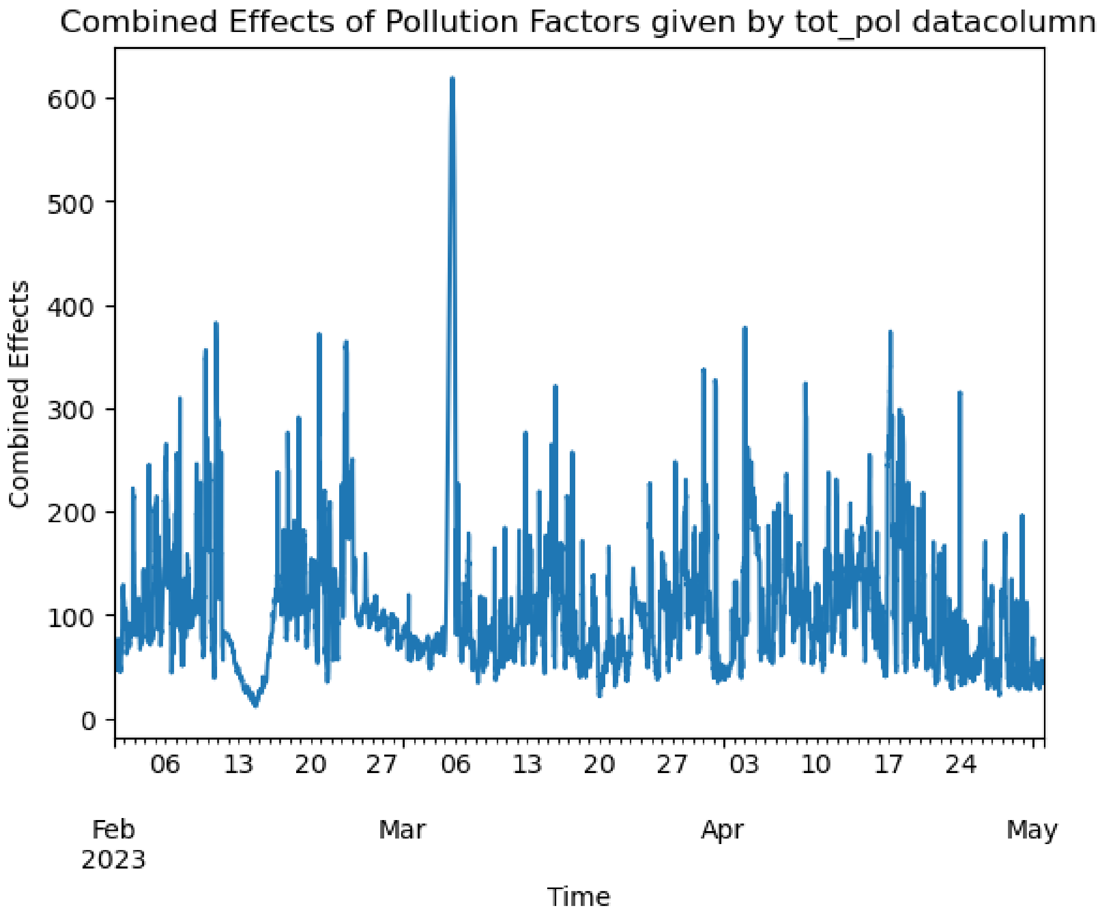
[11]:
```
weights=[0.398,0.166,0.032,0.123,0.094,3,0.075,0.029,0.078,0.0004]
# Define the weights for each column
```

[12]:
```
# Calculate the weighted mean across the columns
df['tot_pol'] = (df.iloc[:, :10] * weights).sum(axis=1)
```

[13]:
```
df
```

[13]:
```
                      PM10  PM2.5    NO    NO2   NOX        CO        SO2  \
from
2023-02-01 00:00:00   95.0   35.0  18.1   90.1  56.2  0.310000   8.200000
2023-02-01 00:15:00   95.0   35.0  18.1   88.0  55.1  0.330000   8.200000
2023-02-01 00:30:00   95.0   35.0  18.1   87.7  55.2  0.380000   8.200000
2023-02-01 00:45:00  122.0   34.0  18.1   88.9  55.7  0.380000   8.200000
2023-02-01 01:00:00  122.0   34.0  18.1   90.0  55.8  0.380000   8.200000
…                      …     …     …     …     …        …          …
2023-05-01 22:45:00   19.0   11.0  17.9  100.0  67.8  0.630000  10.000000
2023-05-01 23:00:00   19.0   11.0  17.9  100.0  67.7  0.570000  10.000000
2023-05-01 23:15:00   19.0   11.0  19.6  100.2  69.2  0.580000   9.900000
2023-05-01 23:30:00   19.0   11.0  20.8  100.2  70.2  0.580000   9.500000
2023-05-01 23:45:00   32.0    6.0  21.8   98.8  70.3  0.828505   8.403109

                      NH3  Ozone  Benzene    tot_pol
from
2023-02-01 00:00:00  17.7   28.1      0.4  64.814560
2023-02-01 00:15:00  18.3   27.1      0.4  64.452260
2023-02-01 00:30:00  19.7   24.9      0.4  64.443760
2023-02-01 00:45:00  21.3   21.9      0.4  75.030760
2023-02-01 01:00:00  22.3   16.7      0.4  74.798860
…                     …     …        …         …
2023-05-01 22:45:00  10.7   26.1      0.1  33.620140
2023-05-01 23:00:00  10.4   30.9      0.1  33.796440
2023-05-01 23:15:00  10.5   29.6      0.1  33.940440
2023-05-01 23:30:00  10.8   30.0      0.1  34.082740
2023-05-01 23:45:00  11.0   33.5      0.1  39.237987

[8640 rows x 11 columns]
```

[14]:
```
df['tot_pol'].plot()
plt.xlabel('Time')
plt.ylabel('Combined Effects')
plt.title('Combined Effects of Pollution Factors given by tot_pol datacolumn')
plt.show()
```



Checking for stationarity in the new column data.

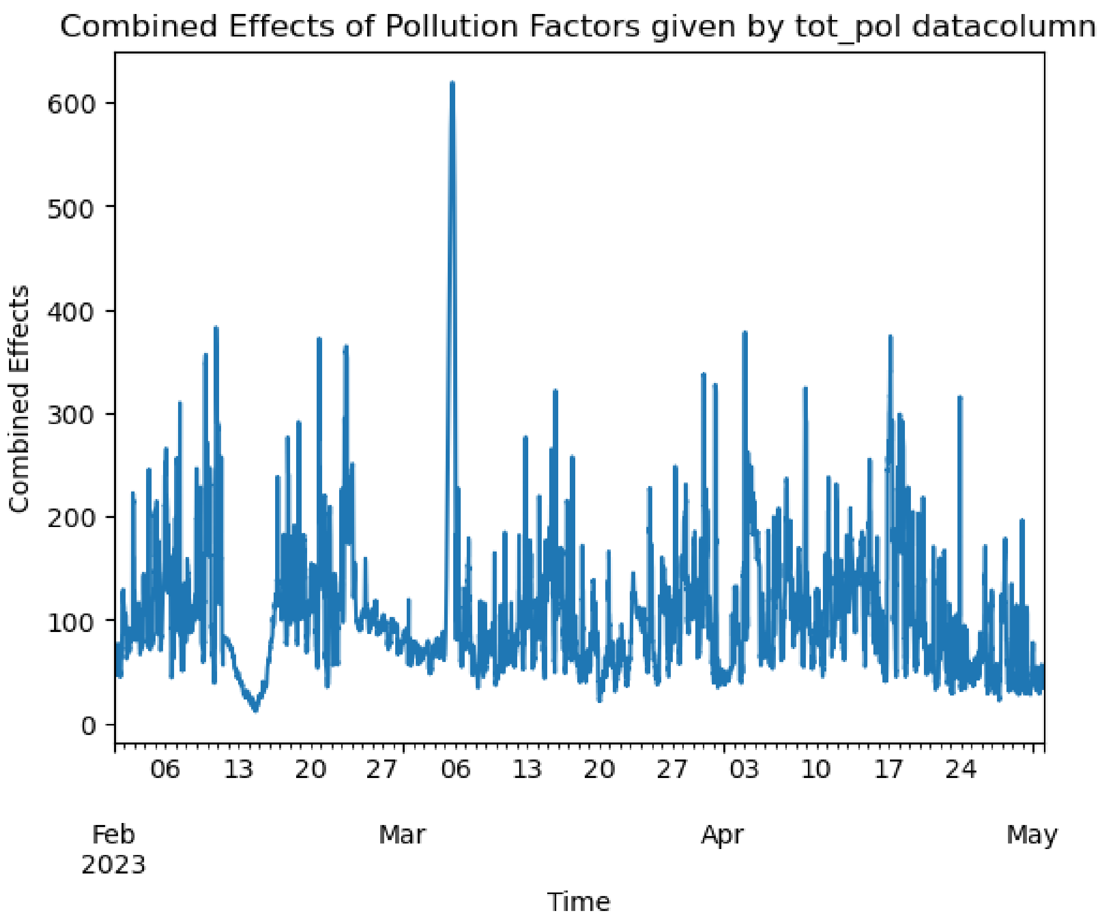
[15]:
```
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(df['tot_pol'])
print(f'p-value: {adf_test[1]}')
```

```
p-value: 5.741498169692062e-13
```

Such a low p-value implies that tot_pol(combined effective data) also follows time-series stationarity.

[16]:
```
# df = df.resample('D').mean()
```

[17]:
```
df['tot_pol'].plot()
plt.xlabel('Time')
plt.ylabel('Combined Effects')
plt.title('Combined Effects of Pollution Factors given by tot_pol datacolumn')
plt.show()
```

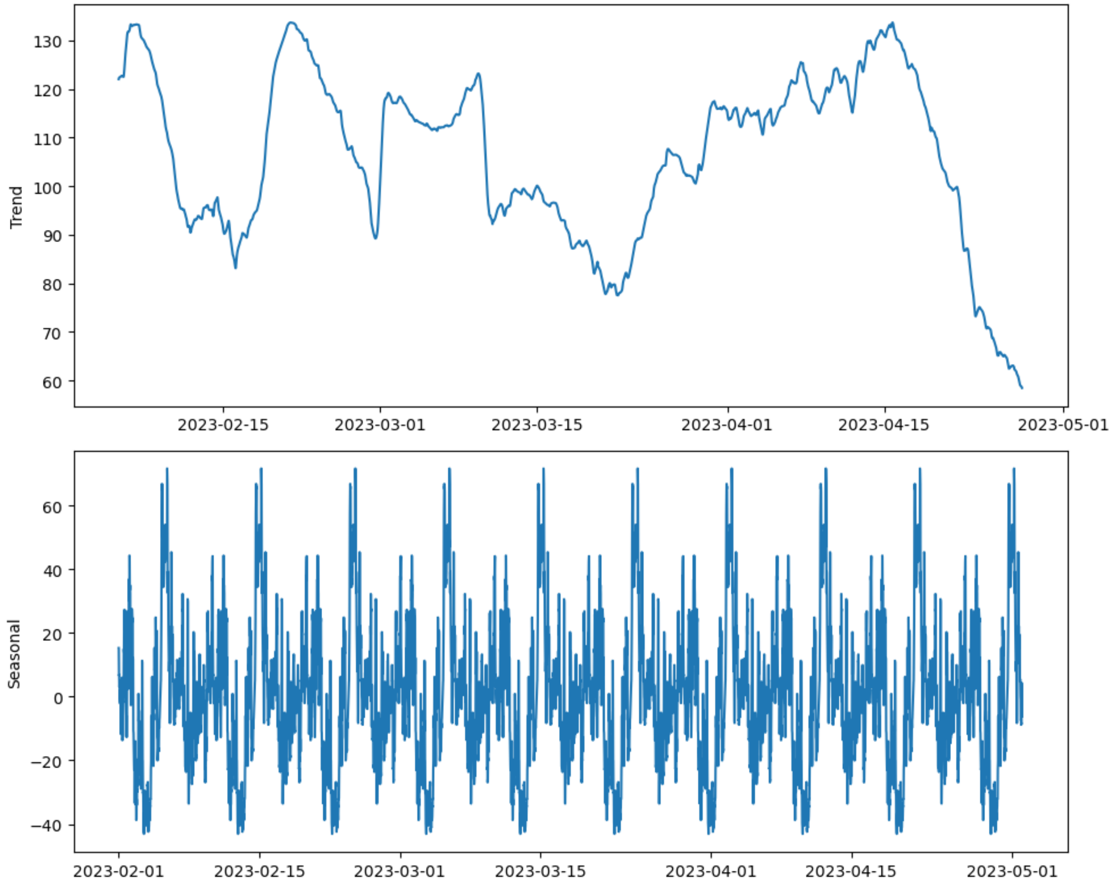


[18]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

[19]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
# Resample the data to daily frequency
# df = df.resample('D').mean()
decomposition = seasonal_decompose(df['tot_pol'], model='additive', period=900)
```

[20]:
```
%matplotlib inline
fig, ax = plt.subplots(2, 1, figsize=(10, 8))
ax[0].plot(decomposition.trend)
ax[0].set_ylabel('Trend')
ax[1].plot(decomposition.seasonal)
ax[1].set_ylabel('Seasonal')
# ax[2].plot(decomposition.resid)
# ax[2].set_ylabel('Residual')
# ax[3].plot(decomposition.observed)
# ax[3].set_ylabel('Observed')
plt.tight_layout()
```

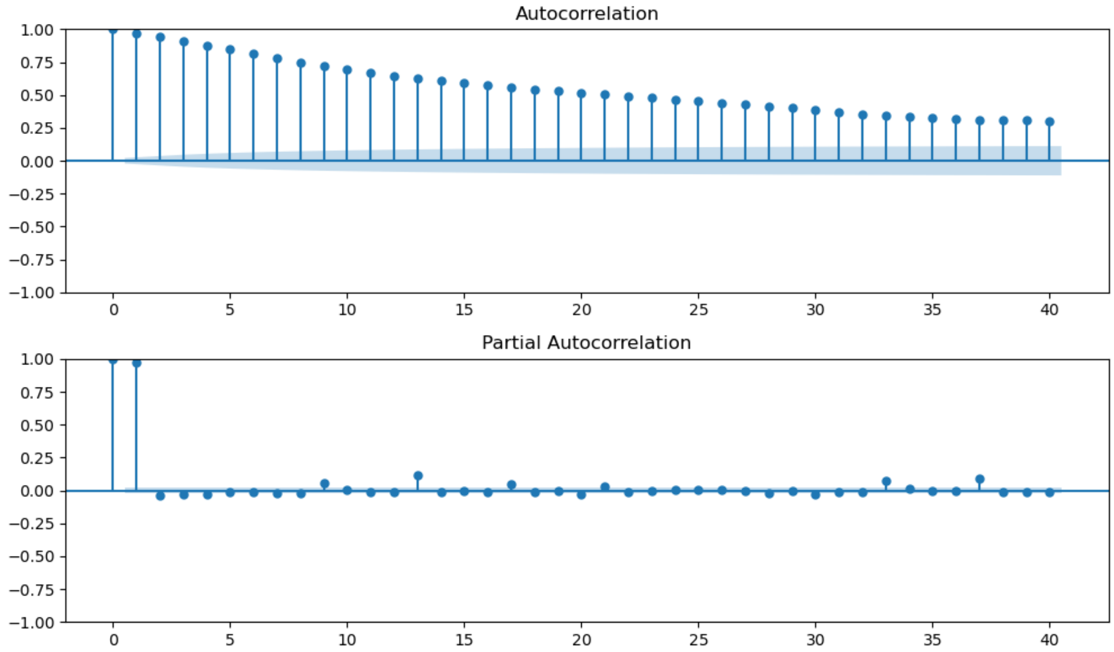
```
plt.show()
```



Above graph captures seasonality present in our data. We can see that the graph has seasonality of 9 days.(pattern repeats after 9 days accross all data).

[21]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

fig, ax = plt.subplots(2, 1, figsize=(10, 6))
plot_acf(df['tot_pol'], ax=ax[0])
ax[0].set_title('Autocorrelation')
plot_pacf(df['tot_pol'], ax=ax[1])
ax[1].set_title('Partial Autocorrelation')
plt.tight_layout()
plt.show()
```

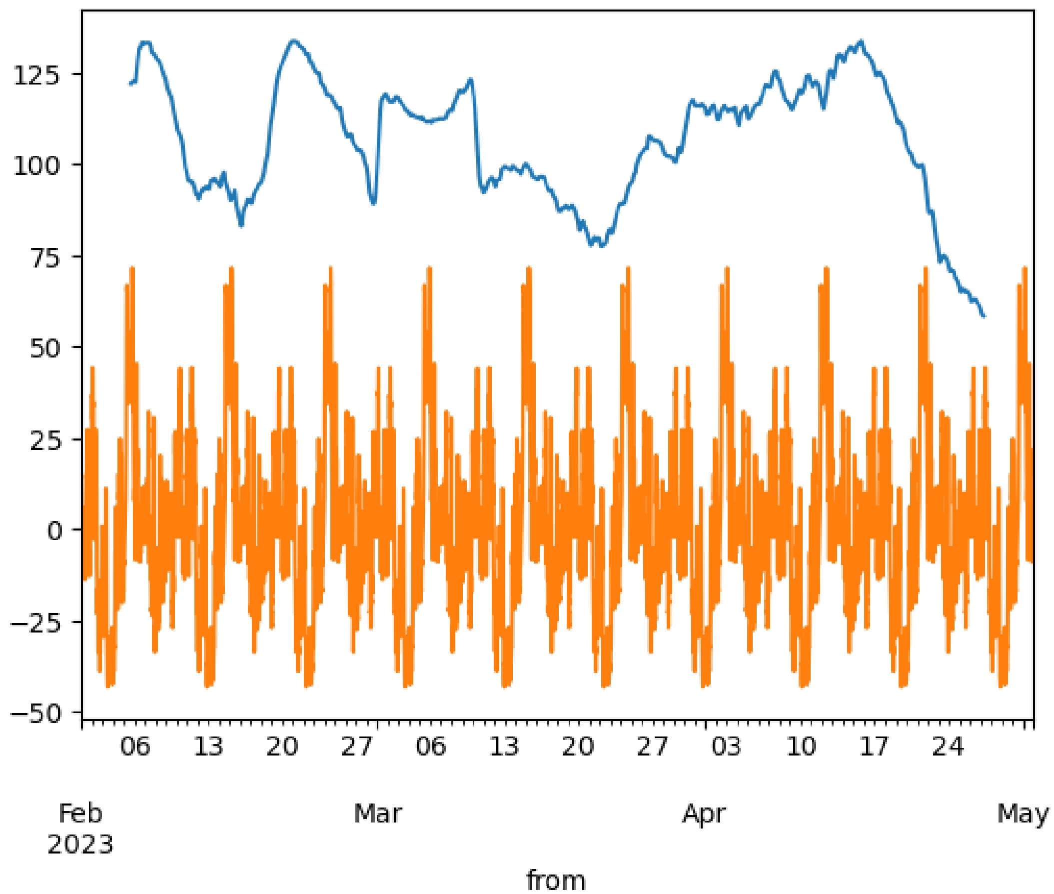
```
C:\ProgramData\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```



[22]:
```python
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

[23]:
```python
trend.plot()
seasonal.plot()
```

[23]: <Axes: xlabel='from'>


125
100
75
50
25
0
−25
−50
06
13
20
27
06
13
20
27
03
10
17
24
Feb
2023
Mar
Apr
May
from

[ ]: