Title: Analysis of Multivariate Time Series Data with Missing Values

Introduction:

The assignment aims to analyse a multivariate dataset consisting of air pollution measurements in the NCL coal fields, specifically focusing on suspended particulate matter (SPM) and reparable particulate matter (RPM). The dataset contains missing values (NA) due to various reasons such as sensor failure and communication link issues. The objective is to plot the time-series data using different techniques, handle the missing values, and explore the suitability of ARMA/ARIMA processes for the dataset.

Converted every column to same unit

Firstly, I converted every column to same unit i.e. (µg/m3).

```
# Conversion factors
ppb_to_microgram_per_m3_nox = 1.91  # Conversion factor for NOX (ppb to µg/m3)
mg_to_microgram_per_m3_co = 1000  # Conversion factor for CO (mg/m3 to µg/m3)

# Convert NOX column from ppb to µg/m3
data_orig['NOX(µg/m3)'] *= ppb_to_microgram_per_m3_nox

# Convert CO column from mg/m3 to µg/m3
data_orig['CO(µg/m3)'] *= mg_to_microgram_per_m3_co
```

1. Missing Data Analysis:

To begin the analysis, the dataset was inspected to determine the presence of missing values. By using the `isnull().sum()` function, the number of missing values in each column was calculated. The missing data entries were denoted as "NA" in the dataset.

```
data_orig.shape
```

```
(8643, 12)
```

```
data_orig.isnull().sum()
```

```
From                0
To                  3
PM10(µg/m3)      1681
PM2.5(µg/m3)      226
NO(µg/m3)        1369
NO2(µg/m3)        416
NOX(µg/m3)        415
CO(µg/m3)         496
SO2(µg/m3)       1451
NH3(µg/m3)        326
Ozone(µg/m3)      453
Benzene(µg/m3)   6195
dtype: int64
```

2. Handling Missing Values:

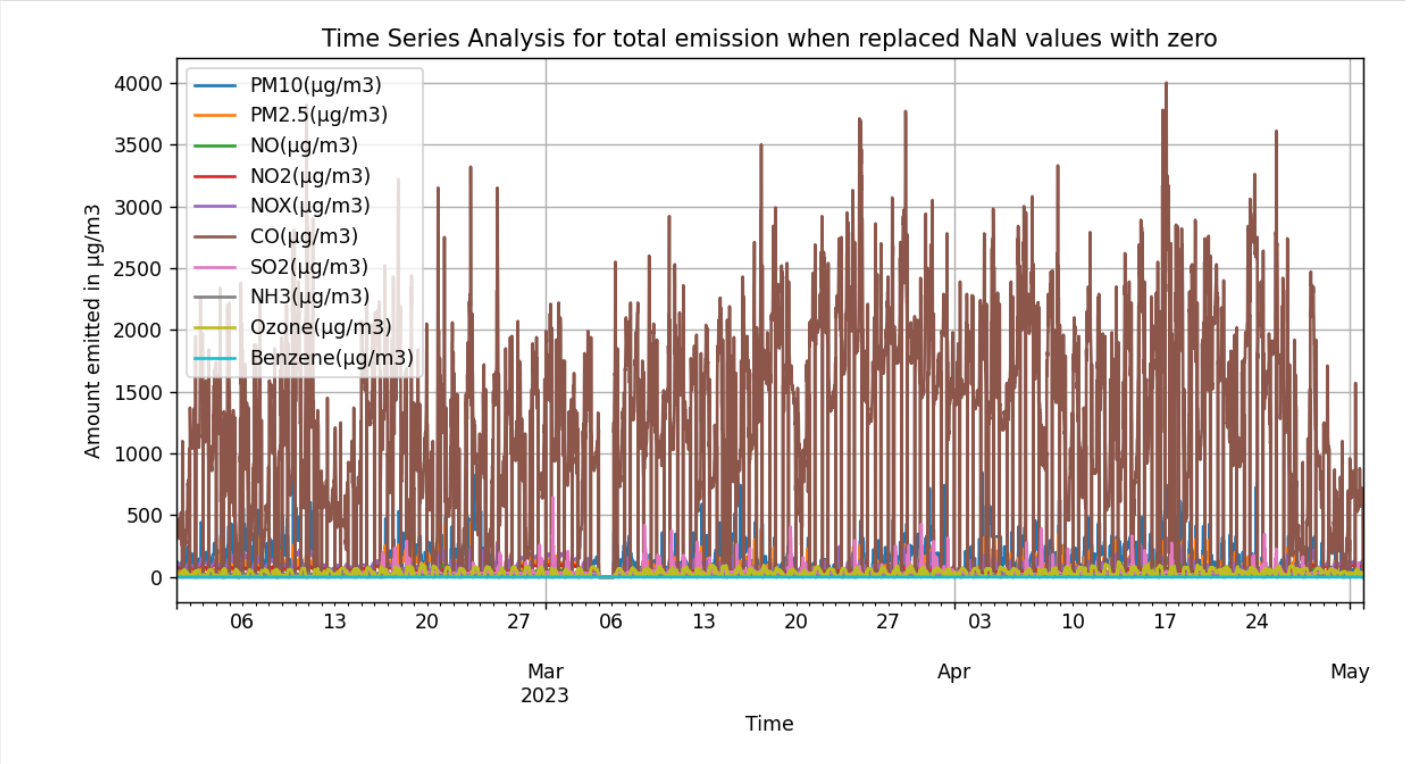
To plot the time-series data accurately, it is necessary to address the missing values. Several approaches were explored:

a. Replacing NA with Zero:

In the first approach, all the NA values were replaced with zero. However, since the dataset contained a substantial number of NA values, this approach could lead to misleading interpretations. Plotting the time series with zero values would result in most observations revolving around zero, making it difficult to draw accurate conclusions or perform meaningful analysis.

```
data = data_orig.fillna(value = 0)
data.head(1)
```

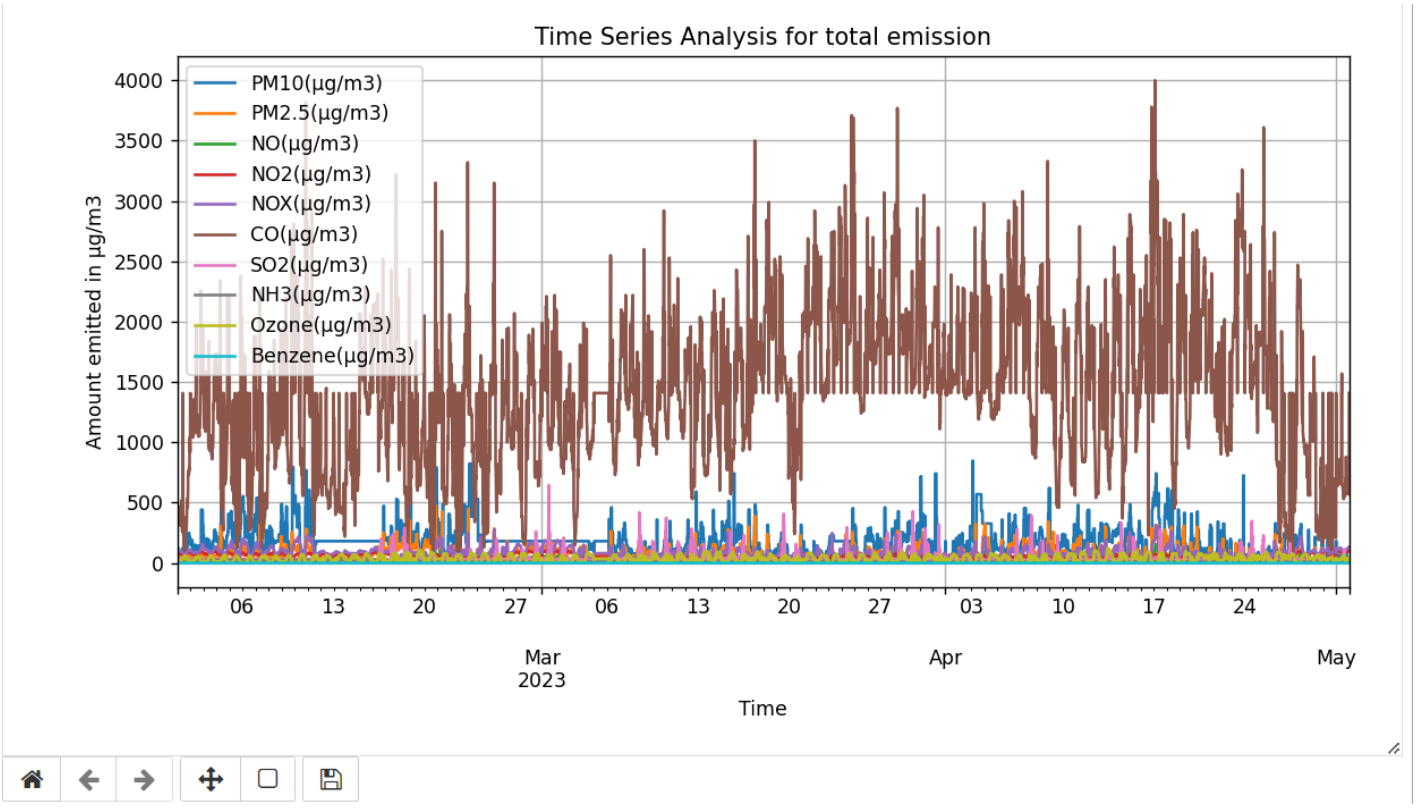
#	From	To	PM10(µg/m3)	PM2.5(µg/m3)	NO(µg/m3)	NO2(µg/m3)	NOX(µg/m3)	CO(µg/m3)	SO2(µg/m3)	NH3(µg/m3)	Ozone(µg/m3)	Benzene(µg/m3)
1	01-02-2023 0.00	01-02-2023 0.15	95.0	35.0	0.0	90.1	107.342	310.0	0.0	17.7	28.1	0.4



b. Replacing NA with Column Mean:

The second approach involved replacing the NA values of each column with the mean value of that column. This approach provides a more reasonable option as it preserves the statistical properties of the data while eliminating missing values. The resulting time series plot can be used to analyse the trends and patterns accurately.

```
column_means = data_orig.mean()
# Fill NaN values in each column with the respective column mean
data_mean = data_orig.fillna(column_means)
data_mean.head(1)
```

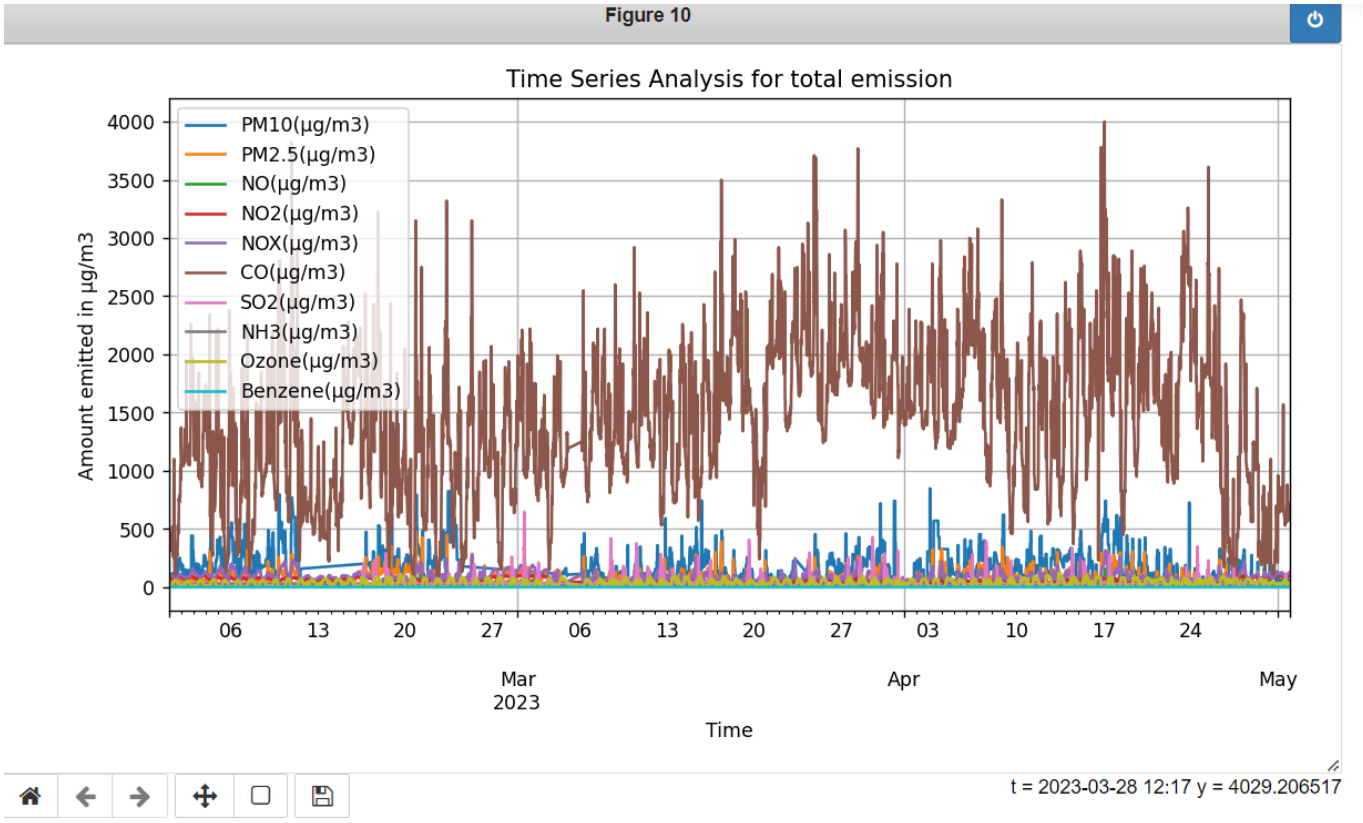


Multivariate Time Series when NaN values is replaced by mean of the column.

c. Interpolation for Missing Data:

Another approach considered was interpolation, which estimates the missing values based on the existing data points. This method provides a more data-driven approach to filling the missing values and can yield more accurate representations of the time series.

```
data_intr = data_orig.interpolate(method = 'spline', order = 1)
data_intr.head(10)
```



Multivariate Time Series when NaN values is replaced by interpolating the values.

3. ARMA/ARIMA Modelling:

To explore the applicability of ARMA/ARIMA processes for the dataset, the following steps were performed:

a. Data Preparation:

For ARIMA modelling, the dataset with NA values replaced by **mean and interpolating** was used. To train the model, the first 6000 rows were selected as the training data, and the remaining 2640 rows were set aside as the testing data. For testing ARIMA model we pick 3 columns PM10, Benzene, CO.

```
t = 6000  # Index to split the data
training_data_mean = data_mean.iloc[:t+1].copy()
testing_data_mean = data_mean.iloc[t+1:].copy()
```

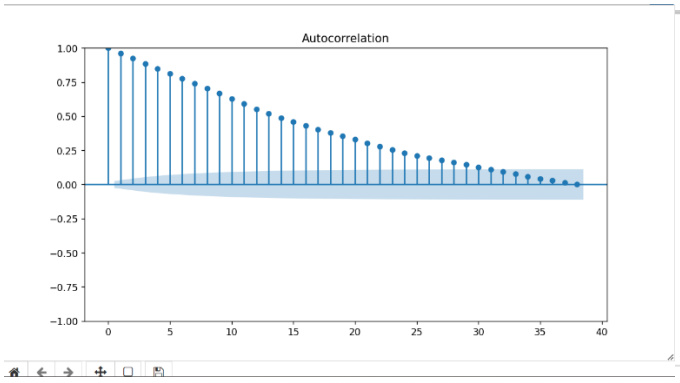
I have applied ARIMA model to dataset in which NaN values were replaced by mean and then dataset values in which NaN values were interpolated. For applying ARIMA first I analysed stationarity, then did differencing and plotted ACF and PACF plot to determine best ARIMA model. Other approach we can use pmdarima library to automatically determine best model. For one case I will determine model manually but for others I will go with pmdarima as it is less time consuming more effective.

Manually determining best model for Benzene (NaN Values replaced my mean)

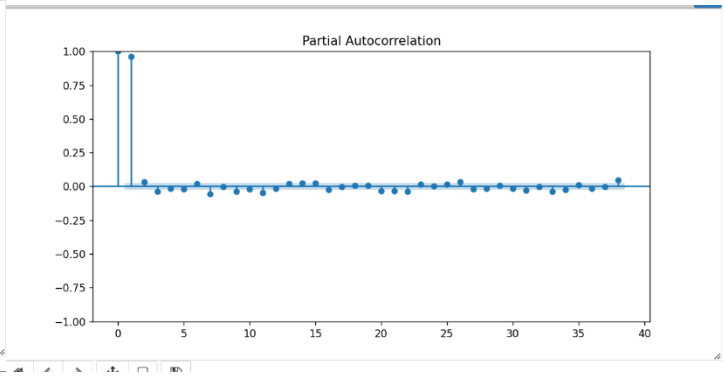
- **Stationarity Analysis:**

The stationarity of the time series was examined using the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots. Additionally, the Augmented Dickey-Fuller (ADF) test was conducted to test the stationarity hypothesis.

```
acf_plot_mean = plot_acf(training_data_mean['Benzene(µg/m3)'])
pacf_plot_mean = plot_pacf(training_data_mean['Benzene(µg/m3)'])
```



Auto- Correlation plot for Benzene



Partial Autocorrelation of Benzene

- **Augmented Dickey-Fuller (ADF)**

```
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(training_data['Benzene(µg/m3)'])
print(f'p-value : {adf_test[1]}')

# Test for stationarity
```

```
p-value : 1.434006505648762e-06
```

- **Differencing:**

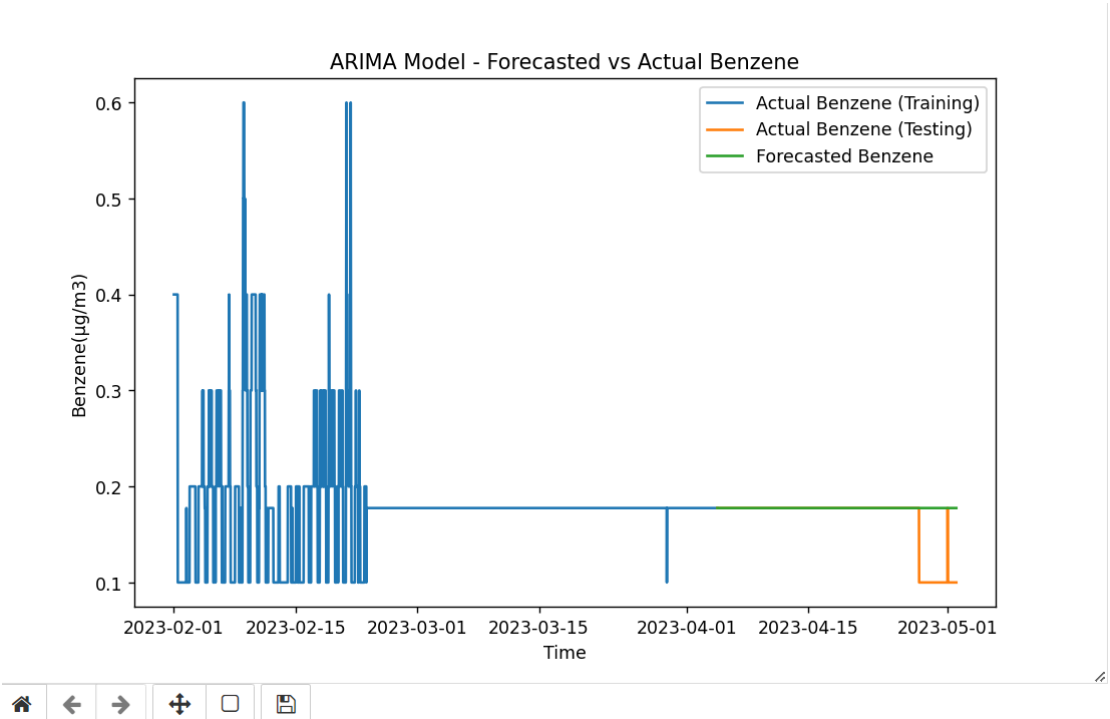
If the time series was found to be non-stationary, differencing was applied to make it stationary. Differencing involves subtracting the previous observation from the current observation.

```
training_data_diff = training_data['Benzene(µg/m3)'].diff().dropna()
training_data_diff.plot()
# Differencing to make stationary
```

Based on the ACF, PACF, and stationarity tests, the appropriate order for the ARIMA model was selected.

Model selected (1,1,4)

```
from statsmodels.tsa.arima.model import ARIMA

order = (1,1,4)  # Replace with the appropriate order values from your ARIMA model
arima_model = ARIMA(training_data_mean['Benzene(µg/m3)'], order=order)
arima_model_fit = arima_model.fit()

# Forecast PM10 values
forecasted_values = arima_model_fit.predict(start=len(training_data), end=len(training_data)+len(testing_data)-1)

# Plot forecasted and actual PM10 values
plt.figure(figsize=(12, 6))
plt.plot(training_data.index, training_data_mean['Benzene(µg/m3)'], label='Actual Benzene (Training)')
plt.plot(testing_data.index, testing_data_mean['Benzene(µg/m3)'], label='Actual Benzene (Testing)')
plt.plot(testing_data.index, forecasted_values, label='Forecasted Benzene')
plt.title('ARIMA Model - Forecasted vs Actual Benzene')
plt.xlabel('Time')
plt.ylabel('Benzene(µg/m3)')
plt.legend()
plt.show()
```

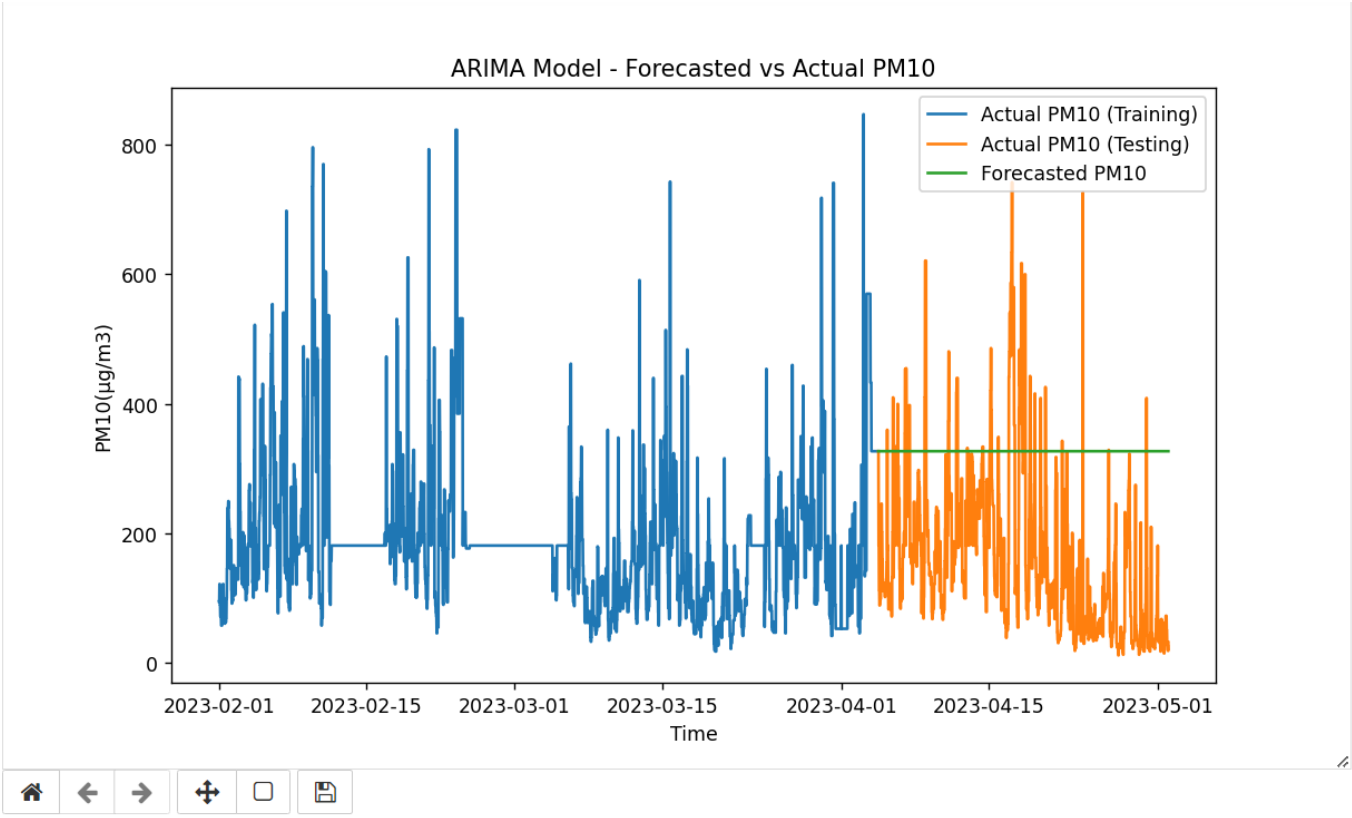


PM10(NaN Values replaced my mean)

```
import pmdarima as pm
auto_arima_mean = pm.auto_arima(training_data_mean['PM10(µg/m3)'], stepwise = False, seasonal = False)
auto_arima_mean
# Automatic method to check best ARIMA model
```

ARIMA
ARIMA(1,1,4)(0,0,0)[0] intercept

```
# AUTOMATIC
# Fit ARIMA model
from statsmodels.tsa.arima.model import ARIMA

order = (1,1,4)  # Replace with the appropriate order values from your ARIMA model
arima_model = ARIMA(training_data_mean['PM10(µg/m3)'], order=order)
arima_model_fit = arima_model.fit()

# Forecast PM10 values
forecasted_values = arima_model_fit.predict(start=len(training_data), end=len(training_data)+len(testing_data)-1)

# Plot forecasted and actual PM10 values
plt.figure(figsize=(12, 6))
plt.plot(training_data.index, training_data_mean['PM10(µg/m3)'], label='Actual PM10 (Training)')
plt.plot(testing_data.index, testing_data_mean['PM10(µg/m3)'], label='Actual PM10 (Testing)')
plt.plot(testing_data.index, forecasted_values, label='Forecasted PM10')
plt.title('ARIMA Model - Forecasted vs Actual PM10')
plt.xlabel('Time')
plt.ylabel('PM10(µg/m3)')
plt.legend()
plt.show()
```

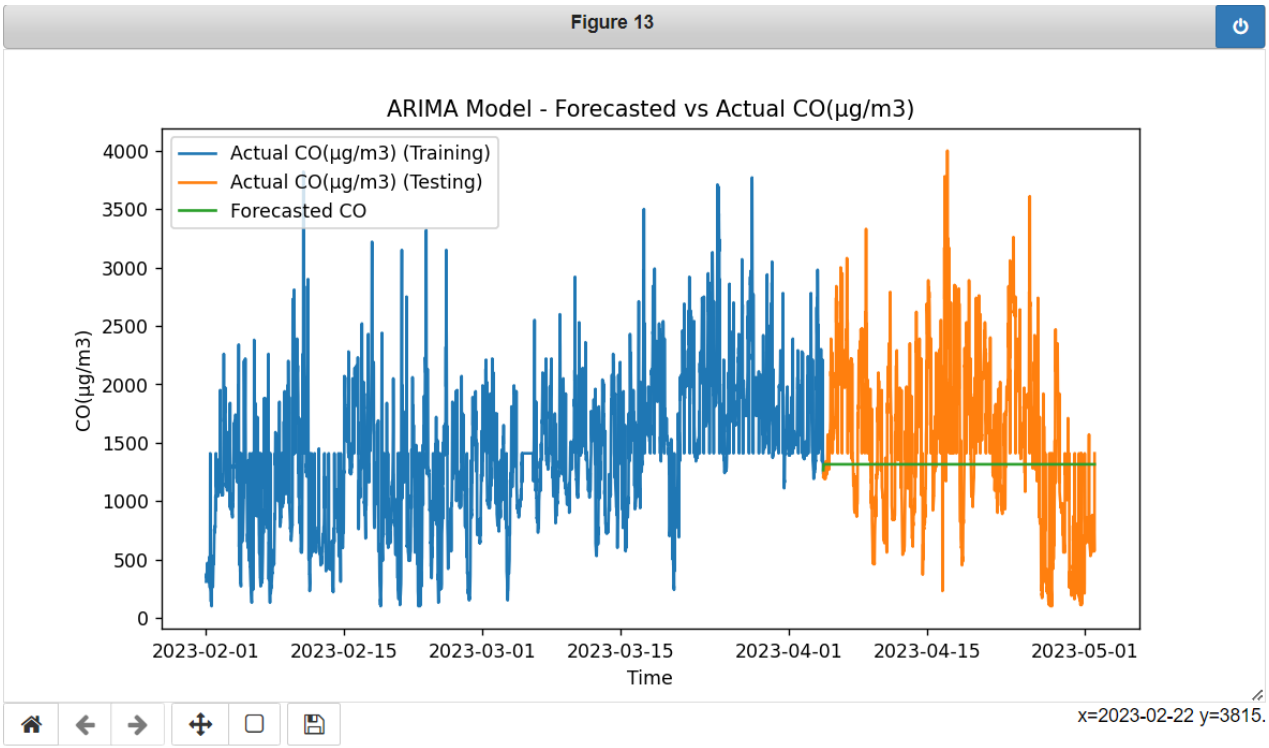


ARIMA Model - Forecasted vs Actual PM10 using p,d,q parameters as 1,1,4

CO(NaN Values replaced my mean)

```
import pmdarima as pm
auto_arima_mean = pm.auto_arima(training_data_mean['CO(µg/m3)'], stepwise = False, seasonal = False)
auto_arima_mean
# Automatic method to check best ARIMA model
```

```
ARIMA
ARIMA(0,1,5)(0,0,0)[0] intercept
```

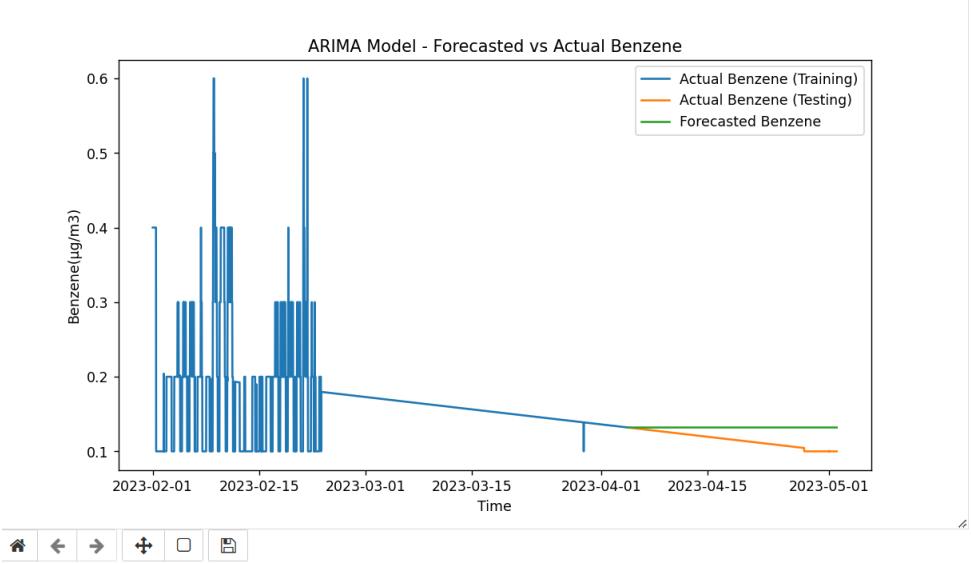


Since pmdarima library is less time consuming and effective we will be using the pmdarima library to automatically select the best model for ARIMA. Now the same above three graph are plotted for the dataset of interpolation.

BENZENE (NaN values by interpolating)

```
import pmdarima as pm
auto_arima = pm.auto_arima(training_data_intr['Benzene(µg/m3)'], stepwise = False, seasonal = False)
auto_arima
```

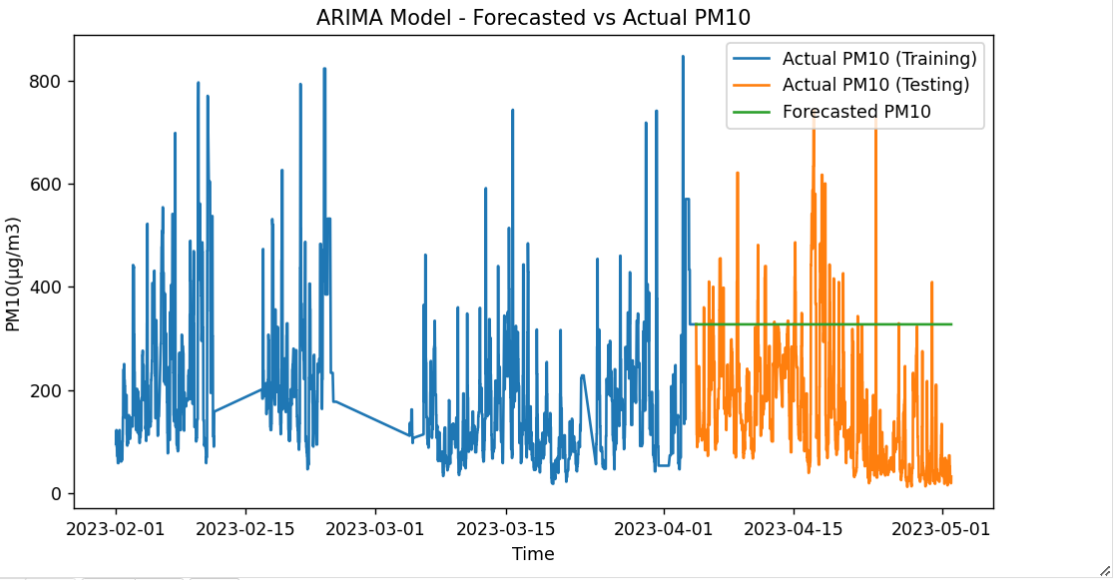
```
ARIMA
ARIMA(5,1,0)(0,0,0)[0] intercept
```



PM10 (NaN values by interpolating)

```
import pmdarima as pm
auto_arima = pm.auto_arima(training_data_intr['PM10(µg/m3)'], stepwise = False, seasonal = False)
auto_arima
```

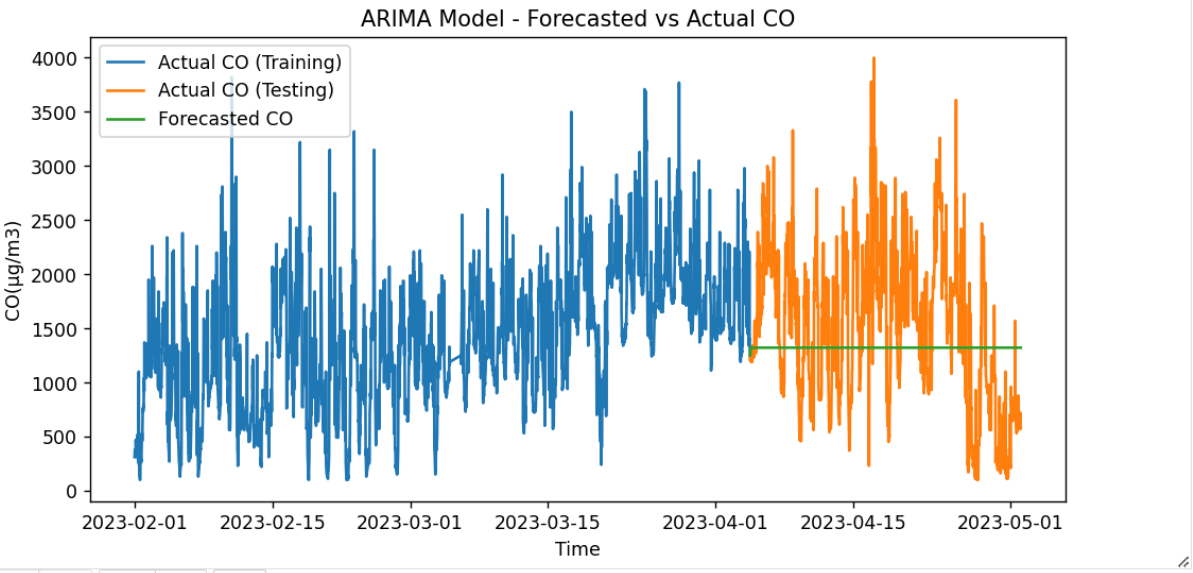
```
ARIMA
ARIMA(0,1,5)(0,0,0)[0] intercept
```



CO (NaN values by interpolating)

```
import pmdarima as pm
auto_arima = pm.auto_arima(training_data_intr['CO(µg/m3)'], stepwise = False, seasonal = False)
auto_arima
```

```
ARIMA
ARIMA(2,1,3)(0,0,0)[0] intercept
```



Conclusion

In conclusion, analysing a multivariate time series dataset with missing values requires careful consideration of the missing data handling techniques. Simply replacing NA values with zero or dropping rows can lead to biased results. Replacing NA values with the mean of the respective column provides a better approach as it maintains the statistical properties of the data. Another method, interpolation, can be used to estimate missing values based on existing data points, resulting in a more accurate representation of the time series.

Q) Why interpolating is better than replacing with mean

- Interpolating missing values is preferable to replacing them with the column mean in datasets with many missing values. It preserves data patterns and captures temporal/spatial dynamics.
- Interpolation minimizes bias by considering existing relationships in the data, while mean replacement can introduce bias and overlook data variability.
- The choice between interpolation and mean replacement depends on the dataset's characteristics and missing data mechanism. Evaluate the dataset and select the appropriate imputation method based on specific requirements.

By following these steps, we can effectively analyse a multivariate time series dataset with missing values, employ different plotting techniques, and explore the suitability of ARMA/ARIMA processes for the dataset.

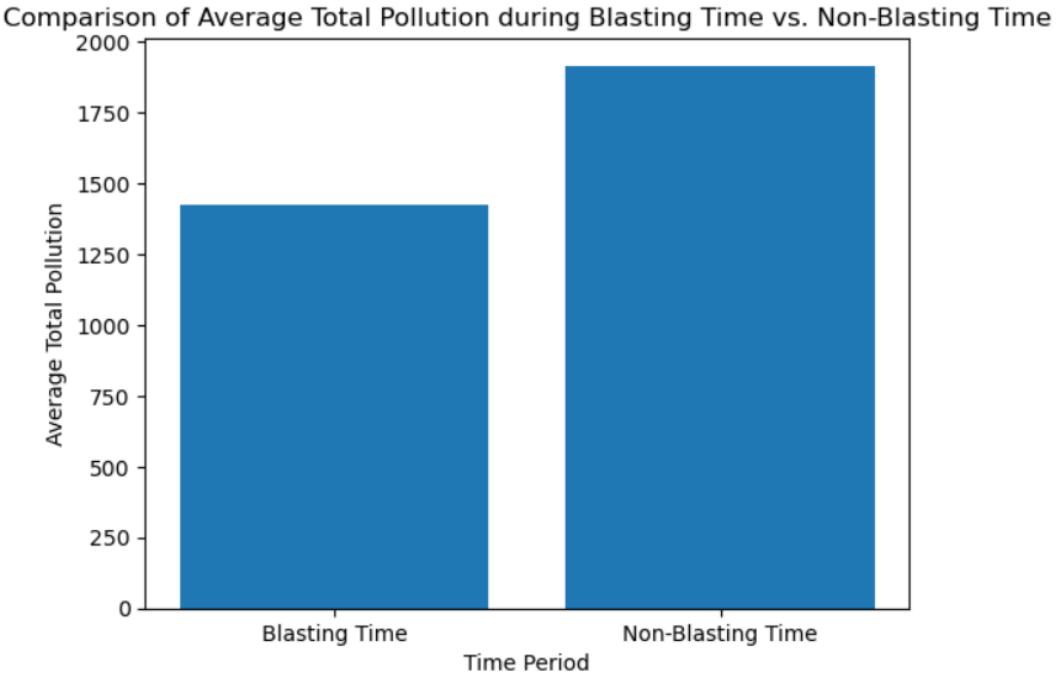
Blasting time analysis

1. Validating the Hypothesis:

To validate the hypothesis that blasting time (13:45 pm to 14:45 pm) has a major effect on air pollution, we compared the average pollution levels during blasting time versus non-blasting time.

We plotted a bar graph comparing the average total pollution during blasting time and non-blasting time. The graph showed that non-blasting time had higher pollution levels than blasting time, **contradicting the initial assumption.**

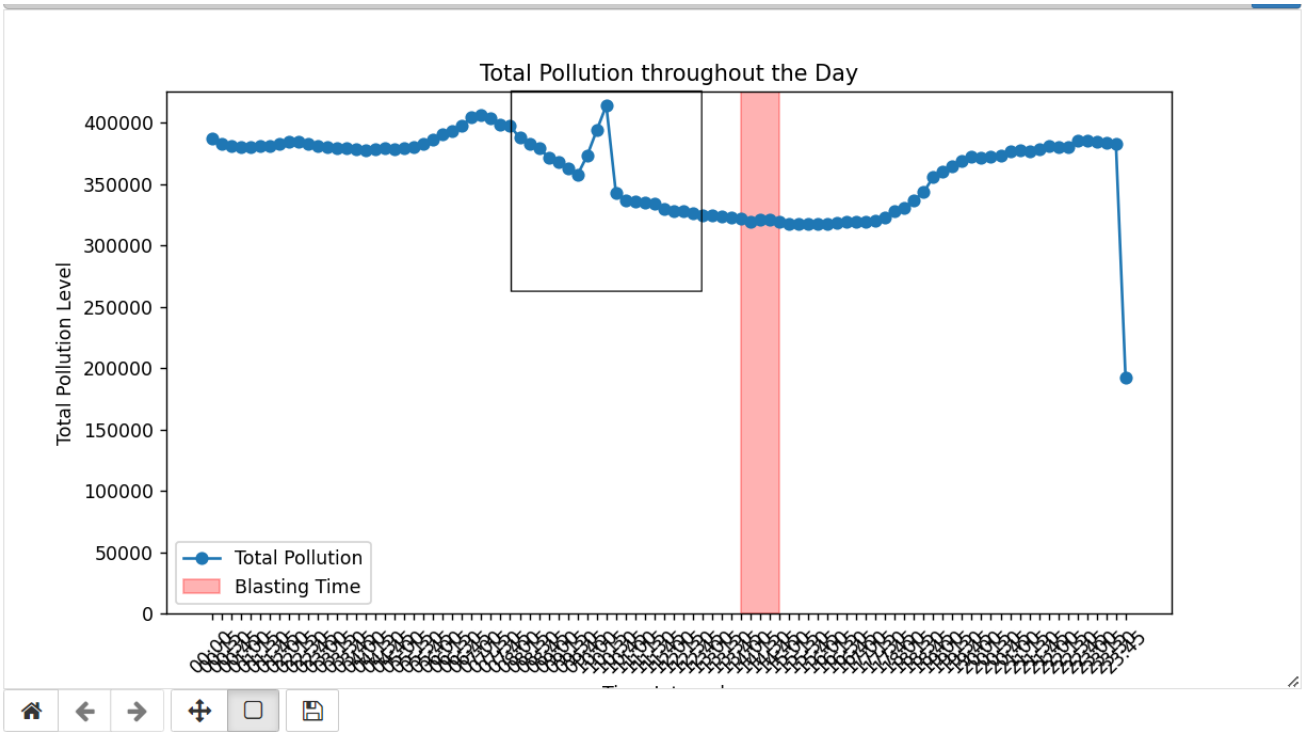
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# Extract the relevant columns for analysis
blasting_time = data_intr['From']
pollution_columns = ['PM10(µg/m3)', 'PM2.5(µg/m3)', 'NO(µg/m3)', 'NO2(µg/m3)', 'NOX(µg/m3)', 'CO(µg/m3)', 'SO2(µg/m3)', 'NH3(µg/m
pollution_data = data_intr[pollution_columns]

# Convert blasting time to datetime format
blasting_time = pd.to_datetime(blasting_time, format='%d-%m-%Y %H.%M')

# Set the blasting time range
blasting_start = pd.to_datetime('13:45', format='%H:%M').time()
blasting_end = pd.to_datetime('14:45', format='%H:%M').time()

# Filter the data for the blasting time range
blasting_data = pollution_data[(blasting_time.dt.time >= blasting_start) & (blasting_time.dt.time <= blasting_end)]

# Calculate the average total pollution for the blasting time range
average_blasting_pollution = blasting_data.sum(axis=1).mean()

# Filter the data for the non-blasting time periods
non_blasting_data = pollution_data[(blasting_time.dt.time < blasting_start) | (blasting_time.dt.time > blasting_end)]

# Calculate the average total pollution for the non-blasting time periods
average_non_blasting_pollution = non_blasting_data.sum(axis=1).mean()

# Create a bar plot to compare the average total pollution for blasting time vs. non-blasting time
time_periods = ['Blasting Time', 'Non-Blasting Time']
average_pollution = [average_blasting_pollution, average_non_blasting_pollution]

plt.bar(time_periods, average_pollution)
plt.xlabel('Time Period')
plt.ylabel('Average Total Pollution')
plt.title('Comparison of Average Total Pollution during Blasting Time vs. Non-Blasting Time')
plt.show()
```



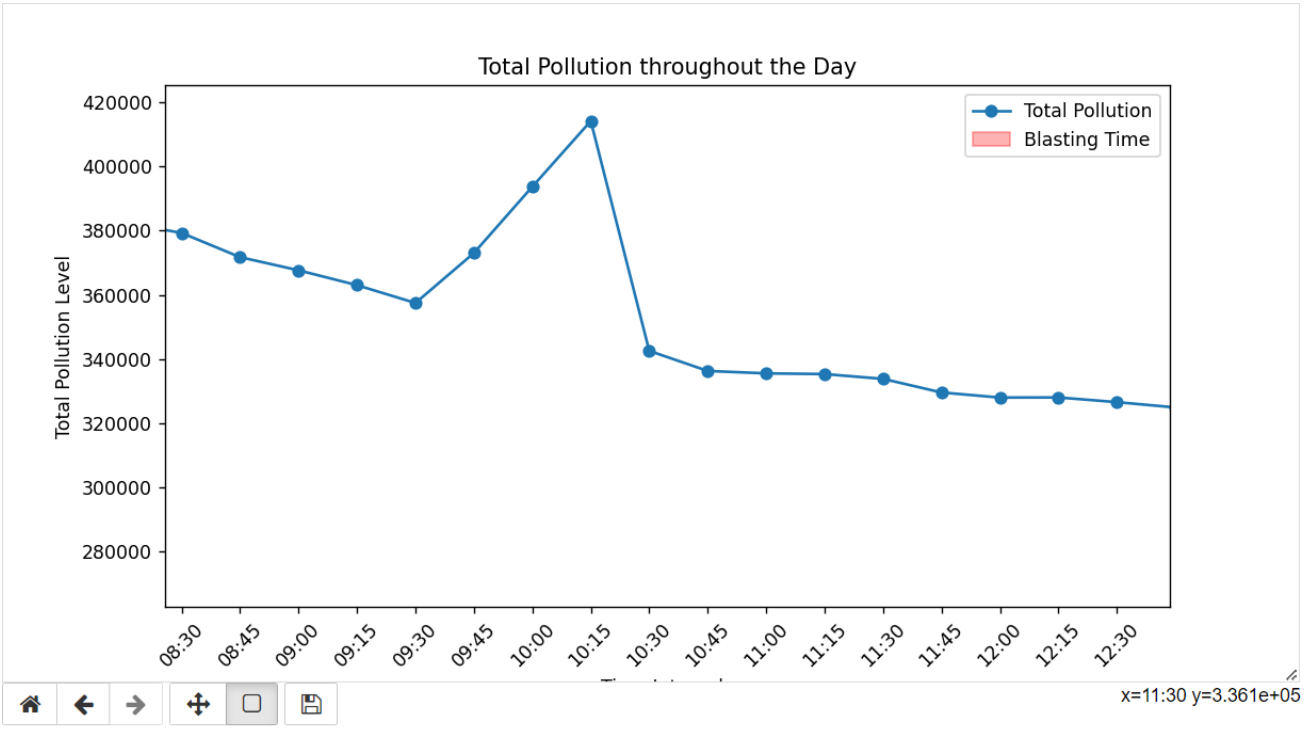
2. Identifying the Blasting Time:

To detect the blasting time from the time-series data, we filtered the dataset for the specified time range (13:45 to 14:45). Plotted graph showing average pollution throughout the day. Graph shows clear spike at some time stamps, after zooming we can see spike is at 9:30 to 10:15 am.

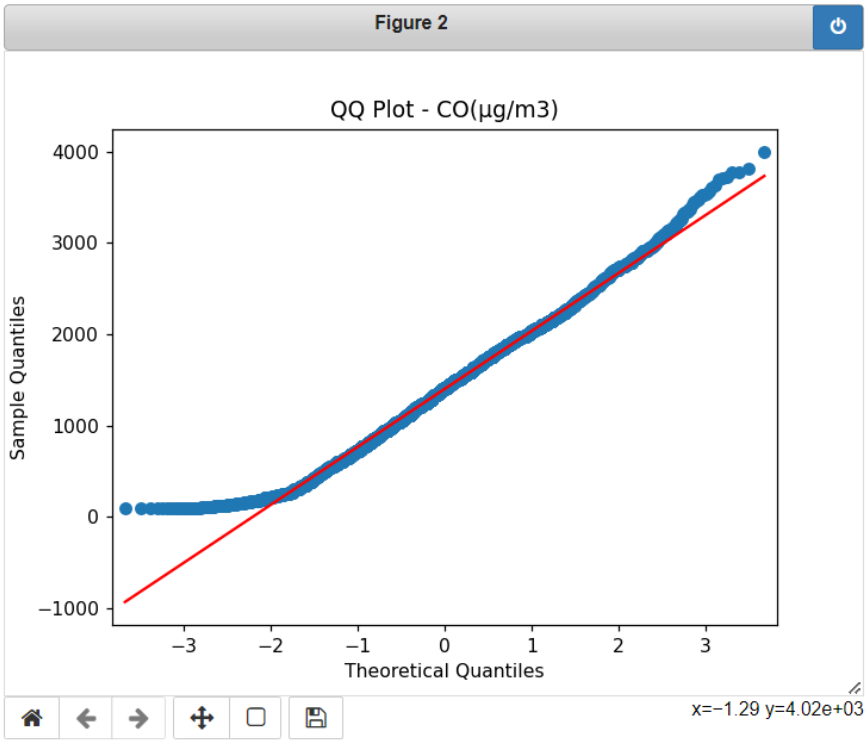
```
import matplotlib.pyplot as plt
from datetime import datetime
%matplotlib notebook

def plot_daily_pollution(dataframe, blasting_start_time, blasting_end_time):
    # Convert the date and time columns to datetime format
    dataframe['From'] = pd.to_datetime(dataframe['From'], format='%d-%m-%Y %H.%M')
    dataframe['To'] = pd.to_datetime(dataframe['To'], format='%d-%m-%Y %H.%M')

    # Generate the time intervals for x-axis labels
    time_intervals = pd.date_range(start='00:00', end='23:59', freq='15min').strftime('%H:%M').tolist()

    # Calculate the total pollution levels for each time interval
    total_pollution = []
    for interval in time_intervals:
        interval_start = datetime.strptime(interval, '%H:%M').time()
        interval_end = (datetime.strptime(interval, '%H:%M') + pd.Timedelta(minutes=15)).time()

        interval_data = dataframe[
            (dataframe['From'].dt.time >= interval_start) &
            (dataframe['To'].dt.time <= interval_end)
        ]
        total_pollution.append(interval_data.loc[:, 'PM10(µg/m3)':'Benzene(µg/m3)'].sum().sum())

    # Create a figure and axis
    fig, ax = plt.subplots()

    # Plot the total pollution levels
    ax.plot(time_intervals, total_pollution, marker='o', linestyle='-', label='Total Pollution')

    # Highlight the blasting time interval
    ax.axvspan(blasting_start_time, blasting_end_time, alpha=0.3, color='red', label='Blasting Time')

    # Set the labels and title
    ax.set_xlabel("Time Intervals")
    ax.set_ylabel("Total Pollution Level")
    ax.set_title("Total Pollution throughout the Day")

    # Add a legend
    ax.legend()

    # Rotate the x-axis labels for better visibility
    plt.xticks(rotation=45)

    # Set the y-axis limit for better visualization
    ax.set_ylim(bottom=0)

    # Display the plot
    plt.show()

# Specify the blasting time range
blasting_start_time = '13:45'
blasting_end_time = '14:45'

# Call the function to plot the daily total pollution levels and highlight the blasting time
plot_daily_pollution(data_intr, blasting_start_time, blasting_end_time)
```

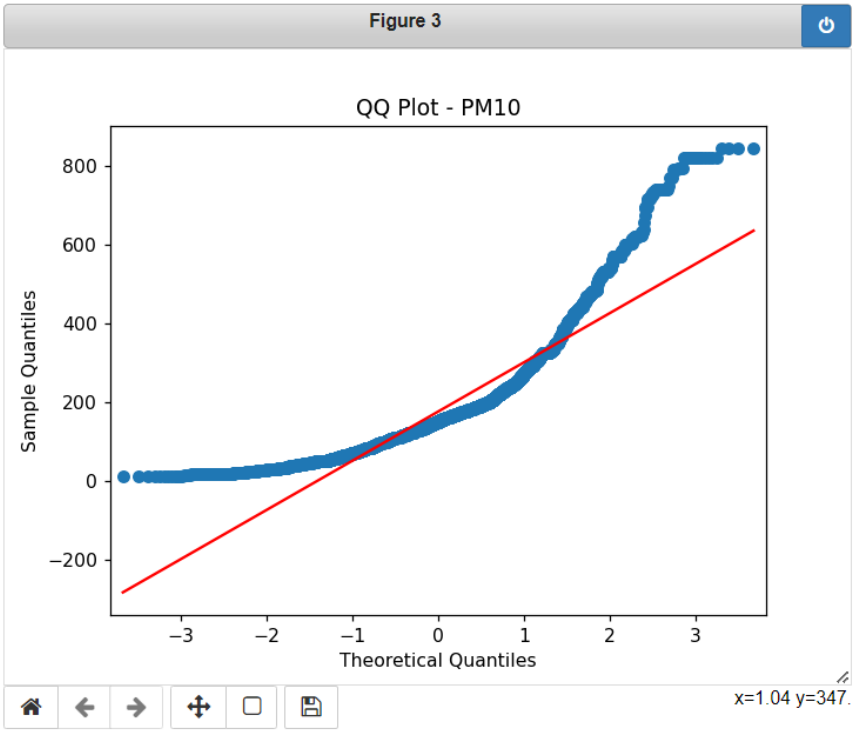


Zooming the marked area



QQ Plots of Pollutants to check their distribution





Question setting

#Forecasting:

1-> Can we use time series methods like AR, MA, ARMA, or ARIMA to forecast future air pollution levels during blasting time?

Ans- Yes, we can use time series methods such as ARIMA to forecast future air pollution levels during blasting time. My analysis, which includes several plots of pollutants and fitting them into an ARIMA model, suggests that these models are capable of predicting air pollution levels with close accuracy.

2-> How accurate are these forecasting models in predicting the air pollution levels?

Ans- The accuracy of forecasting models for predicting air pollution levels using time series methods like AR, MA, ARMA, or ARIMA can vary depending on several factors. These factors include the quality and representativeness of the data, the appropriateness of the chosen model, the characteristics of the specific air pollution data being analysed, and the underlying factors that influence air pollution levels.

In my analysis of several ARIMA models, prediction was not much accurate but it can still be use to predicts as the prediction curve is quite close

#Curve fitting:

1 ->Can we fit a curve to the air pollution data to identify relationships between variables?

2 ->Are there any non-linear patterns or trends in the air pollution data?

3-> What type of curve provides the best fit to the air pollution data?

Tejaswa Mathur

Roll no. 211110