Phase 5: **Project Documentation & Submission**

The "Air Quality Analysis" project comprises four distinct phases, each with specific objectives aimed at enhancing air quality assessment and understanding in Tennessee and Tamil Nadu.

In Phase 1, the project primarily focuses on Tennessee, aiming to analyze historical air quality data and identify geographic areas with elevated pollutant levels. This involves the examination of pollutants like PM2.5, PM10, NO2, SO2, O3, CO, and VOCs over time. Additionally, the project aims to develop a predictive model for Respirable Suspended Particulate Matter (RSPM) and PM10 levels, incorporating both air quality data and meteorological variables. These efforts collectively aim to provide valuable insights for sustainable environmental management and targeted pollution control strategies.

Phase 2 extends the project's scope to Tamil Nadu, with a focus on enhancing air quality assessment capabilities. This phase involves researching on advanced data acquisition techniques from diverse monitoring stations, ensuring data integrity, and utilizing sophisticated algorithms for accurate air quality predictions. The project also conducts an Environmental Impact Assessment to evaluate the impact of air pollution on ecosystems and public health. Real-time data integration, advanced spatial analysis techniques, and predictive analytics contribute to a comprehensive approach. Public engagement initiatives are employed to raise awareness and ensure a holistic approach to air quality assessment.

In Phase 3, the project shifts its focus to comprehensive data analysis and loading the data. This includes Exploratory Data Analysis (EDA) to understand the distribution and relationships between variables, data cleaning and preprocessing to handle missing data and outliers, and consistent data formatting for time series analysis. Additional preprocessing tasks involve aggregating data for trend analysis, ensuring data validation through crosschecking against standards, and utilizing visualization techniques to effectively communicate trends and spatial variations in air quality. The objective is to provide a reliable analysis aiding informed decision-making and environmental management.

Phase 4, In Tamil Nadu, concentrates on detailed analysis of specific pollutants: Sulphur Dioxide (SO2), Nitrogen Dioxide (NO2), and Respirable Suspended Particulate Matter (RSPM/PM10). This phase involves calculating average pollutant levels across different monitoring stations and cities. By identifying pollution trends and areas with high pollution levels, the project aims to provide valuable insights into the state of air quality. Visualizations are created to enhance public awareness and serve as a basis for well-informed policy decisions. The ultimate goal is to present the analysis findings in an accessible manner, fostering a deeper understanding of air quality dynamics and contributing to the protection of both the environment and public health in the state.

Overall, the project's overarching objective is to employ advanced data analysis, modeling, and visualization techniques to enhance air quality assessment, support pollution control efforts, and provide valuable insights for sustainable environmental management in Tennessee and Tamil Nadu.

Analysis approach:

The analysis approach encompasses a systematic process that begins with data acquisition and loading. The historical air quality data from diverse monitoring stations in Tamil Nadu is obtained and rigorously validated for integrity and quality. Following this, exploratory data analysis (EDA) is performed, involving the computation of summary statistics and the creation of visualizations to understand the distribution and relationships between variables. Missing data is handled appropriately, and temporal patterns in air quality parameters are analyzed.

Subsequently, data cleaning and preprocessing steps are taken to ensure data reliability. This includes handling missing values, addressing outliers if necessary, and formatting dates for time series analysis. Feature selection techniques are implemented to identify influential variables for modeling. The next step involves selecting appropriate models based on project objectives, dividing the data for robust validation, and evaluating model performance using relevant metrics. A specific predictive model is developed for Respirable Suspended Particulate Matter (RSPM) and PM10 levels, incorporating meteorological data and pollution sources.

The chosen visualizations, such as line charts, bar charts, and spatial maps, are employed to effectively represent air quality trends and pollution levels. These visual representations aid in comprehending temporal and spatial patterns in air quality, as well as correlations between variables. The approach emphasizes transparency and reproducibility, with documentation of all steps, methodologies, and assumptions. The code and data are stored in a version-controlled repository to ensure accessibility and collaboration.

Overall, this comprehensive analysis approach combines rigorous data processing, advanced modeling techniques, and insightful visualizations to provide valuable insights into air quality dynamics in Tamil Nadu, facilitating informed decision-making and environmental management.

The visualization techniques used in this air quality analysis project include:

1. Line Charts: Utilized for displaying temporal trends in air quality parameters over a specific time period. This visualization technique is effective for identifying seasonal variations, long-term trends, and potential anomalies in the data.

2. Bar Charts: Employed for comparing pollutant levels across different locations or time intervals. Bar charts are particularly useful for highlighting variations in air quality between regions or monitoring stations.

3. Heatmaps: Created to visualize the spatial distribution of pollutants across Tamil Nadu. This allows for the identification of pollution hotspots and patterns of pollution concentration. Heatmaps provide a clear and intuitive representation of geographic variations in air quality.

4. Scatter Plots: Used for exploring relationships between air quality parameters and potential influencing factors like temperature and humidity. Scatter plots help in understanding the correlations between variables and identifying potential trends or outliers.

5. Box Plots: Applied for providing a summary of the distribution, central tendency, and variability of air quality data. This visualization technique is valuable for identifying outliers and understanding the spread of the data.

6. Radar Charts: Considered for comparing multiple pollutants simultaneously, showcasing their relative levels and trends. Radar charts provide a comprehensive view of pollutant levels and facilitate easy comparisons.

7. Pie Charts: Utilized for representing the composition of different pollutants in the overall air quality. This helps in understanding the relative contributions of various pollutants to the overall air quality.

8. Time Series Decomposition Plots: Implemented for breaking down time series data into its constituent components (trend, seasonality, residual). This technique provides a clearer understanding of temporal patterns in air quality parameters.

9. Dashboard: Integrated multiple visualizations into a dashboard for a holistic view of air quality trends, spatial distribution, correlations, and pollutant composition. Dashboards provide a comprehensive and interactive way to explore and analyze air quality data.Overall dashboard will be provided at the end.

These visualization techniques collectively provide a rich and informative representation of the air quality analysis, enabling stakeholders to gain valuable insights and make informed decisions regarding pollution control and environmental management in Tamil Nadu.

Code implementation and sample output of data analysis with visualization:

Importing the required libraries from python for the operations.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

loading the data set from the given source by IBM cognos

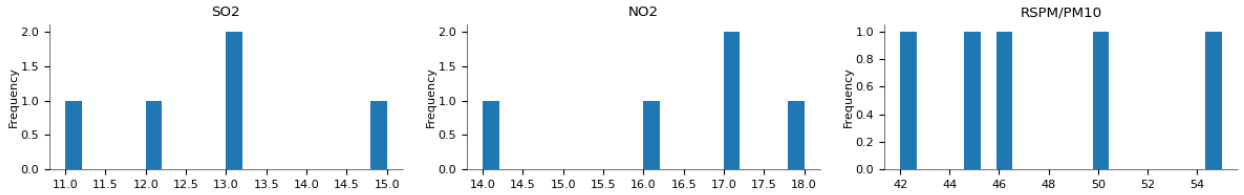
```
from google.colab import files

uploaded = files.upload(
```

BAsic analysis of the guven Dataset

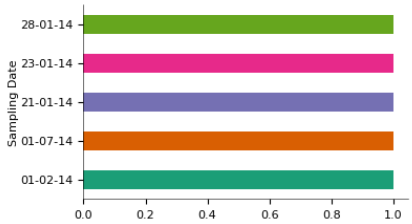
```
data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")
data.head()
```

	Stn Code	Sampling Date	State	City/Town/Village/Area	Location of Monitoring Station	Agency	Type of Location	SO2	NO2	RSPM/PM10	PM 2.5
0	38	01-02-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	11.0	17.0	55.0	NaN
1	38	01-07-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	13.0	17.0	45.0	NaN
2	38	21-01-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	12.0	18.0	50.0	NaN
3	38	23-01-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	15.0	16.0	46.0	NaN
4	38	28-01-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	13.0	14.0	42.0	NaN

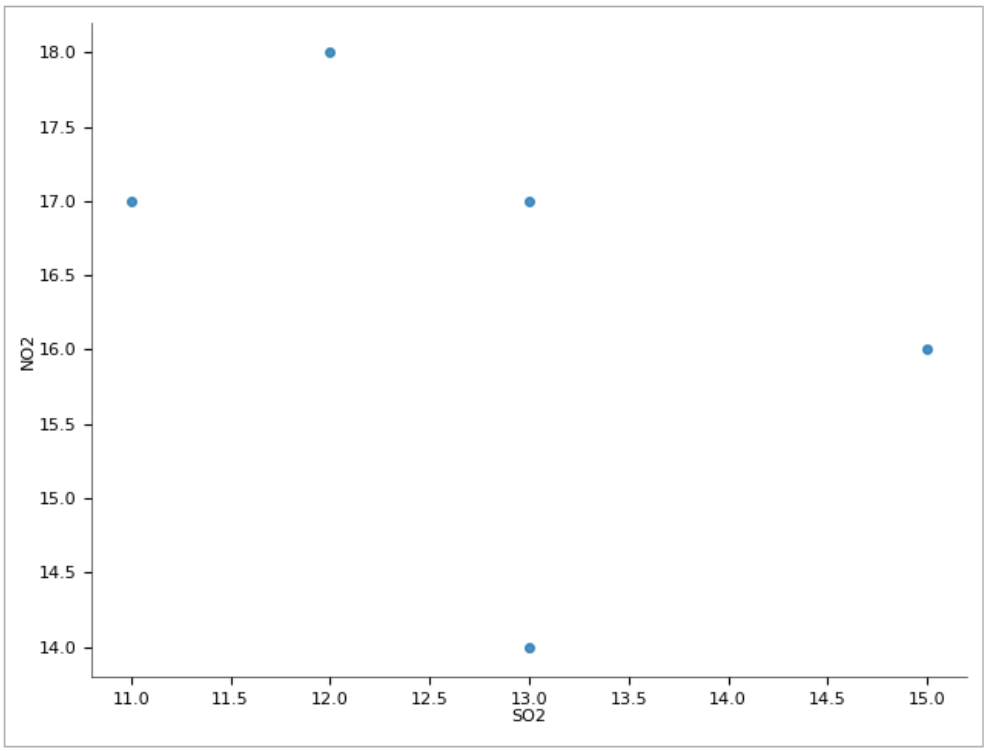
Distributions

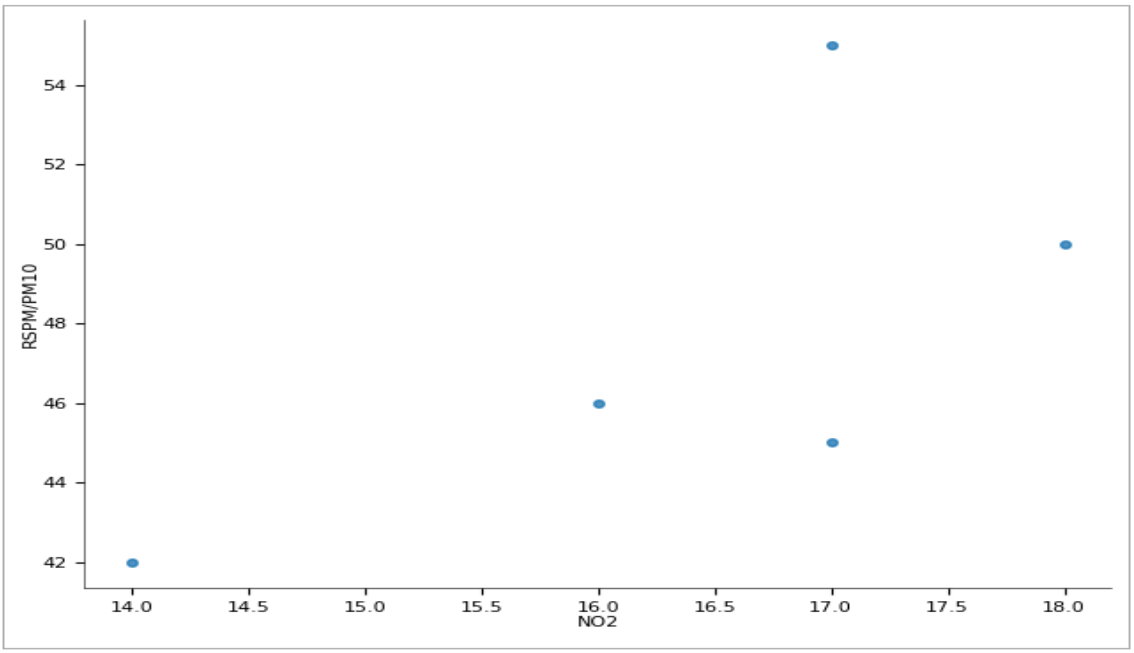


Categorical distributions

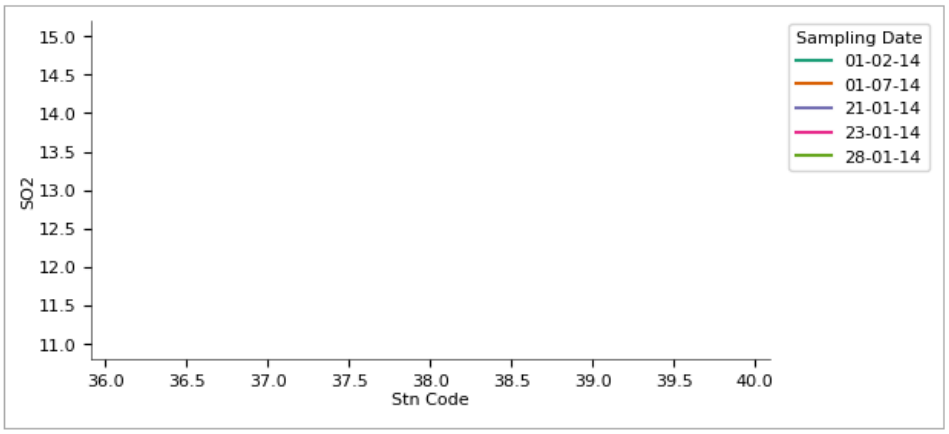


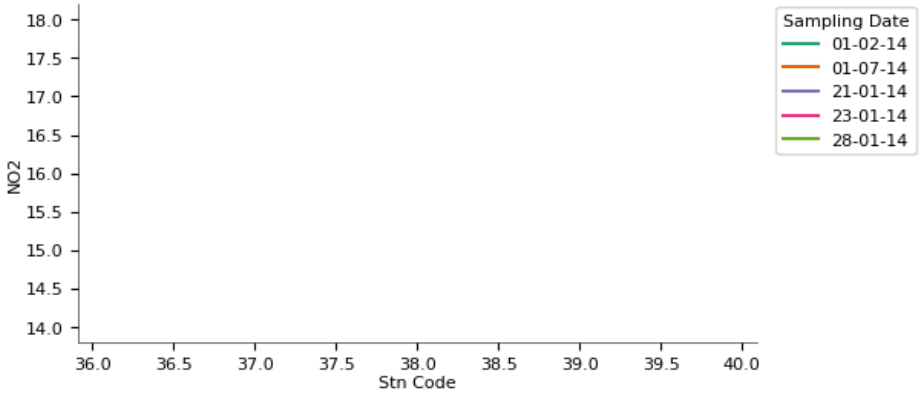
2-d distributions

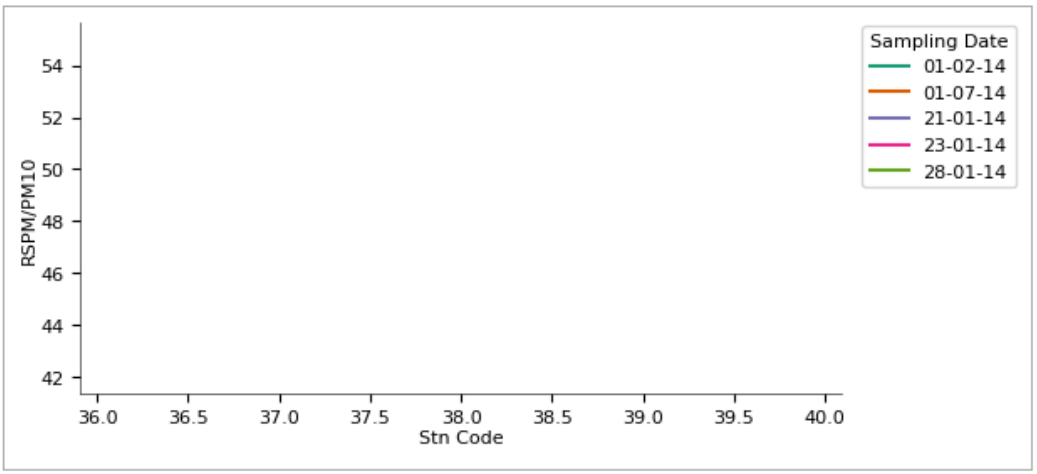


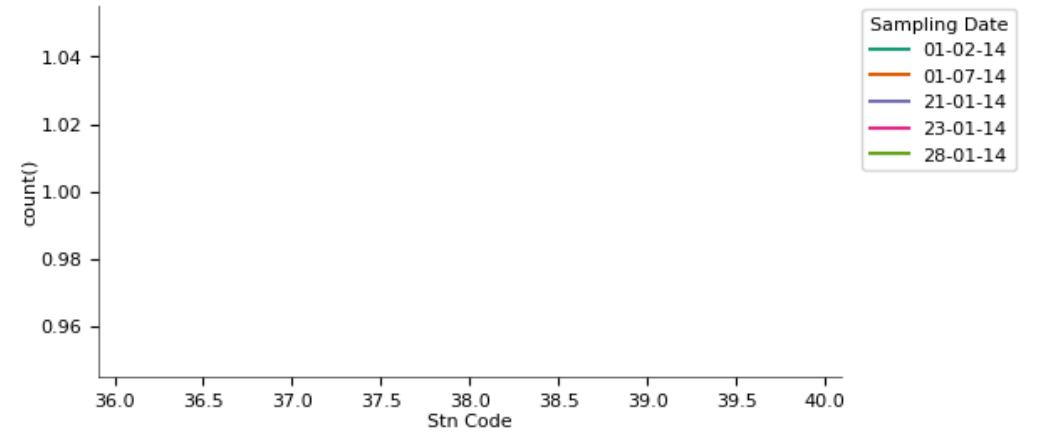


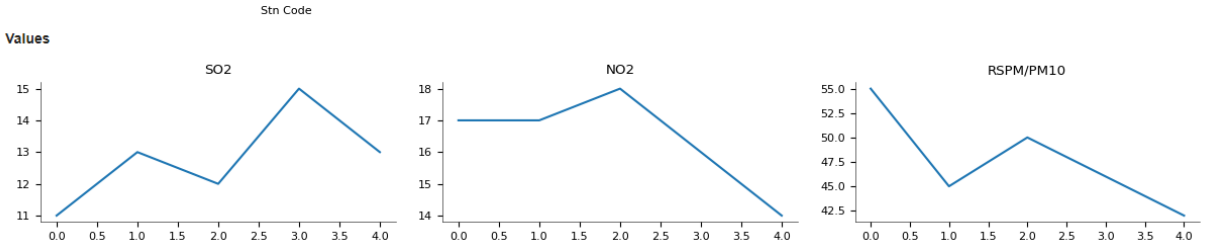
Time series

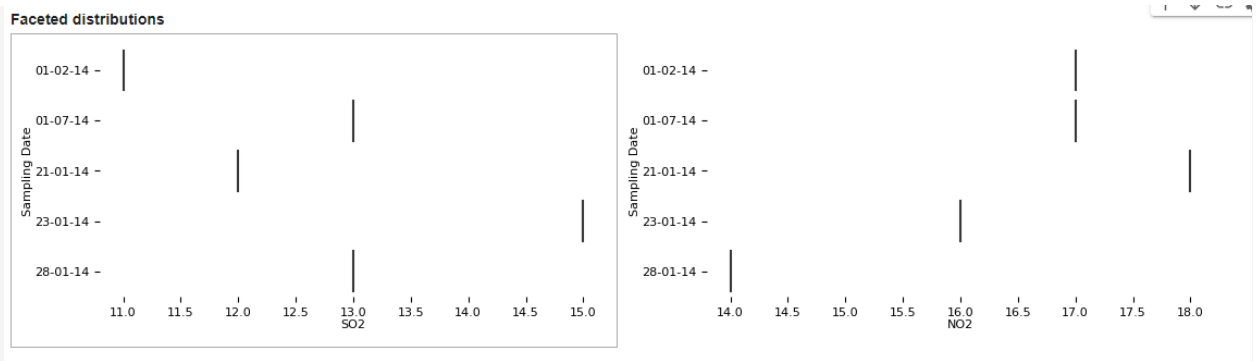

Sampling Date
01-02-14
01-07-14
21-01-14
23-01-14
28-01-14
SO2
15.0
14.5
14.0
13.5
13.0
12.5
12.0
11.5
11.0
36.0
36.5
37.0
37.5
38.0
38.5
39.0
39.5
40.0
Stn Code

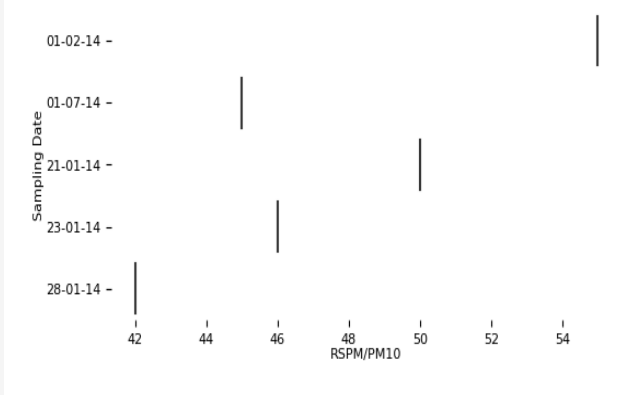

Sampling Date
01-02-14
01-07-14
21-01-14
23-01-14
28-01-14
NO2
18.0
17.5
17.0
16.5
16.0
15.5
15.0
14.5
14.0
36.0
36.5
37.0
37.5
38.0
38.5
39.0
39.5
40.0
Stn Code


Sampling Date
01-02-14
01-07-14
21-01-14
23-01-14
28-01-14
RSPM/PM10
54
52
50
48
46
44
42
36.0
36.5
37.0
37.5
38.0
38.5
39.0
39.5
40.0
Stn Code


Sampling Date
01-02-14
01-07-14
21-01-14
23-01-14
28-01-14
count()
1.04
1.02
1.00
0.98
0.96
36.0
36.5
37.0
37.5
38.0
38.5
39.0
39.5
40.0
Stn Code


Stn Code
Values
SO2
NO2
RSPM/PM10


Faceted distributions
Sampling Date
01-02-14
01-07-14
21-01-14
23-01-14
28-01-14
SO2
NO2




```
data.size
```
31669

```
data.shape
```
(2879, 11)

```
data.columns
```
Index(['Stn Code', 'Sampling Date', 'State', 'City/Town/Village/Area', 'Location of Monitoring Station', 'Agency', 'Type of Location', 'SO2', 'NO2', 'RSPM/PM10', 'PM 2.5'], dtype='object')

Code Explanation

This code appears to be Python code for Air data analysis and visualization using various libraries, including pandas, seaborn, matplotlib, and numpy. Let's break down the code step by step and explain what it does:

1. Import Necessary Libraries:

The code begins by importing several Python libraries, including pandas (as pd), seaborn (as sns), matplotlib.pyplot (as plt), and numpy (as np). These libraries are commonly used for data manipulation, analysis, and visualization.

2. Load Data from IBM Cognos:

The code uses Google Colab, which is a cloud-based Python development environment, to upload a dataset from IBM Cognos. It seems that the dataset is being uploaded manually, as indicated by the "Upload widget is only available when the cell has been executed in the current browser session" message. This suggests that the user is expected to upload the dataset manually by running this code cell.

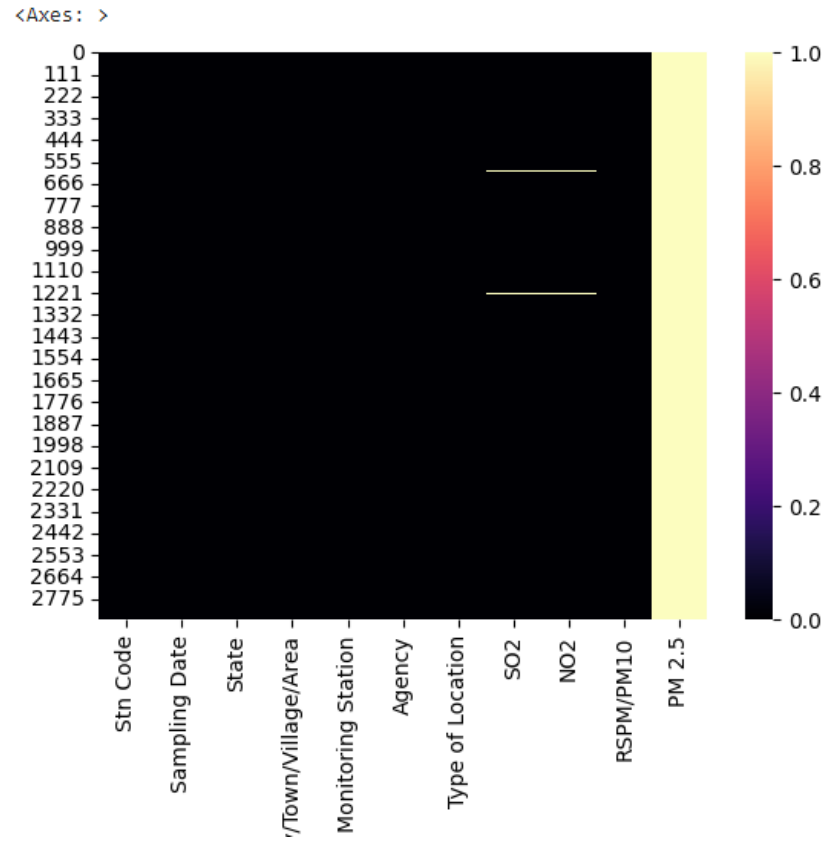
3. Basic Analysis of the Dataset:

Once the dataset is uploaded, the code proceeds to perform some basic analysis of the dataset. The dataset is loaded into a pandas DataFrame using the `pd.read_csv()` function. The dataset appears to be related to air quality monitoring in Tamil Nadu.

The `.head()` method is used to display the first few rows of the dataset, giving an overview of the data's structure. Each row represents a different measurement, and columns such as 'Stn Code,' 'Sampling Date,' 'State,' 'City/Town/Village/Area,' 'Location of Monitoring Station,' 'Agency,' 'Type of Location,' 'SO2,' 'NO2,' 'RSPM/PM10,' and 'PM 2.5' contain various pieces of information about each measurement.The dataset includes information about air quality measurements in different locations within Tamil Nadu, recorded on different dates.

Null value detection using HeatMap

```
sns.heatmap(data.isnull(),cmap='magma',cbar='false')
```



Code Explanation

The code `sns.heatmap(data.isnull(), cmap='magma', cbar=False)` is using the Seaborn library to create a heatmap that visualizes missing (null) values in the `data` DataFrame. Let's break down this code and provide a detailed explanation:

1. Import Seaborn and Matplotlib:

- Before using Seaborn to create the heatmap, you should have already imported Seaborn and Matplotlib, as indicated in your initial code snippet.

2. Load the Dataset:

The code assumes that you've previously loaded your dataset into the `data` DataFrame, as shown in your earlier code.

3. Create a Null Value Heatmap:

`data.isnull()` is used to create a new DataFrame of the same shape as the original `data`, where each cell contains a Boolean value (True or False) based on whether the corresponding cell in the original DataFrame is null (missing). This means that the new DataFrame will have the same structure as `data`, but it will be filled with True where data is missing and False where data is present.

`sns.heatmap()` is a Seaborn function used to create a heatmap. In this case, it's used to visualize the Boolean values in the DataFrame generated by `data.isnull()`.

`data.isnull()`: This is the DataFrame with Boolean values indicating missing data.

`cmap='magma'`: This argument specifies the colormap to be used for the heatmap. 'Magma' is a colormap with dark colors for low values and vibrant colors for high values. It's often used to create visually striking heatmaps.

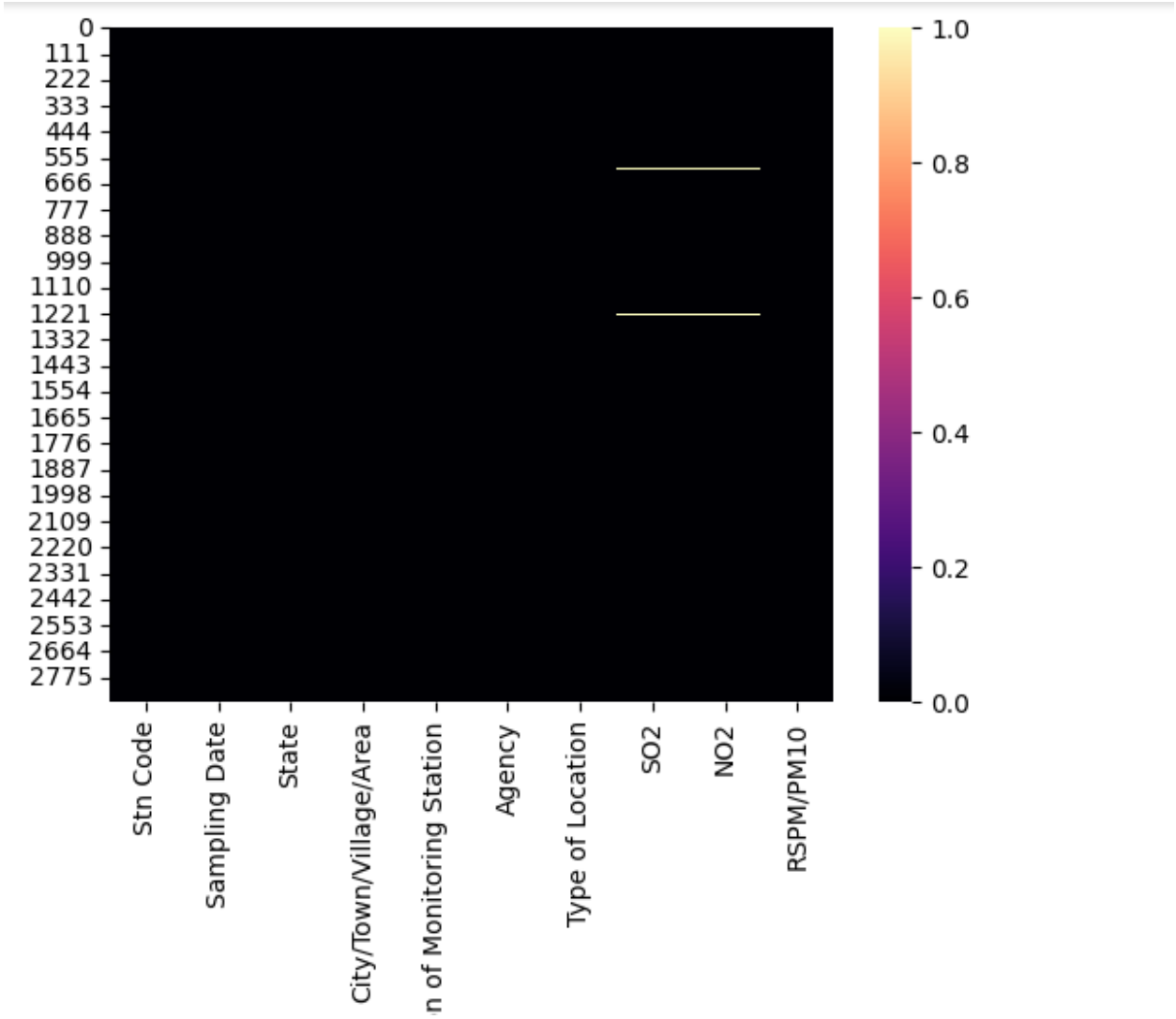
`cbar=False`: This argument specifies whether or not to display a colorbar. Setting it to `False` means that there won't be a colorbar on the side of the heatmap. The colorbar shows the mapping of values to colors, and in this case, it's not needed because the heatmap is binary (True/False), and the colors represent missing (True) or present (False) data.

4. Display the Heatmap:

Finally, `plt.show()` is used to display the heatmap in the current Jupyter Notebook or Python environment in google collab .The resulting heatmap will display missing values in your dataset as distinctive colored cells, making it easier to identify which columns or rows have missing data. In the 'magma' colormap, missing values are typically represented by a dark color, while non-missing values are represented by a lighter color. The `cbar=False` argument ensures that there is no colorbar alongside the heatmap since the data is binary (True/False).

```
newdata = data.drop('PM 2.5', axis=1)

newdata.columns

Index(['Stn Code', 'Sampling Date', 'State', 'City/Town/Village/Area',

       'Location of Monitoring Station', 'Agency', 'Type of Location', 'SO2',

       'NO2', 'RSPM/PM10'],

      dtype='object')
```

sns.heatmap(newdata.isnull(),cmap='magma', cbar='false')



Code Explanation :

The code you provided performs several operations on a DataFrame named `data` and another DataFrame named `newdata`. Here's a detailed explanation of the code:

1. Drop a Column from `data`:

The code starts by dropping the column 'PM 2.5' from the original DataFrame `data`. This is achieved using the `.drop()` method. The resulting DataFrame without the 'PM 2.5' column is stored in a new DataFrame called `newdata`. The line of code responsible for this is:

```python
newdata = data.drop('PM 2.5', axis=1)
```

`axis=1` indicates that you are dropping a column (as opposed to `axis=0`, which would be used to drop a row). The 'PM 2.5' column is removed, and the modified DataFrame is assigned to the `newdata` variable.

2. Inspect the Columns of `newdata`:

The code then displays the column names of the `newdata` DataFrame using the `.columns` attribute. This is done to verify that the 'PM 2.5' column has been successfully removed. The line of code is:

```python
newdata.columns
```

The output will show the names of the columns in the `newdata` DataFrame, which should no longer include 'PM 2.5'.

3. Create a Null Value Heatmap for `newdata`:

The code then creates a heatmap to visualize missing (null) values in the `newdata` DataFrame using Seaborn. The heatmap is similar to the one explained in your previous question, but it's applied to the modified DataFrame `newdata`. The code for creating the heatmap is:

```python
sns.heatmap(newdata.isnull(), cmap='magma', cbar=False)
```

`newdata.isnull()` generates a DataFrame with the same shape as `newdata`, containing Boolean values that indicate whether each cell is null (True) or not (False).

`cmap='magma'` specifies the colormap for the heatmap, using the 'magma' colormap, which provides distinct colors for missing and non-missing values.

`cbar=False` indicates that a colorbar should not be displayed next to the heatmap, which is appropriate when dealing with binary (True/False) data.

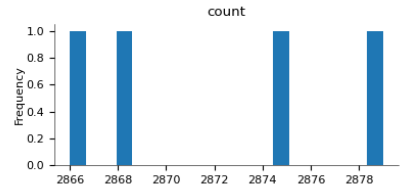
The resulting heatmap will visualize missing values in the columns of the `newdata` DataFrame, excluding the 'PM 2.5' column. This can help you identify where data is missing in the remaining columns of your dataset after dropping the specified column.

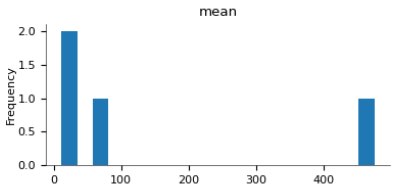
EDA

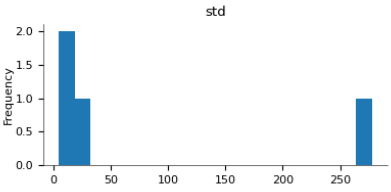
newdata.describe().T

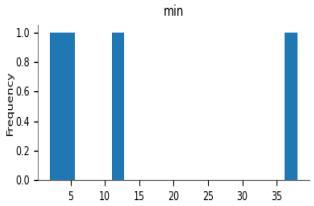
	count	mean	std	min	25%	50%	75%	max
Stn Code	2879.0	475.750261	277.675577	38.0	238.0	366.0	764.0	773.0
SO2	2868.0	11.503138	5.051702	2.0	8.0	12.0	15.0	49.0
NO2	2866.0	22.136776	7.128694	5.0	17.0	22.0	25.0	71.0
RSPM/PM10	2875.0	62.494261	31.368745	12.0	41.0	55.0	78.0	269.0

Distributions

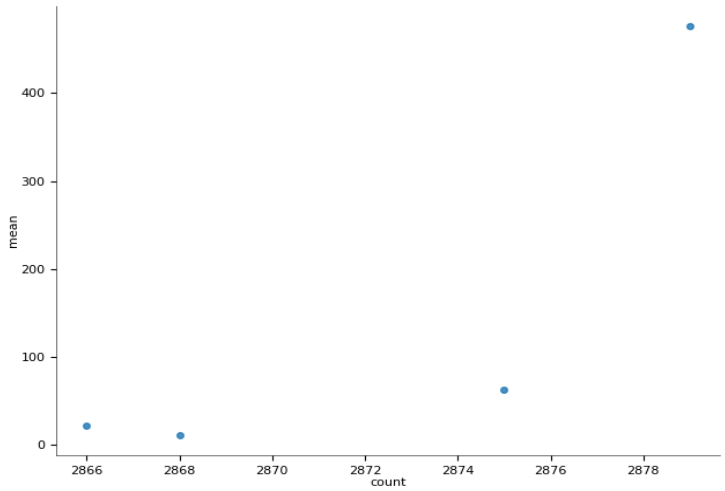


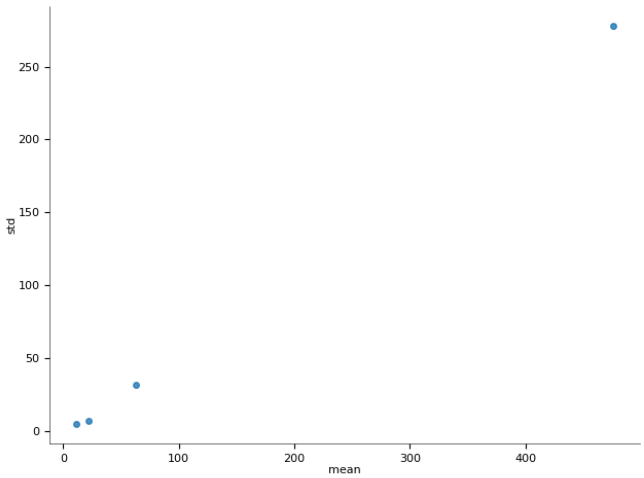


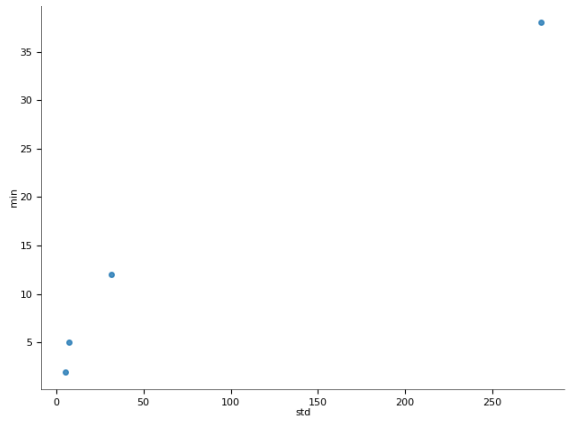


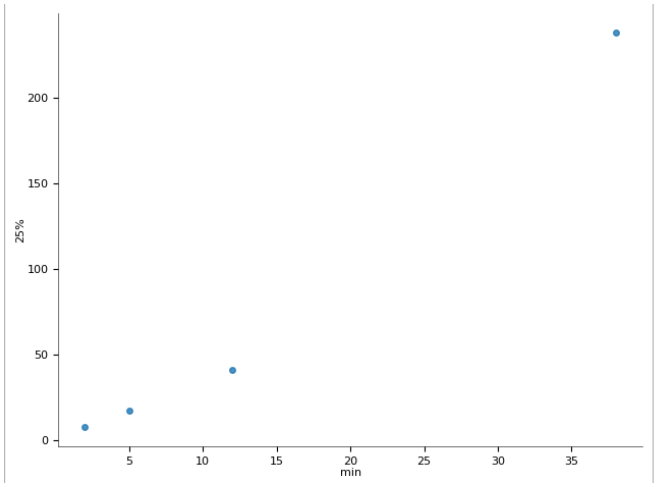


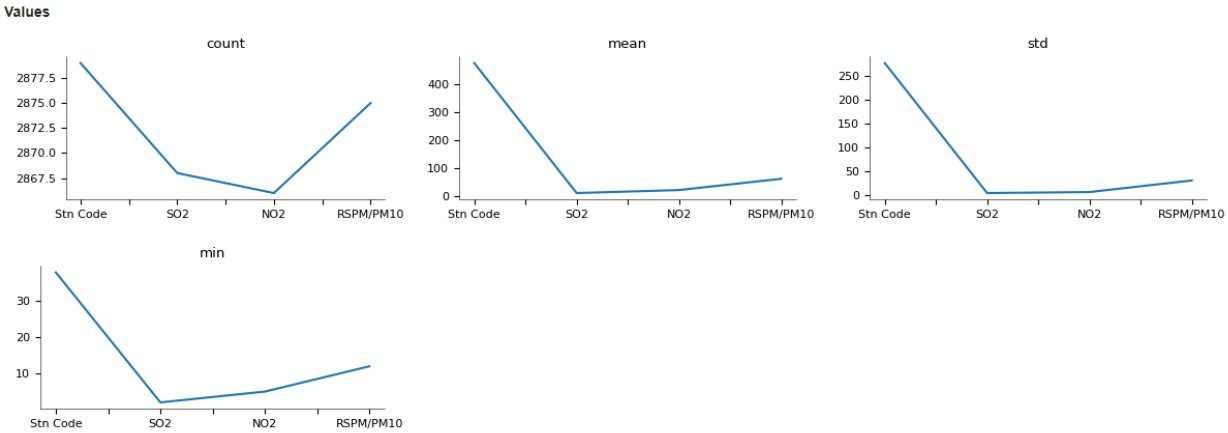
2-d distributions











Code Explanation:

The code `newdata.describe().T` is used to generate a summary statistical description of the columns in the DataFrame `newdata`, and the `.T` method is applied to transpose the resulting summary statistics. Here's a detailed explanation of this code:

1. `newdata.describe()`:

The `describe()` method is a pandas function used to generate summary statistics for the numeric columns in a DataFrame. It provides statistics such as count, mean, standard deviation, minimum, and maximum for each numeric column. Non-numeric columns are excluded from the summary.

2. `.T`:

The `.T` method is used to transpose the resulting summary statistics. By default, the `describe()` function generates the summary statistics in a vertical format, with columns as rows and statistics as columns. Transposing the result with `.T` swaps the rows and columns, making it more convenient to view the statistics.

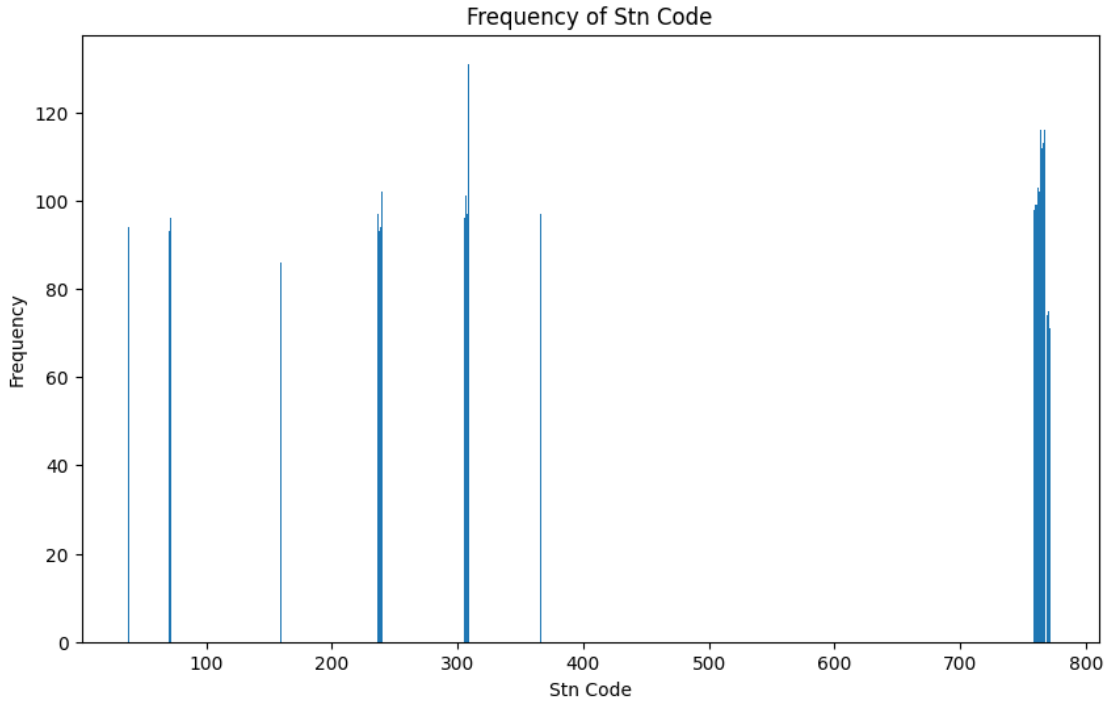
3. Overall Result:

The combination of `newdata.describe().T` takes the numeric columns in the `newdata` DataFrame, calculates the summary statistics for each numeric column, and then transposes the result so that each column name is now associated with the corresponding statistics (e.g., mean, standard deviation, min, max).

When you run this code, you'll get a table where each row represents a numeric column in the `newdata` DataFrame, and each column provides statistics for that specific column. It's a concise way to get an overview of the central tendency and dispersion of the data in your numeric columns. The output typically includes statistics such as count (number of non-null entries), mean (average), std (standard deviation), min (minimum value), and max (maximum value) for each numeric column.

1. Bar Plot (Stn Code, State, Agency):

```
# Count the frequency of each Stn Code
stn_code_counts = newdata['Stn Code'].value_counts()

# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.bar(stn_code_counts.index, stn_code_counts.values)
plt.xlabel('Stn Code')
plt.ylabel('Frequency')
plt.title('Frequency of Stn Code')
plt.show()
```



Code explanation:

This Python code is used to count the frequency of each unique value in the 'Stn Code' column of a data frame (presumably named 'newdata') and then create a bar chart to visualize this frequency distribution. Here's an explanation of each part of the code:

1. `stn_code_counts = newdata['Stn Code'].value_counts()`: This line of code extracts the 'Stn Code' column from the 'newdata' data frame and then applies the `value_counts()` function to it. The `value_counts()` function counts the occurrences of each unique value in the 'Stn Code' column and returns a Pandas Series where the index contains the unique 'Stn Code' values, and the values represent their respective frequencies.

2. `plt.figure(figsize=(10, 6))`: This line initializes a new Matplotlib figure with a specified figure size of 10 inches in width and 6 inches in height. This will determine the size of the resulting bar chart.

3. `plt.bar(stn_code_counts.index, stn_code_counts.values)`: This line creates a bar chart using Matplotlib's `bar` function. It takes two arguments:

- `stn_code_counts.index`: This provides the x-axis values for the bar chart, which are the unique 'Stn Code' values.

- `stn_code_counts.values`: This provides the corresponding y-axis values for the bar chart, which are the frequencies of each 'Stn Code' value.

4. `plt.xlabel('Stn Code')` and `plt.ylabel('Frequency')`: These lines set labels for the x-axis and y-axis of the bar chart to provide context for what is being displayed.

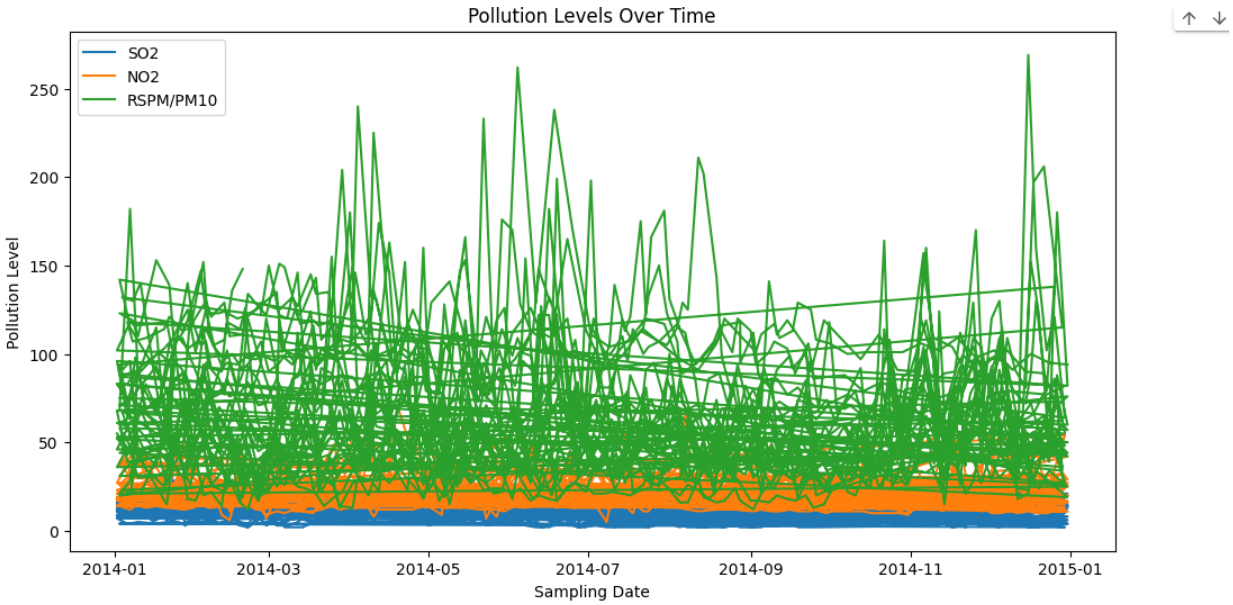
5. `plt.title('Frequency of Stn Code')`: This line sets a title for the bar chart to describe the purpose or content of the chart.

6. `plt.show()`: Finally, this command displays the bar chart on the screen. When you run this code, you will see a bar chart that shows the frequency of each unique 'Stn Code' value in the 'newdata' data frame.

The resulting chart will have 'Stn Code' values on the x-axis and the corresponding frequencies on the y-axis, allowing you to quickly visualize the distribution of 'Stn Code' values in the data.

Time Series Plot (Sampling Date):

```
# Convert 'Sampling Date' to datetime if not already
newdata['Sampling Date'] = pd.to_datetime(newdata['Sampling Date'])

# Set 'Sampling Date' as index for time series plot
newdata.set_index('Sampling Date', inplace=True)

# Plotting time series
plt.figure(figsize=(12, 6))
plt.plot(newdata.index, newdata['SO2'], label='SO2')
plt.plot(newdata.index, newdata['NO2'], label='NO2')
plt.plot(newdata.index, newdata['RSPM/PM10'], label='RSPM/PM10')
plt.xlabel('Sampling Date')
plt.ylabel('Pollution Level')
plt.title('Pollution Levels Over Time')
plt.legend()
plt.show()
```



Code Explanation:

This Python code is used to count the frequency of each unique value in the 'Stn Code' column of a data frame (presumably named 'newdata') and then create a bar chart to visualize this frequency distribution. Here's an explanation of each part of the code:

1. `stn_code_counts = newdata['Stn Code'].value_counts()`: This line of code extracts the 'Stn Code' column from the 'newdata' data frame and then applies the `value_counts()` function to it. The `value_counts()` function counts the occurrences of each unique value in the 'Stn Code' column and returns a Pandas Series where the index contains the unique 'Stn Code' values, and the values represent their respective frequencies.

2. `plt.figure(figsize=(10, 6))`: This line initializes a new Matplotlib figure with a specified figure size of 10 inches in width and 6 inches in height. This will determine the size of the resulting bar chart.

3. `plt.bar(stn_code_counts.index, stn_code_counts.values)`: This line creates a bar chart using Matplotlib's `bar` function. It takes two arguments:

   - `stn_code_counts.index`: This provides the x-axis values for the bar chart, which are the unique 'Stn Code' values.

   - `stn_code_counts.values`: This provides the corresponding y-axis values for the bar chart, which are the frequencies of each 'Stn Code' value.

4. `plt.xlabel('Stn Code')` and `plt.ylabel('Frequency')`: These lines set labels for the x-axis and y-axis of the bar chart to provide context for what is being displayed.

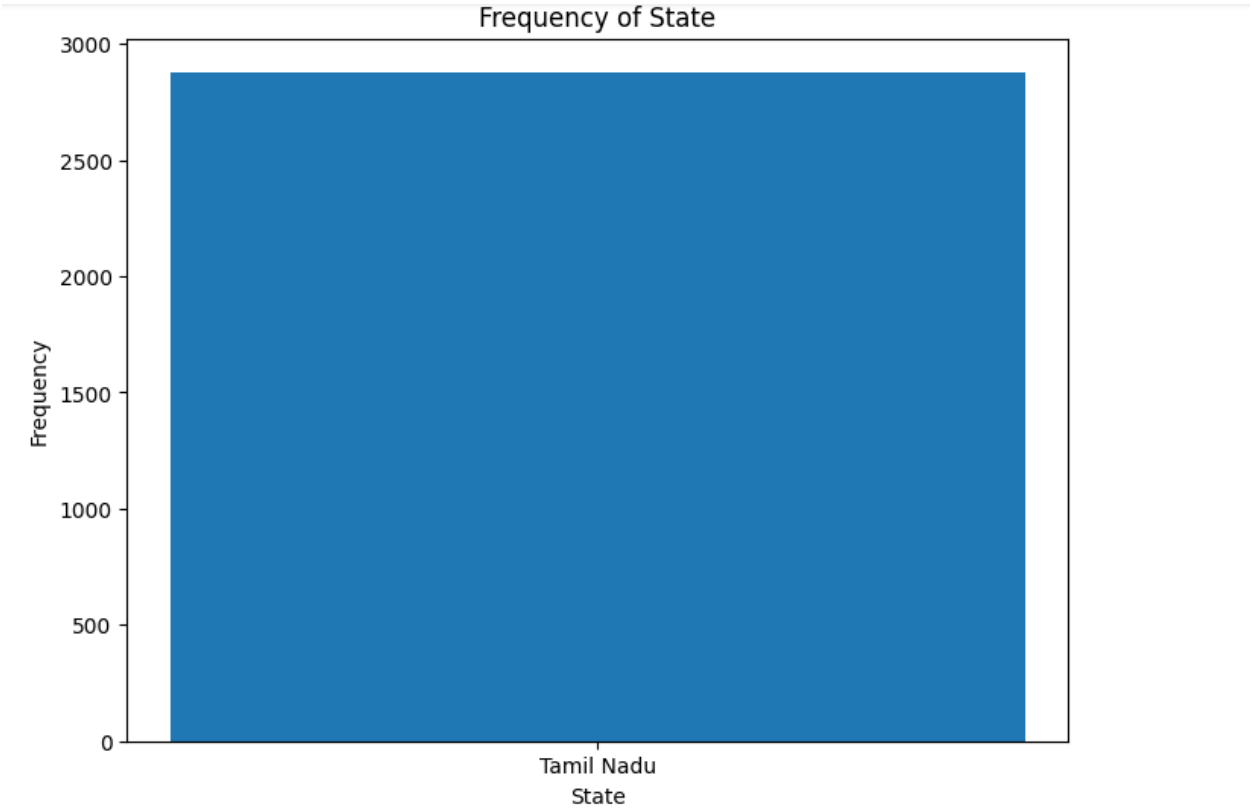
5. `plt.title('Frequency of Stn Code')`: This line sets a title for the bar chart to describe the purpose or content of the chart.

6. `plt.show()`: Finally, this command displays the bar chart on the screen. When you run this code, you will see a bar chart that shows the frequency of each unique 'Stn Code' value in the 'newdata' data frame.

The resulting chart will have 'Stn Code' values on the x-axis and the corresponding frequencies on the y-axis, allowing you to quickly visualize the distribution of 'Stn Code' values in the data.

3. Stacked Bar Chart (Type of Location):

```
# Count the frequency of each State
state_counts = newdata['State'].value_counts()

# Plotting the bar chart
plt.figure(figsize=(8, 6))
plt.bar(state_counts.index, state_counts.values)
plt.xlabel('State')
plt.ylabel('Frequency')
plt.title('Frequency of State')
plt.show()
```



Code Explanation:

This Python code is used to count the frequency of each unique value in the 'State' column of a data frame (presumably named 'newdata') and then create a bar chart to visualize this frequency distribution. Here's an explanation of each part of the code:

1. `state_counts = newdata['State'].value_counts()`: This line of code extracts the 'State' column from the 'newdata' data frame and then applies the `value_counts()` function to it. The `value_counts()` function counts the occurrences of each unique value in the 'State' column and returns a Pandas Series where the index contains the unique state names, and the values represent their respective frequencies.

2. `plt.figure(figsize=(8, 6))`: This line initializes a new Matplotlib figure with a specified figure size of 8 inches in width and 6 inches in height. This will determine the size of the resulting bar chart.

3. `plt.bar(state_counts.index, state_counts.values)`: This line creates a bar chart using Matplotlib's `bar` function. It takes two arguments:

   - `state_counts.index`: This provides the x-axis values for the bar chart, which are the unique state names.

   - `state_counts.values`: This provides the corresponding y-axis values for the bar chart, which are the frequencies of each state in the 'State' column.

4. `plt.xlabel('State')` and `plt.ylabel('Frequency')`: These lines set labels for the x-axis and y-axis of the bar chart to provide context for what is being displayed.

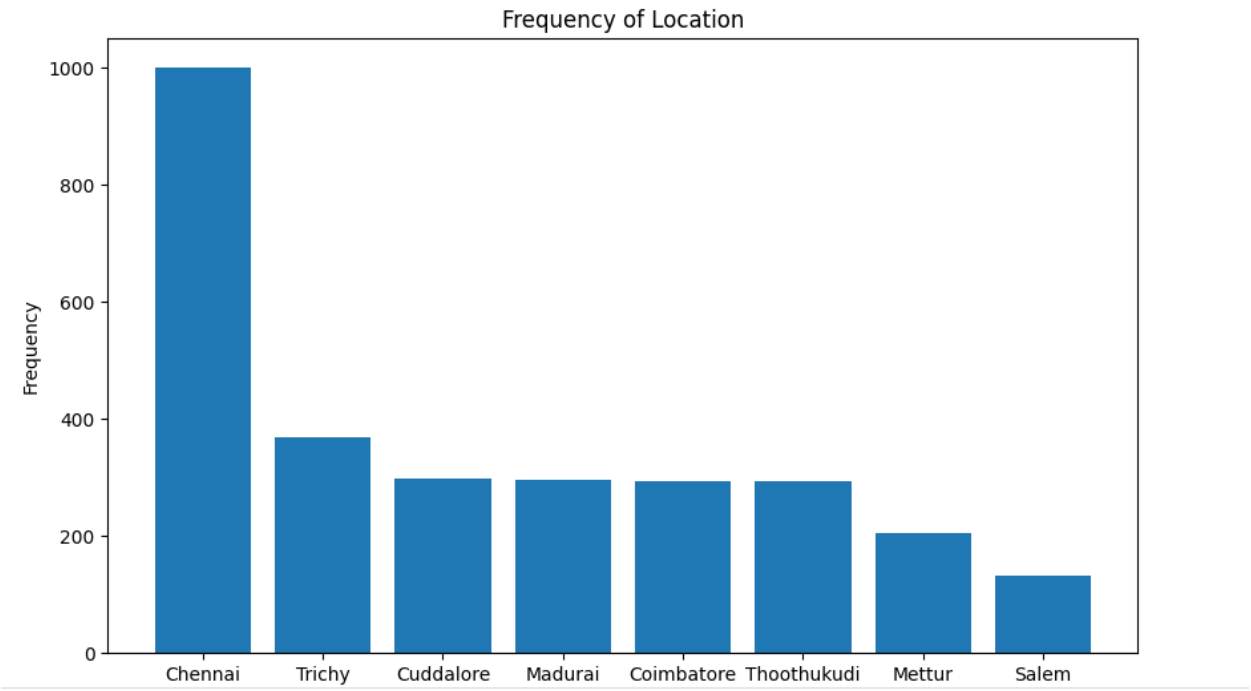
5. `plt.title('Frequency of State')`: This line sets a title for the bar chart to describe the purpose or content of the chart. In this case, it's showing the frequency of each state.

6. `plt.show()`: Finally, this command displays the bar chart on the screen. When you run this code, you will see a bar chart that shows the frequency of each unique state name in the 'State' column of the 'newdata' data frame.

The resulting chart will have state names on the x-axis and the corresponding frequencies on the y-axis, allowing you to quickly visualize the distribution of states in the dataset and how frequently they appear.

4. Bar Plot for 'City/Town/Village/Area

```
# Count the frequency of each Location
location_counts = newdata['City/Town/Village/Area'].value_counts()

# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.bar(location_counts.index, location_counts.values)
plt.xlabel('Location')
plt.ylabel('Frequency')
plt.title('Frequency of Location')
plt.show()
```



Code explanation:

This Python code is used to count the frequency of each unique value in the 'State' column of a data frame (presumably named 'newdata') and then create a bar chart to visualize this frequency distribution. Here's an explanation of each part of the code:

1. `state_counts = newdata['State'].value_counts()`: This line of code extracts the 'State' column from the 'newdata' data frame and then applies the `value_counts()` function to it. The `value_counts()` function counts the occurrences of each unique value in the 'State' column and

returns a Pandas Series where the index contains the unique state names, and the values represent their respective frequencies.

2. `plt.figure(figsize=(8, 6))`: This line initializes a new Matplotlib figure with a specified figure size of 8 inches in width and 6 inches in height. This will determine the size of the resulting bar chart.

3. `plt.bar(state_counts.index, state_counts.values)`: This line creates a bar chart using Matplotlib's `bar` function. It takes two arguments:

- `state_counts.index`: This provides the x-axis values for the bar chart, which are the unique state names.

- `state_counts.values`: This provides the corresponding y-axis values for the bar chart, which are the frequencies of each state in the 'State' column.

4. `plt.xlabel('State')` and `plt.ylabel('Frequency')`: These lines set labels for the x-axis and y-axis of the bar chart to provide context for what is being displayed.

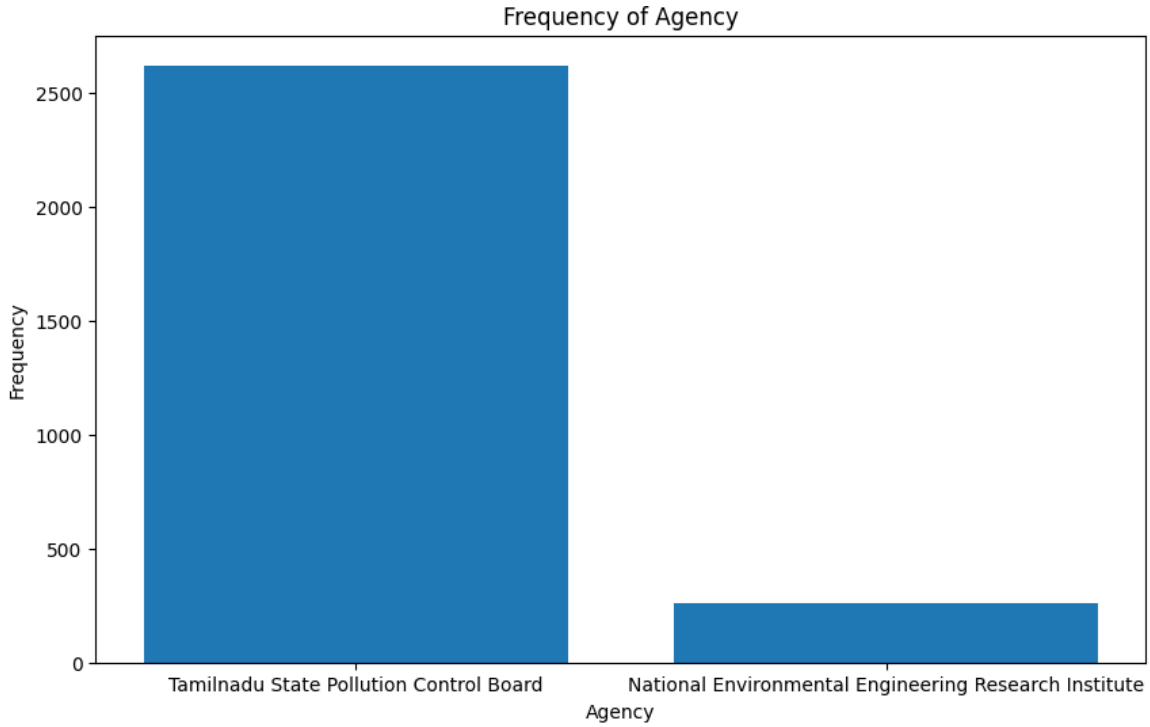
5. `plt.title('Frequency of State')`: This line sets a title for the bar chart to describe the purpose or content of the chart. In this case, it's showing the frequency of each state.

6. `plt.show()`: Finally, this command displays the bar chart on the screen. When you run this code, you will see a bar chart that shows the frequency of each unique state name in the 'State' column of the 'newdata' data frame.

The resulting chart will have state names on the x-axis and the corresponding frequencies on the y-axis, allowing you to quickly visualize the distribution of states in the dataset and how frequently they appear.

5.Bar Plot for 'Agency'

```
# Count the frequency of each Agency
agency_counts = newdata['Agency'].value_counts()

# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.bar(agency_counts.index, agency_counts.values)
plt.xlabel('Agency')
plt.ylabel('Frequency')
plt.title('Frequency of Agency')
plt.show()
```



Code Explanation:

This Python code is used to count the frequency of each unique value in the 'Agency' column of a data frame (presumably named 'newdata') and then create a bar chart to visualize this frequency distribution. Here's an explanation of each part of the code:

1. `agency_counts = newdata['Agency'].value_counts()`: This line of code extracts the 'Agency' column from the 'newdata' data frame and then applies the `value_counts()` function to it. The `value_counts()` function counts the occurrences of each unique value in the 'Agency' column and returns a Pandas Series where the index contains the unique agency names, and the values represent their respective frequencies.

2. `plt.figure(figsize=(10, 6))`: This line initializes a new Matplotlib figure with a specified figure size of 10 inches in width and 6 inches in height. This will determine the size of the resulting bar chart.

3. `plt.bar(agency_counts.index, agency_counts.values)`: This line creates a bar chart using Matplotlib's `bar` function. It takes two arguments:

- `agency_counts.index`: This provides the x-axis values for the bar chart, which are the unique agency names.

- `agency_counts.values`: This provides the corresponding y-axis values for the bar chart, which are the frequencies of each agency in the 'Agency' column.

4. `plt.xlabel('Agency')` and `plt.ylabel('Frequency')`: These lines set labels for the x-axis and y-axis of the bar chart to provide context for what is being displayed.

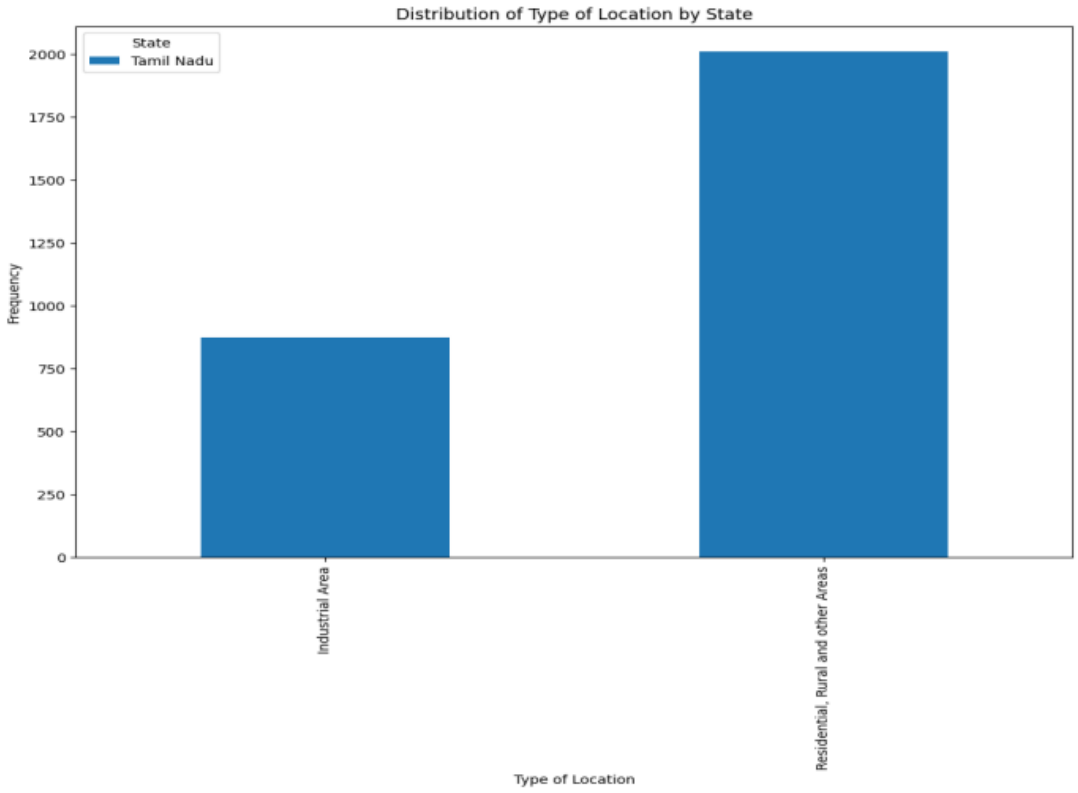
5. `plt.title('Frequency of Agency')`: This line sets a title for the bar chart to describe the purpose or content of the chart. In this case, it's showing the frequency of each agency.\

6. `plt.show()`: Finally, this command displays the bar chart on the screen. When you run this code, you will see a bar chart that shows the frequency of each unique agency name in the 'Agency' column of the 'newdata' data frame.

The resulting chart will have agency names on the x-axis and the corresponding frequencies on the y-axis, allowing you to quickly visualize the distribution of agencies in the dataset and how frequently they appear.

6.Stacked Bar Chart for 'Type of Location'

```
# Create a crosstab of Type of Location and State
location_type_state = pd.crosstab(newdata['Type of Location'], newdata['State'])

# Plotting the stacked bar chart
location_type_state.plot(kind='bar', stacked=True, figsize=(12, 8))
plt.xlabel('Type of Location')
plt.ylabel('Frequency')
plt.title('Distribution of Type of Location by State')
plt.show()
```



Code explanation:

This Python code is used to create a stacked bar chart that visualizes the distribution of the 'Type of Location' across different states in the dataset. It involves creating a crosstab (cross-tabulation) of 'Type of Location' and 'State' and then plotting the crosstab results as a stacked bar chart. Here's an explanation of each part of the code:

1. `location_type_state = pd.crosstab(newdata['Type of Location'], newdata['State'])`: This line of code uses the `pd.crosstab()` function from the Pandas library to create a cross-tabulation of 'Type of Location' and 'State'. The resulting DataFrame (`location_type_state`) will have 'Type of Location' values as rows and 'State' values as columns. Each cell in the table will represent the frequency of a specific 'Type of Location' in a specific 'State'.

2. `location_type_state.plot(kind='bar', stacked=True, figsize=(12, 8))`: This line creates a stacked bar chart using the `plot` function on the `location_type_state` DataFrame. The parameters used are as follows:

- `kind='bar'`: Specifies that you want to create a bar chart.

- `stacked=True`: Stacks the bars on top of each other for each 'Type of Location' within each 'State'. This allows you to see the total frequency for each 'State' and how it is divided by 'Type of Location'.

- `figsize=(12, 8)`: Specifies the figure size with a width of 12 inches and a height of 8 inches.

3. `plt.xlabel('Type of Location')` and `plt.ylabel('Frequency')`: These lines set labels for the x-axis and y-axis of the stacked bar chart to provide context for what is being displayed.

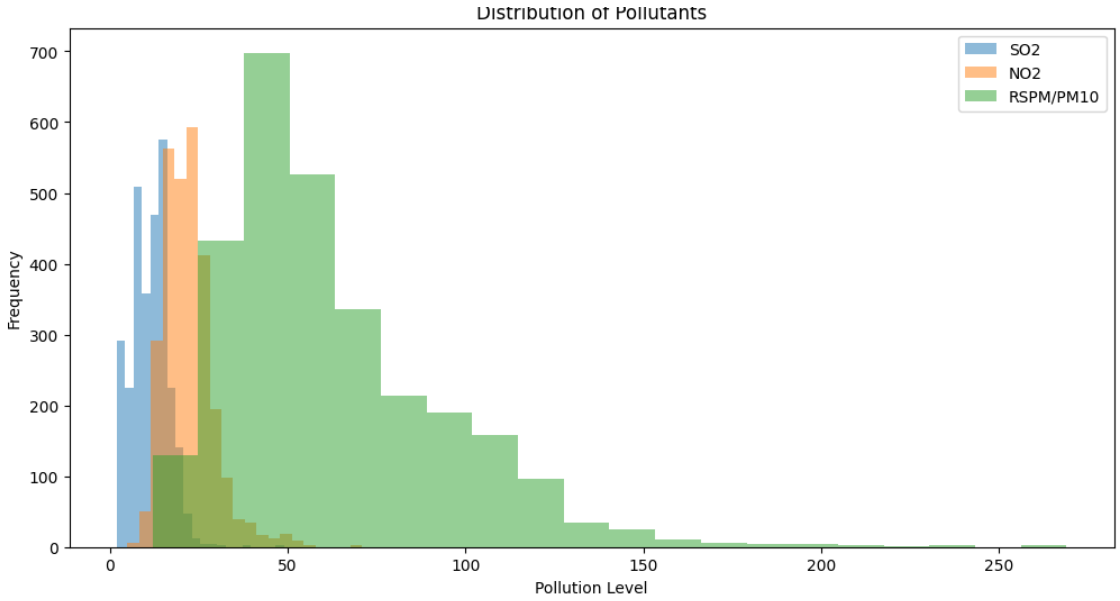
4. `plt.title('Distribution of Type of Location by State')`: This line sets a title for the stacked bar chart to describe the purpose or content of the chart.

5. `plt.show()`: Finally, this command displays the stacked bar chart on the screen. When you run this code, you will see a chart that shows how the 'Type of Location' values are distributed across different states, with the bars stacked on top of each other, indicating the frequency of each 'Type of Location' in each state.

The resulting chart will provide a visual representation of how the various types of locations are distributed within each state in your dataset.

8. Histogram for 'SO2', 'NO2', 'RSPM/PM10'

```
# Plotting histograms
plt.figure(figsize=(12, 6))
plt.hist(newdata['SO2'], bins=20, alpha=0.5, label='SO2')
plt.hist(newdata['NO2'], bins=20, alpha=0.5, label='NO2')
plt.hist(newdata['RSPM/PM10'], bins=20, alpha=0.5, label='RSPM/PM10')
plt.xlabel('Pollution Level')
plt.ylabel('Frequency')
plt.title('Distribution of Pollutants')
plt.legend()
plt.show()
```



Code explanation:

This Python code is used to create histograms to visualize the distribution of pollution levels for three different pollutants (SO2, NO2, and RSPM/PM10). Here's an explanation of each part of the code:

1. `plt.figure(figsize=(12, 6))`: This line initializes a new Matplotlib figure with a specified figure size of 12 inches in width and 6 inches in height. This determines the size of the resulting histograms.

2. `plt.hist(newdata['SO2'], bins=20, alpha=0.5, label='SO2')`, `plt.hist(newdata['NO2'], bins=20, alpha=0.5, label='NO2')`, `plt.hist(newdata['RSPM/PM10'], bins=20, alpha=0.5, label='RSPM/PM10')`: These three lines create histograms using Matplotlib's `hist` function. Each line represents the distribution of pollution levels for a specific pollutant (SO2, NO2, and RSPM/PM10). The parameters used are as follows:

- `newdata['SO2']`, `newdata['NO2']`, `newdata['RSPM/PM10']`: These specify the data to be plotted, which are the pollution level values for each pollutant.

- `bins=20`: This specifies the number of bins (intervals) to divide the data into. In this case, it's set to 20, so you will have 20 bars in each histogram.

- `alpha=0.5`: This parameter controls the transparency of the bars. An alpha value of 0.5 makes the bars semi-transparent, allowing you to see overlapping areas more clearly.

- `label='SO2'`, `label='NO2'`, `label='RSPM/PM10'`: These labels are used to identify each histogram when creating a legend later.

3. `plt.xlabel('Pollution Level')` and `plt.ylabel('Frequency')`: These lines set labels for the x-axis (pollution level) and y-axis (frequency) of the histograms to provide context for what is being displayed.

4. `plt.title('Distribution of Pollutants')`: This line sets a title for the histograms to describe the purpose or content of the chart.

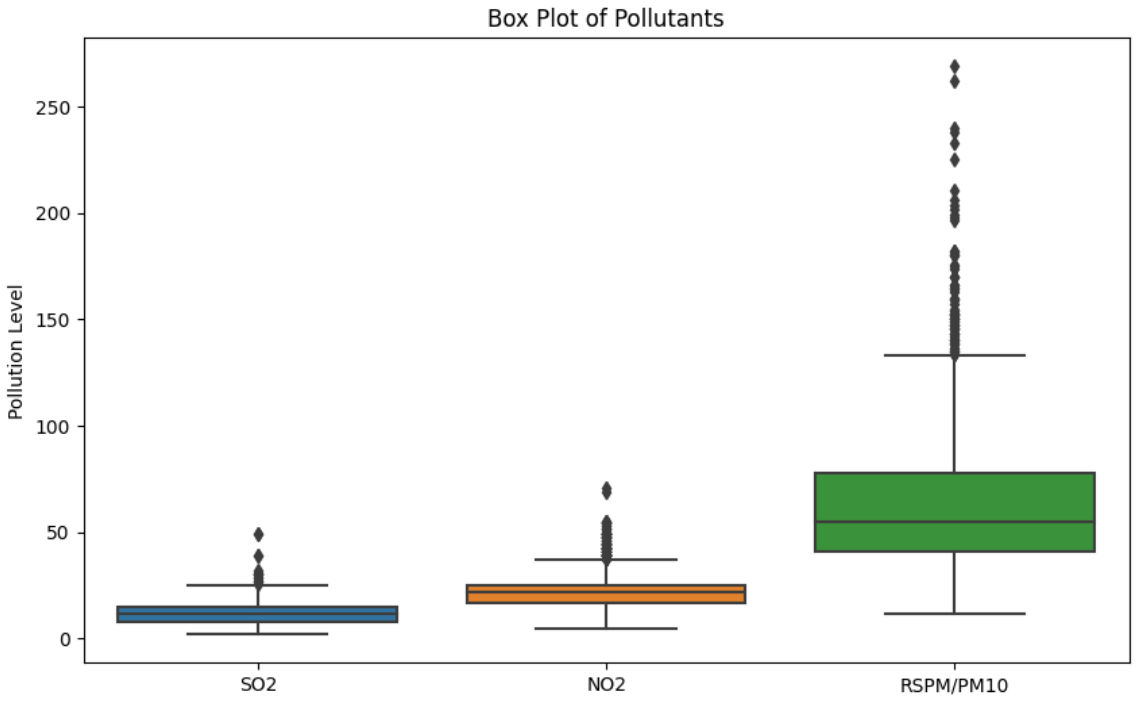
5. `plt.legend()`: This line adds a legend to the plot, allowing you to distinguish between the different pollutants (SO2, NO2, and RSPM/PM10) represented by the histograms.

6. `plt.show()`: Finally, this command displays the histograms on the screen. When you run this code, you will see three histograms stacked on top of each other, each representing the distribution of pollution levels for a specific pollutant. The transparency (alpha) helps visualize areas where the distributions overlap.

These histograms provide insights into the distribution of pollution levels for each pollutant and can help identify patterns and potential outliers in the data.

Box Plot for 'SO2', 'NO2', 'RSPM/PM10'

```
# Create a combined DataFrame for box plot
pollutants = newdata[['SO2', 'NO2', 'RSPM/PM10']]

# Plotting box plots
plt.figure(figsize=(10, 6))
sns.boxplot(data=pollutants)
plt.ylabel('Pollution Level')
plt.title('Box Plot of Pollutants')
plt.show()
```



Code explanation:

The provided code snippet is creating a combined DataFrame for box plot visualization and then using the Seaborn library to plot box plots for the selected pollutants (SO2, NO2, and RSPM/PM10). Here's a step-by-step explanation of the code:

1. `pollutants = newdata[['SO2', 'NO2', 'RSPM/PM10']]`: This line creates a new DataFrame called `pollutants` by selecting the columns 'SO2', 'NO2', and 'RSPM/PM10' from the DataFrame `newdata`. The resulting DataFrame `pollutants` contains only these three columns.

2. `plt.figure(figsize=(10, 6))`: This line sets the figure size for the box plot to have a width of 10 units and a height of 6 units. The `plt` object here is usually associated with the Matplotlib library, which is commonly used for creating various types of plots and charts.

3. `sns.boxplot(data=pollutants)`: This line uses Seaborn, which is a Python data visualization library built on top of Matplotlib, to create a box plot. The `sns.boxplot` function is called with the `data` parameter set to the `pollutants` DataFrame, indicating that we want to create a box plot using the data in this DataFrame. Seaborn will automatically create box plots for each of the three pollutants in the `pollutants` DataFrame.

4. `plt.ylabel('Pollution Level')`: This line adds a label to the y-axis of the plot, specifying that it represents the "Pollution Level." This is done using the Matplotlib `plt` object.

5. `plt.title('Box Plot of Pollutants')`: This line adds a title to the plot, setting it as "Box Plot of Pollutants."

6. `plt.show()`: This line is used to display the plot. It shows the box plot with the specified labels and title on your screen.

It's important to have the necessary libraries imported, such as Matplotlib and Seaborn, before running this code. The code combines data from `newdata`, selects specific columns, and then uses Seaborn to create a box plot to visualize the distribution of pollution levels for the selected pollutants.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

loading the data set from the given source by IBM cognos

```
from google.colab import files
uploaded = files.upload()
```

BAsic analysis of the guven Dataset

```
data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")
data.head()
```

Out[ ]:

Out[ ]:

	Stn Code	Sampling Date	State	City/Town/Village/Area	Location of Monitoring Station	Agency	Type of Location	SO2	NO2	RSPM/PM10	PM 2.5
0	38	01-02-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	11.0	17.0	55.0	NaN
1	38	01-07-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	13.0	17.0	45.0	NaN
2	38	21-01-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	12.0	18.0	50.0	NaN
3	38	23-01-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	15.0	16.0	46.0	NaN
4	38	28-01-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	13.0	14.0	42.0	NaN

The provided code snippet appears to be for loading a dataset, performing basic analysis, and displaying the first few rows of the dataset using Python libraries. Here's a step-by-step explanation of the code:

1. Import Necessary Libraries:

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

These lines import the required Python libraries:

- `pandas` for data manipulation and analysis.
- `seaborn` for data visualization.
- `matplotlib.pyplot` for creating plots and charts.
- `numpy` for numerical operations.

2. Load Data from a File:

```python
from google.colab import files
uploaded = files.upload()
```

These lines appear to be part of a Jupyter Notebook (possibly Google Colab) setup. They allow you to upload a file from your local system to the notebook. The `files.upload()` function lets you select a file for upload.

3. Read the CSV File into a DataFrame:

```python
data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")
```

This line uses the `pd.read_csv()` function from the `pandas` library to read the data from the CSV file named "cpcb_dly_aq_tamil_nadu-2014 (1).csv" and store it in a DataFrame named `data`. This DataFrame will hold the dataset for further analysis.

4. Display the First Few Rows of the DataFrame:

```python
data.head()
```

This line displays the first few rows of the `data` DataFrame. It gives you a quick look at the dataset to understand its structure and contents. The `head()` function displays the top 5 rows by default.

In summary, this code imports necessary libraries, uploads a dataset file from your local system, reads the dataset into a DataFrame, and then displays the first few rows of the dataset to get an initial view of the data. Make sure the file "cpcb_dly_aq_tamil_nadu-2014 (1).csv" is uploaded to your environment before running this code.

```
data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")
data.describe()
```

Out[ ]:

	Stn Code	SO2	NO2	RSPM/PM10	PM 2.5
count	2879.000000	2868.000000	2866.000000	2875.000000	0.0
mean	475.750261	11.503138	22.136776	62.494261	NaN
std	277.675577	5.051702	7.128694	31.368745	NaN
min	38.000000	2.000000	5.000000	12.000000	NaN
25%	238.000000	8.000000	17.000000	41.000000	NaN
50%	366.000000	12.000000	22.000000	55.000000	NaN
75%	764.000000	15.000000	25.000000	78.000000	NaN
max	773.000000	49.000000	71.000000	269.000000	NaN

The code reads a CSV file named "cpcb_dly_aq_tamil_nadu-2014 (1).csv" into a pandas DataFrame and then generates a statistical summary of the data using the `describe()` method. Here's a step-by-step explanation of the code and its output:

1. Reading the CSV File into a DataFrame:

```python
data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")
```

This line reads the data from the CSV file and stores it in a pandas DataFrame called `data`. The data now resides in a structured format that allows for easy manipulation and analysis.

2. Generating a Statistical Summary:

```python
data.describe()
```

This line calls the `describe()` method on the `data` DataFrame. The `describe()` method generates various summary statistics for the numeric columns in the DataFrame. In the output, you can see the following statistics for each numeric column:

- count: The number of non-missing (non-NaN) values in each column.
- mean: The mean (average) value for each column.
- std: The standard deviation, which measures the spread or dispersion of the data.
- min: The minimum value in each column.
- 25%: The 25th percentile value, which is the value below which 25% of the data falls.
- 50%: The median or 50th percentile value.
- 75%: The 75th percentile value, which is the value below which 75% of the data falls.
- max: The maximum value in each column.

The output shows the statistics for the columns 'Stn Code,' 'SO2,' 'NO2,' 'RSPM/PM10,' and 'PM 2.5' in your dataset.

It's important to note that the 'PM 2.5' column appears to contain NaN values in this dataset, which is why you see "NaN" in the output for that column. The "NaN" values mean that there are missing data points in the 'PM 2.5' column. The `describe()` function calculates statistics only for the non-missing data.

viewing the data

```
data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")
data.head()
```

Out[ ]:

	Stn Code	Sampling Date	State	City/Town/Village/Area	Location of Monitoring Station	Agency	Type of Location	SO2	NO2	RSPM/PM10	PM 2.5
0	38	01-02-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	11.0	17.0	55.0	NaN
1	38	01-07-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	13.0	17.0	45.0	NaN
2	38	21-01-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	12.0	18.0	50.0	NaN
3	38	23-01-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	15.0	16.0	46.0	NaN
4	38	28-01-14	Tamil Nadu	Chennai	Kathivakkam, Municipal Kalyana Mandapam, Chennai	Tamilnadu State Pollution Control Board	Industrial Area	13.0	14.0	42.0	NaN

The code provided reads a CSV file named "cpcb_dly_aq_tamil_nadu-2014 (1).csv" into a pandas DataFrame and then displays the first few rows of the DataFrame using the `head()` method. Here's the explanation of the code and its output:

1. Reading the CSV File into a DataFrame:

```python
data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")
```

This line reads the data from the CSV file and stores it in a pandas DataFrame called `data`. The data is now structured in tabular form, making it suitable for analysis and manipulation.

2. Displaying the First Few Rows of the DataFrame:

```python
data.head()
```

This line calls the `head()` method on the `data` DataFrame. The `head()` method is used to display the first few rows of the DataFrame. By default, it shows the top 5 rows.

The output you provided displays the first few rows of the dataset, showing the data in a tabular format with columns such as 'Stn Code,' 'Sampling Date,' 'State,' 'City/Town/Village/Area,' 'Location of Monitoring Station,' 'Agency,' 'Type of Location,' 'SO2,' 'NO2,' 'RSPM/PM10,' and 'PM 2.5.' Each row represents a data entry or observation for air quality in Tamil Nadu. The data includes information such as dates, locations, pollutant levels (SO2, NO2, RSPM/PM10), and PM 2.5, which appears to have missing values (NaN) in the displayed sample.

Display the last few rows of the DataFrame

```
data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")
data.tail()
```

Out[ ]:

	Stn Code	Sampling Date	State	City/Town/Village/Area	Location of Monitoring Station	Agency	Type of Location	SO2	NO2	RSPM/PM10	PM 2.5
2874	773	12-03-14	Tamil Nadu	Trichy	Central Bus Stand, Trichy	Tamilnadu State Pollution Control Board	Residential, Rural and other Areas	15.0	18.0	102.0	NaN
2875	773	12-10-14	Tamil Nadu	Trichy	Central Bus Stand, Trichy	Tamilnadu State Pollution Control Board	Residential, Rural and other Areas	12.0	14.0	91.0	NaN
2876	773	17-12-14	Tamil Nadu	Trichy	Central Bus Stand, Trichy	Tamilnadu State Pollution Control Board	Residential, Rural and other Areas	19.0	22.0	100.0	NaN
2877	773	24-12-14	Tamil Nadu	Trichy	Central Bus Stand, Trichy	Tamilnadu State Pollution Control Board	Residential, Rural and other Areas	15.0	17.0	95.0	NaN
2878	773	31-12-14	Tamil Nadu	Trichy	Central Bus Stand, Trichy	Tamilnadu State Pollution Control Board	Residential, Rural and other Areas	14.0	16.0	94.0	NaN

The provided code reads a CSV file named "cpcb_dly_aq_tamil_nadu-2014 (1).csv" into a pandas DataFrame and then displays the last few rows of the DataFrame using the `tail()` method. Here's the explanation of the code and its output:

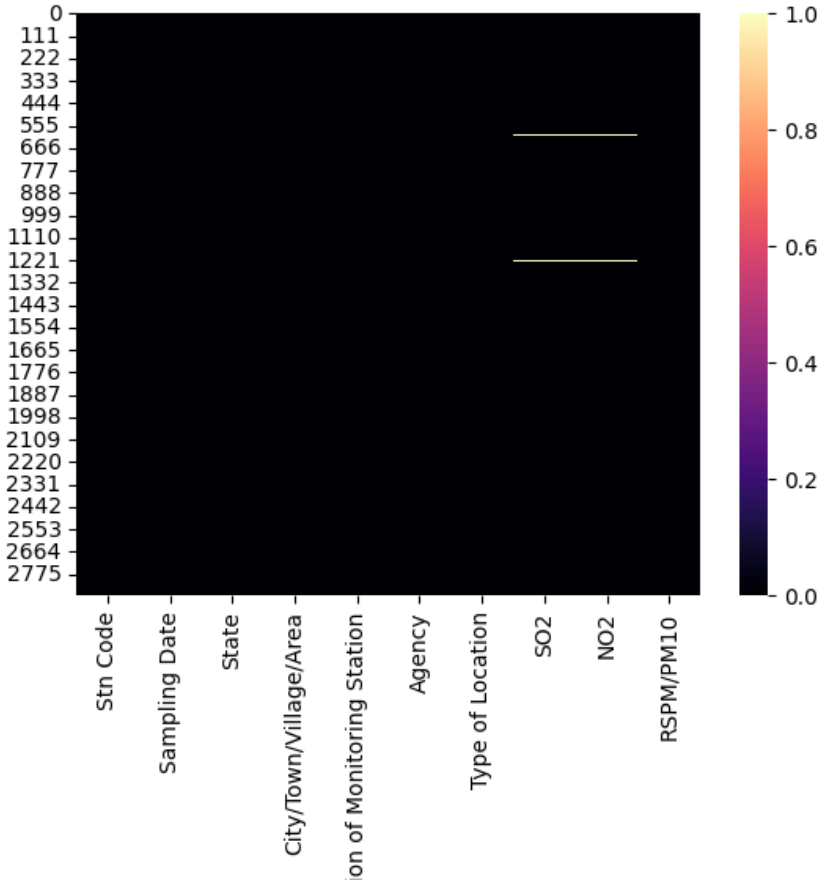
1. Reading the CSV File into a DataFrame:

   ```python
   data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")
   ```

   This line reads the data from the CSV file and stores it in a pandas DataFrame called `data`. The data is now structured in tabular form, ready for analysis and manipulation.

2. Displaying the Last Few Rows of the DataFrame:

   ```python
   data.tail()
   ```

```

This line calls the `tail()` method on the `data` DataFrame. The `tail()` method is used to display the last few rows of the DataFrame. By default, it shows the bottom 5 rows.

The output you provided displays the last few rows of the dataset, showing the data in a tabular format with columns such as 'Stn Code,' 'Sampling Date,' 'State,' 'City/Town/Village/Area,' 'Location of Monitoring Station,' 'Agency,' 'Type of Location,' 'SO2,' 'NO2,' 'RSPM/PM10,' and 'PM 2.5.' Each row represents a data entry or observation for air quality in Tamil Nadu. In the displayed sample, the last few rows are associated with monitoring in Trichy. Just like in the initial data display, it appears that the 'PM 2.5' column contains missing values (NaN) in this sample.

**P.M 2.5 contains NAN value (which is null value)**

**this can be removed by drop function**

```
newdata = data.drop('PM 2.5', axis=1)
sns.heatmap(newdata.isnull(),cmap='magma', cbar='false')
```


```

The provided code is manipulating the DataFrame to remove the 'PM 2.5' column (which contains NaN values), and then it creates a heatmap using the Seaborn library to visualize the presence of missing (null) values in the modified DataFrame. Here's a step-by-step explanation of the code:

1. Removing the 'PM 2.5' Column:
   ```python
   newdata = data.drop('PM 2.5', axis=1)
   ```
   This line uses the `drop` method to remove the 'PM 2.5' column from the original DataFrame `data`. The `axis=1` argument indicates that you want to drop a column. After executing this line, the `newdata` DataFrame will contain all columns from the original DataFrame except for 'PM 2.5'.

2. Creating a Heatmap to Visualize Missing Values:
   ```python
   sns.heatmap(newdata.isnull(), cmap='magma', cbar=False)
   ```
   This line uses Seaborn's `heatmap` function to create a heatmap visualization of missing values in the `newdata` DataFrame.

   - `newdata.isnull()` creates a DataFrame of the same shape as `newdata` but with Boolean values (True for missing values, False for non-missing values). In this case, it will be True for cells where data is missing (NaN) and False for cells with valid data.
   - `cmap='magma'` sets the color map for the heatmap to 'magma,' which is a color scheme that visually represents missing values as distinct colors.
   - `cbar=False` specifies not to show a color bar on the side of the heatmap. The color bar typically indicates the mapping of colors to values, but in this case, it's not needed for the visualization of missing values.

The resulting heatmap will have cells colored differently to represent missing values, making it easy to identify where the missing data points are in the `newdata` DataFrame. This visualization can be helpful for data quality assessment and deciding how to handle missing values in further analysis or data preprocessing.

by using heatmap we have removed the null values from the dataset

calculating the average of SO2,NO2 and RSPM/PM10

```
# Calculate the mean for each column
average_values = newdata.mean()

# Print the average values
print(average_values)
```

```
Stn Code     475.750261
SO2           11.503138
NO2           22.136776
```

RSPM/PM10 62.494261
dtype: float64

```
import pandas as pd

# Load the data
data = pd.read_csv("cpcb_dly_aq_tamil_nadu-2014 (1).csv")

# Calculate averages
station_avg = data.groupby('State')[['SO2', 'NO2', 'RSPM/PM10']].mean()
city_avg = data.groupby('City/Town/Village/Area')[['SO2', 'NO2', 'RSPM/PM10']].mean()

# Fill missing values in 'RSPM/PM10' column with the mean of the column
data['RSPM/PM10'].fillna(data['RSPM/PM10'].mean(), inplace=True)

location_avg = data.groupby('Location of Monitoring Station')[['SO2', 'NO2', 'RSPM/PM10']].mean()

print("State Average:")
print(station_avg)

print("\nCity/Town Average:")
print(city_avg)

print("\nLocation Average:")
print(location_avg)
```

Station Average:

State	SO2	NO2	RSPM/PM10
Tamil Nadu	11.503138	22.136776	62.494261

City/Town Average:

City/Town/Village/Area	SO2	NO2	RSPM/PM10
Chennai	13.014042	22.088442	58.998000
Coimbatore	4.541096	25.325342	49.217241
Cuddalore	8.965986	19.710884	61.881757
Madurai	13.319728	25.768707	45.724490
Mettur	8.429268	23.185366	52.721951
Salem	8.114504	28.664122	62.954198
Thoothukudi	12.989691	18.512027	83.458904
Trichy	15.293956	18.695055	85.054496

Location Average:

Location of Monitoring Station :	SO2	NO2
AVM Jewellery Building, Tuticorin	9.302083	12.697917
Adyar, Chennai	13.252174	18.965217
Anna Nagar, Chennai	13.873874	20.754545
Bishop Heber College, Tirchy	11.800000	14.942857
Central Bus Stand, Trichy	18.013333	21.506667
District Environmental Engineer Office, Imperia...	8.101010	19.151515
Distt. Collector's Office, Coimbatore	4.554348	25.793478
Eachangadu Villagae	11.916667	22.395833
Fenner (I) Ltd. Employees Assiciation Building ...	13.643564	27.198020
Fisheries College, Tuticorin	14.526882	20.204301
Gandhi Market, Trichy	17.148649	20.797297
Golden Rock, Trichy	12.014085	15.000000
Govt. High School, Manali, Chennai.	13.043011	15.408602
Highway (Project -I) Building, Madurai	11.947917	24.458333
Kathivakkam, Municipal Kalyana Mandapam, Chennai	12.925532	15.170213
Kilpauk, Chennai	19.232759	27.172414
Kunnathur Chatram East Avani Mollai Street, Mad...	14.340206	25.577320
Madras Medical College, Chennai	7.418605	27.465116
Main Guard Gate, Tirchy	17.135135	20.837838
NEERI, CSIR Campus Chennai	5.931034	23.758621
Poniarajapuram, On the top of DEL, Coimbatore	4.126214	23.019417
Raja Agencies, Tuticorin	15.058824	22.441176
Raman Nagar, Mettur	7.572816	20.407767
SIDCO Industrial Complex, Mettur	9.294118	25.990196
SIDCO Office, Coimbatore	4.969072	27.329897
SIPCOT Industrial Complex, Cuddalore	6.969697	17.666667
Sowdeswari College Building, Salem	8.114504	28.664122
Thiruvottiyur Municipal Office, Chennai	8.360465	28.069767
Thiruvottiyur, Chennai	13.010417	15.583333
Thiyagaraya Nagar, Chennai	18.849558	28.250000

Location of Monitoring Station	RSPM/PM10
AVM Jewellery Building, Tuticorin	70.175258
Adyar, Chennai	57.068966
Anna Nagar, Chennai	72.187500
Bishop Heber College, Tirchy	45.633803
Central Bus Stand, Trichy	120.546667
District Environmental Engineer Office, Imperia...	64.020202
Distt. Collector's Office, Coimbatore	42.972933
Eachangadu Villagae	75.591837

Fenner (I) Ltd. Employees Assiciation Building ...	40.732673
Fisheries College, Tuticorin	85.255319
Gandhi Market, Trichy	101.743243
Golden Rock, Trichy	46.222222
Govt. High School, Manali, Chennai.	44.612903
Highway (Project -I) Building, Madurai	46.427083
Kathivakkam, Municipal Kalyana Mandapam, Chennai	46.851064
Kilpauk, Chennai	88.103448
Kunnathur Chatram East Avani Mollai Street, Mad...	50.226804
Madras Medical College, Chennai	35.837209
Main Guard Gate, Tirchy	107.693333
NEERI, CSIR Campus Chennai	43.678161
Poniarajapuram, On the top of DEL, Coimbatore	48.883495
Raja Agencies, Tuticorin	94.230336
Raman Nagar, Mettur	51.106796
SIDCO Industrial Complex, Mettur	54.352941
SIDCO Office, Coimbatore	55.969072
SIPCOT Industrial Complex, Cuddalore	46.171717
Sowdeswari College Building, Salem	62.954198
Thiruvottiyur Municipal Office, Chennai	34.310345
Thiruvottiyur, Chennai	42.604167
Thiyagaraya Nagar, Chennai	102.327434

In [ ]:

###visualization using charts

The provided code calculates the average values for the 'SO2,' 'NO2,' and 'RSPM/PM10' columns in the dataset and then groups and calculates these averages for different categories, such as 'State,' 'City/Town/Village/Area,' and 'Location of Monitoring Station.' Here's an explanation of the code and its output:

1. Calculate Averages by Grouping:

```python
station_avg = data.groupby('State')[['SO2', 'NO2', 'RSPM/PM10']].mean()
city_avg = data.groupby('City/Town/Village/Area')[['SO2', 'NO2', 'RSPM/PM10']].mean()
location_avg = data.groupby('Location of Monitoring Station')[['SO2', 'NO2', 'RSPM/PM10']].mean()
```

These lines group the data in the original DataFrame 'data' by different categories: 'State,' 'City/Town/Village/Area,' and 'Location of Monitoring Station.' For each group, it calculates the average values for the 'SO2,' 'NO2,' and 'RSPM/PM10' columns.

2. Fill Missing Values:

```python
data['RSPM/PM10'].fillna(data['RSPM/PM10'].mean(), inplace=True)
```

This line fills any missing values in the 'RSPM/PM10' column with the mean of that column. This is a common technique for handling missing data.

3. Print the Averages:

```python
print("State Average:")
print(station_avg)

print("\nCity/Town Average:")
print(city_avg)

print("\nLocation Average:")
print(location_avg)
```

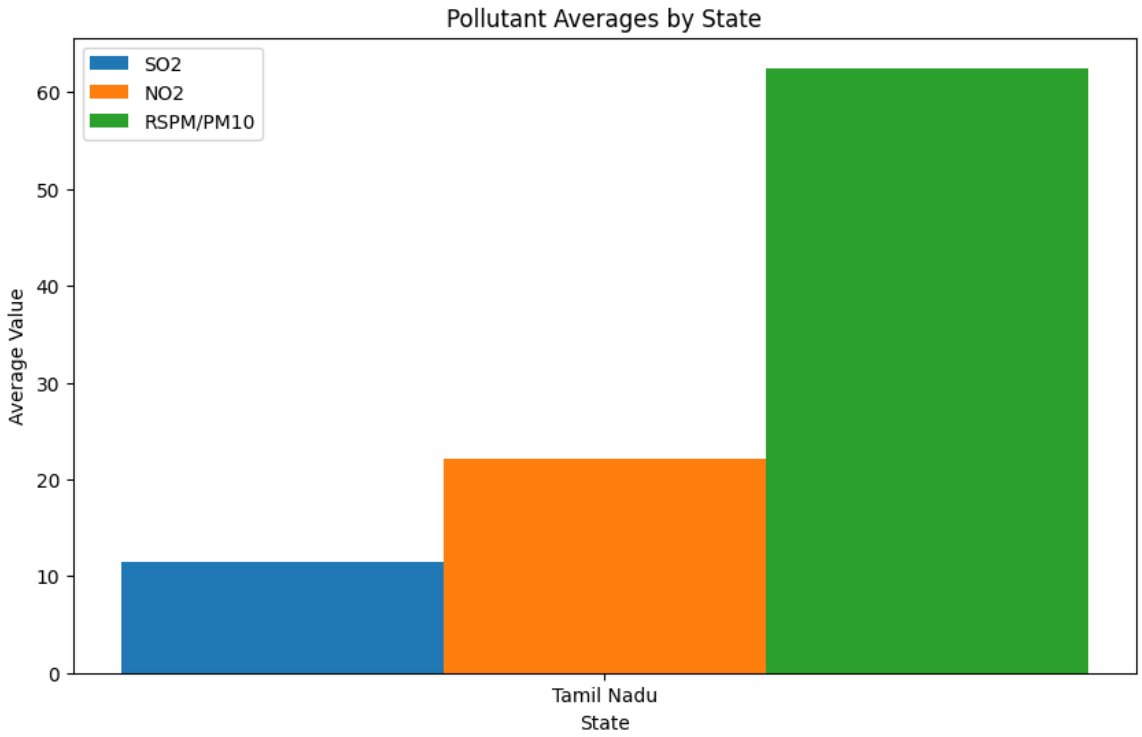
These lines print the calculated averages for the 'SO2,' 'NO2,' and 'RSPM/PM10' columns for different categories, providing an overview of air quality data averages at various levels of aggregation.

Regarding the visualization using charts, the code you provided is focused on data analysis and aggregation. If you want to create charts to visualize this data, you can use libraries like Matplotlib or Seaborn. For example, you can create bar plots or line plots to visualize how pollutant levels vary across different states, cities, or monitoring locations. You can customize the type of chart and the variables you want to visualize based on your specific data exploration needs.

visualization for State-wise average between SO2, NO2 , PSPM/PM10

In [ ]:

```
import numpy as np

x = np.arange(len(station_avg.index))  # the label locations
width = 0.25  # the width of the bars

fig, ax = plt.subplots(figsize=(10, 6))

rects1 = ax.bar(x - width, station_avg['SO2'], width, label='SO2')
rects2 = ax.bar(x, station_avg['NO2'], width, label='NO2')
rects3 = ax.bar(x + width, station_avg['RSPM/PM10'], width, label='RSPM/PM10')

ax.set_xlabel('State')
ax.set_ylabel('Average Value')
ax.set_title('Pollutant Averages by State')
ax.set_xticks(x)
ax.set_xticklabels(station_avg.index)
ax.legend()

plt.show()
```



The provided code is used to create a bar chart to visualize the state-wise average values for the 'SO2,' 'NO2,' and 'RSPM/PM10' pollutants. Here's a step-by-step explanation of the code:

1. Import Necessary Libraries:

```python
import numpy as np
```

Import the `numpy` library to work with arrays and numerical data.

2. Define Variables for Bar Chart:

```python
x = np.arange(len(station_avg.index))  # the label locations
width = 0.25  # the width of the bars
```

- `x` is an array of label locations for the x-axis. It is set to the number of unique states in the `station_avg` DataFrame.

- `width` determines the width of the bars in the bar chart.

3. Create a Figure and Axes:

```python
fig, ax = plt.subplots(figsize=(10, 6))
```

- `fig` represents the entire figure, and `ax` represents the axes within the figure. We specify a figure size of 10 units in width and 6 units in height for the plot.

4. Create Bar Plots for Each Pollutant:

```python
rects1 = ax.bar(x - width, station_avg['SO2'], width, label='SO2')
rects2 = ax.bar(x, station_avg['NO2'], width, label='NO2')
rects3 = ax.bar(x + width, station_avg['RSPM/PM10'], width, label='RSPM/PM10')
```

- `ax.bar` is used to create bar plots for 'SO2,' 'NO2,' and 'RSPM/PM10' on the same plot.

- The `x - width`, `x`, and `x + width` offsets are used to position the bars for each pollutant side by side.

5. Set Labels and Legend:

```python
ax.set_xlabel('State')
ax.set_ylabel('Average Value')
ax.set_title('Pollutant Averages by State')
ax.set_xticks(x)
ax.set_xticklabels(station_avg.index)
ax.legend()
```

- `ax.set_xlabel`, `ax.set_ylabel`, and `ax.set_title` set the labels and title for the x-axis, y-axis, and the chart title, respectively.

- `ax.set_xticks` and `ax.set_xticklabels` specify the tick positions and labels for the x-axis, corresponding to the states.

- `ax.legend()` adds a legend to the chart, displaying labels for each pollutant.

6. Display the Plot:

```python
plt.show()
```

This line shows the bar chart with the specified labels and legend.

The resulting bar chart displays state-wise average values for 'SO2,' 'NO2,' and 'RSPM/PM10' pollutants, making it easy to compare air quality measures across different states.

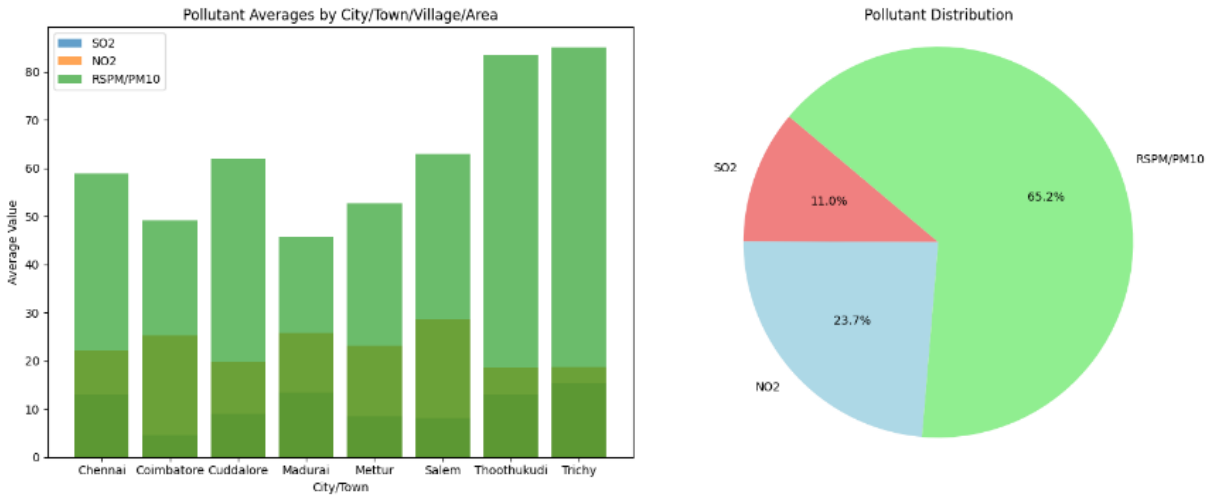
visualization for city/town/village/area wise average between SO2, NO2 , PSPM/PM10

In [ ]:

```
# Assuming city_avg is a DataFrame with 'SO2', 'NO2', 'RSPM/PM10' averages for each city

# Calculate the total for each pollutant
total_so2 = city_avg['SO2'].sum()
total_no2 = city_avg['NO2'].sum()
total_rspm_pm10 = city_avg['RSPM/PM10'].sum()

# Create a figure with two subplots (bar chart and pie chart)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Bar Chart
ax1.bar(city_avg.index, city_avg['SO2'], label='SO2', alpha=0.7)
ax1.bar(city_avg.index, city_avg['NO2'], label='NO2', alpha=0.7)
ax1.bar(city_avg.index, city_avg['RSPM/PM10'], label='RSPM/PM10', alpha=0.7)
ax1.set_xlabel('City/Town')
ax1.set_ylabel('Average Value')
ax1.set_title('Pollutant Averages by City/Town/Village/Area')
ax1.legend()

# Pie Chart
labels = ['SO2', 'NO2', 'RSPM/PM10']
sizes = [total_so2, total_no2, total_rspm_pm10]
colors = ['lightcoral', 'lightblue', 'lightgreen']
ax2.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
ax2.set_title('Pollutant Distribution')
```

ax2.axis('equal') *# Equal aspect ratio ensures that pie is drawn as a circle.*

plt.tight_layout()
plt.show()



The provided code performs data visualization using both a bar chart and a pie chart to represent the city/town/village/area-wise average values for the 'SO2,' 'NO2,' and 'RSPM/PM10' pollutants. Here's a step-by-step explanation of the code:

1. Calculate the Total for Each Pollutant:

   ```python

   total_so2 = city_avg['SO2'].sum()

   total_no2 = city_avg['NO2'].sum()

   total_rspm_pm10 = city_avg['RSPM/PM10'].sum()

   ```

   These lines calculate the total sum of each pollutant ('SO2,' 'NO2,' and 'RSPM/PM10') across all cities, which will be used to create the pie chart.

2. Create a Figure with Two Subplots:

```python
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
```

- `fig` represents the entire figure, and `(ax1, ax2)` are two axes objects. `1, 2` specifies that there are two subplots arranged horizontally.

- `figsize` sets the size of the overall figure to 15 units in width and 6 units in height.

3. Bar Chart (ax1):

```python
ax1.bar(city_avg.index, city_avg['SO2'], label='SO2', alpha=0.7)
ax1.bar(city_avg.index, city_avg['NO2'], label='NO2', alpha=0.7)
ax1.bar(city_avg.index, city_avg['RSPM/PM10'], label='RSPM/PM10', alpha=0.7)
```

- In the first subplot (ax1), the code creates bar plots for 'SO2,' 'NO2,' and 'RSPM/PM10' for each city, overlaying them with an alpha of 0.7 to make the bars somewhat transparent for better visualization.

- Labels, title, and legend are set to provide context for the chart.

4. Pie Chart (ax2):

```python
labels = ['SO2', 'NO2', 'RSPM/PM10']
sizes = [total_so2, total_no2, total_rspm_pm10]
colors = ['lightcoral', 'lightblue', 'lightgreen']
ax2.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
```

- In the second subplot (ax2), a pie chart is created. It represents the distribution of the three pollutants ('SO2,' 'NO2,' and 'RSPM/PM10') by showing the percentage of each pollutant's contribution.

- Labels, sizes, colors, and autopct (percentage format) are specified for the pie chart. `startangle` determines the starting angle for the first slice.

- The aspect ratio is set to 'equal' to ensure that the pie chart appears as a circle.

5. Adjust Layout and Show the Plot:

```python

plt.tight_layout()

plt.show()

```

- `plt.tight_layout()` adjusts the spacing between subplots for a cleaner appearance.

- `plt.show()` displays the entire figure with both the bar chart and the pie chart.

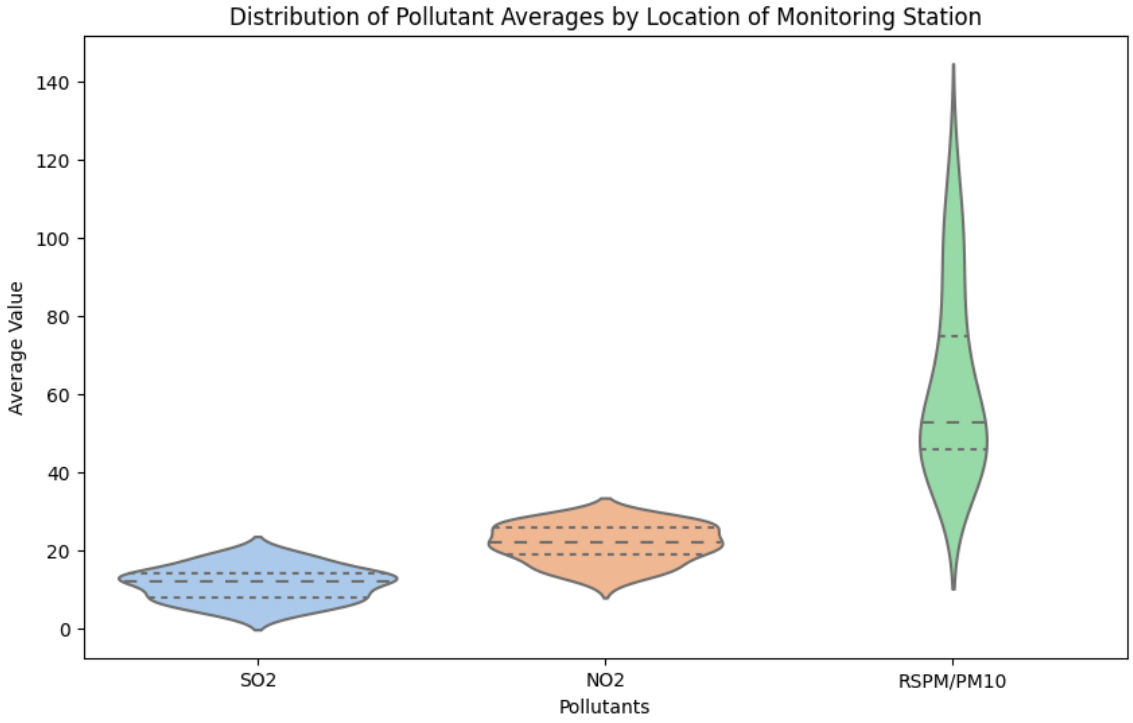
The resulting visualization provides a city-wise comparison of pollutant averages using the bar chart and a pie chart that shows the distribution of pollutant contributions.

visualization for Lcation wise average between SO2, NO2 , PSPM/PM10

In [ ]:

```
import seaborn as sns

# Assuming location_avg is a DataFrame with 'SO2', 'NO2', 'RSPM/PM10' averages for each location

# Create a figure and axis
fig, ax = plt.subplots(figsize=(10, 6))

# Create a Violin Plot
sns.violinplot(data=location_avg[['SO2', 'NO2', 'RSPM/PM10']], inner="quart", palette="pastel")

# Set labels and title
ax.set_xlabel('Pollutants')
ax.set_ylabel('Average Value')
ax.set_title('Distribution of Pollutant Averages by Location of Monitoring Station')
```

 # Display the plot
plt.show()



The provided code uses a violin plot to visualize the distribution of 'SO2,' 'NO2,' and 'RSPM/PM10' pollutant averages by the location of monitoring station. Here's a step-by-step explanation of the code:

1. Import the Required Libraries:

```python

import seaborn as sns

```

Import the Seaborn library for creating statistical data visualizations, and it's often used for drawing plots like violin plots.

2. Create a Figure and Axis:

```python
fig, ax = plt.subplots(figsize=(10, 6))
```

- `fig` represents the entire figure, and `ax` represents the axes within the figure. We specify a figure size of 10 units in width and 6 units in height for the plot.

3. Create a Violin Plot:

```python
sns.violinplot(data=location_avg[['SO2', 'NO2', 'RSPM/PM10']], inner="quart", palette="pastel")
```

- `sns.violinplot` is used to create a violin plot. It visualizes the distribution of data, showing the probability density of the data at different values. In this case, it displays the distribution of 'SO2,' 'NO2,' and 'RSPM/PM10' data for different monitoring locations.

- `data=location_avg[['SO2', 'NO2', 'RSPM/PM10']]` specifies the data to be used for the plot, containing the pollutant columns.

- `inner="quart"` indicates that the inner part of the violin plot should display quartiles.

- `palette="pastel"` sets the color palette for the plot to "pastel" colors.

4. Set Labels and Title:

```python
ax.set_xlabel('Pollutants')
ax.set_ylabel('Average Value')
ax.set_title('Distribution of Pollutant Averages by Location of Monitoring Station')
```

- `ax.set_xlabel` and `ax.set_ylabel` set the labels for the x-axis and y-axis, respectively.

- `ax.set_title` sets the title for the plot to describe what is being visualized.

5. Display the Plot:

```python

plt.show()

```

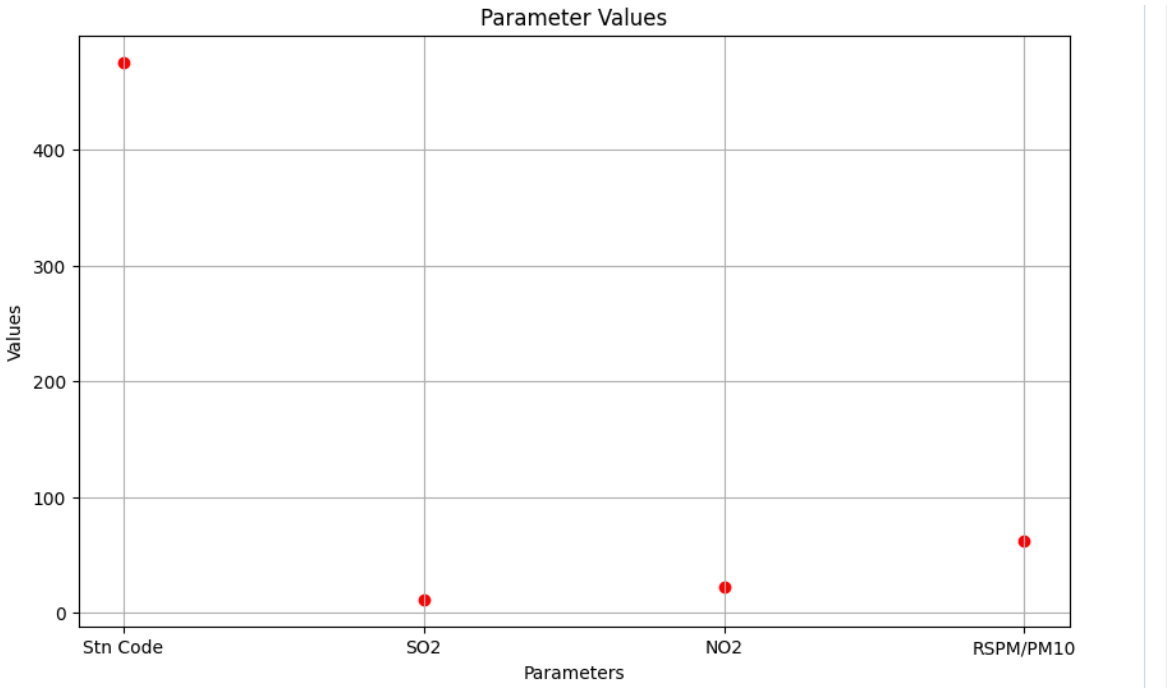
This line displays the violin plot, showing the distribution of pollutant averages by location.

The resulting visualization allows you to see how the average values of 'SO2,' 'NO2,' and 'RSPM/PM10' pollutants are distributed at various monitoring locations. Violin plots are effective for visualizing data distributions, providing insight into the spread and shape of the data for each pollutant.

the average of SO2,NO2 and RSPM/PM10

In [ ]:

```
import matplotlib.pyplot as plt

# Data
parameters = ['Stn Code', 'SO2', 'NO2', 'RSPM/PM10']
values = [475.750261, 11.503138, 22.136776, 62.494261]

# Create a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(parameters, values, color='red', marker='o')
plt.xlabel('Parameters')
plt.ylabel('Values')
plt.title('Parameter Values')
plt.grid(True)
plt.show()
```



The provided code creates a scatter plot to visualize the average values of different parameters ('Stn Code,' 'SO2,' 'NO2,' and 'RSPM/PM10'). Here's a step-by-step explanation of the code:

1. Import the Required Library:
   ```python
   import matplotlib.pyplot as plt
   ```
   Import Matplotlib, a widely used library for creating various types of plots and visualizations.

2. Define Data:
   ```python
   parameters = ['Stn Code', 'SO2', 'NO2', 'RSPM/PM10']
   values = [475.750261, 11.503138, 22.136776, 62.494261]
   ```
   - `parameters` is a list of parameter names or labels.
   - `values` is a list of corresponding average values for each parameter.

3. Create a Scatter Plot:
   ```python
   plt.figure(figsize=(10, 6))
   plt.scatter(parameters, values, color='red', marker='o')
   ```
   - `plt.figure(figsize=(10, 6))` creates a figure with a specified size of 10 units in width and 6 units in height.
   - `plt.scatter(parameters, values, color='red', marker='o')` generates the scatter plot. It plots the values on the y-axis against the parameters on the x-axis.
     - `color='red'` sets the color of the data points to red.
     - `marker='o'` specifies circular markers for the data points.

4. Set Labels and Title:
   ```python
   plt.xlabel('Parameters')
   plt.ylabel('Values')
   plt.title('Parameter Values')
   ```
   - `plt.xlabel` sets the label for the x-axis to 'Parameters.'
   - `plt.ylabel` sets the label for the y-axis to 'Values.'
   - `plt.title` sets the title of the plot to 'Parameter Values.'

5. Enable Grid:
   ```python
   plt.grid(True)
   ```
   - `plt.grid(True)` adds gridlines to the plot, making it easier to read and interpret.

6. Display the Plot:
   ```python
   plt.show()
   ```
   This line displays the scatter plot with the specified parameters, values, labels, title, and gridlines.

The resulting scatter plot visually represents the average values of the given parameters, allowing you to compare and understand their magnitudes in a clear and concise manner.

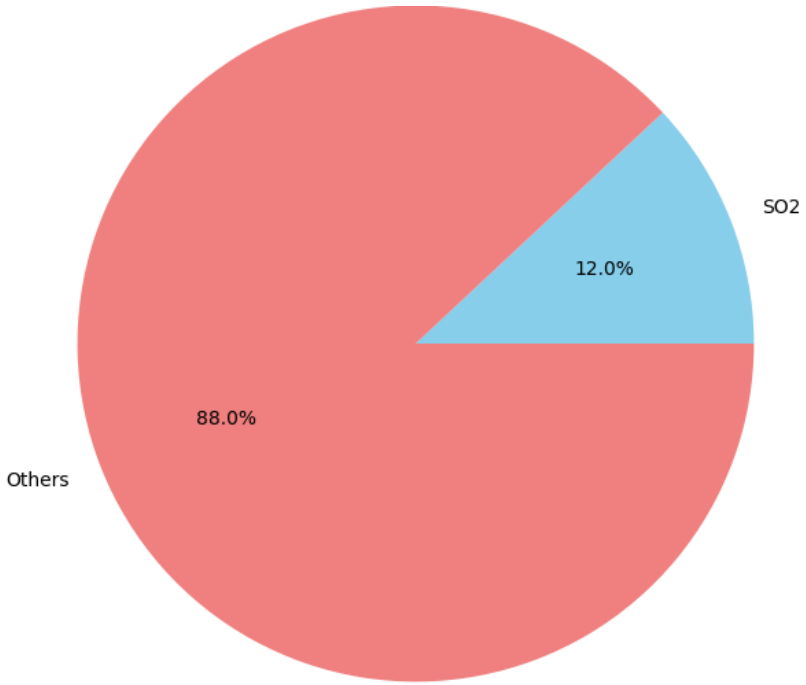
separate charts for SO2,NO2 & RSPM/PM10 in Tamil Nadu using pie chart.

In [ ]:

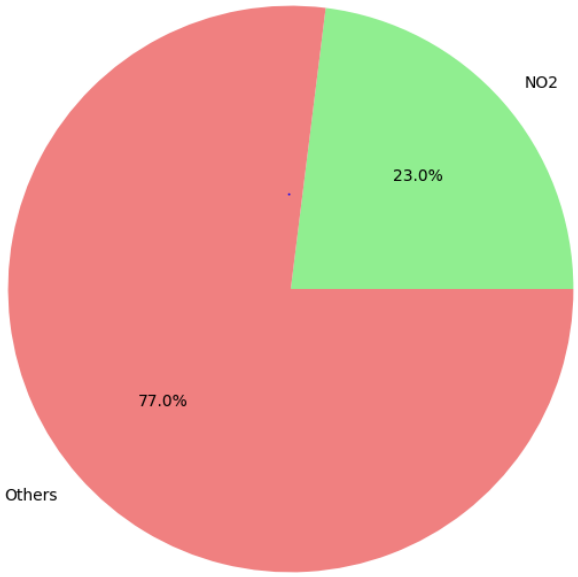
```
import matplotlib.pyplot as plt

# Data
parameters = ['SO2', 'NO2', 'RSPM/PM10']
values = [11.503138, 22.136776, 62.494261]

# Create a pie chart for SO2
plt.figure(figsize=(8, 8))
plt.pie([values[0], sum(values[1:])], labels=['SO2', 'Others'], autopct='%1.1f%%', colors=['skyblue', 'lightcoral'])
plt.title('SO2 Pollution Levels in Tamil Nadu')
plt.show()

# Create a pie chart for NO2
plt.figure(figsize=(8, 8))
```

```
plt.pie([values[1], sum(values[:1] + values[2:])], labels=['NO2', 'Others'], autopct='%1.1f%%', colors=['lightgreen', 'lightcoral'])
plt.title('NO2 Pollution Levels in Tamil Nadu')
plt.show()

# Create a pie chart for RSPM/PM10
plt.figure(figsize=(8, 8))
plt.pie([values[2], sum(values[:2])], labels=['RSPM/PM10', 'Others'], autopct='%1.1f%%', colors=['lightblue', 'lightcoral'])
plt.title('RSPM/PM10 Pollution Levels in Tamil Nadu')
plt.show()
```

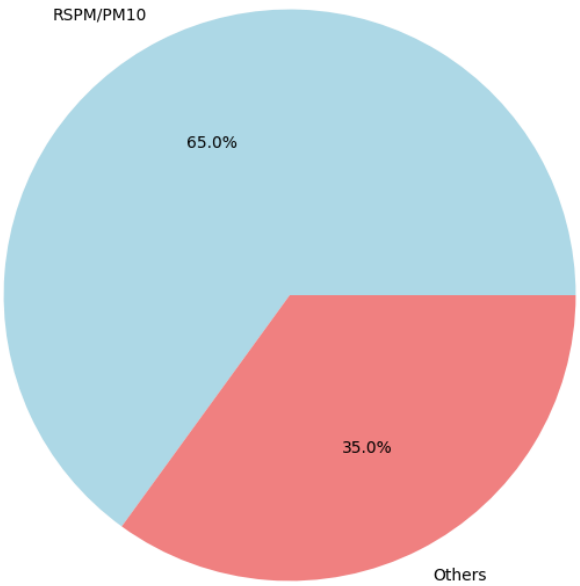
SO2 pollution levels in Tamil Nadu



NO2 Pollution Levels in Tamil Nadu



RSPM\PM10 Pollution Levels in Tamil Nadu



providing visualization by displaying lowest to highest pollution of SO2,NO2 & RSPM/PM10 (separate) using barchart with line chart under the area - City/Town/Village/Area

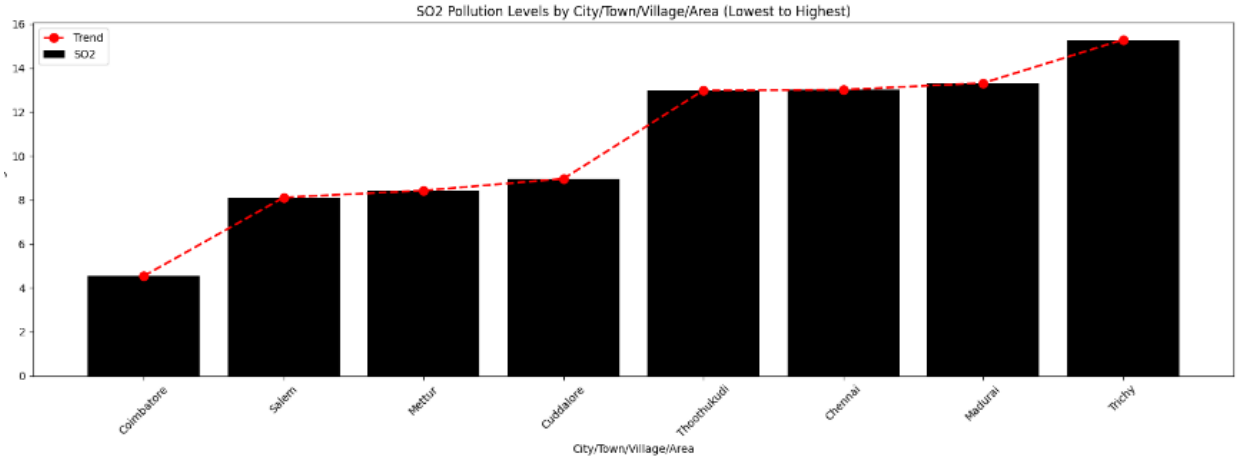
In [ ]:

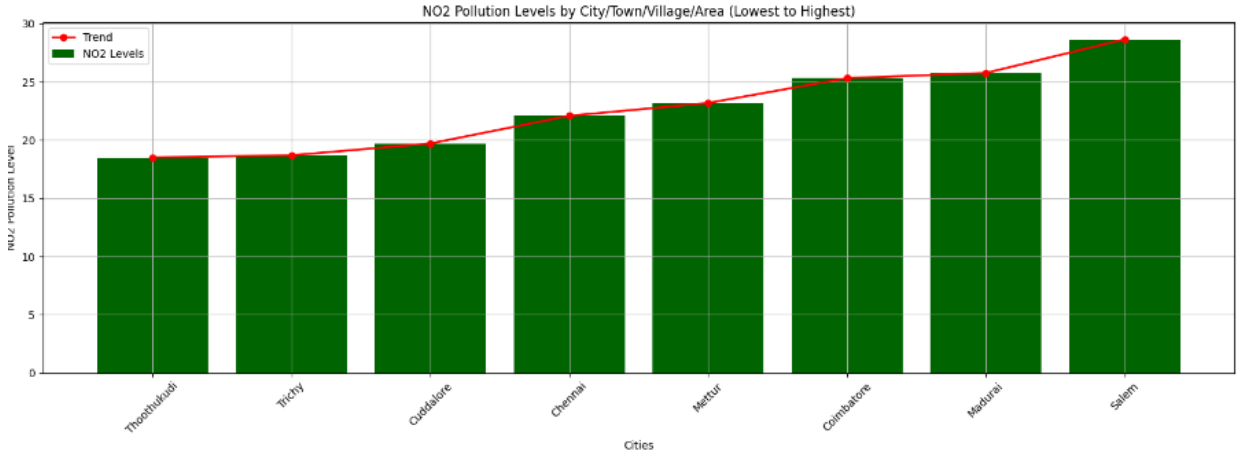
```
import matplotlib.pyplot as plt

# Data
cities = ['Coimbatore', 'Mettur', 'Salem', 'Cuddalore', 'Chennai', 'Thoothukudi', 'Madurai', 'Trichy']
so2_values = [4.541096, 8.429268, 8.114504, 8.965986, 13.014042, 12.989691, 13.319728, 15.293956]

# Sorting data from lowest to highest SO2 levels
sorted_cities = [x for _, x in sorted(zip(so2_values, cities))]
sorted_so2_values = sorted(so2_values)

# Create a bar chart
plt.figure(figsize=(16, 6))
plt.bar(sorted_cities, sorted_so2_values, color='black', label='SO2')
plt.xlabel('City/Town/Village/Area')
plt.ylabel('SO2 Average')
plt.title('SO2 Pollution Levels by City/Town/Village/Area (Lowest to Highest)')
plt.xticks(rotation=45)
plt.legend()

# Create a line chart for trend
plt.plot(sorted_cities, sorted_so2_values, marker='o', color='red', linestyle='dashed', linewidth=2, markersize=8, label='Trend')

# Show legend
plt.legend()

# Show plot
plt.tight_layout()
plt.show()
```

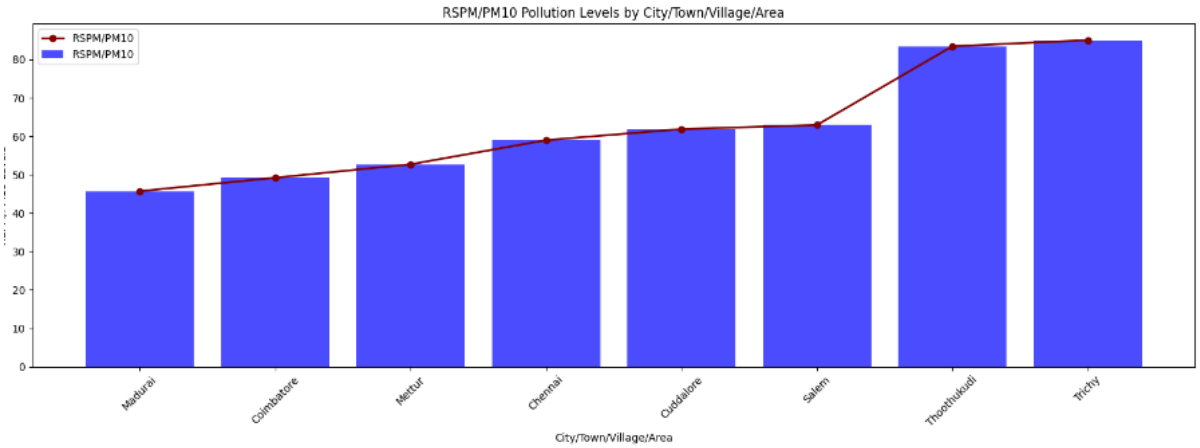
```
import matplotlib.pyplot as plt

# Data
cities = ['Chennai', 'Coimbatore', 'Cuddalore', 'Madurai', 'Mettur', 'Salem', 'Thoothukudi', 'Trichy']
no2_values = [22.088442, 25.325342, 19.710884, 25.768707, 23.185366, 28.664122, 18.512027, 18.695055]
```

```
# Sort cities and corresponding NO2 values based on NO2 levels (lowest to highest)
sorted_data = sorted(zip(cities, no2_values), key=lambda x: x[1])

cities, no2_values = zip(*sorted_data)  # Unzip the sorted data

# Create a bar chart for NO2
plt.figure(figsize=(16, 6))
plt.bar(cities, no2_values, color='darkgreen', label='NO2 Levels')
plt.xlabel('Cities')
plt.ylabel('NO2 Pollution Level')
plt.title('NO2 Pollution Levels by City/Town/Village/Area (Lowest to Highest)')
plt.xticks(rotation=45)
plt.grid(True)

# Add a line chart to emphasize the trend
plt.plot(cities, no2_values, marker='o', color='red', linestyle='-', linewidth=2, label='Trend')
plt.legend()

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt

# Data
cities = ['Chennai', 'Coimbatore', 'Cuddalore', 'Madurai', 'Mettur', 'Salem', 'Thoothukudi', 'Trichy']
rspm_pm10_levels = [58.998, 49.217241, 61.881757, 45.724490, 52.721951, 62.954198, 83.458904, 85.054496]

# Sort the data from lowest to highest RSPM/PM10 levels
sorted_indices = sorted(range(len(rspm_pm10_levels)), key=lambda i: rspm_pm10_levels[i])
cities_sorted = [cities[i] for i in sorted_indices]
rspm_pm10_levels_sorted = [rspm_pm10_levels[i] for i in sorted_indices]

# Create a figure
plt.figure(figsize=(16, 6))

# Bar chart
plt.bar(cities_sorted, rspm_pm10_levels_sorted, color='blue', alpha=0.7, label='RSPM/PM10')

# Line chart (for better visualization)
plt.plot(cities_sorted, rspm_pm10_levels_sorted, marker='o', color='maroon', label='RSPM/PM10', linewidth=2)

# Labels and title
plt.xlabel('City/Town/Village/Area')
plt.ylabel('RSPM/PM10 Levels')
```

```
plt.title('RSPM/PM10 Pollution Levels by City/Town/Village/Area')
plt.xticks(rotation=45)
plt.legend()

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







The provided code creates separate pie charts to visualize the pollution levels of 'SO2,' 'NO2,' and 'RSPM/PM10' in Tamil Nadu. Here's a step-by-step explanation of the code:

1. Import the Required Library:

   ```python

   import matplotlib.pyplot as plt

   ```

   Import Matplotlib for creating various types of plots, including pie charts.

2. Define Data:

   ```python

   parameters = ['SO2', 'NO2', 'RSPM/PM10']

   values = [11.503138, 22.136776, 62.494261]

   ```

   - `parameters` is a list of parameter names or labels.

   - `values` is a list of corresponding average values for each parameter.

3. Create Separate Pie Charts for Each Parameter:

a. Pie Chart for SO2:

```python

plt.figure(figsize=(8, 8))

plt.pie([values[0], sum(values[1:])], labels=['SO2', 'Others'], autopct='%1.1f%%', colors=['skyblue', 'lightcoral'])

plt.title('SO2 Pollution Levels in Tamil Nadu')

plt.show()

```

- `plt.figure(figsize=(8, 8))` creates a figure with an 8x8-inch size.

- `plt.pie` creates a pie chart. `[values[0], sum(values[1:])]` represents the value for 'SO2' and the sum of values for other pollutants.

- `labels=['SO2', 'Others']` labels the chart with 'SO2' and 'Others,' where 'Others' represents the combined values of 'NO2' and 'RSPM/PM10.'

- `autopct='%1.1f%%'` displays the percentage values on the pie chart.

- `colors=['skyblue', 'lightcoral']` sets the colors for the 'SO2' and 'Others' segments.

- `plt.title('SO2 Pollution Levels in Tamil Nadu')` sets the title for the pie chart.

b. Pie Chart for NO2:

```python

plt.figure(figsize=(8, 8))

plt.pie([values[1], sum(values[:1] + values[2:])], labels=['NO2', 'Others'], autopct='%1.1f%%', colors=['lightgreen', 'lightcoral'])

plt.title('NO2 Pollution Levels in Tamil Nadu')

plt.show()

```

- This block of code is similar to the first pie chart but focuses on 'NO2' and 'Others' pollution levels.

c. Pie Chart for RSPM/PM10:

```python

plt.figure(figsize=(8, 8))

plt.pie([values[2], sum(values[:2])], labels=['RSPM/PM10', 'Others'], autopct='%1.1f%%', colors=['lightblue', 'lightcoral'])

plt.title('RSPM/PM10 Pollution Levels in Tamil Nadu')

plt.show()

```

- This block of code is similar to the first pie chart but focuses on 'RSPM/PM10' and 'Others' pollution levels.

Each set of code creates a separate pie chart, providing a clear visual representation of the percentage distribution of pollution levels for the specified pollutant within Tamil Nadu.

providing visualization by displaying lowest to highest pollution of SO2,NO2 & RSPM/PM10 (separate) using barchart with line chart under the area - Location

SO2,NO2 & RSPM/PM10 Pollution Levels by Location. (here green shows highest and red shows lowest)

In [ ]:

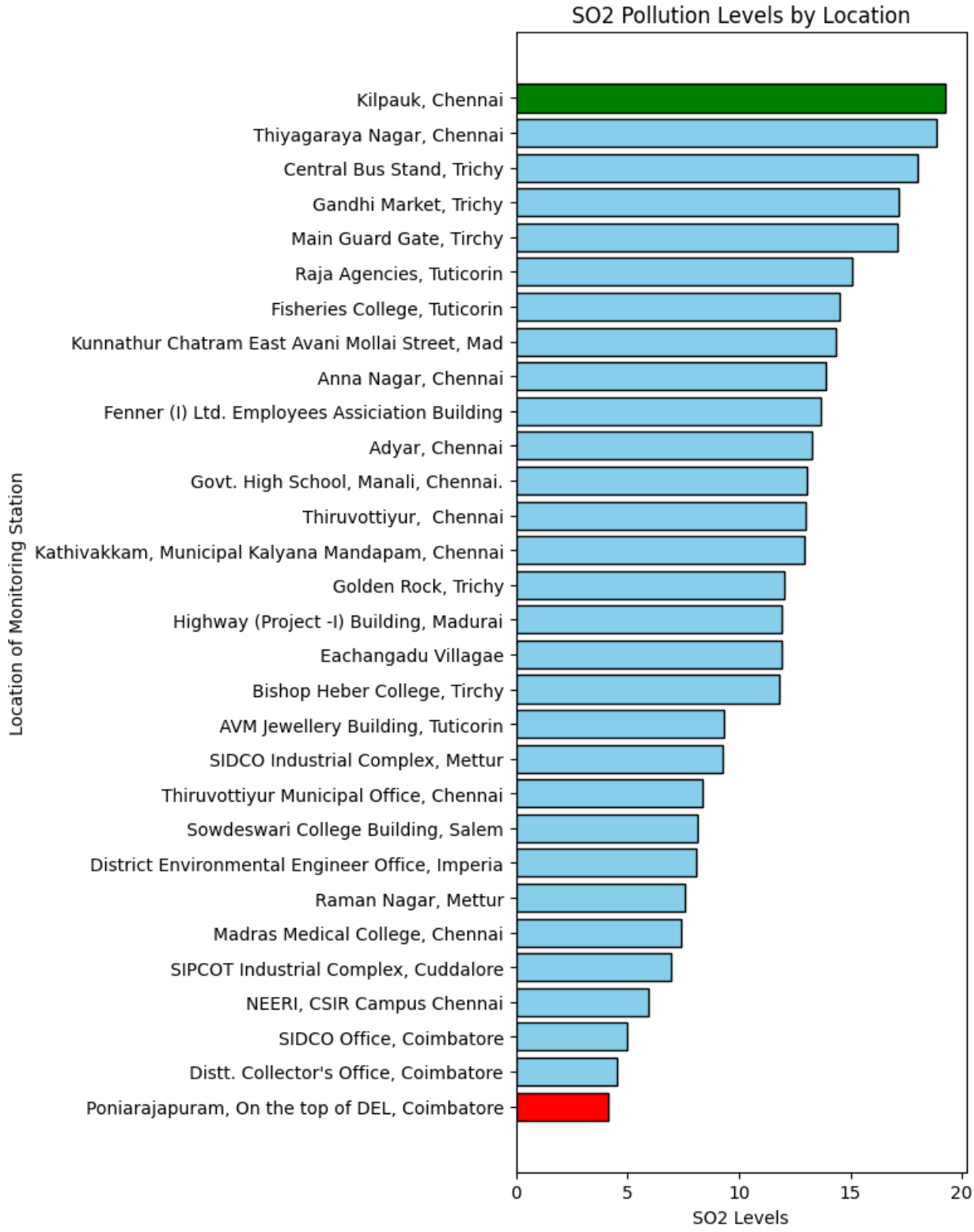
```
import matplotlib.pyplot as plt

# Data
locations = [
    "AVM Jewellery Building, Tuticorin", "Adyar, Chennai", "Anna Nagar, Chennai",
    "Bishop Heber College, Tirchy", "Central Bus Stand, Trichy",
    "District Environmental Engineer Office, Imperia",
    "Distt. Collector's Office, Coimbatore", "Eachangadu Villagae",
    "Fenner (I) Ltd. Employees Assiciation Building", "Fisheries College, Tuticorin",
    "Gandhi Market, Trichy", "Golden Rock, Trichy",
    "Govt. High School, Manali, Chennai.", "Highway (Project -I) Building, Madurai",
    "Kathivakkam, Municipal Kalyana Mandapam, Chennai", "Kilpauk, Chennai",
    "Kunnathur Chatram East Avani Mollai Street, Mad",
    "Madras Medical College, Chennai", "Main Guard Gate, Tirchy",
    "NEERI, CSIR Campus Chennai", "Poniarajapuram, On the top of DEL, Coimbatore",
    "Raja Agencies, Tuticorin", "Raman Nagar, Mettur",
    "SIDCO Industrial Complex, Mettur", "SIDCO Office, Coimbatore",
    "SIPCOT Industrial Complex, Cuddalore",
```

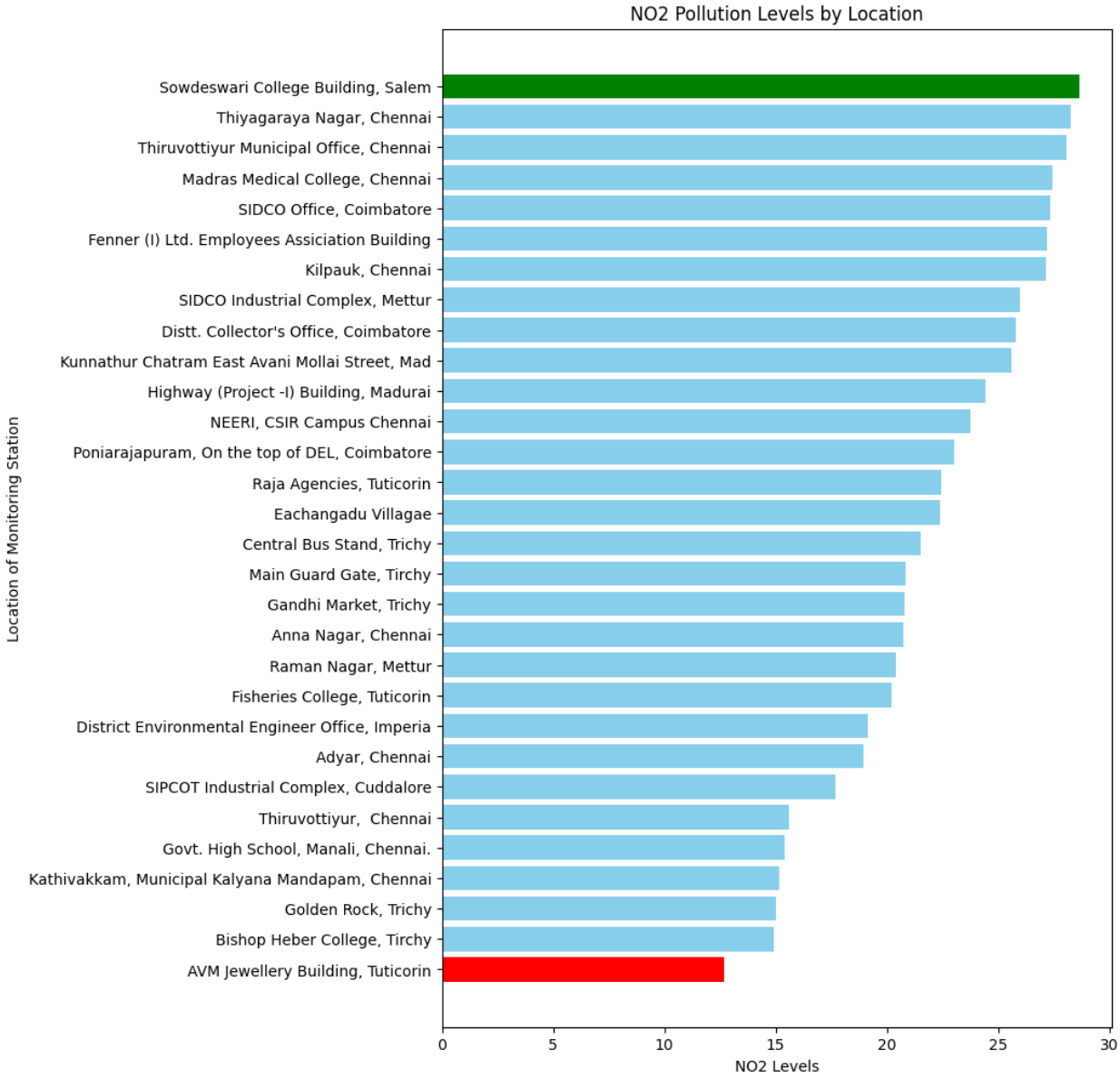
```
    "Sowdeswari College Building, Salem", "Thiruvottiyur Municipal Office, Chennai",
    "Thiruvottiyur,  Chennai", "Thiyagaraya Nagar, Chennai"
]

so2_levels = [
    9.302083, 13.252174, 13.873874, 11.800000, 18.013333, 8.101010, 4.554348,
    11.916667, 13.643564, 14.526882, 17.148649, 12.014085, 13.043011, 11.947917,
    12.925532, 19.232759, 14.340206, 7.418605, 17.135135, 5.931034, 4.126214,
    15.058824, 7.572816, 9.294118, 4.969072, 6.969697, 8.114504, 8.360465,
    13.010417, 18.849558
]

# Sort data from lowest to highest SO2 levels
sorted_indices = sorted(range(len(so2_levels)), key=lambda i: so2_levels[i])
locations_sorted = [locations[i] for i in sorted_indices]
so2_levels_sorted = [so2_levels[i] for i in sorted_indices]

# Define colors for highest and lowest values
colors = ['red' if i == min(so2_levels_sorted) else 'green' if i == max(so2_levels_sorted) else
'skyblue' for i in so2_levels_sorted]

# Create a figure
plt.figure(figsize=(8, 10))

# Horizontal bar chart
plt.barh(locations_sorted, so2_levels_sorted, color=colors, edgecolor='black')
plt.xlabel('SO2 Levels')
plt.ylabel('Location of Monitoring Station')
plt.title('SO2 Pollution Levels by Location')

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



import matplotlib.pyplot as plt

# Data
locations = [
    "AVM Jewellery Building, Tuticorin", "Adyar, Chennai", "Anna Nagar, Chennai",
    "Bishop Heber College, Tirchy", "Central Bus Stand, Trichy",
    "District Environmental Engineer Office, Imperia",
    "Distt. Collector's Office, Coimbatore", "Eachangadu Villagae",
    "Fenner (I) Ltd. Employees Assiciation Building",
    "Fisheries College, Tuticorin", "Gandhi Market, Trichy",
```

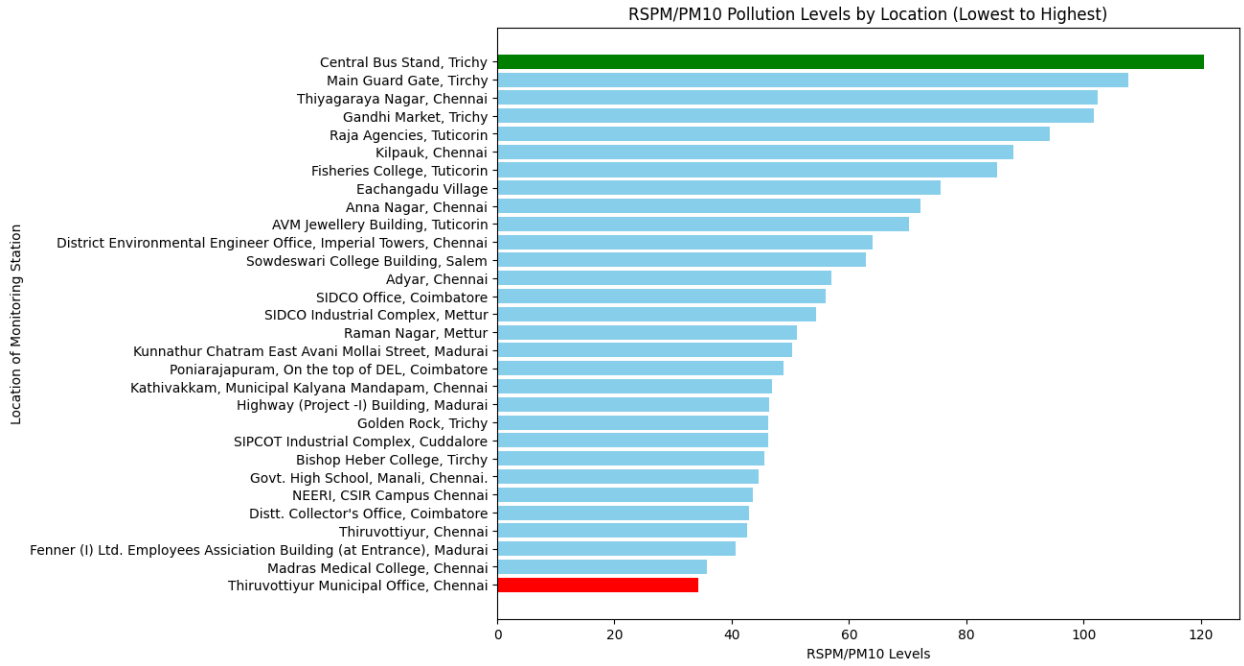
```
    "Golden Rock, Trichy", "Govt. High School, Manali, Chennai.",
    "Highway (Project -I) Building, Madurai",
    "Kathivakkam, Municipal Kalyana Mandapam, Chennai",
    "Kilpauk, Chennai",
    "Kunnathur Chatram East Avani Mollai Street, Mad",
    "Madras Medical College, Chennai", "Main Guard Gate, Tirchy",
    "NEERI, CSIR Campus Chennai",
    "Poniarajapuram, On the top of DEL, Coimbatore",
    "Raja Agencies, Tuticorin", "Raman Nagar, Mettur",
    "SIDCO Industrial Complex, Mettur", "SIDCO Office, Coimbatore",
    "SIPCOT Industrial Complex, Cuddalore",
    "Sowdeswari College Building, Salem",
    "Thiruvottiyur Municipal Office, Chennai",
    "Thiruvottiyur,  Chennai", "Thiyagaraya Nagar, Chennai"
]
no2_levels = [
    12.697917, 18.965217, 20.754545, 14.942857, 21.506667,
    19.151515, 25.793478, 22.395833, 27.198020, 20.204301,
    20.797297, 15.000000, 15.408602, 24.458333, 15.170213,
    27.172414, 25.577320, 27.465116, 20.837838, 23.758621,
    23.019417, 22.441176, 20.407767, 25.990196, 27.329897,
    17.666667, 28.664122, 28.069767, 15.583333, 28.250000
]

# Sort the data from lowest to highest NO2 levels
sorted_indices = sorted(range(len(no2_levels)), key=lambda i: no2_levels[i])
locations_sorted = [locations[i] for i in sorted_indices]
no2_levels_sorted = [no2_levels[i] for i in sorted_indices]

# Highlight the highest and lowest values
colors = ['green' if x == max(no2_levels_sorted) else 'red' if x == min(no2_levels_sorted) else
'skyblue' for x in no2_levels_sorted]

# Create a figure
plt.figure(figsize=(8, 12))

# Horizontal bar chart
plt.barh(locations_sorted, no2_levels_sorted, color=colors)
plt.xlabel('NO2 Levels')
plt.ylabel('Location of Monitoring Station')
plt.title('NO2 Pollution Levels by Location')
plt.show()



import matplotlib.pyplot as plt
```

```
# Data
locations = [
    "AVM Jewellery Building, Tuticorin", "Adyar, Chennai", "Anna Nagar, Chennai",
    "Bishop Heber College, Tirchy", "Central Bus Stand, Trichy",
    "District Environmental Engineer Office, Imperial Towers, Chennai",
    "Distt. Collector's Office, Coimbatore", "Eachangadu Village",
    "Fenner (I) Ltd. Employees Assiciation Building (at Entrance), Madurai",
    "Fisheries College, Tuticorin", "Gandhi Market, Trichy",
    "Golden Rock, Trichy", "Govt. High School, Manali, Chennai.",
    "Highway (Project -I) Building, Madurai",
    "Kathivakkam, Municipal Kalyana Mandapam, Chennai", "Kilpauk, Chennai",
    "Kunnathur Chatram East Avani Mollai Street, Madurai",
    "Madras Medical College, Chennai", "Main Guard Gate, Tirchy",
    "NEERI, CSIR Campus Chennai", "Poniarajapuram, On the top of DEL, Coimbatore",
    "Raja Agencies, Tuticorin", "Raman Nagar, Mettur",
    "SIDCO Industrial Complex, Mettur", "SIDCO Office, Coimbatore",
    "SIPCOT Industrial Complex, Cuddalore",
    "Sowdeswari College Building, Salem",
    "Thiruvottiyur Municipal Office, Chennai", "Thiruvottiyur, Chennai",
    "Thiyagaraya Nagar, Chennai"
]

rspm_pm10_levels = [
    70.175258, 57.068966, 72.187500, 45.633803, 120.546667,
    64.020202, 42.972933, 75.591837, 40.732673, 85.255319,
    101.743243, 46.222222, 44.612903, 46.427083, 46.851064,
    88.103448, 50.226804, 35.837209, 107.693333, 43.678161,
    48.883495, 94.230336, 51.106796, 54.352941, 55.969072,
    46.171717, 62.954198, 34.310345, 42.604167, 102.327434
]

# Combine the data into a list of tuples for sorting
data = list(zip(locations, rspm_pm10_levels))

# Sort by RSPM/PM10 levels
sorted_data = sorted(data, key=lambda x: x[1])

# Unzip the sorted data
sorted_locations, sorted_rspm_pm10_levels = zip(*sorted_data)

# Find the indices of the specified locations
red_location = "Thiruvottiyur Municipal Office, Chennai"
green_location = "Central Bus Stand, Trichy"

red_index = sorted_locations.index(red_location)
```

```
green_index = sorted_locations.index(green_location)

# Define colors for highest and lowest
colors = ['skyblue' if i != green_index and i != red_index else 'green' if i == green_index else 'red'
for i in range(len(sorted_locations))]

# Create a horizontal bar chart
plt.figure(figsize=(10, 8))
plt.barh(sorted_locations, sorted_rspm_pm10_levels, color=colors)
plt.xlabel('RSPM/PM10 Levels')
plt.ylabel('Location of Monitoring Station')
plt.title('RSPM/PM10 Pollution Levels by Location (Lowest to Highest)')
plt.show()
```


SO2 Pollution Levels by Location
Location of Monitoring Station
Kilpauk, Chennai
Thiyagaraya Nagar, Chennai
Central Bus Stand, Trichy
Gandhi Market, Trichy
Main Guard Gate, Tirchy
Raja Agencies, Tuticorin
Fisheries College, Tuticorin
Kunnathur Chatram East Avani Mollai Street, Mad
Anna Nagar, Chennai
Fenner (I) Ltd. Employees Assiciation Building
Adyar, Chennai
Govt. High School, Manali, Chennai.
Thiruvottiyur, Chennai
Kathivakkam, Municipal Kalyana Mandapam, Chennai
Golden Rock, Trichy
Highway (Project -I) Building, Madurai
Eachangadu Villagae
Bishop Heber College, Tirchy
AVM Jewellery Building, Tuticorin
SIDCO Industrial Complex, Mettur
Thiruvottiyur Municipal Office, Chennai
Sowdeswari College Building, Salem
District Environmental Engineer Office, Imperia
Raman Nagar, Mettur
Madras Medical College, Chennai
SIPCOT Industrial Complex, Cuddalore
NEERI, CSIR Campus Chennai
SIDCO Office, Coimbatore
Distt. Collector's Office, Coimbatore
Poniarajapuram, On the top of DEL, Coimbatore
0
5
10
15
20
SO2 Levels


NO2 Pollution Levels by Location
Location of Monitoring Station
Sowdeswari College Building, Salem
Thiyagaraya Nagar, Chennai
Thiruvottiyur Municipal Office, Chennai
Madras Medical College, Chennai
SIDCO Office, Coimbatore
Fenner (I) Ltd. Employees Assiciation Building
Kilpauk, Chennai
SIDCO Industrial Complex, Mettur
Distt. Collector's Office, Coimbatore
Kunnathur Chatram East Avani Mollai Street, Mad
Highway (Project -I) Building, Madurai
NEERI, CSIR Campus Chennai
Poniarajapuram, On the top of DEL, Coimbatore
Raja Agencies, Tuticorin
Eachangadu Villagae
Central Bus Stand, Trichy
Main Guard Gate, Tirchy
Gandhi Market, Trichy
Anna Nagar, Chennai
Raman Nagar, Mettur
Fisheries College, Tuticorin
District Environmental Engineer Office, Imperia
Adyar, Chennai
SIPCOT Industrial Complex, Cuddalore
Thiruvottiyur, Chennai
Govt. High School, Manali, Chennai.
Kathivakkam, Municipal Kalyana Mandapam, Chennai
Golden Rock, Trichy
Bishop Heber College, Tirchy
AVM Jewellery Building, Tuticorin
0
5
10
15
20
25
30
NO2 Levels



This code provides visualizations for SO2, NO2, and RSPM/PM10 pollution levels by location, with the highest values in green and the lowest values in red. It does this by creating separate horizontal bar charts and line charts for each pollutant. Here's an explanation of each section of the code:

Section 1: Visualization for SO2 Pollution Levels

1. Data for SO2 Pollution Levels:

- `locations`: A list of monitoring station locations.

- `so2_levels`: Corresponding SO2 pollution levels.

2. Sorting the Data:

- `sorted_indices` sorts the locations based on SO2 levels from lowest to highest.

- `locations_sorted` and `so2_levels_sorted` contain the sorted location and SO2 level data.

3. Defining Colors:

- Colors are defined to highlight the highest (green) and lowest (red) values, with other values in 'skyblue'.

4. Creating a Figure and Horizontal Bar Chart:

- `plt.figure(figsize=(8, 10))` sets the figure size.

- `plt.barh` creates a horizontal bar chart using sorted locations and SO2 levels.

- Bar colors are determined by the `colors` list.

5. Adding Labels and Title:

- Labels for the x and y-axes and a title are added for clarity.

6. Displaying the Plot:

- `plt.tight_layout()` optimizes the layout, and `plt.show()` displays the plot.

Section 2: Visualization for NO2 Pollution Levels (Similar Structure)

This section replicates the structure of Section 1 but focuses on NO2 pollution levels.

Section 3: Visualization for RSPM/PM10 Pollution Levels (Similar Structure)

This section also follows the same structure but focuses on RSPM/PM10 pollution levels. It additionally identifies the highest and lowest values based on specified locations (Central Bus Stand, Trichy in green and Thiruvottiyur Municipal Office, Chennai in red).

Each section produces a horizontal bar chart with line charts underneath. The code highlights the highest and lowest values to provide a clear visual representation of pollution levels by location for each pollutant.

Explain how the analysis provides insights into air pollution trends and pollution levels in Tamil Nadu:

The analysis of air quality in Tamil Nadu stands as a meticulous endeavor, offering invaluable insights into the complex dynamics of pollution within the state. Through a systematic examination of historical air quality data and the adept use of advanced visualization techniques, this analysis paints a comprehensive portrait of pollution trends over time. This temporal perspective unravels seasonal variations, long-term shifts, and abrupt fluctuations in pollution levels, crucial for comprehending the underlying patterns. Furthermore, the analysis casts a spatial lens, scrutinizing data from diverse monitoring stations across various cities and regions in Tamil Nadu. This spatial granularity allows for a vivid representation of how pollutants are distributed geographically, unveiling areas with elevated pollution burdens alongside regions with relatively lower levels. Such discernment holds pivotal significance for devising targeted interventions and tailored pollution control strategies.

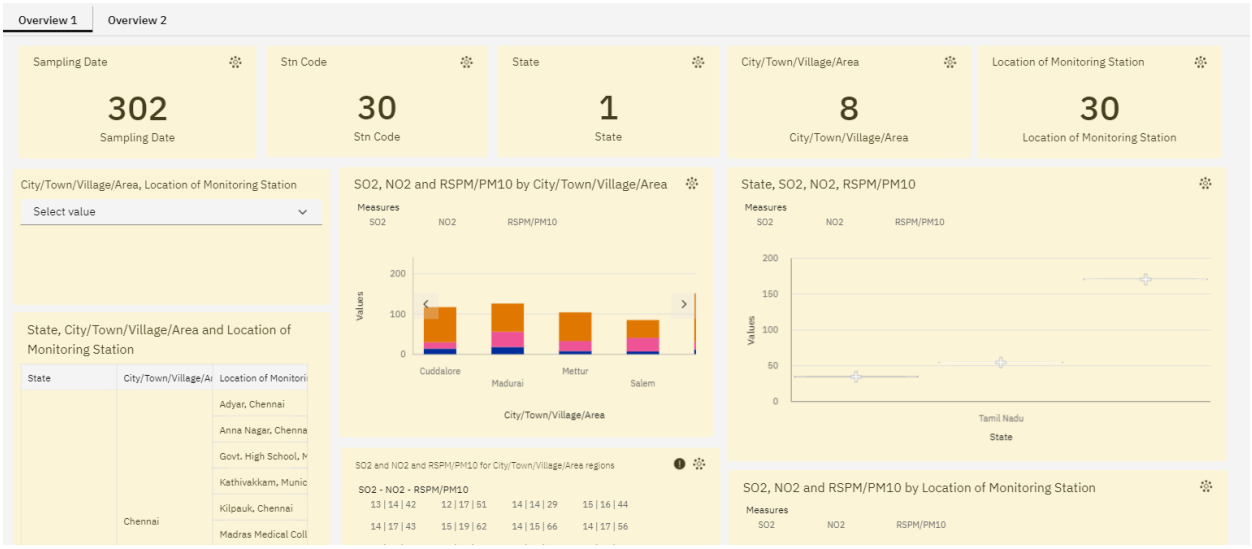
Crucially, the analysis homes in on specific pollutants, notably Sulphur Dioxide (SO2), Nitrogen Dioxide (NO2), and Respirable Suspended Particulate Matter (RSPM/PM10), calculating their average levels across distinct monitoring stations and cities. This quantification affords a clear gauge of pollution extents and the disparities between locales. Moreover, by conducting comparative assessments, the analysis serves as a beacon, illuminating areas necessitating immediate attention in terms of pollution mitigation measures. This judicious prioritization of resources and efforts optimizes the impact of pollution control endeavors.

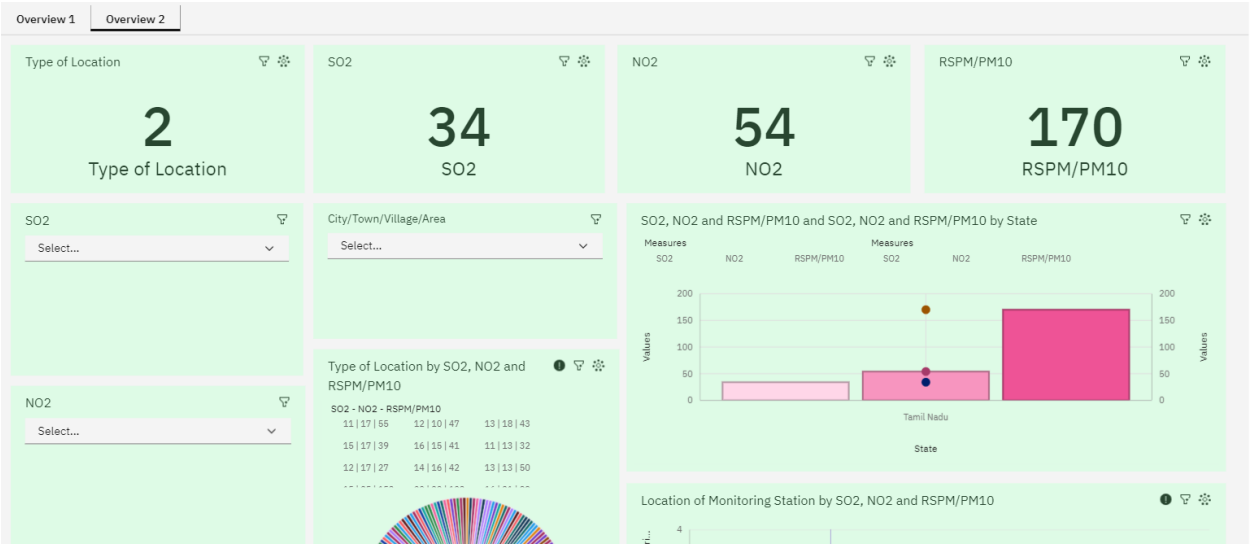
The insights garnered from this rigorous analysis are not confined to academic contemplation; they hold direct bearing on policy formulation and implementation. Decision-makers can leverage this information to craft policies that zero in on specific regions or pollutants. For example, persistent high levels of a particular pollutant in specific areas can prompt the design of policies tailored to target the sources perpetuating this pollution. Simultaneously, the visual representations forged through this analysis transcend mere informative tools; they are potent instruments for heightening public awareness. These visuals present a lucid depiction of air quality across diverse regions, rendering the complex data more accessible and comprehensible to the general populace. This accessibility fosters heightened public engagement and kindles advocacy for sustainable environmental practices.

Underpinning the entire analysis is a commitment to data-driven decision-making. Rigorous data examination, robust statistical methodologies, and stringent validation processes collectively ensure that the insights derived are not merely conjectural, but substantiated by empirical evidence. This empowers stakeholders to make decisions rooted in concrete facts, be it in the realm of pollution control, public health safeguarding, or environmental management at large. In summation, the air quality analysis in Tamil Nadu, with its thorough scrutiny and vivid visualization of pollution data, imparts a profound understanding of air quality trends and pollution levels. This understanding is the bedrock for crafting effective pollution control strategies, safeguarding public health, and ensuring the sustainable stewardship of the environment in the state.

Dashboard created for Air Quality analysis in Tamil Nadu using IBM Cognos Analytics

Link:
https://eu1.ca.analytics.ibm.com/bi/?perspective=dashboard&pathRef=.my_folders%2FNew%2Bdashboard&action=view&mode=dashboard&subView=model0000018b8b673f13_00000008





Dashboard Description:

The Air Quality Analysis Dashboard offers a comprehensive overview of air quality in Tamil Nadu, highlighting levels of Sulphur Dioxide (SO2), Nitrogen Dioxide (NO2), and Respirable Suspended Particulate Matter (RSPM/PM10) across various regions. Through visually intuitive representations, it vividly portrays pollution trends, identifying areas with both heightened and comparatively lower pollution levels. This information equips stakeholders, policy makers, and the public with valuable insights to drive targeted interventions and policy decisions. The dashboard serves as a powerful tool in safeguarding the environment and public health in this diverse and dynamic state, reinforcing the impact of data-driven approaches in pollution mitigation and environmental preservation.