**AIR QUALITY ANALYSIS IN TAMILNADU**

OBJECTIVES:

The main goal of air quality analysis in Tamil Nadu, as in any region, is to safeguard public health and the environment by ensuring that the air quality meets national and international standards and is conducive to the well-being of its residents.To assess, monitor, and manage air quality in Tamil Nadu to mitigate the adverse health and environmental impacts of air pollution, and to promote clean and sustainable development within the state.The main goal of air quality analysis in Tamil Nadu, and any region, is ultimately to create an environment where residents can enjoy clean and healthy air, free from the adverse effects of air pollution, and where the natural environment is preserved for future generations.

**STEPS TO DESIGN :**

STEP1: Data Preprocessing:

Data preprocessing is a crucial step in air quality analysis as it involves cleaning, transforming, and preparing raw data for meaningful analysis. In the context of air quality analysis in Tamil Nadu, here are some common data preprocessing steps:

Data Collection: Gather air quality data from various monitoring stations located across Tamil Nadu. This data may include measurements of pollutants like PM2.5, PM10, SO2, NO2, CO, O3, and meteorological data (e.g., temperature, humidity, wind speed, and wind direction).

Data Cleaning:

Handle missing data: Identify and handle missing values in the dataset. This might involve interpolation, removal of incomplete records, or imputation with reasonable estimates.

Outlier detection: Identify and address outliers, which can significantly affect the analysis. Outliers could be due to measurement errors or unusual events. Consider strategies such as winsorization or removal of extreme values.

Data Transformation:

Data normalization: Normalize the data to a common scale, ensuring that different pollutants' measurements are comparable. Common techniques include min-max scaling or z-score standardization.

Data aggregation: Aggregate data at different time scales (e.g., hourly, daily, monthly) to provide a more comprehensive view of air quality trends and fluctuations.

Feature engineering: Create derived features or variables, such as daily averages, peak pollution hours, or pollution indices, to simplify the analysis.

Data Integration:

Merge datasets: Integrate air quality data with relevant meteorological and geographical data, which can help identify correlations and environmental influences on air quality.

STEP 2:JUPYTER ENVIRONMENT :

By using Jupyter environment , we are visualize the data .

Environment Setup: Ensure that your Jupyter environment is properly configured

and ready for data analysis.

Library Installation: Install the necessary Python libraries like Pandas, Matplotlib,

Seaborn, and NumPy for data analysis and visualization.

STEP 3 : DATA VISUALIZATION:

Generate exploratory data visualizations to better understand the dataset, detect patterns, and identify potential issues. Visualizations can include time series plots, scatter plots, and box plots.

Chart Creation: Design and create charts and graphs using Matplotlib and Pandas to represent the air quality analysis in Tamilnadu.

Statistical Analysis: Utilize Python& #39;s capabilities for statistical analysis, such as calculating mean values, and standard deviations.

STEP 4 : CODE EXPLAINATION:

The provided Python code demonstrates how to perform data visualization using the Matplotlib library within a Jupyter Notebook environment. The code takes sample air quality data in the form of a dictionary, converts it into a Pandas DataFrame, and then creates a variety of plots to visualize the data.

Here's a step-by-step explanation of the code:

1. Import the necessary libraries, Pandas for data manipulation and Matplotlib for data visualization.

2. Define sample air quality data in a dictionary called `data`. This data represents air quality measurements in Chennai, Tamil Nadu. It includes attributes such as station code, sampling date, state, city, location of the monitoring station, agency, type of location, and various pollutant levels (SO2, NO2, RSPM/PM10, and PM2.5). The data is structured as key-value pairs, where each key represents a column, and the corresponding values are lists containing the data points.

3. Create a Pandas DataFrame named `df` using the sample data. This DataFrame is essentially a tabular representation of the data, making it easy to work with and analyze.

4. Convert the "Sampling Date" column to datetime objects using `pd.to\_datetime` to ensure that the dates are recognized as date-time data, making it easier to work with time-based visualizations.

5. Create a Matplotlib figure with a size of 12x6 inches to accommodate multiple subplots.

6. Subplot 1 (Top Left):

- Create a line plot showing the levels of SO2 (Sulfur Dioxide) and NO2 (Nitrogen Dioxide) over time (x-axis). The data points are marked with blue and green circles, and each line has a label for identification.

- Set labels, title, and add a legend to the subplot.

7. Subplot 2 (Top Right):

- Create a bar plot showing the levels of RSPM/PM10 (Respirable Suspended Particulate Matter/Particulate Matter with a diameter of 10 micrometers or less) over time. The bars are displayed in red.

- Set labels and title for the subplot.

8. Subplot 3 (Bottom Left):

- Create a box plot displaying the distribution of SO2 and NO2 levels. This allows for visualizing the spread and central tendency of the data.

- Set labels and title for the subplot.

9. Subplot 4 (Bottom Right):

- Create a scatter plot showing the relationship between SO2 levels on the x-axis and RSPM/PM10 levels on the y-axis. Data points are marked in magenta.

- Set labels and title for the subplot.

10. Utilize `plt.tight\_layout()` to ensure proper spacing between subplots.

11. Finally, display the entire figure with all the subplots using `plt.show()`.

The code generates a set of visualizations that help analyze air quality data over time, comparing different pollutants and their relationships. It provides a visual representation of the data that can aid in understanding trends, correlations, and patterns in air quality.

**PROGRAM:**

import pandas as pd

import matplotlib.pyplot as plt

# Sample data

data = {

"Stn Code": [72] \* 13,

"Sampling Date": ["01-03-2014", "01-08-2014", "01-10-2014", "17-01-2014", "22-01-2014", "24-01-2014", "29-01-2014", "31-01-2014", "02-05-2014", "02-07-2014", "02-12-2014", "14-02-2014", "19-02-2014"],

"State": ["Tamil Nadu"] \* 13,

"City/Town/Village/Area": ["Chennai"] \* 13,

"Location of Monitoring Station": ["Thiruvottiyur, Chennai"] \* 13,

"Agency": ["Tamilnadu State Pollution Control Board"] \* 13,

"Type of Location": ["Industrial Area"] \* 13,

"SO2": [13, 11, 13, 16, 12, 14, 14, 14, 13, 14, 13, 16, 10],

"NO2": [17, 16, 17, 18, 14, 18, 16, 15, 17, 17, 18, 16, 18],

"RSPM/PM10": [31, 40, 30, 31, 35, 38, 49, 36, 54, 51, 44, 41, 46],

"PM 2.5": ["NA"] \* 13

}

# Create a DataFrame

df = pd.DataFrame(data)

# Convert "Sampling Date" to a datetime object

df['Sampling Date'] = pd.to\_datetime(df['Sampling Date'], format='%d-%m-%Y')

# Data Visualization

plt.figure(figsize=(12, 6))

# Line plot for SO2 and NO2 over time

plt.subplot(2, 2, 1)

plt.plot(df['Sampling Date'], df['SO2'], marker='o', color='b', label='SO2')

plt.plot(df['Sampling Date'], df['NO2'], marker='o', color='g', label='NO2')

plt.xlabel('Sampling Date')

plt.ylabel('SO2 and NO2 Levels')

plt.title('SO2 and NO2 Levels Over Time')

plt.legend()

# Bar plot for RSPM/PM10

plt.subplot(2, 2, 2)

plt.bar(df['Sampling Date'], df['RSPM/PM10'], color='r')

plt.xlabel('Sampling Date')

plt.ylabel('RSPM/PM10 Levels')

plt.title('RSPM/PM10 Levels Over Time')

# Box plot for SO2 and NO2

plt.subplot(2, 2, 3)

plt.boxplot([df['SO2'], df['NO2']], labels=['SO2', 'NO2'])

plt.ylabel('Levels')

plt.title('Box Plot of SO2 and NO2')

# Scatter plot between SO2 and RSPM/PM10

plt.subplot(2, 2, 4)

plt.scatter(df['SO2'], df['RSPM/PM10'], color='m')

plt.xlabel('SO2 Levels')

plt.ylabel('RSPM/PM10 Levels')

plt.title('Scatter Plot of SO2 vs. RSPM/PM10')

plt.tight\_layout()

plt.show()

OUTPUT:

