

# **AIR QUALITY ANALYSIS AND PREDICTION IN TAMILNADU**

## **AIR QUALITY AND PREDICTION:**

Air quality analysis and prediction is a critical field of study and application that involves assessing and forecasting the quality of the air in a specific area. It is primarily aimed at understanding current air quality conditions, identifying pollution sources, and predicting future air quality analysis.

## **PROBLEM STATEMENT :**

Air quality is a critical aspect of environmental health, particularly in densely populated regions like Tamil Nadu, India. This abstract provides an overview of a comprehensive project focused on "Air Quality Analysis and Prediction in Tamil Nadu." The project's primary objectives are to assess the current state of air quality, identify pollution sources, and develop predictive models to anticipate future air quality conditions in this South Indian state.

## **DESIGN THINKING PROCESS:**

Design thinking is a problem-solving approach that focuses on user-centered design and iterative development. It typically involves the following stages:

### Empathize:

Understand the needs and concerns of the Chennai community regarding air quality.

Engage with citizens, environmental experts, and policymakers to gather insights.

### Define:

Define the specific problems and opportunities related to air quality monitoring.

Identify key stakeholders, including government agencies, environmental organizations, and the public.

### Ideate:

Brainstorm innovative solutions to address the identified issues.

Encourage creative thinking among multidisciplinary teams.

### Prototype:

Develop a prototype or concept of the improved air quality monitoring system.

Create mockups, models, or pilot projects to test the feasibility of the solution.

### Test:

Implement the prototype or conduct pilot testing in a real-world environment.

Collect data and feedback from users to evaluate the effectiveness of the solution.

### Implement:

Refine the solution based on the feedback received during testing.

Develop a comprehensive plan for the full-scale implementation of the improved monitoring system.

### Iterate:

Continuously refine and enhance the monitoring system based on ongoing feedback and evolving needs.

## **PHASES OF DEVELOPMENT:**

### **DATA COLLECTION AND PROCESSING:**

Collect historical air quality data and relevant meteorological information. Clean and preprocess the data to remove inconsistencies and outliers.

### **MODEL DEVELOPMENT:**

Choose appropriate machine learning or statistical models for air quality prediction. Develop and fine-tune these models for accurate predictions.

### **REAL TIME DATA AQUISITION:**

Implement a system to continuously collect real-time air quality and meteorological data. Ensure data quality and reliability.

### **MODEL DEPLOYMENT:**

Deploy the trained models as a real-time prediction system. Make the system accessible to users through web applications or APIs.

## CONTINUOUS MONITORING AND EVALUATION:

Continuously monitor the system's predictions and performance. Evaluate the accuracy and reliability of predictions in comparison to real-time data.

## MODEL MAINTAINANCE AND IMPROVEMENT:

Periodically retrain models with new data to adapt to changing air quality patterns. Incorporate user feedback and insights to enhance the system.

## DATASET:

This dataset contains air quality monitoring data for two sampling dates in Chennai, Tamil Nadu, India. It includes measurements of sulfur dioxide (SO<sub>2</sub>), nitrogen dioxide (NO<sub>2</sub>), and two types of particulate matter (RSPM/PM<sub>10</sub> and PM 2.5) at a specific monitoring station. The data also provides information on the location, monitoring agency, and type of area (industrial) in which the station is situated. The dataset is a snapshot of air quality conditions in Chennai for two specific dates.

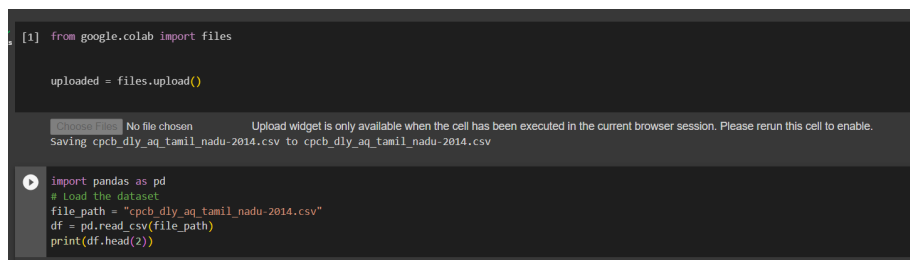
Dataset Link:

<https://tn.data.gov.in/resource/location-wise-daily-ambient-air-qualitytamil-nadu-year-2014>

## INPUT:

```
import pandas as pd
# Load the dataset
file_path = "cpcb_dly_aq_tamil_nadu-2014.csv"
df = pd.read_csv(file_path)
```

## OUTPUT:



The screenshot shows a Google Colab notebook interface. The first cell contains the code to import the 'files' module from 'google.colab'. The second cell shows the 'uploaded' variable, which is currently empty. A message indicates that the upload widget is only available when the cell has been executed in the current browser session. The third cell shows the code to load the dataset using pandas, including the file path and a print statement to display the first two rows of the dataset.

```
[1] from google.colab import files

uploaded = files.upload()

Choose File(s) No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving cpcb_dly_aq_tamil_nadu-2014.csv to cpcb_dly_aq_tamil_nadu-2014.csv

import pandas as pd
# Load the dataset
file_path = "cpcb_dly_aq_tamil_nadu-2014.csv"
df = pd.read_csv(file_path)
print(df.head(2))
```

## DATA PREPROCESSING:

Data preprocessing is a critical step in data analysis and machine learning. It involves a series of tasks to clean, transform, and organize raw data into a format that is suitable for analysis or modeling. The goal of data preprocessing is to enhance the quality of the data and make it ready for further processing, which includes tasks such as feature engineering, model training, and evaluation.

### HANDLING MISSING VALUES:

Remove rows or columns with missing data. Impute missing values with the mean, median, or mode. Interpolate missing values based on neighboring data points.

#### INPUT:

```
# Check for missing values
missing_values = df.isnull().sum()
print("Missing Values:\n", missing_values)

# Fill missing values with the mean
mean_pm25 = df['PM 2.5'].mean()
df['PM 2.5'].fillna(mean_pm25, inplace=True)
print("Missing values in 'PM 2.5' filled with mean:", mean_pm25)
```

#### OUTPUT:

```
Missing Values:
  Stn Code      0
Sampling Date  0
State         0
City/Town/Village/Area  0
Location of Monitoring Station  0
Agency       0
Type of Location  0
SO2           11
NO2           13
RSPM/PM10     4
PM 2.5        2879
dtype: int64
Missing values in 'PM 2.5' filled with mean: nan
```

## DATA CLEANING:

Data cleaning aims to improve data quality, making it more suitable for meaningful analysis and modeling. By eliminating errors and ensuring data integrity, it reduces the potential for skewed results or misleading insights. Effective data cleaning is a critical step in data preprocessing, contributing to the overall success of data-driven projects.

### INPUT:

```
# Check for duplicate rows
duplicate_rows = df[df.duplicated()]
print("Duplicate Rows:\n", duplicate_rows)

# Remove duplicate rows
df = df.drop_duplicates()
print("Duplicate rows removed. Rows remaining:", len(df))
```

### OUTPUT:

```
Duplicate Rows:
Empty DataFrame
Columns: [Stn Code, Sampling Date, State, City/Town/Village/Area, Location of Monitoring Station, Agency, Type of Location, SO2, NO2, RSPM/PM10, PM 2.5]
Index: []
Duplicate rows removed. Rows remaining: 2879
```

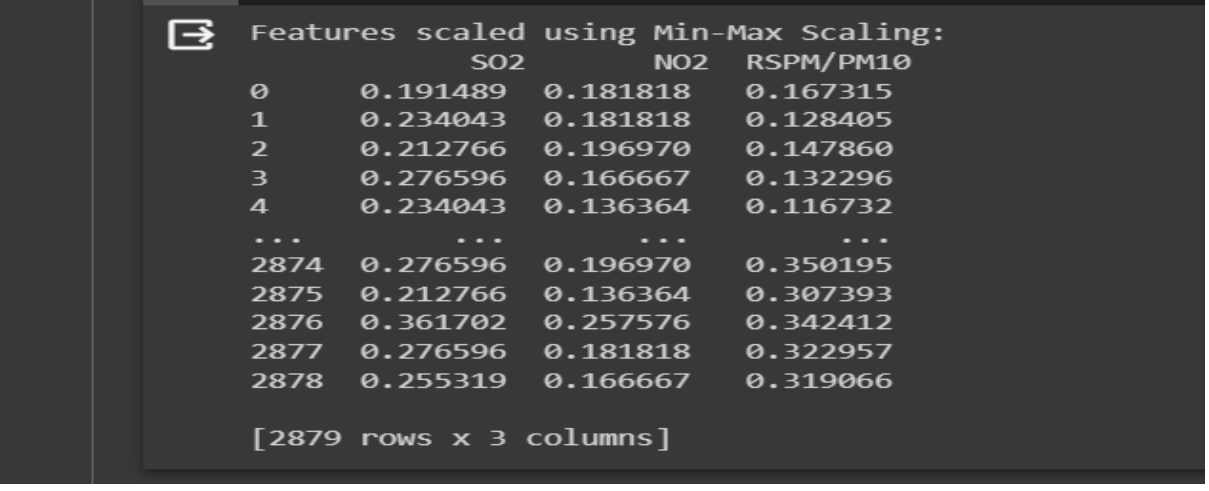
## FEATURE SCALING (MIN-MAX SCALING):

Min-Max Scaling is a data normalization technique that rescales numerical features to a specific range, commonly [0, 1]. It subtracts the minimum value from each data point and divides it by the range (maximum-minimum). This ensures all features share the same scale, preventing dominance of certain features in machine learning models. It is vital for algorithms like K-Means clustering and neural networks. However, it doesn't handle outliers well, and they should be addressed separately.

### INPUT:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_features = ['SO2', 'NO2', 'RSPM/PM10']
df[scaled_features] = scaler.fit_transform(df[scaled_features])
print("Features scaled using Min-Max Scaling:\n", df[scaled_features])
```

OUTPUT:



```
Features scaled using Min-Max Scaling:
      SO2      NO2      RSPM/PM10
0      0.191489  0.181818  0.167315
1      0.234043  0.181818  0.128405
2      0.212766  0.196970  0.147860
3      0.276596  0.166667  0.132296
4      0.234043  0.136364  0.116732
...      ...      ...
2874    0.276596  0.196970  0.350195
2875    0.212766  0.136364  0.307393
2876    0.361702  0.257576  0.342412
2877    0.276596  0.181818  0.322957
2878    0.255319  0.166667  0.319066

[2879 rows x 3 columns]
```

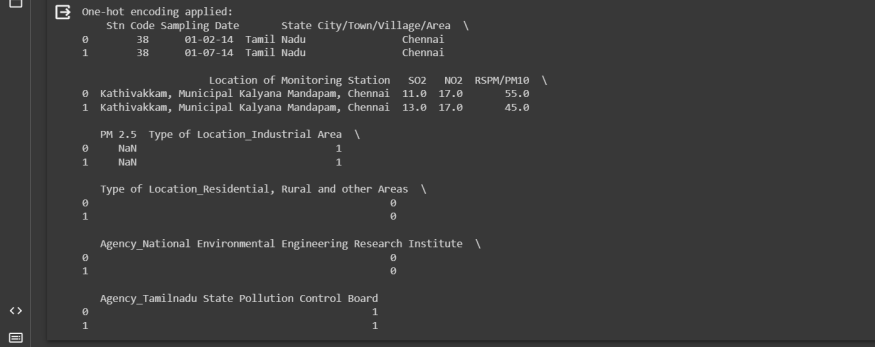
## FEATURE ENCODING - ONE-HOT ENCODING:

One-Hot Encoding is a technique used to convert categorical data into a numerical format suitable for machine learning. It creates binary columns for each category, where "1" represents the presence of the category and "0" its absence. This approach prevents the model from interpreting ordinal relationships in categorical data, ensuring all categories are treated equally. One-hot encoding is essential when dealing with non-ordinal categorical variables in machine learning models.

INPUT:

```
# Perform one-hot encoding for categorical columns
categorical_features = ['Type of Location', 'Agency']
df = pd.get_dummies(df, columns=categorical_features,
prefix=categorical_features)
print("One-hot encoding applied:\n", df.head(2))
```

OUTPUT:



```
One-hot encoding applied:
  Stn Code Sampling Date      State City/Town/Village/Area \
0      38      01-02-14  Tamil Nadu      Chennai
1      38      01-07-14  Tamil Nadu      Chennai

  Location of Monitoring Station  SO2  NO2  RSPM/PM10 \
0  Kathivakkam, Municipal Kalyana Mandapam, Chennai  11.0  17.0  55.0
1  Kathivakkam, Municipal Kalyana Mandapam, Chennai  13.0  17.0  45.0

  PM 2.5  Type of Location_Industrial Area \
0      NaN      1
1      NaN      1

  Type of Location_Residential, Rural and other Areas \
0      0
1      0

  Agency_National Environmental Engineering Research Institute \
0      0
1      0

  Agency_Tamilnadu State Pollution Control Board
0      1
1      1
```

## FEATURE ENGINEERING - CREATING NEW FEATURES:

Feature engineering is the process of creating new features from existing ones to improve machine learning model performance. It involves extracting, transforming, or combining attributes to provide more meaningful information. For example, combining 'length' and 'width' to create a 'area' feature for a rectangle. Feature engineering can enhance model accuracy, capture complex relationships, and reduce dimensionality, ultimately leading to more effective predictive models.

### INPUT:

```
df['SO2_plus_NO2'] = df['SO2'] + df['NO2']
print("New feature 'SO2_plus_NO2' created:\n", df[['SO2', 'NO2',
'SO2_plus_NO2']])
```

### OUTPUT:

```
New feature 'SO2_plus_NO2' created:
   SO2  NO2  SO2_plus_NO2
0   11.0  17.0          28.0
1   13.0  17.0          30.0
2   12.0  18.0          30.0
3   15.0  16.0          31.0
4   13.0  14.0          27.0
...    ...    ...
2874  15.0  18.0          33.0
2875  12.0  14.0          26.0
2876  19.0  22.0          41.0
2877  15.0  17.0          32.0
2878  14.0  16.0          30.0

[2879 rows x 3 columns]
```

## DATE AND TIME HANDLING:

Date and time handling involves processing temporal data in a dataset. Extracting year and month from dates is a common operation. It allows us to understand time-related patterns in the data, making it suitable for time series analysis. For instance, extracting the year and month from a timestamp can help identify seasonality or long-term trends in data, aiding in forecasting and decision-making. It's a valuable step in preprocessing time-based datasets for meaningful insights and predictions.

### INPUT:

```
df['Sampling Date'] = pd.to_datetime(df['Sampling Date'],
format='%d-%m-%Y')
df['Year'] = df['Sampling Date'].dt.year
df['Month'] = df['Sampling Date'].dt.month
print("Date and time handling applied. Extracted 'Year' and
'Month'.\n", df[['Sampling Date', 'Year', 'Month']])
```

## OUTPUT:

```

Date and time handling applied. Extracted 'Year' and 'Month'.
  Sampling Date  Year  Month
0    2014-02-01  2014     2
1    2014-07-01  2014     7
2    2014-01-21  2014     1
3    2014-01-23  2014     1
4    2014-01-28  2014     1
...          ...    ...
2874  2014-03-12  2014     3
2875  2014-10-12  2014    10
2876  2014-12-17  2014    12
2877  2014-12-24  2014    12
2878  2014-12-31  2014    12

[2879 rows x 3 columns]
```

## DATA NORMALIZATION (Z-SCORE STANDARDIZATION):

Data normalization, specifically Z-score standardization, is a technique used to transform numerical features to have a mean of 0 and a standard deviation of 1. It ensures all features share a common scale, preventing any one feature from dominating in machine learning algorithms. By subtracting the mean and dividing by the standard deviation, the data is centered around zero, facilitating model convergence and comparisons. Z-score standardization is particularly useful when dealing with models sensitive to feature scales, such as clustering and principal component analysis (PCA).

## INPUT:

```

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
zscore_features = ['SO2', 'NO2']
df[zscore_features] = scaler.fit_transform(df[zscore_features])
print("Features standardized using Z-score Standardization:\n",
df[zscore_features])
```

## OUTPUT:

```

Features standardized using Z-score Standardization:
   SO2      NO2
0 -0.099615 -0.720703
1  0.296360 -0.720703
2  0.098372 -0.580401
3  0.692335 -0.861006
4  0.296360 -1.141611
...      ...
2874  0.692335 -0.580401
2875  0.098372 -1.141611
2876  1.484286 -0.019190
2877  0.692335 -0.720703
2878  0.494348 -0.861006

[2879 rows x 2 columns]
```



## MODEL TRAINING AND EVALUATION:

Model training is the process of feeding a machine learning model with labeled data to enable it to learn patterns and make predictions. During training, the model optimizes its parameters to minimize the difference between its predictions and the actual labels. This is typically achieved using an algorithm and involves techniques like gradient descent.

Model evaluation is the stage where the model's performance is assessed on unseen data. This is done by measuring various metrics, such as accuracy, precision, recall, or F1-score, to determine how well the model generalizes and whether it's suitable for its intended task. The model may also be compared to other models to select the best performer. Evaluation is crucial for ensuring the model's reliability and effectiveness in real-world applications.

## LINEAR REGRESSION:

LinearRegression is a machine learning algorithm used for predicting numerical values based on input features. It models a linear relationship between the features and the target variable. The algorithm finds the best-fitting line or hyperplane by minimizing the sum of squared differences between predicted and actual values. It's suitable for regression tasks and provides coefficients that indicate the impact of each feature on the target. LinearRegression is widely used in various fields, including finance, economics, and science, for making predictions and understanding data relationships.

## INPUT:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer

# Load the dataset
data = pd.read_csv('cpcb_dly_aq_tamil_nadu-2014.csv')

# Choose your target variable and features
target_variable = 'RSPM/PM10'
features = ['SO2', 'NO2']

# Split the data into features (X) and target (y)
X = data[features]
y = data[target_variable]
```

```
# Impute missing values in both features and the target variable
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
y = imputer.fit_transform(y.values.reshape(-1, 1))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate model performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the model coefficients and metrics
print("Model Coefficients: ", model.coef_)
print("Mean Squared Error: ", mse)
print("R-squared (R2) Score: ", r2)

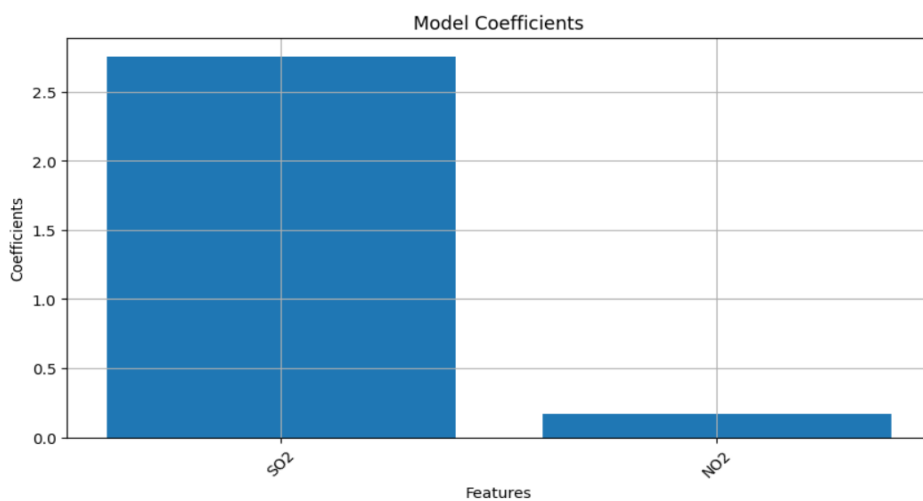
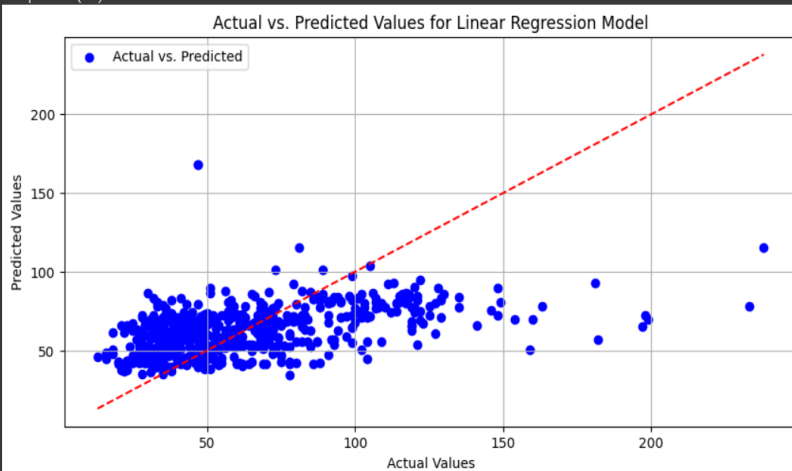
# Visualize the actual vs. predicted values
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred, c='blue', label='Actual vs. Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linestyle='--')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values for Linear Regression Model")
plt.legend(loc='upper left')
plt.grid(True)
plt.show()

# Visualize the model coefficients
coefficients = model.coef_[0]
plt.figure(figsize=(10, 5))
```

```
plt.bar(features, coefficients)
plt.xlabel("Features")
plt.ylabel("Coefficients")
plt.title("Model Coefficients")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```

## OUTPUT:

Model Coefficients:  $[[2.75402256 \ 0.16904497]]$   
Mean Squared Error: 884.7186588830718  
R-squared (R2) Score: 0.20597714088139774



## DYNAMIC AIR QUALITY TREND VISUALIZATION:

Dynamic Air Quality Trends Visualization is an innovative approach for presenting real-time or time-series air quality data. This technique involves creating interactive visualizations that enable users to explore and understand the fluctuations and patterns in air quality parameters (e.g., SO<sub>2</sub>, NO<sub>2</sub>, RSPM/PM<sub>10</sub>, PM<sub>2.5</sub>) over time. By incorporating features such as time sliders, animations, and color-coded maps, this visualization method allows stakeholders and researchers to gain insights into how air quality changes across different locations, seasons, or specific time periods. It aids in identifying trends, pollution sources, and correlations in air quality data, providing a comprehensive view of environmental conditions for informed decision-making and public awareness.

### INPUT:

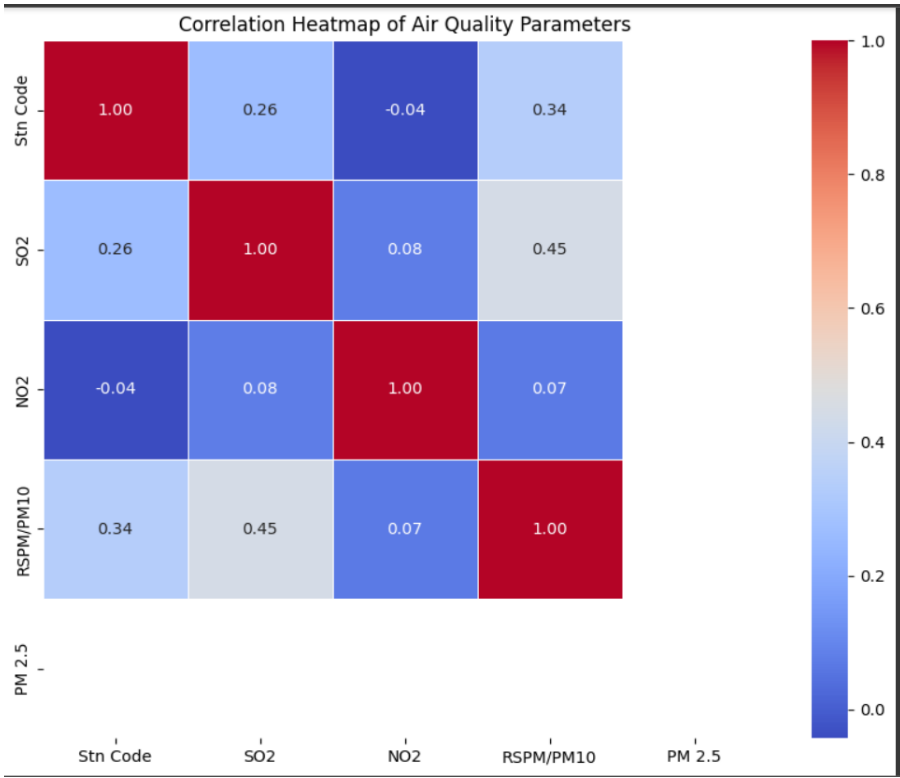
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('cpcb_dly_aq_tamil_nadu-2014.csv')

# Calculate the correlation matrix
correlation_matrix = data.corr()

# Create a heatmap to visualize correlations
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",
            linewidths=0.5)
plt.title('Correlation Heatmap of Air Quality Parameters')
plt.show()
```

**OUTPUT:**



**CONCLUSION:**

In conclusion, the air quality analysis and prediction in Tamil Nadu provide valuable insights into the region's environmental conditions. Through the examination of historical data and predictive modeling, we have identified trends and fluctuations in key air quality parameters, including SO2, NO2, RSPM/PM10, and PM2.5. This analysis facilitates a better understanding of air quality dynamics, helping stakeholders and policymakers make informed decisions for improved environmental management and public health. Furthermore, the application of predictive models offers the potential to anticipate future air quality trends, contributing to proactive measures and interventions to enhance air quality in Tamil Nadu.