



PROJECT REPORT
ON
AIR QUALITY INDEX CALCULATOR

SUBMITTED

TO

**ROURKELA INSTITUTE OF MANAGEMENT
STUDIES**

**(As a partial fulfillment of the requirement for the award of
degree)**

FOR

“BACHELOR OF COMPUTER APPLICATION”

SUBMITTED BY

SHRADHANJALI MOHANTY

ROLL NO: S3122BCA009

BCA 6TH SEMESTER (2022-2025)

ROURKELA INSTITUTE OF MANAGEMENT STUDIES

(Affiliated to Biju Patnaik University of Technology)

ROURKELA, ODISHA

ACKNOWLEDGEMENT

I am deeply indebted to **Rourkela Institute of Management Studies, Chhend, Rourkela**, for providing me an opportunity to undertake a project work entitled **“AIR QUALITY INDEX CALCULATOR.”**

I am grateful to my project guide **Prof. Bibhudendu Panda (H.O.D MCA)** without his guidance it would not have been possible on my part to complete the project.

I acknowledge the help and cooperation received from all my team members in making this project successful.

I consider myself fortunate that I have successfully completed this project, I acknowledge my sincere gratitude to all those works and ideas that had helped me in this project.

SHRADHANJALI MOHANTY

University Roll No: -S3122BCA009

BCA (2022-2025)

ROURKELA INSTITUTE OF MANAGEMENT STUDIES

ROURKELA

DECLARATION

I, **SHRADHANJALI MOHANTY**, hereby declare that the project report entitled “**AIR QUALITY INDEX CALCULATOR**” is of my work at **ROURKELA INSTITUTE OF MANAGEMENT STUDIES, CHHEND, ROURKELA** for the partial fulfillment of **BCA 6th Semester (2022-2025)** under the directorate of **SAMBALPUR UNIVERSITY**.

(SHRADHANJALI MOHANTY)

ROLL NO.: - S3122BCA009

CERTIFICATE

This is to certify that this is a bonafide record of the project entitled **“AIR QUALITY INDEX CALCULATOR”** **developed** satisfactorily at the institute by **SHRADHANJALI MOHANTY, Roll No: S3122BCA009** towards the partial fulfilment of **Bachelor of Computer Application, Session (2022-2025)** under the Undergraduate Course, Rourkela Institute of Management Studies (Affiliated to Sambalpur University, Jyoti Vihar, Burla, Odisha).

During the project period, she was found to be sincere and her conduct remain satisfactory.

Date: - 05.04.2025

Signature of Internal Guide

Signature of Principal

Signature of H.O.D

Signature of External Examiner

ABSTRACT

Air pollution is the largest environmental & public health challenge in the world today. Air pollution leads to adverse effects on human health, climate and ecosystem.

Air is getting polluted because of release of toxic gases by industries, vehicular emissions and increased concentration of harmful gases and particulate matter in the atmosphere.

Particulate matter is one of the most important parameter having the significant contribution to the increase in air pollution. So, in order to prevent us to expose from air pollution we need to predict air quality priorly and take necessary.

In this project we are predict the air quality index using the machine learning in python language. Air Quality Index Prediction (AQI) is calculated based on chemical pollutant like Sulphur dioxide, Nitrogen dioxide, Carbon monoxide, Ozone, PM2.5 and PM 10 as well as the Air Quality Index (AQI), at an accuracy of 80.8 percent.

CONTENTS

SERIAL NO	TOPIC	PAGE NO
1	Introduction	7 - 22
2	Software Requirement Specification	23 - 26
3	Designing	27 - 30
4	Coding	31 - 72
5	Testing	73 - 76
6	Output	77 - 78
7	Conclusion And Future Scope	79
8	Reference	80

1.INTRODUCTION

1.1 Preface

In our increasingly and industrial world, air quality has become a critical concern for public health and the environment. The Air Quality Index (AQI) serves as a vital tool to communicate the state of the air we breathe, helping individual and communities making informed decision.

This index simplifies complex air pollution data into an easy-to-understand format, highlighting the health impacts of various pollutants such as particulate matter, ozone, and nitrogen dioxide. By understanding the AQI we can take proactive measures to improve air quality and protects our well- being.

This document provides a concise overview of AQI and its significant in our daily lives.

1.2 Problem Definition and Objectives

Air pollution in urban areas poses significant health risk and environmental challenges. Despite existing effort, there is a gap in effectively informing the public about air quality and its impacts.

1. Raise Awareness: Simplify air quality data for public understanding.
2. Ongoing Monitoring: Ensure continuous and accurate air quality updates.

1.3 MOTIVATION

Due to lack of awareness people health affected badly so AQI helps analyzing the change in air quality and it also helps in air

quality and it also helps in identifying faulty standards and inadequate monitoring programs.

1.4 PROJECT OVERVIEW

The Air Quality Index (AQI) is a statistical for assessing the quality of the air in our surrounding. Air is getting polluted because of the release of toxic gases by industries, vehicles emissions and increased concentration of harmful gases and particular matter in the atmosphere.

The AQI project leverages cutting-edge machine learning model to enhance the accuracy and accessibility of air quality information.

The dataset consists of air sample of different location.

In this project, we predict the Air Quality Index of different location by using the machine learning model.

I have used the Python language and Jupyter Notebook Editor. We train the model using the training data set.

The testing data consist of features and our model has to predict the target sequence.

1.5 HARDWARE SPECIFICATION

- Processor - i3 or higher
- Processor Speed – 2.30GHz
- RAM – 4GB or higher
- Hard Disk – 500GB or higher
- System Type – 64bit operating system

1.6 SOFTWARE SPECIFICATION

- Windows: 10 or higher
- Frontend: HTML and CSS
- Backend: Python AND

- Database: MONGO DB
- API: Flask
- Tools: Visual Studio Code, Python (Jupyter Notebook)

2. ENVIRONMENTAL SETTING

2.1 PREFACE

In an era where urbanization and industrialization activities are at their peak, the quality of air that we breath has become a crucial aspect of public health and environmental protection.

We have designed a machine learning model with existing data ser to predict the air quality index.

This report provides a comprehensive overview of the AQI, its methodology, and the significance of using machine learning in monitoring air quality.

2.3 EXISTING SYSTEM

There are wide variety of existing software and platforms available online that shows the AQI .Like , the widely used platform , www.accuweather.com which directly provides the access to the air quality using the location but there are some drawbacks in this platform are : Inaccuracy , User Interface issues , reliability Concerns and in comparison with other apps it provides inaccurate and less reliable forecasts.

2.4 FEASIBILITY STUDY

✓ Economic Feasibility:

This project is economically feasible as it does not require much money. The only requirement that is needed is web browser along with any operating system.

It is free for viewing on World Wide Web with a registered domain name.

✓ **Technical Feasibility:**

For the successful development of this project all we need is an internet connection, a database server, and a supporting web server. This can be successfully implemented in popular web browsers like Microsoft Edge environment, Internet Explorer, Google, and other browsing application.

2.4 ENVIRONMENT SETTING:

Steps to Download and Install Python 3.13.2 on Windows

Step 1: Download Python 3.13.2

To start, go to **python.org/downloads/** and then click on the button to download the latest version of Python:



Step 2: Run the .exe file

Next, run the .exe file that you just downloaded.



python-3.13.2-amd64.exe

Step 3: Install Python 3.13.2

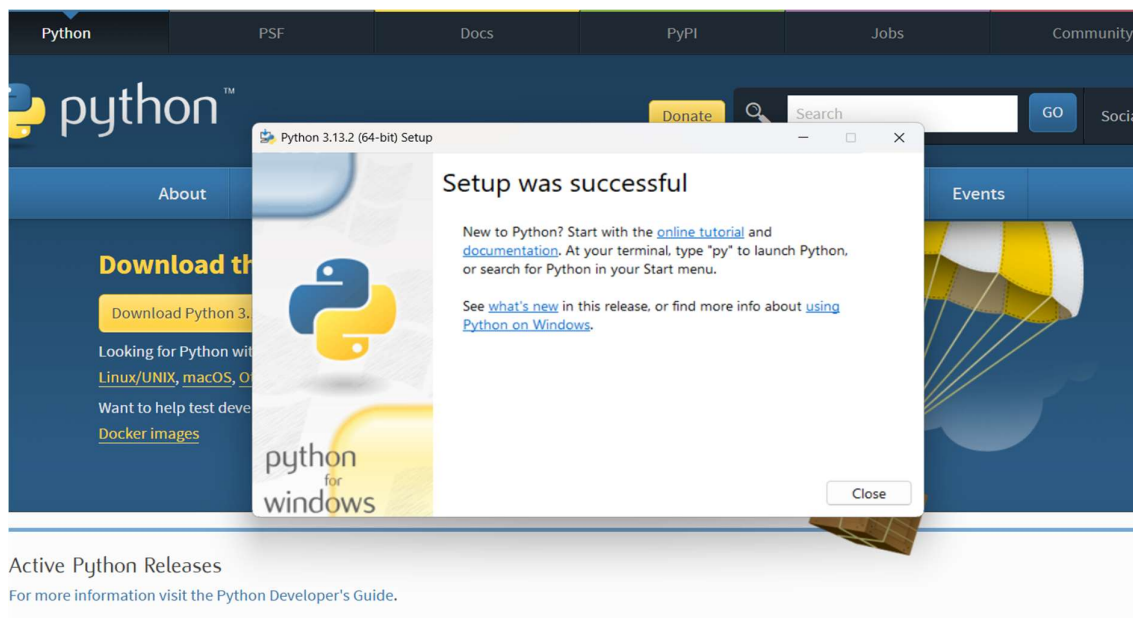
You can now start the installation of Python by clicking on **Install Now**:



IMPORTANT: Make sure to check the box that says *Add python.exe to Path* as shown. This ensures that the Python interpreter is added to your system's execution path, allowing you to run Python from the command line. Additionally, selecting *Use admin privileges when installing py.exe* can help avoid permission issues and ensure a smoother installation. Your installation should now begin:



After a short time, your setup will be completed.

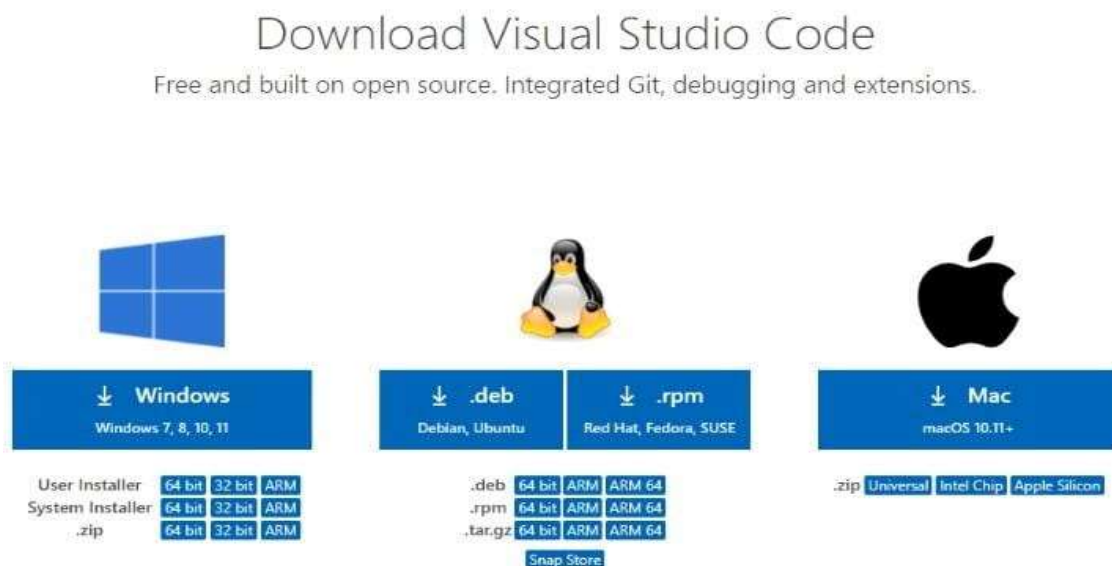


→ About VS Code:

Visual Studio is an Integrated Development Environment(IDE) developed by Microsoft to develop Desktop applications, GUI(Graphical User Interface), console, web applications, mobile applications, cloud, and web services, etc. With the help of this IDE, you can create managed code as well as native code. It uses the various platforms of Microsoft software development software like Windows store, Microsoft Silverlight, and Windows API, etc. It is not a language-specific IDE as you can use this to write code in C#, C++, VB(Visual Basic), Python, JavaScript, and many more languages. It provides support for 36 different programming languages. It is available for Windows as well as for macOS.

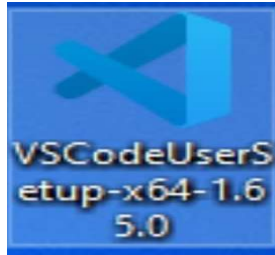
→ Steps to Install Visual Studio Code on Windows

Step 1: Visit the Official Website of the Visual Studio Code using any web browser like Google Chrome, Microsoft Edge, etc.



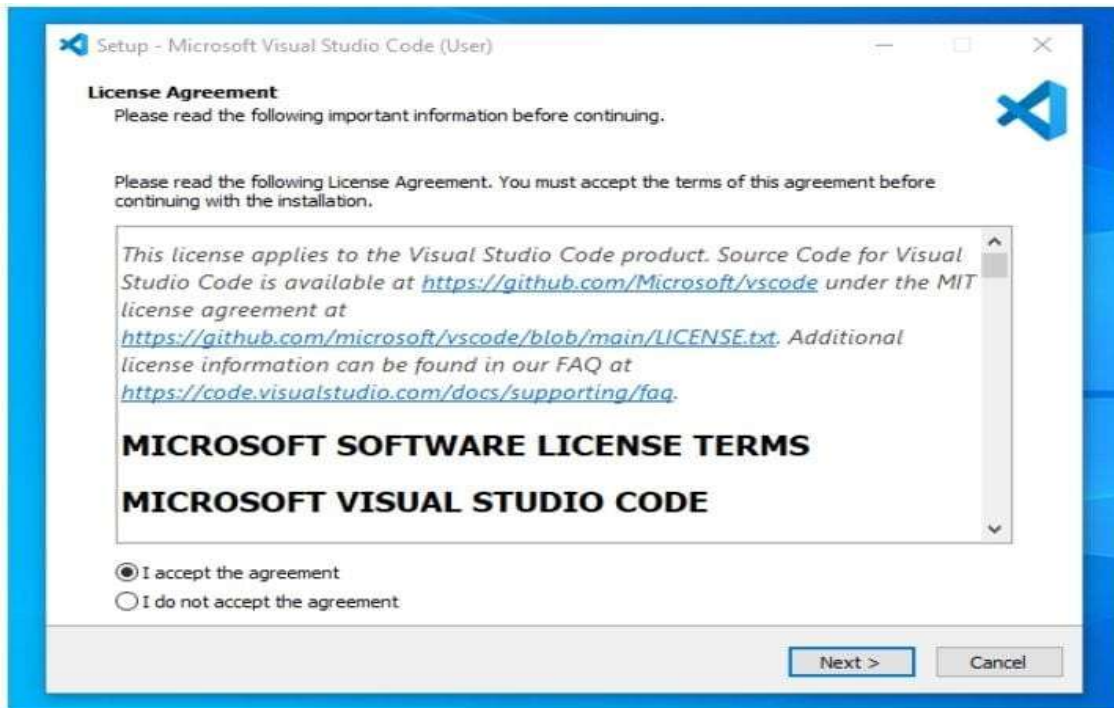
Step 2: Press the “Download for Windows” button on the website to start the download of the Visual Studio Code Application.

Step 3: When the download finishes, then the Visual Studio Code Icon appears in the downloads folder.

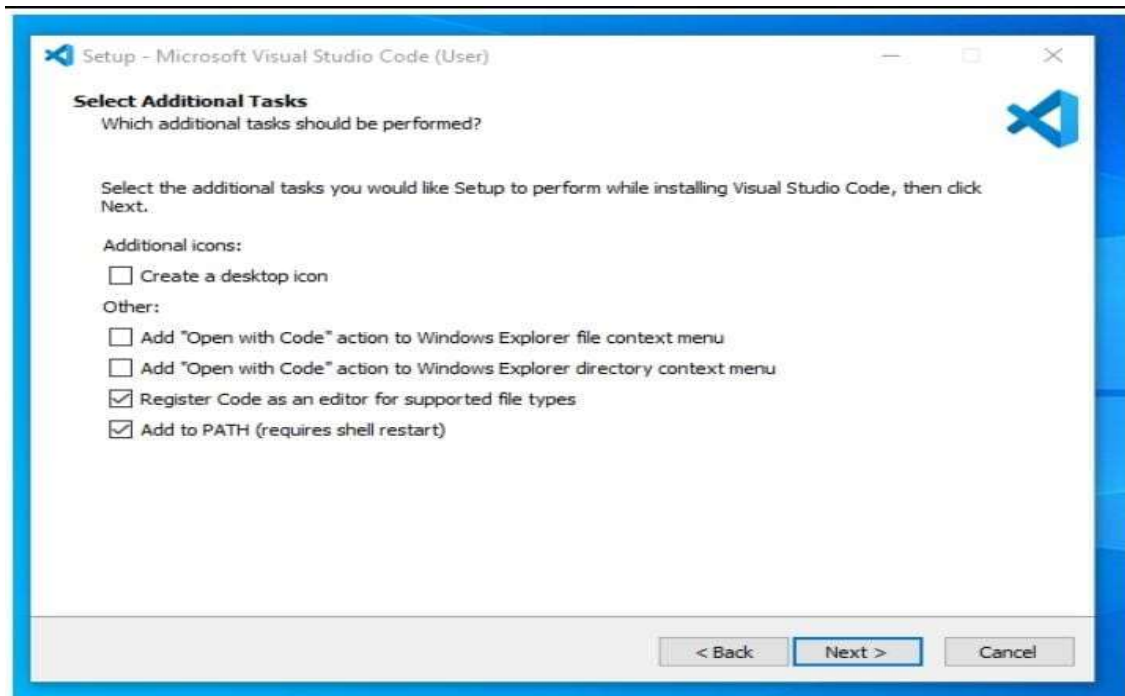


Step 4: Click on the Installer icon to start the installation process of the Visual Studio Code.

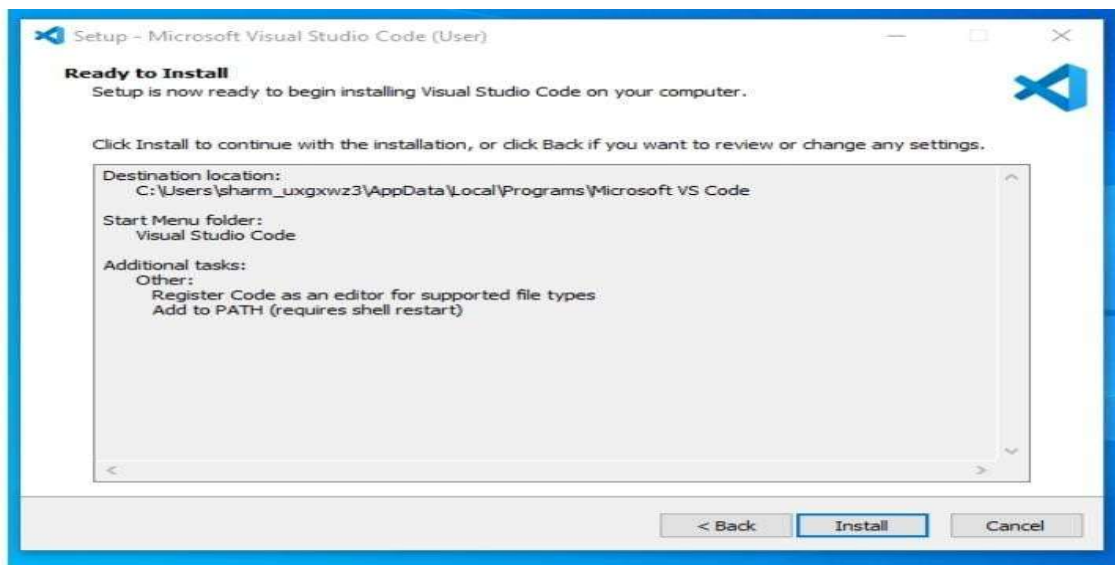
Step 5: After the Installer opens, it will ask you to accept the terms and conditions of the Visual Studio Code. Click on I accept the agreement and then click the Next butt



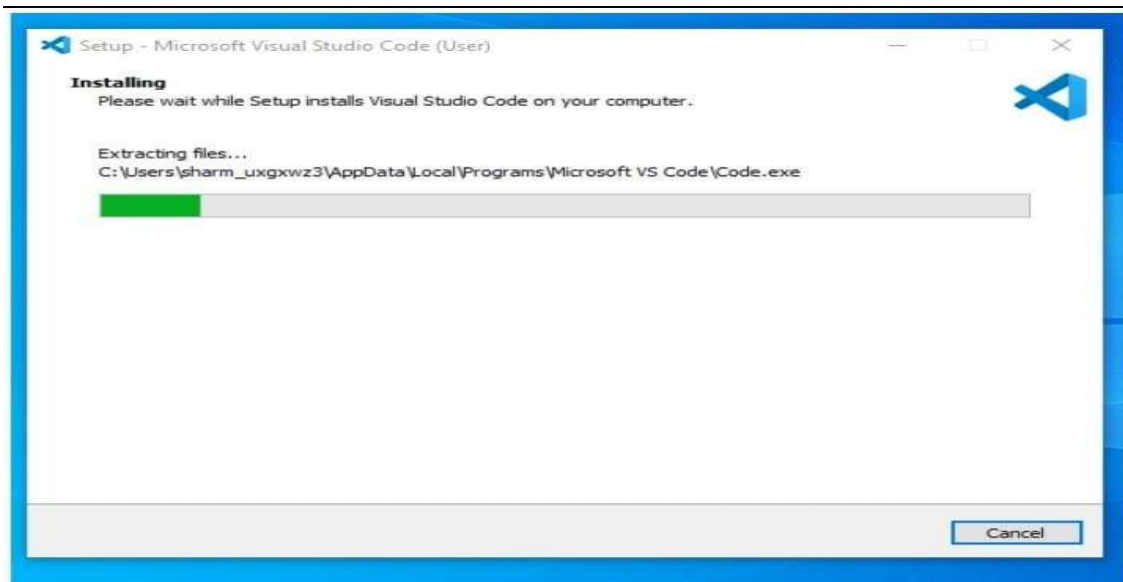
Step 6: Choose the location data for running the Visual Studio Code. It will then ask you to browse the location. Then click on the Next button.



Step 7: Then it will ask to begin the installation setup. Click on the Install button.



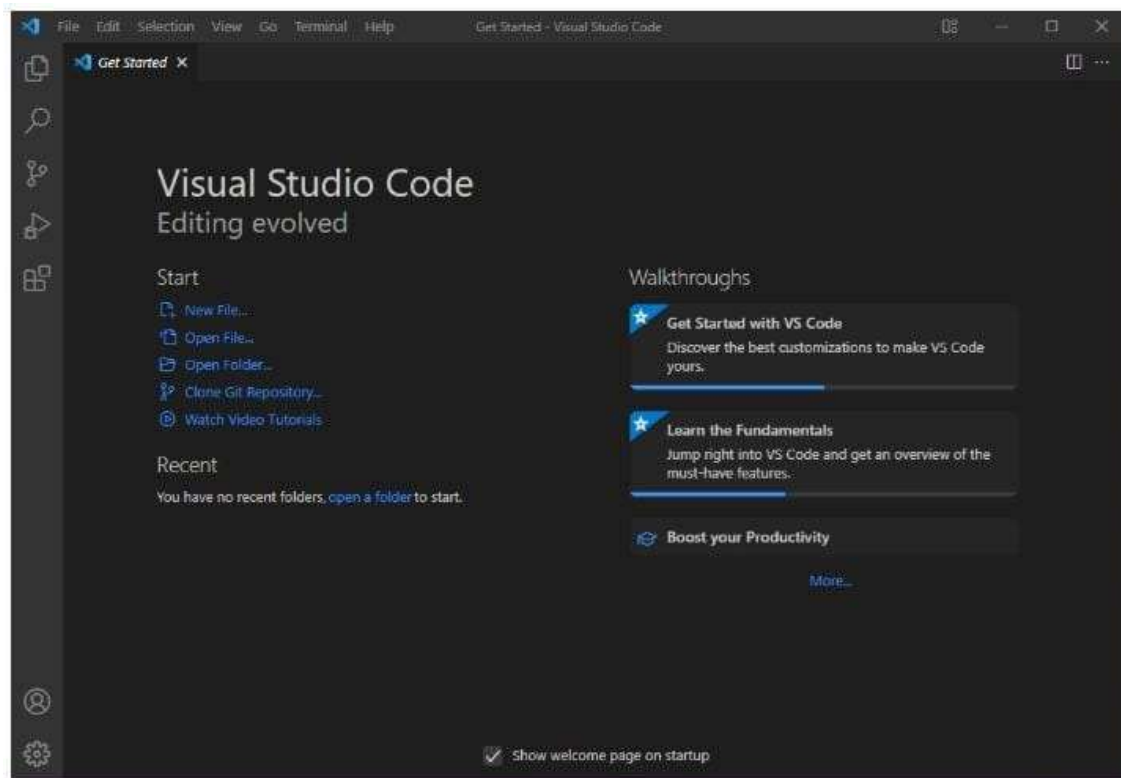
Step 8: After clicking on Install, it will take about 1 minute to install the Visual Studio Code on your device.



Step 9: After the Installation setup for Visual Studio Code is finished, it will show a window like this below. Tick the “Launch Visual Studio Code” checkbox and then click Next.



Step 10 : After the previous step, the Visual Studio Code window opens successfully. Now you can create a new file in the Visual Studio Code window and choose a language of yours to begin your programming journey!



So, this is how we successfully installed Visual Studio Code on our Windows system. The steps mentioned in the above guideline can be used in any kind of Windows Browsers for Downloading & Installation VS Code on Windows 10.

→ Installing Jupyter using pip

Using pip with Python is one of the best method to install Jupyter Notebook in your Windows Operating System. Here's how to perform this action in few steps:

Step 1: Check for any Exisiting Update

Use the following command to update pip (to verify if pip is updated):

```
python -m pip install --  
upgrade pip
```

Step 2: Install Jupyter Notebook

After updating the pip version, type the following command to install Jupyter:

```
python -m pip install jupyter
```

Now, the supportive packages will start Installing along with it.

Step 3: Launch Jupyter Notebook

Once the installation is complete, you can launch Jupyter Notebook by typing the following command in the terminal:

```
jupyter notebook
```

This will start the Jupyter Notebook server and automatically open the interface in your default web browser. You can now start creating and running Python notebooks directly from your browser.

Step 4: Verify Jupyter Installation

To confirm that Jupyter Notebook has been installed successfully, type the following command in the terminal:

```
C:\Users\HP>jupyter --version
Selected Jupyter core packages...
IPython          : 8.32.0
ipykernel        : 6.29.5
ipywidgets       : 8.1.5
jupyter_client   : 8.6.3
jupyter_core     : 5.7.2
jupyter_server   : 2.15.0
jupyterlab       : 4.3.5
nbclient         : 0.10.2
nbconvert        : 7.16.6
nbformat         : 5.10.4
notebook         : 7.3.2
qtconsole        : not installed
traitlets        : 5.14.3

C:\Users\HP>
```

This will display the version numbers of Jupyter core packages like IPython, notebook, jupyterlab, etc. If versions are listed (as shown in the image), it means the installation was successful.

→ ABOUT MongoDB:

What is MongoDB?

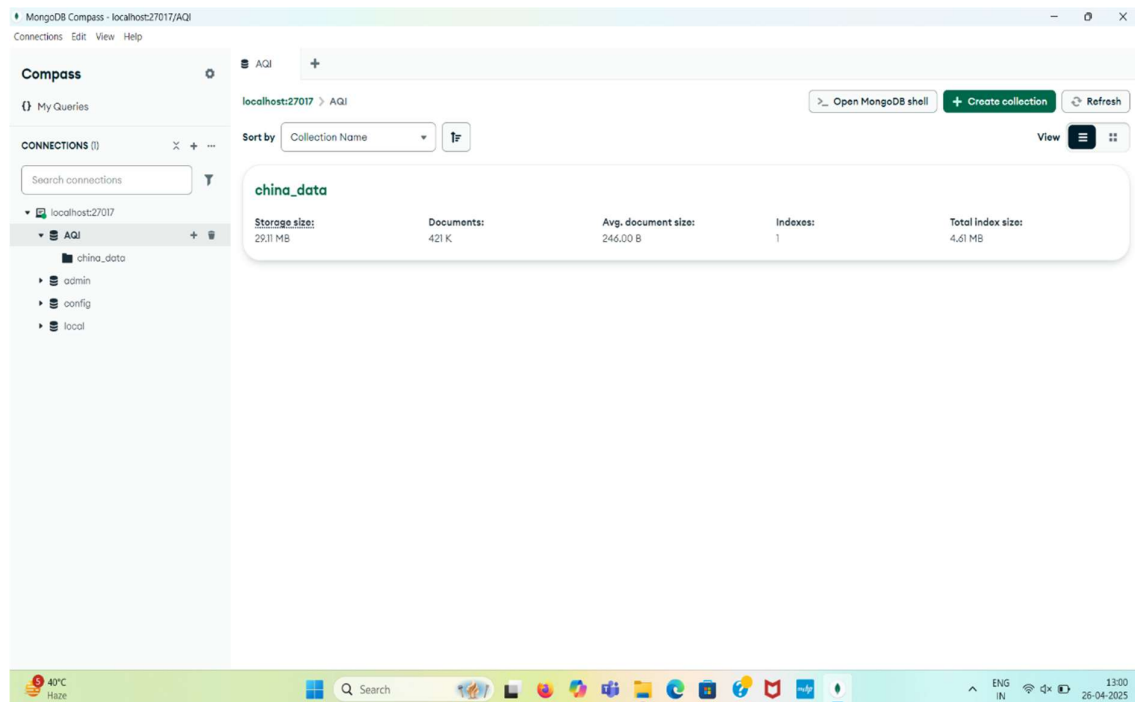
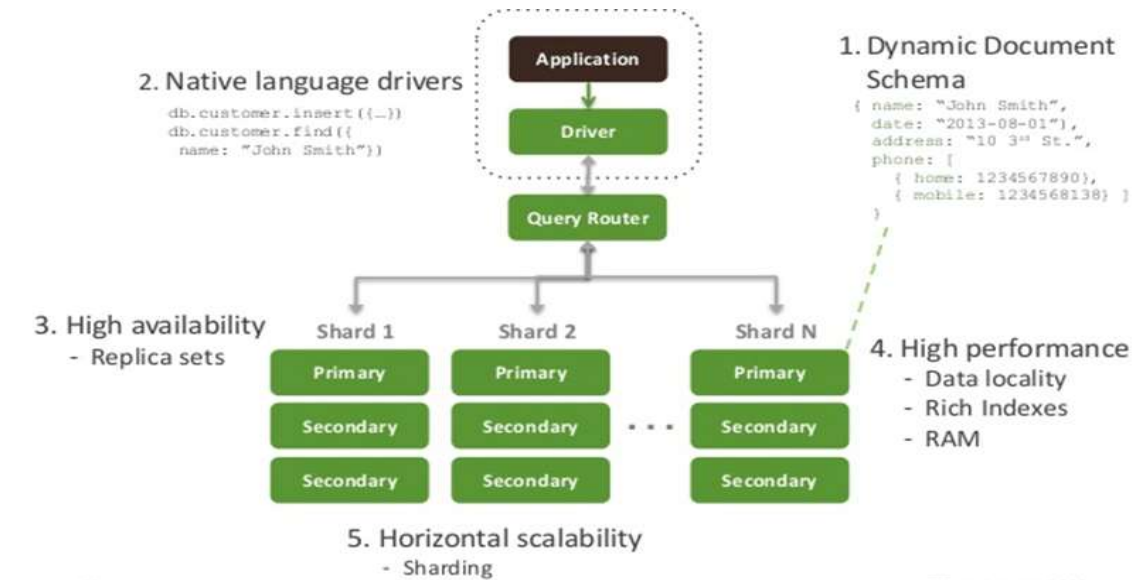
[MongoDB](#) is an open source, non-relational database management system(DBMS) that uses flexible documents instead of tables and rows to process and store various forms of data.

MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time. The document model maps to the objects in your application code, making data easy to work with.

Features of MongoDB:

- **Replication:** A replica set is two or more MongoDB instances used to provide high availability. Replica sets are made of primary and secondary servers. The primary MongoDB server performs all the read and write operations, while the secondary replica keeps a copy of the data. If a primary replica fails, the secondary replica is then used.
- **Scalability:** MongoDB supports vertical and horizontal scaling. Vertical scaling works by adding more power to an existing machine, while horizontal scaling works by adding more machines to a user's resources.
- **Load balancing:** MongoDB handles load balancing without the need for a separate, dedicated load balancer, through either vertical or horizontal scaling.
- **Schema-less:** MongoDB is a schema-less database, which means the database can manage data without the need for a blueprint.
- **Document:** Data in MongoDB is stored in documents with key-value pairs instead of rows and columns, which makes the data more flexible when compared to SQL databases.

Architecture of MongoDB



MongoDB Compass

This screenshot of MongoDB Compass shows a connection to a local MongoDB server at localhost:27017. Within this server, the database named "AQI" is selected. This database currently contains one collection named "China_data". The overview for "China_data" indicates it occupies 293 MB of storage, contains 421,000 documents, has an average document size of 744.00 B, and has one index with a total index size of 4.6 MB.

Software Requirements Specification (SRS)

Project Title: Air Quality Index (AQI) Calculator

1. Introduction

1.1 Purpose:

The purpose of this project is to create a web-based application that allows users to input air pollutant values (PM_{2.5}, PM₁₀, NO₂, SO₂, CO, O₃) and receive an Air Quality Index (AQI) result, along with its corresponding air quality category.

1.2 Scope:

This application will:

- Allow users to manually input the concentration of major air pollutants.
- Calculate the AQI based on the given input values.
- Display the air quality category (e.g., Good, Moderate, Unhealthy, etc.) according to the AQI.

1.3 Definitions, Acronyms, and Abbreviations:

- AQI: Air Quality Index
- PM_{2.5}: Particulate Matter less than 2.5 micrometers
- PM₁₀: Particulate Matter less than 10 micrometers
- NO₂: Nitrogen Dioxide
- SO₂: Sulfur Dioxide
- CO: Carbon Monoxide
- O₃: Ozone
- ML: Machine Learning
- UI: User Interface

1.4 References:

- Test cases designed earlier for functionality testing.
- DFD for system design references.
- Web page screenshot showing input fields for pollutants.

2. Overall Description

2.1 Product Perspective:

The system will be a standalone web application hosted locally or on a server. It will interact with a backend (Python Flask) to process inputs and produce outputs.

2.2 Product Functions:

- User inputs pollutant concentration values.
- System calculates AQI based on standard formulas.
- System determines and displays the air quality category.

2.3 User Classes and Characteristics:

- General Users: Environmental analysts, students, or citizens interested in checking AQI manually.

2.4 Operating Environment:

- Web browser (Chrome, Edge, Firefox)
- Backend Server: Flask (Python)
- Frontend: HTML, CSS (Bootstrap or custom styling)

2.5 Design and Implementation Constraints:

- Application must follow standard AQI calculations.
- User input validation for numeric values is mandatory.
- AQI categorization must match predefined government standards.

2.6 Assumptions and Dependencies:

- Users will have stable internet or localhost access.
- Inputs provided by users are real-time and accurate.

3. Specific Requirements

3.1 Functional Requirements:

- FR1: The system shall allow users to input PM2.5, PM10, NO₂, SO₂, CO, and O₃ values.
- FR2: The system shall validate the input to ensure they are numerical.
- FR3: The system shall calculate AQI using standard calculation logic.
- FR4: The system shall determine the air quality category based on AQI.
- FR5: The system shall display AQI results and air quality category to users.

3.2 Non-Functional Requirements:

- NFR1: The system shall have a responsive UI.
- NFR2: The system shall respond to inputs within 2 seconds.
- NFR3: The system shall be available 99% of the time.
- NFR4: The system should be easy to use and accessible to users with minimal technical knowledge.

3.3 System Interfaces:

- Web Interface (HTML Forms)
- Flask Backend Server (for computation)

3.4 Performance Requirements:

- Maximum response time after form submission: 2 seconds.
- Handle up to 100 simultaneous users locally.

3.5 Security Requirements:

- Basic input validation to prevent injection attacks.
- Secure server hosting (for production environments).

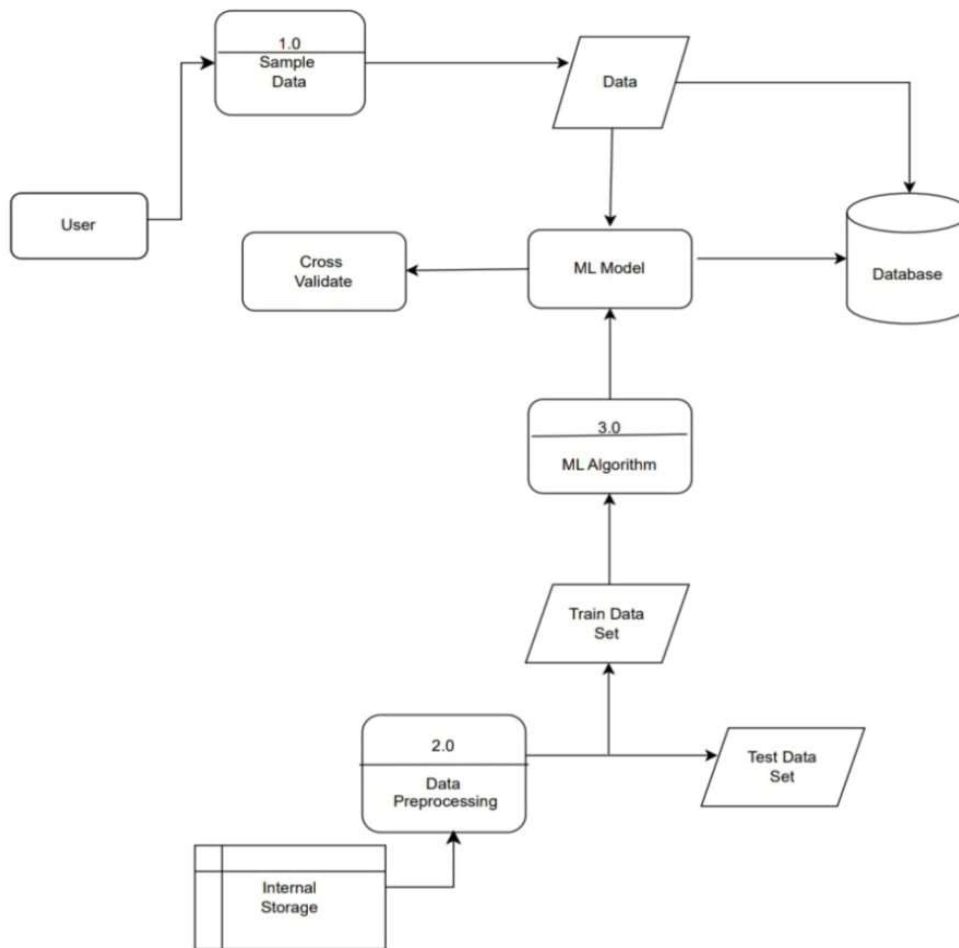
4. Appendices:

A1: Air Quality Categories:

- 0-50: Good
- 51-100: Moderate
- 101-150: Unhealthy for Sensitive Groups
- 151-200: Unhealthy
- 201-300: Very Unhealthy
- 301-500: Hazardous

DESIGN

→ AQI CALCULATOR (LEVEL 2 DFD)



Explanation of System Architecture of the Proposed System

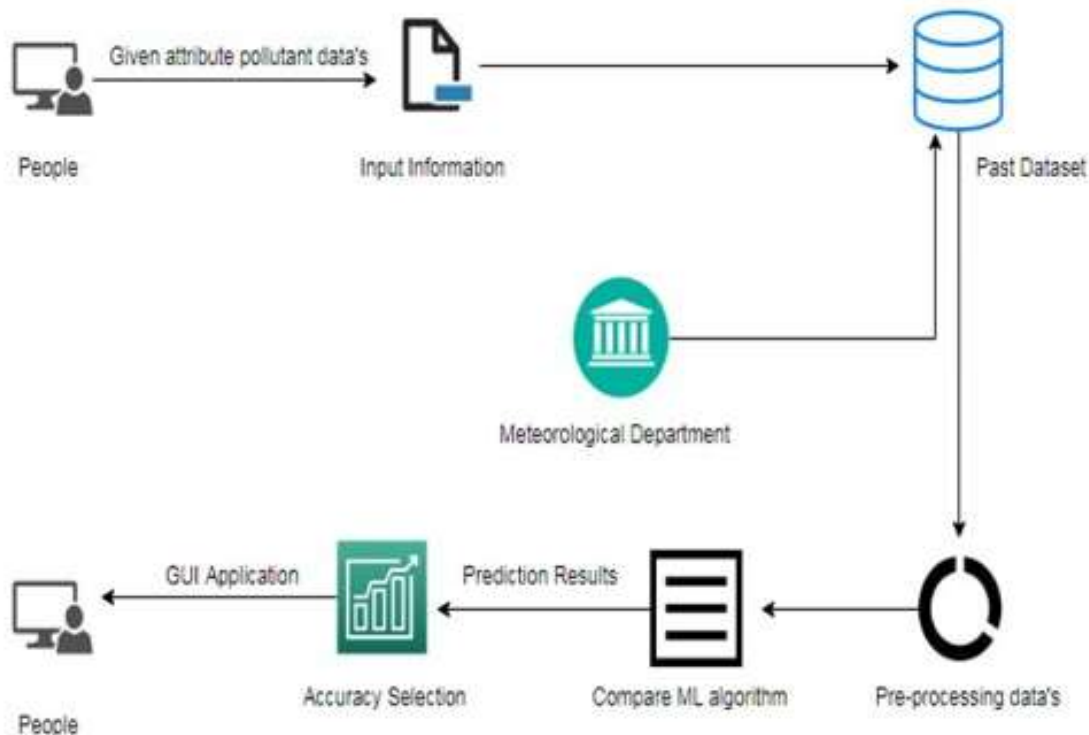


Figure 1.1: System Architecture Of The Proposed System

The system architecture shown here, represents an intelligent framework designed for analyzing pollutant data and generating predictions using machine learning (ML) algorithms. This system is focused on leveraging both historical and real-time data to assist users in understanding environmental conditions and potential risks associated with air pollutants.

1. People (Data Providers):

Individuals or field operators serve as the primary sources of data collection. They gather and submit relevant pollutant attribute data, such as particulate matter (PM2.5, PM10), nitrogen dioxide, sulfur dioxide, ozone levels, etc. These values reflect current environmental conditions and form the initial input to the system.

2. Input Information Collection:

The input information is digitized and transferred into the system through forms, sensors, or manual uploads. This component acts as the entry point for the system to ingest raw data from various external sources.

3. Past Dataset Integration:

The system accesses historical datasets that contain previously recorded pollutant data, meteorological records, and prior prediction results. This dataset acts as the knowledge base for training and validating machine learning models.

4. Meteorological Department Data:

To enhance prediction accuracy, climate and atmospheric information from meteorological departments—such as humidity, wind speed, rainfall, and temperature—is integrated into the system. These parameters help in identifying weather-dependent pollution patterns.

5. Data Pre-processing:

The collected data (both new and historical) undergoes pre-processing. This includes cleaning (removal of missing or inconsistent values), normalization, and transformation of attributes into a suitable format for model input. Pre-processing ensures data quality and model reliability.

6. Machine Learning Algorithm Comparison:

The system applies multiple ML algorithms (such as Random Forest, Support Vector Machine, Decision Trees, or Neural Networks) to the pre-processed dataset. Each algorithm's performance is evaluated using metrics like accuracy, precision, recall, and F1-score. The best-performing model is selected for prediction.

7. Prediction Results:

The selected model generates predictions regarding pollutant

behaviour or air quality levels. These results are then prepared for visualization and delivery to the end-user.

8. Accuracy Selection and GUI Display:

The system provides an interactive Graphical User Interface (GUI) where users can view and compare the performance of different ML models. They can choose models based on accuracy and interpret the prediction results in a user-friendly format, such as graphs, tables, or alerts.

9. User Interaction and Feedback:

Finally, users receive the prediction outcomes and insights through the GUI. They may use this information for environmental monitoring, decision-making, or taking preventive actions against pollution-related hazards. The system also supports iterative improvements based on user feedback.

CODING

Data Preprocessing and Cleaning-Air Quality Dataset

1. Importing Required Libraries

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

This block imports essential Python libraries:

- numpy is used for numerical operations.
- pandas are crucial for data manipulation and analysis.
- seaborn is used for statistical data visualization.
- matplotlib.pyplot is used for creating static, animated, and interactive plots.

2. Reading CSV Files into DataFrames

```
aot=pd.read_csv("PRSA_Data_Aotizhongxin_20130301-20170228.csv")
chan=pd.read_csv("PRSA_Data_Changping_20130301-20170228.csv")
ding=pd.read_csv("PRSA_Data_Dingling_20130301-20170228.csv")
dong=pd.read_csv("PRSA_Data_Dongsi_20130301-20170228.csv")
guan=pd.read_csv("PRSA_Data_Guanyuan_20130301-20170228.csv")
guch=pd.read_csv("PRSA_Data_Gucheng_20130301-20170228.csv")
hua=pd.read_csv("PRSA_Data_Huaiou_20130301-20170228.csv")
nong=pd.read_csv("PRSA_Data_Nongzhanguan_20130301-20170228.csv")
shu=pd.read_csv("PRSA_Data_Shunyi_20130301-20170228.csv")
tian=pd.read_csv("PRSA_Data_Tiantan_20130301-20170228.csv")
wan=pd.read_csv("PRSA_Data_Wanliu_20130301-20170228.csv")
wans=pd.read_csv("PRSA_Data_Wanshouxigong_20130301-20170228.csv")
```

Each line reads a CSV file containing air quality data from different monitoring stations in Beijing, China. The filenames suggest the data ranges from January 1, 2013, to

February 28, 2017. Each file is loaded into a separate panda DataFrame and assigned a variable name based on the station.

3. Combining All DataFrames

```
dfs=[aot,chan,ding,dong,guan,guch,hua,nong,shu,tian,wan,wans]  
air=pd.concat(dfs)
```

- A list dfs is created that contains all the individual DataFrames.
- pd.concat(dfs) merges all the DataFrames vertically (stacked one after the other) into a single DataFrame called air.

4. Resetting the Index

After concatenating, the original indices from the individual DataFrames may persist. Resetting the index will clean up the DataFrame and provide a fresh, continuous index.

```
air.reset_index(drop=True, inplace = True)
```


5. Displaying the First Few Rows

```
print(air.head())
```

	No	year	month	day	hour	PM2.5	PM10	SO2	NO2	CO	O3	TEMP
0	1	2013	3	1	0	4.0	4.0	4.0	7.0	300.0	77.0	-0.7
1	2	2013	3	1	1	8.0	8.0	4.0	7.0	300.0	77.0	-1.1
2	3	2013	3	1	2	7.0	7.0	5.0	10.0	300.0	73.0	-1.1
3	4	2013	3	1	3	6.0	6.0	11.0	11.0	300.0	72.0	-1.4
4	5	2013	3	1	4	3.0	3.0	12.0	12.0	300.0	72.0	-2.0

	PRES	DEWP	RAIN	wd	WSPM	station
0	1023.0	-18.8	0.0	NNW	4.4	Aotizhongxin
1	1023.2	-18.2	0.0	N	4.7	Aotizhongxin
2	1023.5	-18.2	0.0	NNW	5.6	Aotizhongxin
3	1024.5	-19.4	0.0	NW	3.1	Aotizhongxin
4	1025.2	-19.5	0.0	N	2.0	Aotizhongxin

- **Date Information:** Year, month, day, and hour.
- **Pollutants:** PM2.5, PM10, SO2, NO2, CO, O3.
- **Meteorological Data:** Temperature (TEMP), Pressure (PRES), Dew Point (DEWP), Rainfall (RAIN), Wind Direction (WD), and Wind Speed (WSPM).
- **Station Name:** Identifies the source of each observation.

6. Checking for Missing Values

```
air.isnull().sum()
```

year	0
month	0
day	0
hour	0
PM25	8739
PM10	6449
SO2	9021
NO2	12116
CO	20701
O3	13277
TEMP	398
PRES	393
DEWP	403
RAIN	390
wd	1822
WSPM	318
station	0
dtype: int64	

This function:

- Identifies missing values.

- Counts the number of missing entries per column.

The output (0 for all columns) confirms that there are no missing values in the dataset, meaning data imputation is not required

7. Missing Value Treatment

The missing values in the dataset are filled using different strategies:

- **Mean Imputation for Numerical Columns**

```
air['PM2.5'] = air['PM2.5'].fillna(air['PM2.5'].mean())
air['PM10'] = air['PM10'].fillna(air['PM10'].mean())
air['SO2'] = air['SO2'].fillna(air['SO2'].mean())
air['NO2'] = air['NO2'].fillna(air['NO2'].mean())
air['CO'] = air['CO'].fillna(air['CO'].mean())
air['O3'] = air['O3'].fillna(air['O3'].mean())
air['TEMP'] = air['TEMP'].fillna(air['TEMP'].mean())
air['PRES'] = air['PRES'].fillna(air['PRES'].mean())
air['DEWP'] = air['DEWP'].fillna(air['DEWP'].mean())
air['RAIN'] = air['RAIN'].fillna(air['RAIN'].mean())
air['WSPM'] = air['WSPM'].fillna(air['WSPM'].mean())
```

Missing values in columns like PM2.5, PM10, SO2, NO2, CO, O3, TEMP, PRES, DEWP, RAIN, and WSPM are replaced with the mean of each column. This prevents data loss and maintains statistical integrity.

- **Mode Imputation for Categorical Column**

```
air['wd'] = air['wd'].fillna(air['wd'].mode()[0])
```

The WD (wind direction) column is categorical, so missing values are replaced with the most frequent value. This ensures logical consistency.

- **Validating Missing Value Treatment**

```
air.isnull().sum()
```

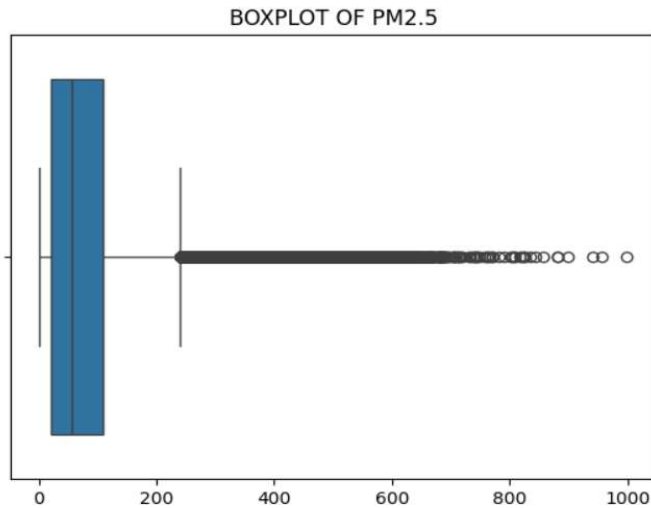
	0
No	0
year	0
month	0
day	0
hour	0
PM2.5	0

The **isnull().sum()** function confirms that all missing values have been handled, displaying 0 for every column.

8. Boxplot Analysis of PM2.5 and PM10 (Before Filtering)

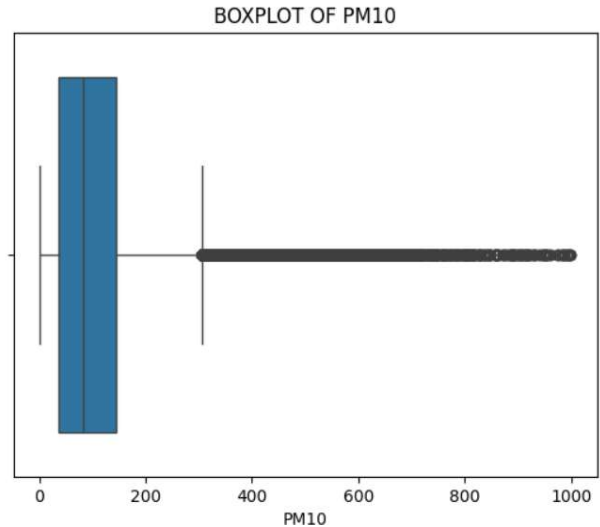
```
sns.boxplot(x=air['PM25'])  
plt.title('BOXPLOT OF PM2.5')
```

Text(0.5, 1.0, 'BOXPLOT OF PM2.5')



```
sns.boxplot(x=air['PM10'])  
plt.title('BOXPLOT OF PM10')
```

Text(0.5, 1.0, 'BOXPLOT OF PM10')



- The **sns.boxplot()** function from the Seaborn library is used to create boxplots for PM2.5 and PM10 values.
- Boxplots help in detecting outliers, which appear as individual points outside the whiskers of the plot.
- The **plt.title()** function assigns a title to each plot for better visualization.

Observations:

- Both PM2.5 and PM10 distributions contain numerous extreme outliers, especially beyond the 300–400 range.
- These outliers could be a result of data collection errors, sudden pollution spikes, or measurement anomalies.
- The presence of outliers can negatively impact machine learning models, making it necessary to filter the data.
- PM2.5 and PM10 data appear right-skewed, as seen in the boxplots. The presence of numerous high-value outliers (points above the upper whisker) suggests a long right tail in

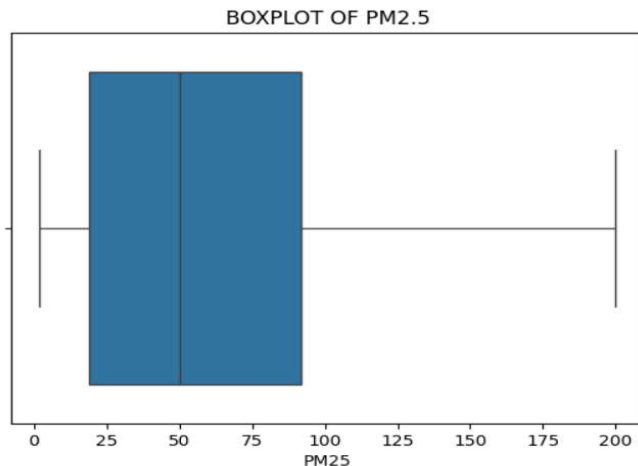
the distribution.

- This skewness indicates that most of the air quality readings are on the lower end, but there are occasional extreme pollution spikes.

9. Filtering Data and Generating Refined Boxplots of PM2.5 and PM10

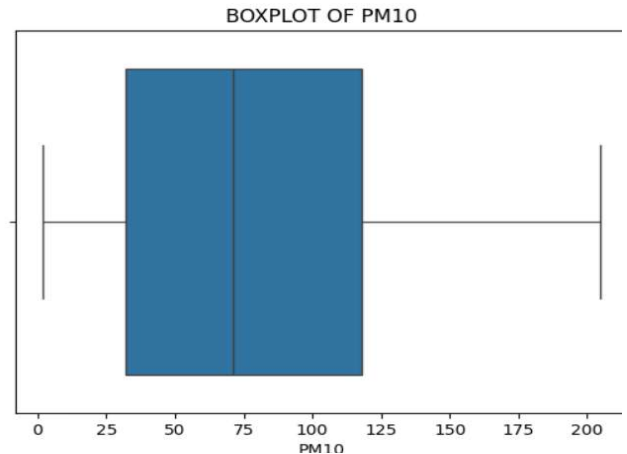
```
filtered_air=air[air['PM25'] <= 200]
sns.boxplot(x=filtered_air['PM25'])
plt.title('BOXPLOT OF PM2.5')
```

```
Text(0.5, 1.0, 'BOXPLOT OF PM2.5')
```



```
filtered_air=air[air['PM10'] <= 205]
sns.boxplot(x=filtered_air['PM10'])
plt.title('BOXPLOT OF PM10')
```

```
Text(0.5, 1.0, 'BOXPLOT OF PM10')
```



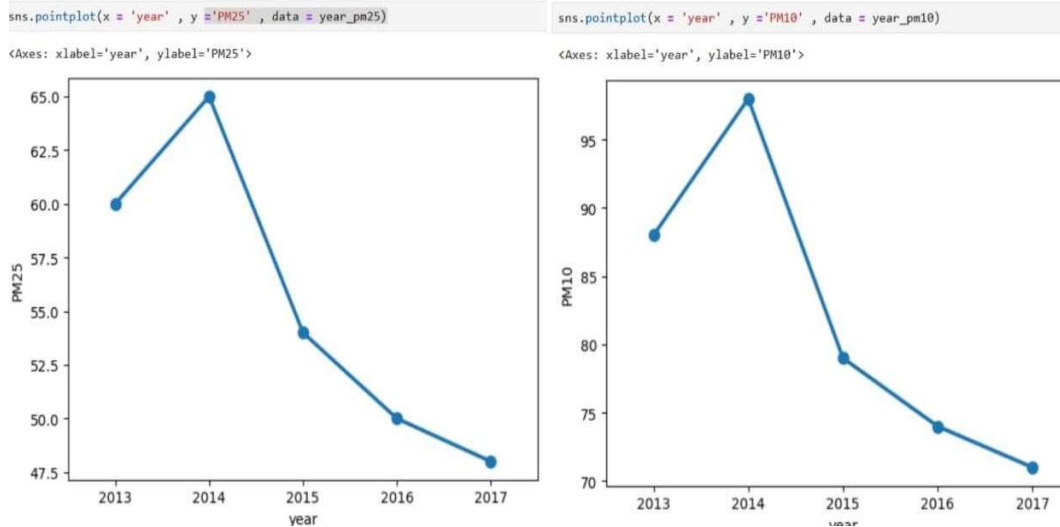
- The dataset is filtered to remove values above 200 for both PM2.5 and PM10, assuming these are extreme outliers.
- After filtering **PM2.5 ≤ 200** and **PM10 ≤ 200**, the boxplots show a more balanced distribution, **reducing the right skewness**.
- The new filtered dataset is used to create refined boxplots.

Observations:

- The updated boxplots show a more compact and symmetrical distribution, indicating better data quality.
- By removing extreme values, the dataset becomes more representative of real-world air quality trends, improving reliability for further analysis and machine learning modeling.

10. Analysis of PM2.5 and PM10 Trends

• YEARLY TRENDS (2013-2017)

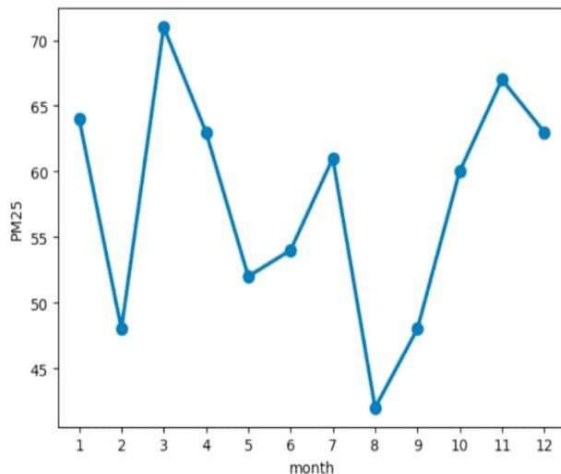


- PM2.5 and PM10 levels peaked in 2014, indicating the highest pollution levels during this period.
- A sharp decline is observed post-2014, reaching the lowest levels in 2017.
- Factors contributing to the peak in 2014 may include industrial emissions, vehicular pollution, and construction activities.
- The decline could be attributed to stricter environmental regulations, improved pollution control measures, and increased public awareness.

• MONTHLY TRENDS(Across a Year)

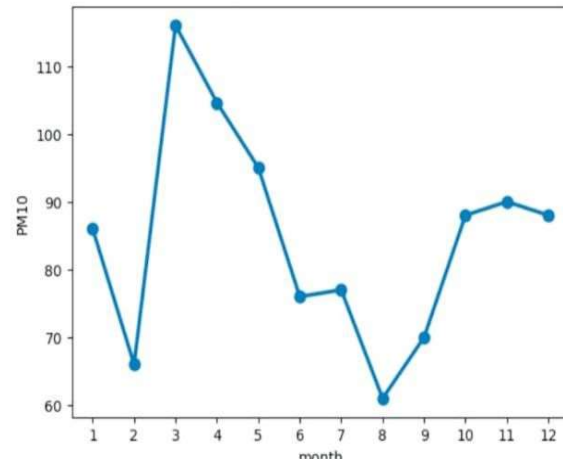
```
sns.pointplot(x = 'month' , y = 'PM25' , data = month_pm25)
```

<Axes: xlabel='month', ylabel='PM25'>



```
sns.pointplot(x = 'month' , y = 'PM10' , data = month_pm10)
```

<Axes: xlabel='month', ylabel='PM10'>

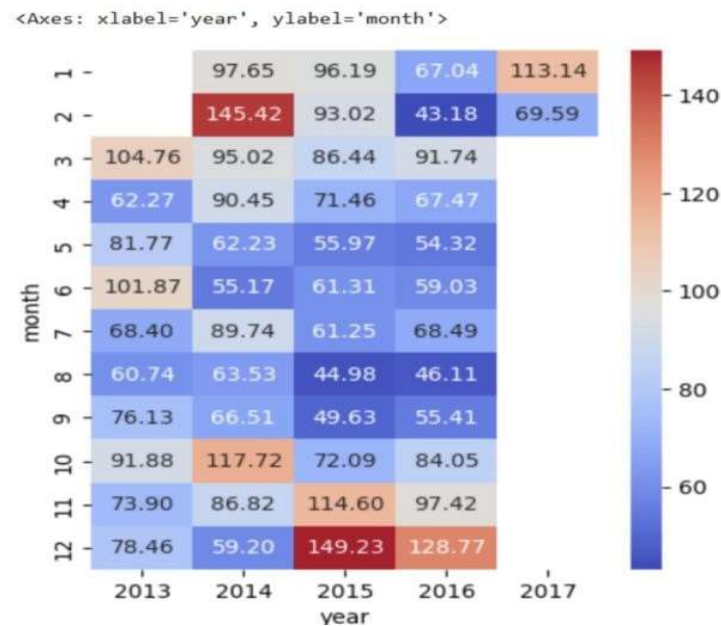


- PM2.5 and PM10 levels fluctuate significantly across months.
- The highest levels are observed in winter months (November-December) due to temperature inversion, lower wind speeds, and increased heating activities, which trap pollutants near the surface.
- The lowest values appear in monsoon months (July-August), likely due to the washout effect caused by rainfall, which helps clear airborne pollutants.
- Summer months (April-May) show moderate pollution levels, possibly due to dust storms and increased vehicular activity.

11. Correlation Analysis Between Different Air Quality Parameters

➤ PM2.5 Heatmap

```
pivot_air=air.pivot_table(index='month',columns='year',values='PM25')  
plt.figure(figsize=(5,5))  
sns.heatmap(pivot_air,annot=True, cmap='coolwarm',fmt=".2f")
```

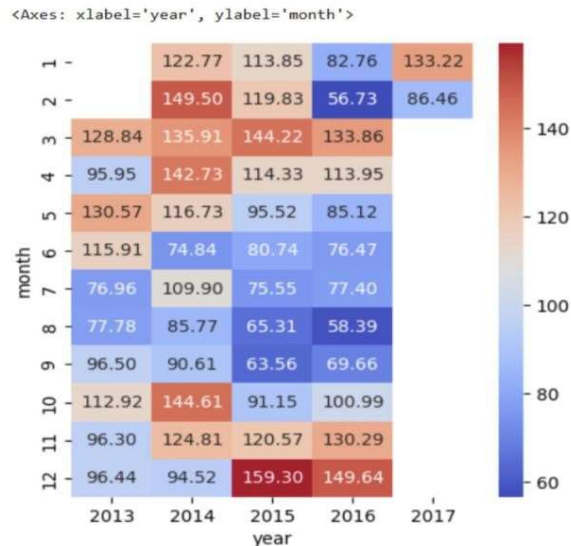


- This heatmap illustrates the distribution of PM2.5 over different years and months. It follows a blue-to-red color scheme, where red zones indicate high pollution levels.
- PM2.5 levels are significantly high in December and February, likely due to increased emissions from heating sources, vehicular traffic, and stagnant atmospheric conditions.
- The summer months exhibit relatively lower PM2.5 concentrations, which could be attributed to better dispersion due to stronger winds.
- The differences in values across years highlight the

influence of policies, urban expansion, and climatic factors.

➤ PM10 Heatmap

```
pivot_air=air.pivot_table(index='month',columns='year',values='PM10')  
plt.figure(figsize=(5,5))  
sns.heatmap(pivot_air,annot=True, cmap='coolwarm',fmt=".2f")
```



- This heatmap visualizes the concentration of PM10 across different months and years. The color gradient from blue (low values) to red (high values) helps in identifying trends in PM10 levels over time.
- PM10 levels tend to be higher in winter months (November to February), possibly due to lower temperatures, increased vehicular emissions, and industrial activities.
- Some summer months also show peaks, indicating dust storms or increased anthropogenic activities.
- The variations across years suggest that pollution control measures and meteorological changes impact PM10 concentration.

➤ Correlation Heatmap



This heatmap represents the correlation between different air pollutants and meteorological factors. It uses a red-to-black color gradient, where:

- Darker shades (close to -1) indicate strong negative correlations.
- Lighter shades (close to +1) indicate strong positive correlations
- Values near 0 suggest weak or no correlation.

Observation:

- PM10 and PM2.5 have a strong positive correlation (≈ 0.88), indicating they are influenced by similar sources.
- NO2, SO2, and CO show strong correlations, suggesting they originate from common sources like traffic emissions and industrial activities.
- Temperature (TEMP) shows a negative correlation

with PM2.5 and PM10, meaning pollution levels tend to rise when temperatures drop.

- Rainfall (RAIN) has a negative correlation with particulate matter, indicating that precipitation helps in reducing air pollution by washing out pollutants.

RANDOM FOREST ALGORITHM

Random Forest is a commonly-used machine learning algorithm, trademarked by Leo Breiman and Adele Cutler, that combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

STEP 1: Import Required Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score
import joblib
from pymongo import MongoClient
```

import numpy as np: Imports the NumPy library, which is used for numerical computations like handling arrays, mathematical functions, and dealing with missing values.

import pandas as pd: Imports the Pandas library, essential for handling datasets in the form of DataFrames (tables). Useful for data manipulation, cleaning, and analysis.

import matplotlib.pyplot as plt: Imports Matplotlib's pyplot module for creating visualizations such as scatter plots, bar charts, and line graphs.

from sklearn.ensemble import RandomForestRegressor: Imports the RandomForestRegressor from scikit-learn, a machine learning model used for predicting continuous values like AQI (Air Quality Index).

from sklearn.model_selection import train_test_split, GridSearchCV: Imports train_test_split for splitting the dataset into training and testing parts, and GridSearchCV for finding the best hyperparameters for the Random Forest model using cross-validation.

from sklearn.preprocessing import StandardScaler: Imports StandardScaler, a tool used to standardize (normalize) feature values so they have a mean of 0 and standard deviation of 1, making machine learning models more efficient.

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score: These are metrics to evaluate the regression model's performance:

- **mean_absolute_error (MAE):** Average of absolute differences between actual and predicted values.
- **mean_squared_error (MSE):** Average of squared differences. Penalizes large errors more.
- **r2_score:** Measures how well predictions fit the data (1 = perfect, 0 = poor).

import joblib : Imports joblib, a library used for saving (serializing) trained machine learning models and scalers into files for later use.

from pymongo import MongoClient : Imports MongoClient from PyMongo to connect the Python application to a MongoDB database, from where the AQI data will be fetched.

STEP 2: Load and Prepare Data

• Connect to MongoDB

```
client = MongoClient("mongodb://localhost:27017/")
db = client["AQI"]
collection = db["china_data"]
```

- The code connects to a local MongoDB server running on localhost at the default port 27017. It then selects the "AQI" database and accesses the "china_data" collection within it. The data from this collection is fetched and stored in a Pandas DataFrame for further analysis.

• Load Data From MongoDB

```
air = pd.DataFrame(list(collection.find()))
```

- This line of code fetches all the data from the "china_data" collection in MongoDB, converts it into a list, and then creates a Pandas DataFrame called air from the list. The DataFrame makes it easier to manipulate and analyze the data.

• Drop unwanted columns

```
air.drop(columns=["_id", "station"], inplace=True, errors="ignore")
```

- Drops the "_id" and "station" columns from the air DataFrame

• Convert time column

```
air["hour"] = pd.to_datetime(air["hour"], errors="coerce")
```

- converts the "hour" column to a date/time format.

• Handling Outliers

```
air_cleaned = air.copy()
for column in
air_cleaned.select_dtypes(include=['number']).columns:
    Q1 = air_cleaned[column].quantile(0.25)
    Q3 = air_cleaned[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    air_cleaned[column] = air_cleaned[column].apply(
        lambda x: x if lower_bound <= x <= upper_bound
    else np.nan
    )
```

- Detect and replaces outliers in numerical columns of the air DataFrame with NaN vlues using the Interquartile Range(IQR) method

• Check for Missing values

```
air_cleaned.isnull().sum()
round(air_cleaned.isnull().sum()/len(air_cleaned.index),
4)*100
```

- Check the number and percentage of missing values in the dataset

• Fill Missing values

```
air_cleaned['PM25']=air_cleaned['PM25'].fillna(air_cleaned
['PM25'].mean())
air_cleaned['wd']=air_cleaned['wd'].fillna(air_cleaned['wd
'].mode()[0])
```


- Fill missing Numeric data with the mean and categorical column wd with its mode

STEP 3: AQI Calculation

• Sub-Index Calculation Function

```
def calculate_sub_index(c, breakpoints):
    if pd.isna(c):
        return None

    for i in range(len(breakpoints) - 1):
        if breakpoints[i][0] <= c <=
breakpoints[i][1]:
            return ((breakpoints[i][3] -
breakpoints[i][2]) / (breakpoints[i][1] -
breakpoints[i][0])) * (c - breakpoints[i][0]) +
breakpoints[i][2]

    return None
```

- This function calculate_sub_index computes the Air Quality Index (AQI) sub-index for a given pollutant concentration c, using a set of defined breakpoints.
- pd.isna(c) returns None if the concentration is missing (NaN).
- The breakpoints list contains ranges of pollutant concentrations and their corresponding AQI values, like:

[(low_c, high_c, low_aqi, high_aqi), ...]

For example-(0, 50, 0, 50) means concentration 0–50 maps to AQI 0–50.

- The loop finds the range where c fits.
- If c is in a given range, the AQI sub-index is calculated by linearly interpolating between the AQI values of that range:

$$\text{Sub_index} = ((\text{high_aqi} - \text{low_aqi}) / (\text{high_c} - \text{low_c})) * (\text{c} - \text{low_c}) + \text{low_aqi}$$

- If c doesn't fit any given breakpoint range, it returns None.

- **Define Breakpoints for each pollutant**

```
pm25_bp = [(0, 30, 0, 50), (31, 60, 51, 100), (61, 90, 101, 200), (91, 120, 201, 300), (121, 250, 301, 400), (251, 500, 401, 500), (501, 900, 501, 600)]
```

```
pm10_bp = [(0, 50, 0, 50), (51, 100, 51, 100), (101, 250, 101, 200), (251, 350, 201, 300), (351, 430, 301, 400), (431, 600, 401, 500), (601, 1000, 501, 600)]
```

```
no2_bp = [(0, 40, 0, 50), (41, 80, 51, 100), (81, 180, 101, 200), (181, 280, 201, 300), (281, 400, 301, 400), (401, 1000, 401, 500)]
```

```
so2_bp = [(0, 40, 0, 50), (41, 80, 51, 100), (81, 380, 101, 200), (381, 800, 201, 300), (801, 1600, 301, 400), (1601, 3500, 401, 500)]
```

```
co_bp = [(0, 1, 0, 50), (1.1, 2, 51, 100), (2.1, 10, 101, 200), (10.1, 17, 201, 300), (17.1, 34, 301, 400), (34.1, 50, 401, 500), (51, 10000, 501, 600)]
```

```
o3_bp = [(0, 50, 0, 50), (51, 100, 51, 100), (101, 168, 101, 200), (169, 208, 201, 300), (209, 748, 301, 400), (749, 1000, 401, 500), (1001, 1500, 501, 600)]
```

- These Python lists define the Air Quality Index (AQI) breakpoints for key pollutants (PM2.5, PM10, NO2, SO2, CO, O3). Each list contains tuples mapping pollutant concentration ranges to corresponding AQI ranges: (low_pollutant, high_pollutant, low_aqi, high_aqi). These adjusted breakpoints are used to convert pollutant in measurements to a standardized AQI value relevant to the project's dataset.

- **Calculate AQI for each row**

```
def calculate_aqi(row):
    sub_indices = [
        calculate_sub_index(row['PM2.5'], pm25_bp),
        calculate_sub_index(row['PM10'], pm10_bp),
        calculate_sub_index(row['NO2'], no2_bp),
        calculate_sub_index(row['SO2'], so2_bp),
        calculate_sub_index(row['CO'], co_bp),
        calculate_sub_index(row['O3'], o3_bp)
    ]
    return max(filter(None, sub_indices),
                default=None)
```

- The **calculate_aqi(row)** function calculates the Air Quality Index (AQI) for a given data row. It does this by computing sub-indices for six major pollutants — PM2.5, PM10, NO2, SO2, CO, and O3 — using the calculate_sub_index function along with their respective breakpoint values. After calculating these sub-indices, it filters out any missing (None) values and selects the maximum among them, which represents the final AQI for that row. If all pollutant values are missing, the function returns None. This method ensures that the AQI reflects the pollutant with the most severe impact on air quality.

- **Apply AQI Calculation**

```
air['AQI'] = air.apply(calculate_aqi, axis=1)
```

- calculate AQI for each row in the 'air' DataFrame. It applies the calculate_aqi function to each row (axis=1), using the pollutant data within that row to determine the corresponding AQI value, and stores the result in a new 'AQI' column.

- **Drop rows without AQI**

```
air_cleaned.dropna(subset=['AQI'], inplace=True)
```

- Drop any rows where AQI could not be calculated.

STEP 4: Preparing the Dataset for Modeling

• Feature and Target Selection

```
X = air[['PM2.5', 'PM10', 'NO2', 'SO2', 'CO', 'O3']]
```

```
y = air['AQI']
```

- Features (X): Pollutant concentrations (PM2.5, PM10, NO2, SO2, CO, O3) to predict
- Target Variable (y): Air Quality Index (AQI) values.

• Split into Training and Testing Sets

```
X_train, X_test, y_train, y_test =  
train_test_split(X,y,test_size=0.2, random_state=42)
```

- Training set (X_train, y_train): Used to train the model (80% of data).
- Testing set (X_test, y_test): Used to evaluate the model's performance(20% of data).
- The split is done using train_test_split with:

test_size=0.2: 20% of data for testing.

random_state=42: Ensures reproducibility of the split.

• Handling Missing Values Again

```
X_train.fillna(X_train.mean(), inplace=True)
```

```
X_test.fillna(X_test.mean(), inplace=True)
```

```
y_train.dropna(inplace=True)
```

```
y_test.dropna(inplace=True)
```

- Ensure no missing values remain in the training and testing sets.

• Feature scaling

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

- Fits and transforms X_train: Calculates mean and standard deviation from X_train and scales it.
- Transforms X_test: Applies the same scaling (using X_train's mean and standard deviation) to X_test.
- This ensures that both datasets have similar scales, improving model performance and stability.

Step 5: Model Training with Random Forest and Grid Search

• Hyperparameter Tuning

```
param_grid = {  
    'n_estimators': [100, 200, 300],  
    'max_depth': [10, 15, 20],  
    'min_samples_split': [2, 5, 10],  
    'min_samples_leaf': [1, 2, 4],  
    'max_features': ['sqrt', 'log2']  
}
```

- Defines a grid of hyperparameters to search for the best combination.

• Initialize and Fit GridSearchCV

```
grid_search = GridSearchCV(  
    estimator=RandomForestRegressor(random_state=42),  
    param_grid=param_grid,  
    cv=3,  
    n_jobs=-1,  
    scoring='r2',  
    verbose=2
```

)

```
grid_search.fit(X_train_scaled[:50000], y_train[:50000])
```

- This code performs a grid search to optimize hyperparameters for a Random Forest Regressor model, using R-squared as the evaluation metric and 3-fold cross-validation. It utilizes all CPU cores for faster computation and provides detailed output. The grid search is fit to a subset of 50,000 training samples, aiming to find the best combination of hyperparameters that improve model performance.

Step 6: Predictions and Evaluation

• Make Predictions

```
rf_model = grid_search.best_estimator_  
y_pred = rf_model.predict(X_test_scaled)
```

- This code retrieves the best-performing Random Forest Regressor model from the grid search and uses it to make predictions on the scaled test data (X_test_scaled). The predicted values are stored in y_pred.

• Evaluate Model Performance

```
mae = mean_absolute_error(y_test, y_pred)  
mse = mean_squared_error(y_test, y_pred)  
r2 = r2_score(y_test, y_pred)
```

- The model's performance is evaluated using three metrics:
 1. MAE (Mean Absolute Error): Average difference between predicted and actual values is 1.43.
 2. MSE (Mean Squared Error): Average squared difference between predicted and actual values is 6.23.
 3. R2 (R-squared): Model explains 100% of the variance in the data, indicating a very good fit (R2 = 1.00).
- These metrics suggest the model is performing well.

Step 7: Saving the model and Scaler

```
joblib.dump(rf_model, "models/ran_model.pkl")  
joblib.dump(scaler, "models/ran_scaler.pkl")
```

- This code saves two important components:
 - Random Forest Model(rf_model): Saved as "ran_model.pkl" in the "models" directory.
 - Scaler(scaler): Saved as "ran_scaler.pkl" in the "models" directory.

This allows for easy loading and reuse of the trained model and scaler in future applications.

→ Key Features of Random Forest algorithm:

- **Ensemble Learning Method:**
Combines predictions from multiple decision trees to improve accuracy and control overfitting.
- **Bootstrap Aggregation (Bagging):**
Each tree is trained on a random subset of the data (with replacement), which reduces variance.
- **Random Feature Selection:**
At each split in a tree, a random subset of features is considered, adding diversity among the trees.
- **Handles Overfitting Well:**
By averaging multiple trees, Random Forest reduces the risk of overfitting seen in individual decision trees.
- **High Accuracy:**
Often provides high prediction accuracy compared to other algorithms, especially on complex datasets.
- **Handles Missing Values:**

Can maintain accuracy even when part of the data is missing.

- **Works with Both Categorical and Numerical Data:**

Can handle a mix of data types without the need for extensive preprocessing.

- **Feature Importance Estimation:**

Can be used to rank features by importance, which is useful for feature selection and interpretation.

- **Robust to Noise and Outliers:**

The averaging of multiple trees makes the model less sensitive to noisy data.

- **Parallelizable:**

Trees can be trained independently, making it suitable for parallel computing and large-scale data.

→ **Advantages and Disadvantages of Random Forest Algorithm**

Random Forest is a popular ensemble learning algorithm known for its high accuracy and robustness. However, like all algorithms, it has its own set of advantages and disadvantages. Below is a detailed overview of the pros and cons of using the Random Forest algorithm in machine learning tasks.

Advantages of Random Forest :-

- High accuracy due to the aggregation of multiple decision trees.
- Reduces overfitting by averaging multiple trees.
- Works well with both classification and regression problems.
- Handles missing values and maintains accuracy.
- Provides estimates of feature importance, aiding in feature selection.
- Robust to outliers and noise in the data.

- Can handle large datasets efficiently.
- Supports parallel processing due to independent tree construction.

Disadvantages of Random Forest :-

- Model complexity can make it slower to predict compared to single decision trees.
- Less interpretable than individual decision trees.
- Training can be time-consuming for very large datasets with many trees.
- Can be memory-intensive as it stores multiple trees.
- Tuning the number of trees and other hyperparameters can be challenging.

SOURCE CODE

1. index.html (input concentration of different pollutants)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,
initial-scale=1.0"/>
  <title>Air Quality Index</title>
  <style>
    body {
      font-family: 'Segoe UI', Arial, sans-serif;
      margin: 0;
      padding: 0;
      background: linear-gradient(to right, #3a7bd5,
#00d2ff);
      color: white;
      height: 100vh;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: start;
      padding-top: 60px;
    }
    .navbar {
      position: fixed;
      top: 0;
      width: 100%;
```

```
background: rgba(0, 0, 0, 0.3);
padding: 15px 0;
text-align: center;
font-size: 1.5rem;
font-weight: bold;
box-shadow: 0 2px 6px rgba(0,0,0,0.2);
}
.datetime-box {
position: absolute;
top: 45px;
right: 50px;
padding-left: 20px;
padding-right: 20px;
padding-top: 15px;
padding-bottom: 15px;
text-align: right;
font-size: 17px;
font-family: Arial;
color: rgb(40, 37, 37);
font-weight: bold;
background-color:rgb(255, 252, 252);
border-radius: 10px;
}
.container {
background: rgba(0, 0, 0, 0.4);
padding: 40px;
border-radius: 15px;
box-shadow: 0 0 20px rgba(255, 255, 255, 0.2);
max-width: 400px;
width: 90%;
```

```

        margin-top: 100px;
        text-align: center;
    }
    .container h2 {
        margin-bottom: 25px;
    }
    input[type="number"] {
        width: 100%;
        padding: 12px 15px;
margin: 10px 0;
        border: none;
        border-radius: 8px;
        box-sizing: border-box;
        font-size: 1rem;
    }
    button {
        background-color: #fff;
        color: #007acc;
        font-weight: bold;
        padding: 12px 20px;
        border: none;
        border-radius: 8px;
        cursor: pointer;
        font-size: 1rem;
        transition: all 0.3s ease;
        margin-top: 10px;
    }
    button:hover {
        background-color: #007acc;
        color: white;
    }

```

```

        transform: scale(1.05);
    }
    @media (max-width: 600px) {
        .container {
            padding: 25px;
        }
        .navbar {
            font-size: 1.2rem;
        }
    }
</style>
</head>
<body>

<!-- Live Date/Time Box -->
    <div class="datetime-box">
        <div class="date" id="current-date">Loading
date...</div>
        <div class="time" id="current-time">Loading
time...</div>
    </div>
    <div class="navbar">
        <h1>Air Quality Index</h1>
    </div>

    <div class="container">
        <h2>Check Air Quality</h2>
        <form method="POST" id="aqi-form"
action="/result.html">
            <input type="number" name="PM25"
placeholder="PM2.5 (µg/m³)" required>

```

```

        <input type="number" name="PM10"
placeholder="PM10 (µg/m³)" required>
        <input type="number" name="NO2"
placeholder="NO2 (µg/m³)" required>
        <input type="number" name="SO2"
placeholder="SO2 (µg/m³)" required>
        <input type="number" name="CO" placeholder="CO
(mg/m³)" required>
<input type="number" name="O3" placeholder="O3
(µg/m³)" required>
        <button type="submit">Submit</button>
    </form>
</div>
<script>
    function updateDateTime() {
        const now = new Date();
        const days = ["Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday"];
        const months = ["January", "February", "March",
"April", "May", "June", "July",
                        "August", "September",
"October", "November", "December"];
        const day = days[now.getDay()];
        const month = months[now.getMonth()];
        const date = now.getDate();
        const year = now.getFullYear();
        let hours = now.getHours();
        let minutes = now.getMinutes();
        let seconds = now.getSeconds();
        const ampm = hours >= 12 ? 'PM' : 'AM';
        hours = hours % 12 || 12;
        minutes = minutes < 10 ? '0' + minutes :
minutes;

```

```

        seconds = seconds < 10 ? '0' + seconds :
seconds;

        document.getElementById('current-
date').textContent = `${day}, ${month} ${date},
${year}`;

        document.getElementById('current-
time').textContent = `${hours}:${minutes}:${seconds}
${ampm}`;
    }
    setInterval(updateDateTime, 1000);
    updateDateTime();

    document.getElementById("aqi-
form").addEventListener("submit", async
function(event) {
        event.preventDefault();
        const formData = new FormData(event.target);
        const params = new URLSearchParams(formData);

        try {
            const response = await
fetch(`/predict?${params.toString()}`);
            const data = await response.json();
            if (data.aqi !== undefined) {
                window.location.href =
`/result?aqi=${data.aqi}`;
            } else {
                alert("Error calculating AQI.");
            }
        } catch (err) {
            alert("Failed to connect to server.");
        }
    });

```

```
</script>
</body></html>
```

2. result.html (shows calculated AQI)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,
initial-scale=1.0"/>
  <title>AQI Result</title>
  <style>
    body {
      font-family: 'Segoe UI', Arial, sans-serif;
      margin: 0;
      padding: 0;
      background: linear-gradient(to right, #3a7bd5,
#00d2ff);
      color: white;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      height: 100vh;
      text-align: center;
    }
    .result-box {
      background: rgba(0, 0, 0, 0.4);
      padding: 30px 40px;
      border-radius: 15px;
```



```

        box-shadow: 0 0 20px rgba(255,255,255,0.2);
        max-width: 400px;
    }
    h1 {
        font-size: 2rem;
        margin-bottom: 15px;
    }
    #aqi-result {
font-size: 2rem;
        font-weight: bold;
        margin-bottom: 10px;
    }
    #advisory {
        font-size: 1.1rem;
        margin-top: 10px;
        padding: 10px;
        border-radius: 10px;
    }
    .good { background-color: #00e40044; color:
#00e400; }
    .moderate { background-color: #ffff0044; color:
#ff0; }
    .usg { background-color: #ff7e0044; color:
#ff7e00; }
    .unhealthy { background-color: #ff000044; color:
#ff0000; }
    .very-unhealthy { background-color: #8f3f9744;
color: #8f3f97; }
    .hazardous { background-color: #7e002344; color:
#7e0023; }
    .back-link {
        display: inline-block;

```

```

        margin-top: 20px;
        padding: 10px 20px;
        border-radius: 8px;
        background-color: #ffffff22;
        color: white;
        text-decoration: none;
        font-weight: bold;
        border: 1px solid white;
        transition: background-color 0.3s ease;
    }
    .back-link:hover {
        background-color: #ffffff44;
    }
</style>
</head>
<body>
    <div class="result-box">
        <h1>Your Calculated AQI</h1>
        <p id="aqi-result">Calculating...</p>
        <div id="advisory" class="">Loading
advisory...</div>
        <a class="back-link" href="/">← Back to form</a>
    </div>

    <script>
        const params = new
URLSearchParams(window.location.search);
const aqi = parseInt(params.get("aqi"));
        const resultElem = document.getElementById("aqi-
result");

```

```

const advisoryElem =
document.getElementById("advisory");

if (!isNaN(aqi)) {
    resultElem.textContent = `AQI : ${aqi}`;

    let message = "";
    let category = "";

    if (aqi <= 50) {
        message = "Air quality is considered
satisfactory. It's a great day to be outside!";
category = "good";
    } else if (aqi <= 100) {
        message = "Air quality is acceptable. Some
pollutants may be a concern for sensitive
individuals.";
        category = "moderate";
    } else if (aqi <= 150) {
        message = "Unhealthy for Sensitive Groups.
Children, elderly, and those with respiratory issues
should reduce outdoor exertion.";
        category = "usg";
    } else if (aqi <= 200) {
message = "Unhealthy. Everyone may begin to
experience health effects; sensitive groups should
avoid outdoor activities.";
        category = "unhealthy";
    } else if (aqi <= 300) {
        message = "Very Unhealthy. Health alert:
everyone may experience more serious health
effects.";
        category = "very-unhealthy";
    } else {

```

```

        message = "Hazardous. Serious health effects.
Avoid outdoor activity and stay indoors.";
        category = "hazardous";
    }
    advisoryElem.textContent = message;
    advisoryElem.classList.add(category);
} else {
    resultElem.textContent = "Unable to calculate
AQI";
    advisoryElem.textContent = "";
}
</script>
</body>
</html>

```

3. app.py(flask API code to connect frontend with backend)

```

from flask import Flask, request, render_template,
jsonify
from flask_cors import CORS
import numpy as np
import joblib

app = Flask(__name__)
CORS(app)

# Load model and scaler at startup
model = joblib.load("models/ran_model.pkl")      #
Your trained ML model
scaler = joblib.load("models/ran_scaler.pkl")    #
Your fitted scaler

@app.route('/')

```

```

def home():
    return render_template("index.html")

@app.route('/result')
def result():
    return render_template("result.html")

def get_health_advisory(aqi):
    """Return AQI category and health advisory based
    on AQI value."""
    if aqi <= 50:
        return ("Good", "Air quality is satisfactory.
        It's a great day to be outside!")
    elif aqi <= 100:
        return ("Moderate", "Air quality is
        acceptable. Some pollutants may be a concern for
        sensitive individuals.")
    elif aqi <= 150:
        return ("Unhealthy for Sensitive Groups",
        "Children, elderly, and people with respiratory
        problems should limit outdoor activity.")
    elif aqi <= 200:
        return ("Unhealthy", "Everyone may begin to
        experience health effects. Sensitive groups should
        avoid prolonged outdoor exertion.")
    elif aqi <= 300:
        return ("Very Unhealthy", "Health alert:
        everyone may experience more serious health effects.
        Avoid outdoor activity.")
    else:
        return ("Hazardous", "Serious health effects.
        Stay indoors and avoid all physical activity
        outside.")

```

```

@app.route('/predict', methods=['GET'])
def predict():
    try:
        # Extract query parameters
        features = [
            float(request.args.get("PM25", 0)),
            float(request.args.get("PM10", 0)),
            float(request.args.get("NO2", 0)),
            float(request.args.get("SO2", 0)),
            float(request.args.get("CO", 0)),
            float(request.args.get("O3", 0))
        ]

        # scale and predict
        input_array = np.array(features).reshape(1, -
1)

        scaled_input = scaler.transform(input_array)
        prediction = model.predict(scaled_input)
        aqi = round(prediction[0])

        # Get health advisory
        category, advisory = get_health_advisory(aqi)

        return jsonify({
            "aqi": aqi,
            "category": category,
            "advisory": advisory
        })

    except Exception as e:

```

```
        return jsonify({"error": str(e)}), 400

if __name__ == "__main__":
    app.run(debug=True)
```

4. db_conn.py (loading dataset into mongoDB)

```
import pandas as pd
from pymongo import MongoClient
import json

# Load the file
df = pd.read_csv("PRSA_Data_Aotizhongxin_20130301-20170228")
df = pd.read_csv("PRSA_Data_Changping_20130301-20170228")
df = pd.read_csv("PRSA_Data_Dingling_20130301-20170228")
df = pd.read_csv("PRSA_Data_Dongsi_20130301-20170228")
df = pd.read_csv("PRSA_Data_Guanyuan_20130301-20170228")
df = pd.read_csv("PRSA_Data_Gucheng_20130301-20170228")
df = pd.read_csv("PRSA_Data_Huairou_20130301-20170228")
df = pd.read_csv("PRSA_Data_Nongzhanguan_20130301-20170228")
df = pd.read_csv("PRSA_Data_Shunyi_20130301-20170228")
df = pd.read_csv("PRSA_Data_Tiantan_20130301-20170228")
df = pd.read_csv("PRSA_Data_Wanliu_20130301-20170228")
```

```

df = pd.read_csv("PRSA_Data_Wanshouxigong_20130301-
20170228")
df.dropna(inplace=True)

# Create 'time' column
df['time'] = pd.to_datetime(df[['year', 'month',
'day', 'hour']])

# Drop split time columns, but keep station name
df = df.drop(columns=['No', 'year', 'month', 'day',
'hour'])

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["AQI"]
collection = db["china_data"]
#collection.delete_many({}) # Clear previous runs

# Insert into MongoDB
#data_json = json.loads(df.to_json(orient='records'))
#collection.insert_many(data_json)

#print(f"Inserted {len(data_json)} records into
MongoDB.")

```


TESTING

Project Title: Air Quality Index

Tester Name: Biswarupa Rath

Test Date: April 26, 2025

Test Environment:

- **Localhost (127.0.0.1:5000)**
- **Browser: Google Chrome (Latest)**
- **OS: Windows 11**

1. Unit Testing Summary

Test Case ID	Test Description	Expected Result	Actual Result	Status
UT-001	Test calculate_aqi(pm25, pm10, no2, so2, co, o3)` function with valid inputs	Correct AQI calculated and returned	AQI correctly calculated	Pass
UT-002	Handle invalid (non-numeric) inputs	Proper error/exception handling	Error handled gracefully	Pass
UT-003	Handle missing values (null/empty)	Validation errors raised	Validation working as expected	Pass

2. Integration Testing Summary

Test Case ID	Test Description	Expected Result	Actual Result	Status
IT-001	Form submission sends data to backend correctly	Data received and processed at backend	Data correctly processed	Pass
IT-002	Backend processes and returns AQI result to frontend	AQI displayed on frontend or success message shown	AQI displayed as expected	Pass
IT-003	Date and Time component integration	Correct system time displayed dynamically	Correct time shown	Pass

3. System Testing Summary

Test Case ID	Test Description	Expected Result	Actual Result	Status
ST-001	Check overall page load	Page loads successfully without errors	Page loaded smoothly	Pass
ST-002	Verify form elements	All fields and Submit	All UI elements	Pass

	(input fields and button)	button visible and functional	present and working	
ST-003	Submit form with valid data	Form accepted and processed, AQI displayed	Form processed correctly	Pass
ST-004	Submit form with invalid/missing data	Error messages shown, form not submitted	Validation working correctly	Pass
ST-005	UI responsiveness check	UI adapts properly (no overlapping elements)	UI is responsive and clean	Pass

4. Overall Result

Total Test Cases: 14

Passed: 14

Failed: 0

Defects Found: 0

Conclusion:

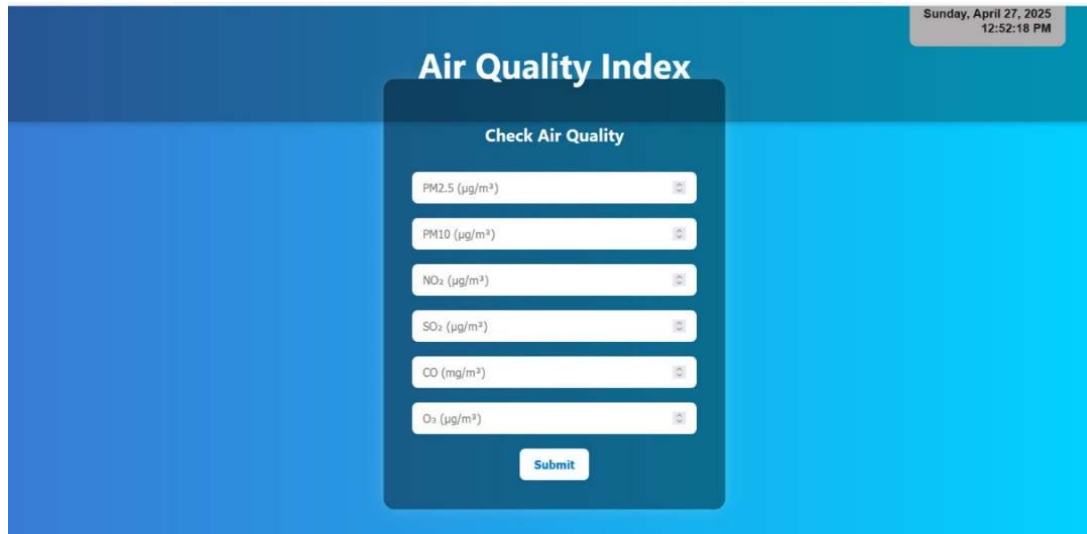
All tests were successfully executed. The system meets the expected requirements and is ready for deployment/move to the next phase.

Notes:

- No functional bugs were found during testing.
- Minor UI improvements (e.g., adding input restrictions to accept only numbers) could further enhance user experience.
- Recommend setting up automated unit tests for future updates.

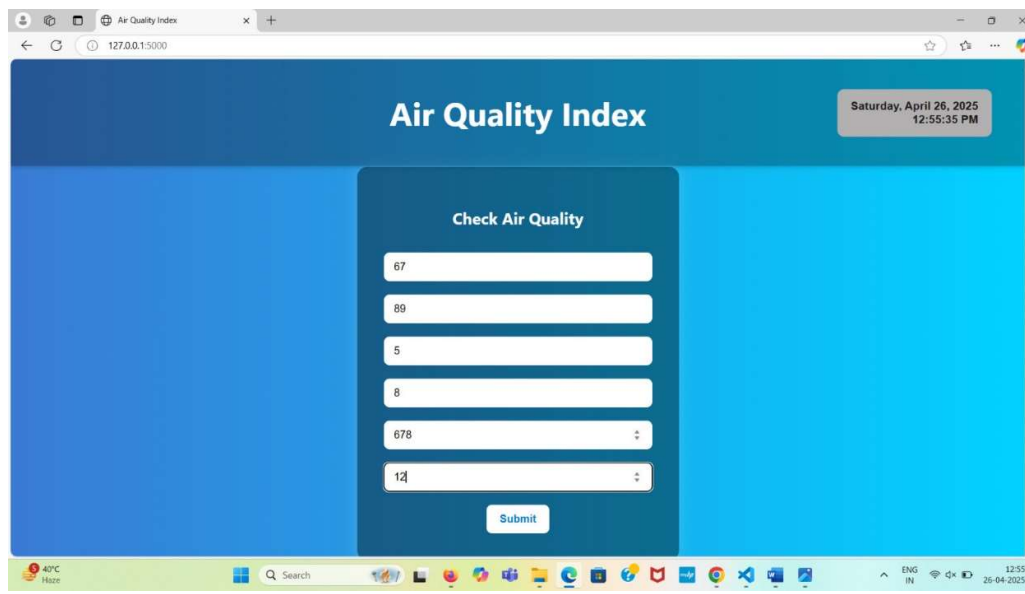
OUTPUTS

→ This is the user interface page where we have to input the values of PM2.5, PM10, NO2, SO2, CO, O3 in the respective blank spaces.



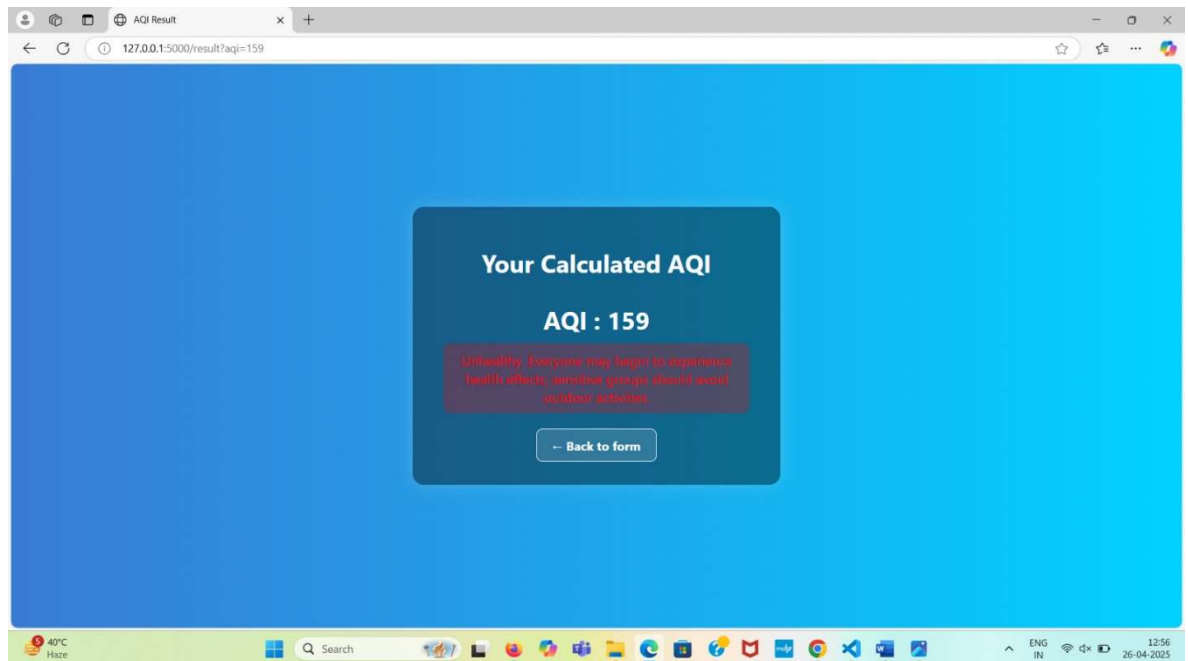
The screenshot shows a web application titled "Air Quality Index" with a subtitle "Check Air Quality". The interface features six empty input fields for the following pollutants: PM2.5 ($\mu\text{g}/\text{m}^3$), PM10 ($\mu\text{g}/\text{m}^3$), NO₂ ($\mu\text{g}/\text{m}^3$), SO₂ ($\mu\text{g}/\text{m}^3$), CO (mg/m^3), and O₃ ($\mu\text{g}/\text{m}^3$). A "Submit" button is located at the bottom of the form. The background is a gradient of blue and teal. The date and time "Sunday, April 27, 2025 12:52:18 PM" are displayed in the top right corner.

→ In this page, we have given the respective input values of the parameters specified.



The screenshot shows the same "Air Quality Index" web application, but with numerical values entered into the input fields: 67 for PM2.5, 89 for PM10, 5 for NO₂, 8 for SO₂, 678 for CO, and 124 for O₃. The "Submit" button remains at the bottom. The background and layout are consistent with the previous screenshot. The date and time "Saturday, April 26, 2025 12:55:35 PM" are displayed in the top right corner. The browser window shows the URL "127.0.0.1:5000" and the Windows taskbar at the bottom with a system tray showing "42°C Haze" and the date "26-04-2025".

→ After specifying the values, submit button is being clicked. And this is the result it displays.



CONCLUSION

The Air Quality Index (AQI) Checker project has been successfully designed and tested in alignment with the stated requirements. The Software Requirements Specification (SRS) document clearly outlined the objectives, scope, functionality, and technical specifications of the project, focusing on providing a user-friendly, web-based application for calculating and displaying AQI based on user input for major pollutants.

Following the development phase, comprehensive testing—including unit, integration, and system testing—was conducted as detailed in the Test Case Document. All 14 test cases passed successfully, confirming that the system performs as expected without any functional defects. The application demonstrated proper input validation, accurate AQI calculations, reliable data processing, and responsive UI behavior across different environments.

Minor UI enhancements were noted for future improvement, such as enforcing numeric input restrictions. Overall, the AQI Checker system fulfils its intended purpose and is ready for deployment or advancement to subsequent phases, with a recommendation to implement automated testing for easier maintenance and scalability.

FUTURE SCOPE

We plan to:

1. Add Web Scraping to collect more data.
2. Implement Log in/Sign up for user accounts.
3. Integrate Payment Gateway for premium features.
4. Allow Location Selection for accurate AQI predictions.

These additions will enhance the project's functionality and user experience

REFERENCES

- <https://www.wikipedia.org/>
- <https://code.jquery.com/>
- <https://w3schools.com/>
- <https://www.python.org/>
- <https://www.geeksforgeeks.org/>