A Project Report on

Pollution Tracker

Submitted By:

Khushi Yashodhara, NNM23VL033
Krishna M Nair, NNM23VL034
Liron Crasta, NNM23VL035

NITTE (Deemed to be University) | NMAM INSTITUTE OF TECHNOLOGY
Department of Electronics Engineering (VLSI Design and Technology)

December - 2024

NITTE (Deemed to be University) | NMAM INSTITUTE OF TECHNOLOGY

Department of Electronics Engineering (VLSI Design and Technology)

CERTIFICATE

This is to certify that Khushi Yashodhara (NNM23VL033), Krishna M Nair (NNM23VL034), Liron Crasta (NNM23VL035) are bonafide students of N.M.A.M. Institute of Technology, Nitte, who have submitted a project report entitled "Pollution Tracker" as part of the Project-based Python Programming Lab, in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Electronics Engineering(VLSI Design and Technology)during the year 2024-2025.

Name of the Examiner **Signature with date**

Chapter 1

ABSTRACT

The Pollution Tracker is a Python-based mini-project designed to assess and monitor air quality by calculating the Air Quality Index (AQI) using pollutant data such as PM2.5, PM10, CO2, and SO2. The program evaluates pollutant concentrations against predefined thresholds, assigns AQI categories, and provides tailored recommendations to mitigate exposure and reduce pollution. Using file handling techniques, it processes input data, logs AQI values over time in a CSV format, and outputs results in a structured text file for easy reference. This project demonstrates the application of Python in environmental monitoring, emphasizing simplicity, efficiency, and practical utility. By enabling users to track and understand pollution trends, the project highlights the role of technology in fostering awareness and promoting environmental responsibility.

Contents

Chapter 2

INTRODUCTION

2.1 Background

Air pollution is a growing global concern, significantly impacting public health, climate, and ecosystems. Monitoring air quality and understanding pollutant levels are essential to mitigate these effects. While advanced tools exist, they are often complex or expensive, making them inaccessible for smaller communities or educational purposes.

2.2 Problem Statement

Despite the availability of advanced air quality monitoring systems, there is a lack of affordable and accessible tools for small-scale or educational purposes. Individuals, researchers, and educators often face challenges in monitoring local air pollution trends due to the technical and financial barriers posed by existing solutions. This limitation prevents widespread awareness of air quality and hinders efforts to promote sustainable practices. There is a pressing need for a user-friendly, standalone tool that can process pollution data, calculate AQI, categorize pollution levels, and provide actionable recommendations to reduce emissions and exposure risks. Such a tool should be easy to use, require minimal resources, and offer functionalities like historical data logging to help users track pollution trends over time.

2.3 Objectives

- Develop a Python-based tool to calculate the Air Quality Index (AQI) from pollutant data.
- Categorize pollution levels and offer actionable suggestions for mitigation.
- Log historical AQI data to track pollution trends over time.
- Ensure the program is simple, accessible, and suitable for educational and small-scale applications.

2.4 Scope

The Pollution Tracker project is designed for individuals, researchers, and educators to monitor air quality and analyze pollution trends. It provides a cost-effective solution for small-scale air quality assessments without relying on expensive hardware or complex software, promoting awareness and encouraging sustainable practices to reduce environmental impact.

Chapter 3

LITERATURE REVIEW

3.1 Existing Pollution Trackers

Existing pollution trackers, such as mobile apps and online platforms (e.g., AQI monitors like AirVisual and government dashboards), typically feature real-time monitoring, interactive maps, and forecasts of air quality data. They often integrate sensors and APIs to provide users with accurate AQI values, pollutant breakdowns, and health-related recommendations. However, these systems can be complex, resource-intensive, or reliant on specific hardware, limiting their accessibility for educational purposes or small-scale projects.

3.2 Python in Educational Projects

Python is widely recognized for its simplicity and versatility, making it an ideal choice for developing pollution trackers. Its robust standard library and support for file handling, data processing, and numerical calculations enable efficient implementation of core functionalities such as AQI computation and data logging. Previous projects have leveraged Python to create air quality monitoring tools, showcasing its capability to analyze pollutant data, provide actionable insights, and address real-world environmental challenges effectively.

3.3 Advantages of Using Python for Tracking pollution

- **Ease of Learning:** Python's readable syntax and extensive documentation make it ideal for beginners to grasp programming fundamentals.
- **Versatility:** Python supports various tasks, such as data processing, file handling, and visualization, making it suitable for comprehensive projects like pollution trackers.
- **Modularity:** Python encourages the use of functions and modules, promoting code reusability and maintainability.
- **Community Support:** Python has a large community, offering numerous tutorials, resources, and forums for troubleshooting.

Chapter 4

METHODOLOGY

4.1 Analysis and Design

The Pollution Tracker project is designed to address the need for accessible tools to monitor air quality by calculating the Air Quality Index (AQI) from pollutant data such as PM2.5, PM10, CO2, and SO2. The system is modular, comprising an input module to read and validate data from a text file, a processing module to calculate individual and overall AQI values using predefined thresholds, and an output module to display results, store them in a text file, and log historical data in a CSV file. The algorithm determines AQI values using pollutant concentration ratios and identifies the highest AQI to classify overall air quality into categories with actionable suggestions. This structured design ensures the program is user-friendly, scalable, and effective for educational or small-scale applications.

4.2 Development Environment

- **Programming Language:** Python 3.x
- **Libraries Used:**
  - os for file handling
  - datetime for timestamping logged data
- **VS Code, PyCharm, or IDLE:** Popular editors that provide features like syntax highlighting, debugging tools, and extensions to enhance productivity.
- **File Formats:**
  - Text Files (.txt): Used for input pollutant data for simplicity and compatibility and for saving AQI results and suggestions in a structured format.
  - CSV Files (.csv): Used for logging historical AQI data, enabling easy analysis and visualization.

4.3 Algorithms and Techniques

The system employs simple algorithms to manage pollution tracking operations:

- **Data Input and Validation:** The program reads pollutant concentration data from a text file and ensures that the input file exists and contains valid numerical values for pollutants like PM2.5, PM10, CO2, and SO2.
- **AQI Calculation:** For each pollutant, the AQI is calculated using the formula:

  AQI = (Pollutant Value/Threshold Value) ×100
- **Categorization and Suggestion Mapping:** The overall AQI is mapped to predefined ranges corresponding to pollution categories (e.g., Good, Moderate, Unhealthy).A lookup table is used to associate each category with a specific suggestion to reduce pollution exposure and emissions.
- **Data Output and Logging:** The final results, including AQI values, categories, and suggestions, are saved to an output text file for user reference.Historical AQI data, along with timestamps, are logged into a CSV file for tracking pollution trends over time.
- **Exception Handling:** The program includes mechanisms to handle errors, such as missing input files or invalid data. It ensures that the program remains robust and provides meaningful feedback to the user in case of issues.

Chapter 5

IMPLEMENTATION

5.1 Setup and Configuration

To set up the Pollution Tracker, ensure that Python 3.x is installed on the system. The project uses only Python's standard libraries, such as os, datetime, and csv, so no additional installations are required. Download or create the project files, including the Python script, and navigate to the project directory.

5.2 Development Stages

1. **Data Initialization:** The program checks if the input file with pollutant data exists. If not, it creates a sample file with default pollutant values. This ensures the program runs even without initial data.

2. **Data Input and Validation:** The program reads data from the input file. It checks if the data is correct. If the file is missing or the data is invalid, an error is raised.

3. **AQI Calculation:** Then it calculates the AQI for each pollutant. It uses predefined thresholds to convert pollutant concentrations into AQI values. Higher AQI means worse air quality.Next, overall AQI is calculated by taking the highest AQI value among the pollutants. The worst pollutant sets the overall air quality.

4. **AQI Categorization and Suggestions:** Based on the overall AQI, the program categorizes the air quality (e.g., Good, Hazardous). It also gives suggestions to reduce pollution or limit exposure.

5. **Result Output:** The program saves the AQI results to a text file. This file includes the overall AQI, category, suggestions, and individual pollutant AQIs. Users can review the results at any time.

6. **Historical Logging:** Logs each AQI calculation with a timestamp in a CSV file. This allows tracking pollution trends over time.

7. **Error Handling:** Includes error handling to manage issues like missing files or invalid data. It provides clear messages to guide users in fixing problems.

5.3 Challenges and Solutions

- **Handling Invalid Input Files:** Implemented input validation and exception handling to manage cases where the input file is missing, corrupted, or contains incorrect data, ensuring the program does not crash and provides clear error messages to guide users.
- **Ensuring Accurate AQI Calculation:** Developed a system to calculate the AQI for multiple pollutants using predefined thresholds, ensuring that the overall air quality is determined by the most harmful pollutant. This prevents inaccurate assessments of air quality.
- **Data Persistence for Historical Logging:** Used CSV files to log AQI data with timestamps, allowing historical tracking of pollution levels over time. This ensures that past data can be easily accessed and analyzed even after the program ends.
- **Handling Multiple Pollutants Simultaneously:** Addressed the challenge of processing multiple pollutants by calculating individual AQI values for each pollutant and selecting the highest value to determine the overall air quality. This ensures the program accurately reflects the most critical pollutant in the air.

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Presentation of Results

The results portrayed in the given photos demonstrate how the Pollution Tracker efficiently calculates the Air Quality Index (AQI) for various pollutants, categorizes the air quality, and provides actionable suggestions. The program executes its functions in an organized manner, from reading pollutant data to calculating individual and overall AQI, assigning appropriate categories, and saving the results in user-friendly formats for further analysis and tracking.

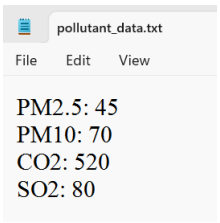

Figure 6.1: Input entered into the file

In the fig 6.1, the values of the pollutants are entered into a .txt file. After running the code, we can see the output in fig 6.2, which displays the overall AQI value,shows which pollution category the place falls under and gives helpful suggestions to minimize pollution.

```
In [1]: runfile('C:/Users/HP/OneDrive/Desktop/Python Project/pollution_tracker.py', wdir='C:/Users/HP/OneDrive
Project')
Results have been written to: aqi_results.txt
Logged data to: aqi_log.csv

AQI calculation complete.
Overall AQI: 140.0
Pollution Category: Unhealthy for Sensitive Groups
Suggestion: Limit outdoor activities for sensitive groups; consider using air purifiers indoors.
Individual AQI Values: {'PM2.5': 128.57142857142858, 'PM10': 140.0, 'CO2': 104.0, 'SO2': 106.66666666666667}
```

Figure 6.2: Output displayed after running the code

THe fig 6.3 and fig 6.4 display the results stored in both text file and logged in excel sheet allowing users to track pollution trends over time.

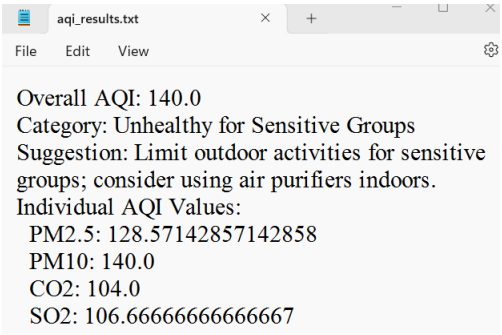

Figure 6.3: Result stored in file

B	C	D	E	F	G	H
140	Unhealthy for Sensitive Groups	Limit outdoor activities for sensitive groups; consider using air purifiers indoors.	128.5714	140	104	106.6666667
140	Unhealthy for Sensitive Groups	Limit outdoor activities for sensitive groups; consider using air purifiers indoors.	128.5714	140	104	106.6666667
140	Unhealthy for Sensitive Groups	Limit outdoor activities for sensitive groups; consider using air purifiers indoors.	128.5714	140	104	106.6666667

Figure 6.4: Result logged in excel sheet

6.2 Analysis of Results

Thus this project efficiently calculates and categorizes air quality based on pollutant data, providing reliable AQI values for pollutants like PM2.5, PM10, CO2, and SO2. The system accurately determines the overall air quality by using the highest AQI value from multiple pollutants and assigns it to predefined categories like ”Good,” ”Moderate,” or ”Unhealthy.” It further enhances user experience by offering actionable suggestions to reduce exposure to pollution based on the AQI category. Additionally, the program logs AQI data with timestamps in a CSV file, allowing users to track pollution trends over time and analyze patterns.Overall, the Pollution Tracker provides a useful and insightful tool for monitoring air quality, with the potential for future enhancements to broaden its capabilities.

6.3 Discussion

The development of the Pollution Tracker project signifies the culmination of programming skills with real-world environmental applications. The use of the Python language turned out to be a good choice as it is easy to code, has good file-handling capabilities, and is efficient in managing computations. The AQI enables users to monitor air quality and understand pollution levels through categorization and actionable suggestions. Thus, the logging of historical data by this program offers insight into trends on pollution and encourages further long-term analysis toward well-informed decisions. This function could potentially be built upon using visualization tools like graphs or charts to illustrate the data more vividly. Overall, the Pollution Tracker acts as a stepping stone for creating more sophisticated environmental monitoring tools while demonstrating practical application of programming in solving global issues.

Chapter 7

CONCLUSION

The Pollution Tracker project successfully demonstrates how Python can be leveraged to develop an efficient and accessible tool for monitoring air quality. By calculating the Air Quality Index (AQI) based on pollutant data and providing actionable suggestions to mitigate pollution, the project addresses both environmental awareness and practical problem-solving. Its modular design, file-based data handling, and historical logging ensure usability and scalability, making it suitable for educational purposes and small-scale applications. This project not only highlights the power of technology in tackling real-world issues but also fosters a deeper understanding of environmental challenges, encouraging users to adopt sustainable practices for a healthier future.

Bibliography

[1] Python Documentation. *https://docs.python.org/3/.*

[2] GeeksforGeeks. ”Python File Handling Tutorial” *https://www.geeksforgeeks.org*

[3] W3Schools. ”Python CSV Module” *https://www.w3schools.com*

[4] Environmental Protection Agency (EPA), *Air Quality Index (AQI) Standards and Guidelines*

APPENDIX: Code used in the project

```
import csv
from datetime import datetime

# Define AQI thresholds for different pollutants
AQI_THRESHOLDS = {
    "PM2.5": {"standard": 35, "max_aqi": 500},
    "PM10": {"standard": 50, "max_aqi": 500},
    "CO2": {"standard": 500, "max_aqi": 500},
    "SO2": {"standard": 75, "max_aqi": 500}
}

# Define pollution categories and suggestions
POLLUTION_CATEGORIES = {
    (0, 50): ("Good", "Air quality is satisfactory. Maintain clean energy
       practices."),
    (51, 100): ("Moderate", "Consider carpooling or using public transport
       to reduce emissions."),
    (101, 150): ("Unhealthy for Sensitive Groups", "Limit outdoor
       activities for sensitive groups; consider using air purifiers
       indoors."),
    (151, 200): ("Unhealthy", "Avoid outdoor activities; reduce vehicle
       usage, consider working from home."),
    (201, 300): ("Very Unhealthy", "Avoid going outside; use air purifiers
       and masks indoors; reduce industrial activities."),
    (301, 500): ("Hazardous", "Stay indoors; minimize all emissions;
       consider emergency air quality measures.")
}

# Load pollutant values from a text file
def load_pollutant_data(file_path):
    try:
        pollutant_data = {}
        with open(file_path, 'r') as file:
            for line in file:
                key, value = line.strip().split(": ")
                pollutant_data[key] = float(value)
        return pollutant_data
    except FileNotFoundError:
        print("Error: Input file not found.")
        return None
    except ValueError:
        print("Error: Invalid data format.")
        return None

# Calculate AQI for a single pollutant
def calculate_individual_aqi(value, standard, max_aqi):
    return min(max_aqi, (value / standard) * 100)

# Determine the AQI category and suggestion based on the overall AQI
def get_aqi_category_and_suggestion(aqi):
    for (low, high), (category, suggestion) in POLLUTION_CATEGORIES.items()
       :
        if low <= aqi <= high:
            return category, suggestion
    return "Invalid AQI", "No suggestions available."
```

```
# Main function to calculate AQI from pollutant data
def calculate_overall_aqi(pollutant_data):
    aqi_values = {}

    for pollutant, value in pollutant_data.items():
        if pollutant in AQI_THRESHOLDS:
            standard = AQI_THRESHOLDS[pollutant]["standard"]
            max_aqi = AQI_THRESHOLDS[pollutant]["max_aqi"]
            aqi_values[pollutant] = calculate_individual_aqi(value,
                standard, max_aqi)

    overall_aqi = max(aqi_values.values())
    category, suggestion = get_aqi_category_and_suggestion(overall_aqi)
    return overall_aqi, category, suggestion, aqi_values

# Write results to an output text file
def write_results_to_file(output_path, overall_aqi, category, suggestion,
   aqi_values):
    with open(output_path, 'w') as file:
        file.write(f"Overall AQI: {overall_aqi}\n")
        file.write(f"Category: {category}\n")
        file.write(f"Suggestion: {suggestion}\n")
        file.write("Individual AQI Values:\n")
        for pollutant, aqi in aqi_values.items():
            file.write(f"  {pollutant}: {aqi}\n")
    print("Results have been written to:", output_path)

# Log AQI data for tracking pollution over time
def log_aqi_data(log_file, overall_aqi, category, suggestion, aqi_values):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    log_data = [timestamp, overall_aqi, category, suggestion] + list(
       aqi_values.values())

    with open(log_file, 'a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(log_data)
    print("Logged data to:", log_file)

# Main program
def main(input_file, output_file, log_file):
    pollutant_data = load_pollutant_data(input_file)

    if pollutant_data:
        overall_aqi, category, suggestion, aqi_values =
           calculate_overall_aqi(pollutant_data)
        write_results_to_file(output_file, overall_aqi, category,
           suggestion, aqi_values)
        log_aqi_data(log_file, overall_aqi, category, suggestion,
           aqi_values)

        # Display AQI and suggestion
        print("\nAQI calculation complete.")
        print(f"Overall AQI: {overall_aqi}")
        print(f"Pollution Category: {category}")
        print(f"Suggestion: {suggestion}")
        print("Individual AQI Values:", aqi_values)

# Run the main function
```

```
if _name_ := "_main_":
    input_file = "pollutant_data.txt"  # Input file path (plain text)
    output_file = "aqi_results.txt"    # Output file path (plain text)
    log_file = "aqi_log.csv"           # Log file path (CSV)
    main(input_file, output_file, log_file)
```