Yihang Fu
yf329
yihang.fu@yale.edu

Final Project
BIS 634 Computational Methods for Informatics

2024/12/15

1 Introduction

Air pollution is one of the most pressing challenges of the 21st century, posing significant threats to human health, ecosystems, and the global climate. Air quality is determined by the presence of harmful pollutants such as particulate matter (PM2.5 and PM10), ozone (O3), nitrogen dioxide (NO2), carbon monoxide (CO), and sulfur dioxide (SO2). Prolonged exposure to poor air quality has been linked to respiratory illnesses, cardiovascular diseases, and reduced life expectancy, making air pollution a critical environmental and public health issue.
This project focuses on the development of an advanced Air Quality Index (AQI) prediction system aimed at providing accurate and actionable insights into air quality trends. By leveraging a combination of machine learning models and historical air quality data, the system enables stakeholders to make informed decisions for public health advisories, policy formulation, and environmental management. The project integrates data visualization techniques, dynamic user interactions, and a robust prediction engine to enhance accessibility and usability for end users.
The system provides two primary functionalities:

1. **Manual AQI Prediction:** Users can input specific pollutant concentration levels (PM2.5, PM10, O3, NO2, CO, SO2) to obtain the predicted AQI and an assessment of the air quality.
2. **Automated AQI Prediction:** Users can specify a city name to fetch real-time pollutant data via external APIs, enabling the system to predict the AQI and visualize historical trends.

Additionally, the project incorporates a comparative analysis feature, allowing users to evaluate air quality trends between two cities over a specified time range. This feature not only visualizes AQI patterns but also provides text-based conclusions to facilitate better understanding of the differences in air quality.
The overarching goal of this project is to demonstrate the effective application of data science and web technologies in addressing real-world environmental challenges. By combining predictive modeling, interactive web design, and meaningful data visualization, the AQI prediction system serves as a tool for raising awareness and supporting initiatives to combat the adverse effects of air pollution. The figure 1 shows the overall pipeline of the interactive website.
The server API is built with Flask and handles all the backend tasks, like getting data from external APIs, running the AQI prediction model, and sending the results back to the front-end. It fetches geolocation and air quality data using OpenWeather APIs and uses a saved Random Forest model to predict AQI. The web front-end is simple and user-friendly, made with HTML, CSS, and Bootstrap. Users can input data, choose cities for comparison, and see predictions, plots, and advice all in one place. Figure 2 is the homepage of our website.

2 Key Features

3 Key Features

3.1 Prediction

The prediction functionality of the Air Quality Index (AQI) system is a core feature that provides users with insights into air quality based on pollutant concentration data. We provide two ways to provide a future forecast for uses. Users can `manually type` in the data of the air pollutants, and the website will return an expected prediction value and corresponding suggestions. Users can also `type in the city name` and we will call an external API called OpenWeather [1]to first derive the geo-location of the target city and use the coordinates to call the `Climate Forecast` function. Finally the API will also return a json with expected prediction results.

[1]https://openweathermap.org/api

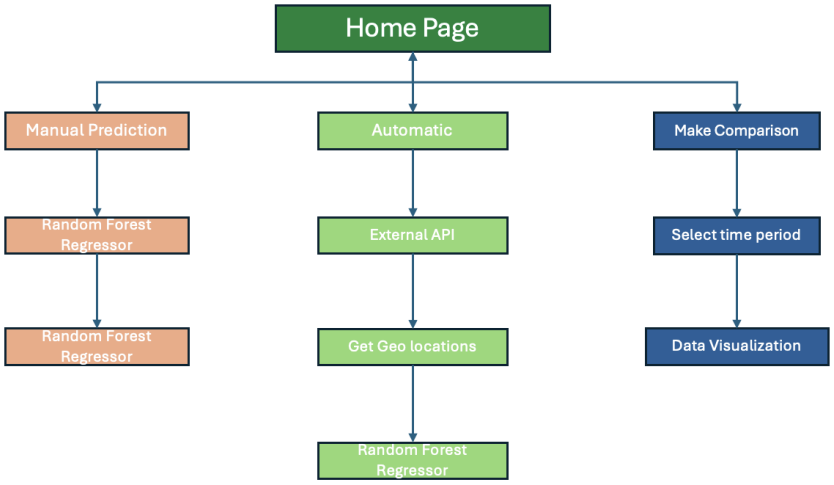


Figure 1: Overview of the project

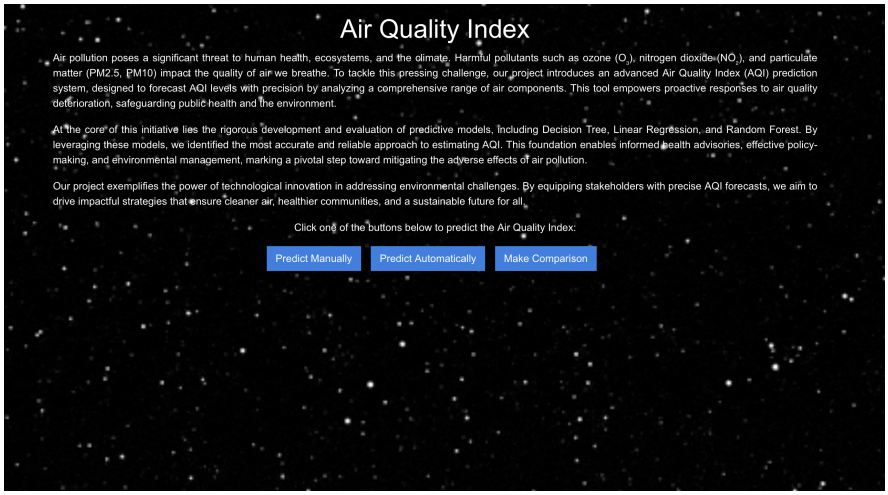


Figure 2: Homepage

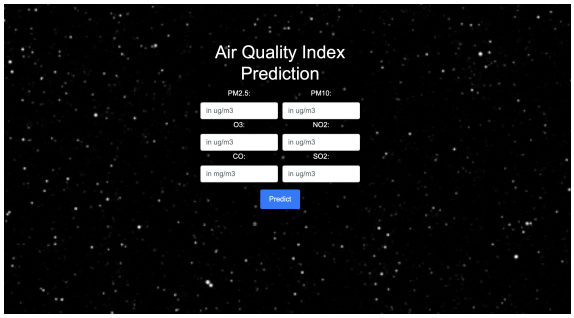
3.1.1 Model and Methodology

For the first option of the users, we provide a **Random Forest Regressor** driven AQI prediction. Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the average prediction of the individual trees. Its ability to handle non-linear relationships and missing data makes it particularly suitable for environmental datasets. The data [2] we used to train the model is an open-access data set provided by the Energy Policy Institute at the University of Chicago (EPIC), with the `CC-BY-4.0 license`. Using global, satellite-derived data, the AQLI provides a detailed picture of air pollution levels and their health impact around the world, tracking changes from 2000 to 2022.
To ensure the prediction model operates effectively, comprehensive data preprocessing steps were undertaken:

- **Handling Missing Data:** Missing or empty data values in the dataset were imputed using the mean value of the corresponding pollutant. This ensures that the model receives a complete dataset without introducing significant bias or variance.
- **Feature Scaling** Normalization and Standardization are applied to ensure uniformity when integrating data from external sources.

The preprocessed data was then used to train the Random Forest Regressor model. Once the training was completed, the model was serialized and saved as a `joblib` file, allowing efficient integration into the web application for real-time predictions. As shown in figure 3, users can type in the air pollutants data manually and the system will return the results and corresponding suggestions.

[2] https://github.com/aqli-epic/aqli-update/tree/main/data



(a) Example manual inputs of air pollutant data

(b) Shown results of the manual prediction

Figure 3: Manual prediction webpage.

3.2 Using External APIs for fetching geo-location and climate data

The automatic AQI prediction feature leverages external APIs to provide real-time predictions without requiring users to manually input pollutant data. This section describes the integration of external APIs to retrieve geolocation and air quality data.

3.2.1 Geolocation Data Retrieval

To fetch air quality data for a user-specified city, the system first needs to determine the geographic coordinates (latitude and longitude) of the city. This is accomplished using the **OpenWeather Geocoding API**, which provides geolocation data for cities worldwide. The data was well-annotated with metadata, including information such as timestamps, geolocation (latitude and longitude), pollutant concentrations, and AQI values. This metadata enabled efficient data filtering, robust analysis, and seamless integration with external APIs. Additionally, it enhanced the accuracy of visualizations and supported the extraction of valuable insights, such as seasonal trends and regional comparisons in air quality.

1. **User Input:** The user enters the name of the desired city into the web interface.
2. **API Request:** The system constructs an HTTP GET request to the Geocoding API endpoint with the city name as a query parameter:

   ```
   http://api.openweathermap.org/geo/1.0/direct?q={city_name}&limit=1&appid={API_KEY}
   ```

   Here, `city_name` is the user-provided city name, and `API_KEY` is the API authentication token.
3. **API Response:** The API returns a JSON object containing the latitude and longitude of the specified city. For example:

   ```
   [
       {
           "name": "London",
           "lat": 51.5074,
           "lon": -0.1278,
           "country": "GB"
       }
   ]
   ```

4. **Data Extraction:** The latitude (`lat`) and longitude (`lon`) values are extracted from the API response and used to query the Air Quality API.

3.2.2 Fetching Air Quality Data

After obtaining the geolocation data, the system retrieves pollutant concentration levels using the **OpenWeather Air Pollution API**.

1. **API Request:** The system sends an HTTP GET request to the Air Pollution API endpoint, including the latitude and longitude values obtained earlier:

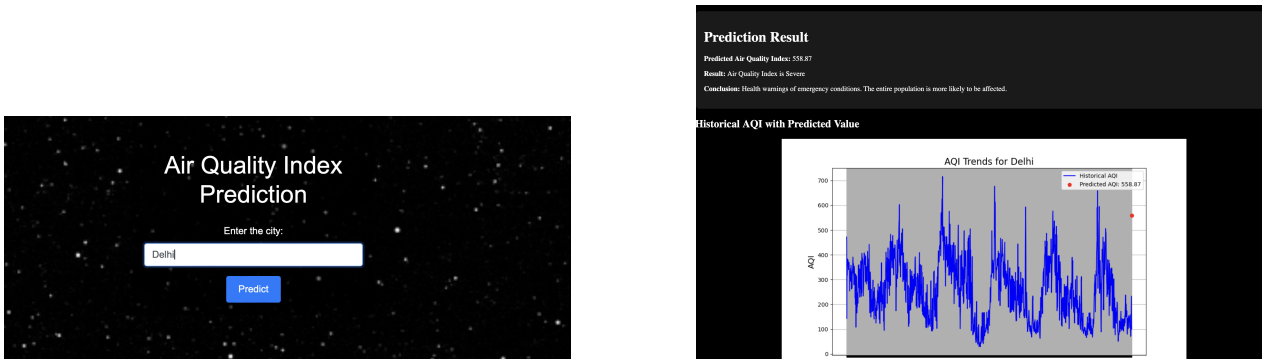
`http://api.openweathermap.org/data/2.5/air_pollution?lat={lat}&lon={lon}&appid={API_KEY}`

2. **API Response:** The API returns a JSON object containing real-time pollutant concentration data for the specified location. For example:

```
{
    "list": [
        {
            "components": {
                "pm2_5": 12.0,
                "pm10": 20.5,
                "o3": 40.2,
                "no2": 15.7,
                "co": 0.3,
                "so2": 5.2
            },
            "main": {
                "aqi": 3
            }
        }
    ]
}
```

3. **Data Extraction:** The pollutant concentrations (`pm2_5`, `pm10`, `o3`, `no2`, `co`, `so2`) are extracted from the JSON response and passed to the trained Random Forest Regressor model for AQI prediction.

With the fetched data, we can then transit to the Random Forest Regressor model we built up before. As shown in figure 4, users can manually type in the city name and will get the prediction results and a line plot with historical and predictive values.



(a) Example input

(b) Shown results of the prediction

Figure 4: Automatic Prediction webpage

3.3 Make Comparison

The **Make Comparison** feature of the AQI prediction system allows users to compare air quality trends between two cities over a specified time period. This functionality is designed to provide insightful visualizations and meaningful conclusions regarding air quality differences between the selected cities.

3.3.1 User Interaction Workflow

The workflow for the Make Comparison feature is as follows:

1. **City Selection:** The user is prompted to input the names of two cities they wish to compare.
2. **Time Range Selection:** The user can choose a specific time period for comparison, such as the past 7, 15, 30, 60, or 90 days.

3. **Data Filtering:** The system retrieves historical AQI data for the two cities and filters it to match the selected time range.
4. **Visualization:** A comparative line plot is generated, showing the AQI trends for both cities over the specified time range.
5. **Text-Based Conclusion:** The system computes the average AQI for each city during the selected time period and generates a text-based conclusion summarizing the differences in air quality.

3.3.2 Data Processing and Visualization

The system processes the historical AQI data for the two cities as follows:

- **Data Filtering:** The data is filtered to include only the rows corresponding to the selected cities and time range. This ensures that the visualization and analysis are focused on the user-specified parameters.
- **Plot Generation:** A comparative line plot is generated using `Matplotlib`. The x-axis represents the dates in the selected time range, and the y-axis represents the AQI values. The AQI trends for the two cities are represented by distinct lines with different colors (e.g., blue for City 1 and red for City 2).
- **Dynamic X-Axis Formatting:** The x-axis labels are dynamically adjusted to avoid overcrowding and improve readability.

3.3.3 Error Handling and Validation

To ensure a smooth user experience, the system incorporates robust error handling mechanisms:

- **Invalid City Names:** If one or both cities are not found in the dataset, the system informs the user and prompts for valid city names.
- **Insufficient Data:** If one or both cities lack data for the selected time range, the system notifies the user and suggests trying a different time range or cities.
- **Empty Results:** If the data retrieval process fails or yields empty results, the system displays an appropriate error message.

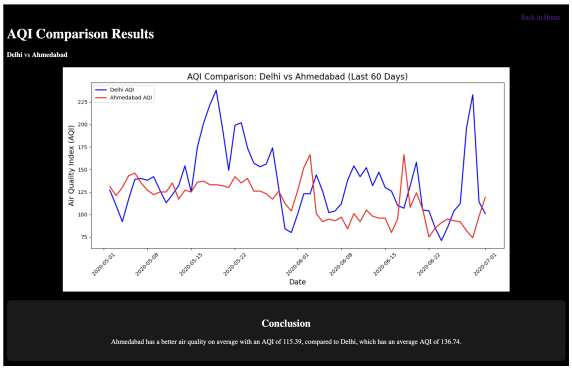
3.3.4 Significance of the Feature

The Make Comparison feature provides users with a deeper understanding of air quality differences between cities. By offering both visual and textual insights, this feature empowers users to:

- Identify cities with better air quality over a specific time range.
- Assess the impact of seasonal and temporal variations on air quality.
- Make informed decisions regarding travel, health precautions, or policy recommendations.

This feature 5 combines dynamic data visualization with automated analysis, making it a valuable tool for exploring and understanding air quality trends in a comparative context.

(a) Example make-comparison input



(b) Shown comparison results

Figure 5: Make Comparison demo webpage.

4 Interesting Findings and hypotheses

During the development and analysis of the AQI prediction system, several intriguing patterns and insights were uncovered in the air quality data. These findings highlight the complexities of air pollution dynamics and provide valuable context for interpreting AQI trends.

4.1 Seasonal Trends in Air Quality Index

One of the most notable findings is the presence of distinct **seasonal patterns** in AQI data. In one year, the AQI is relatively low during Spring and Autumn while increased in Winter. These **hypotheses** (no validation yet) can be attributed to specific seasonal activities and climatic conditions:

- **Agricultural Activities:** During certain seasons, particularly in autumn, air quality tends to worsen due to activities such as crop burning by farmers. This practice releases large amounts of particulate matter (PM2.5 and PM10) and other pollutants into the atmosphere, significantly degrading air quality in agricultural regions.
- **Winter Heating Demand:** In colder months, the increased use of heating systems, including the burning of coal and wood, contributes to elevated levels of pollutants such as carbon monoxide (CO) and particulate matter. Combined with temperature inversions that trap pollutants near the surface, this leads to a noticeable spike in AQI values during winter.
- **Improved Air Quality in Monsoon:** In regions with a monsoon season, AQI values show significant improvement due to the cleansing effects of rainfall, which helps settle particulate matter and reduce pollutant concentrations in the air.

4.2 Socioeconomic and Population Density Impact on Air Quality

Another significant observation is the relationship between socioeconomic development, population density, and air quality. These factors interact in complex ways to influence AQI trends:

- **Developed vs. Underdeveloped Regions:**
  - **Developed Regions:** High-income, urbanized areas often have higher baseline pollution levels due to industrial activity and vehicular emissions. However, stricter environmental regulations, advanced pollution control technologies, and cleaner energy usage mitigate extreme AQI events, leading to more stable air quality trends.
  - **Underdeveloped Regions:** In contrast, underdeveloped regions may experience lower average AQI levels due to reduced industrialization. However, the lack of stringent regulations, reliance on inefficient fuels, and practices such as crop burning contribute to more significant fluctuations and severe AQI peaks during certain periods.
- **Impact of Population Density:**
  - **High-Density Areas:** Urban centers with dense populations exhibit consistently higher AQI values due to increased energy consumption, vehicular traffic, and industrial activities. The impact is compounded by localized emissions.
  - **Low-Density Areas:** Rural or sparsely populated regions generally have better air quality. However, they may experience seasonal spikes due to natural events (e.g., wildfires) or localized agricultural activities.
  - **Nonlinear Relationship:** The relationship between population density and air quality is not strictly linear. Cities with effective public transport systems and stricter emissions controls often outperform less populated regions with inadequate environmental policies.

These findings (together with the hypothesis) emphasize the importance of balancing development and environmental protection while addressing air quality challenges across diverse regions.

A code implementation

```
from flask import Flask, request, jsonify, render_template
import joblib
import requests
import pandas as pd
import matplotlib
matplotlib.use('Agg')

import matplotlib.pyplot as plt
import os


app = Flask(__name__)

# Load our trained model
model = joblib.load('/Users/yihang/Desktop/BIS_634/finalproject/AQI-prediction-Using-Flask-Web-App-main/.ipynb_checkpoints/airquality.joblib')
data = pd.read_csv('/Users/yihang/Desktop/BIS_634/finalproject/AQI-prediction-Using-Flask-Web-App-main/.ipynb_checkpoints/data.csv')
#data['Date'] = pd.to_datetime(data['Date'])  # Ensure Date is in datetime format
# OpenWeather API Key
API_KEY = 'bd5e378503939ddaee76f12ad7a97608'

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

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

@app.route('/predict_manually', methods=['POST','GET'])
def predict_manually():
    if request.method == 'POST':
        # Extract data from form
        pm25 = float(request.form['PM2.5'])
        pm10 = float(request.form['PM10'])
        o3 = float(request.form['O3'])
        no2 = float(request.form['NO2'])
        co = float(request.form['CO'])
        so2 = float(request.form['SO2'])

        # Prepare data for prediction
        sample = [[pm25, pm10, o3, no2, co, so2]]
        prediction = model.predict(sample)[0]

        # Determine Air Quality Index based on prediction
        result, conclusion = determine_air_quality(prediction)

        # Return the result
        return render_template('manual_results.html', prediction=prediction, result=result, conclusion=conclusion)
    else:
        return render_template('index.html')


@app.route('/predict_automatically', methods=['GET', 'POST'])
def predict_automatically():
    if request.method == 'POST':
        city_name = request.form.get('city_name')
        if not city_name:
            error_message = "Missing city name parameter"
            error_code = 400
            return render_template('error.html', error=error_message, error_code=error_code), 400

        # Fetch air quality data for the input city, then send the geo data to get the air quality data. Then we use our model to get the prediction results.
        geocode_url = f"http://api.openweathermap.org/geo/1.0/direct?q={city_name}&limit=1&appid={API_KEY}"
        geocode_response = requests.get(geocode_url)
        if geocode_response.status_code != 200:
            error_message = "Failed to fetch location data"
            error_code = 500
            return render_template('error.html', error=error_message, error_code=error_code), 500
```

```
69
70          geocode_data = geocode_response.json()
71          if not geocode_data:
72              error_message = "City not found"
73              error_code = 404
74              return render_template('error.html', error=error_message, error_code=error_code)
    , 404
75
76          lat = geocode_data[0]['lat']
77          lon = geocode_data[0]['lon']
78
79          air_quality_url = f"http://api.openweathermap.org/data/2.5/air_pollution?lat={lat}&
    lon={lon}&appid={API_KEY}"
80          air_quality_response = requests.get(air_quality_url)
81          if air_quality_response.status_code != 200:
82              error_message = "Failed to fetch Air Quality Index data"
83              error_code = 500
84              return render_template('error.html', error=error_message, error_code=error_code)
    , 500
85
86          air_quality_data = air_quality_response.json()['list'][0]['components']
87          sample = [
88              [air_quality_data['pm2_5'], air_quality_data['pm10'], air_quality_data['o3'],
89               air_quality_data['no2'], air_quality_data['co'], air_quality_data['so2']]
90          ]
91          prediction = round(model.predict(sample)[0], 2)
92
93          # Generate a line plot for historical AQI of the city
94          city_data = data[data['City'].str.lower() == city_name.lower()]
95          if city_data.empty:
96              error_message = "No historical data available for this city."
97              error_code = 404
98              return render_template('error.html', error=error_message, error_code=error_code)
    , 404
99
100         plt.figure(figsize=(10, 6))
101         plt.plot(city_data['Date'], city_data['AQI'], label='Historical AQI', color='blue')
102         plt.scatter([city_data['Date'].iloc[-1]], [prediction], color='red', label=f'
    Predicted AQI: {prediction}', zorder=5)
103         plt.title(f'AQI Trends for {city_name}', fontsize=16)
104         plt.xlabel('Date', fontsize=14)
105         plt.ylabel('AQI', fontsize=14)
106         plt.legend()
107         plt.grid(True)
108
109         # Save the plot to the static folder
110         plot_path = os.path.join(app.root_path, 'static', 'aqi_plot.png')
111         print(f"Saving plot to: {plot_path}")  # Debug statement
112         plt.savefig(plot_path)
113         plt.close()
114
115         # Render results template with the plot
116         result, conclusion = determine_air_quality(prediction)
117         return render_template(
118             'results.html',
119             prediction=prediction,
120             result=result,
121             conclusion=conclusion,
122             plot_url=plot_path
123         )
124     else:
125         return render_template('city.html')
126
127 @app.route('/compare', methods=['GET', 'POST'])
128 def compare():
129     if request.method == 'POST':
130         # Get form data
131         city1 = request.form['city1']
132         city2 = request.form['city2']
133         time_range = int(request.form['time_range'])  # Get the selected time range in days
134
135         # Filter AQI data for the selected cities
136         city1_data = data[data['City'].str.lower() == city1.lower()]
137         city2_data = data[data['City'].str.lower() == city2.lower()]
138
139         # Check if data exists for both cities
140         if city1_data.empty or city2_data.empty:
```

```
            error_message = "One or both cities not found in the dataset. Please try again."
            return render_template('compare.html', error_message=error_message)

        # Convert the 'Date' column to datetime (with day-first format)
        city1_data['Date'] = pd.to_datetime(city1_data['Date'], dayfirst=True)
        city2_data['Date'] = pd.to_datetime(city2_data['Date'], dayfirst=True)

        # Filter data for the selected time range
        end_date = city1_data['Date'].max()  # Get the latest date in the data
        start_date = end_date - pd.Timedelta(days=time_range)

        city1_data = city1_data[(city1_data['Date'] >= start_date) & (city1_data['Date'] <= end_date)]
        city2_data = city2_data[(city2_data['Date'] >= start_date) & (city2_data['Date'] <= end_date)]

        # Check if both datasets have data in the filtered range
        if city1_data.empty or city2_data.empty:
            error_message = f"One or both cities do not have data for the past {time_range} days. Please try another range or cities."
            return render_template('compare.html', error_message=error_message)

        # Calculate average AQI
        city1_avg_aqi = city1_data['AQI'].mean()
        city2_avg_aqi = city2_data['AQI'].mean()

        # Generate text-based conclusion
        if city1_avg_aqi < city2_avg_aqi:
            conclusion = (
                f"{city1} has a better air quality on average with an AQI of {city1_avg_aqi:.2f}, "
                f"compared to {city2}, which has an average AQI of {city2_avg_aqi:.2f}."
            )
        elif city1_avg_aqi > city2_avg_aqi:
            conclusion = (
                f"{city2} has a better air quality on average with an AQI of {city2_avg_aqi:.2f}, "
                f"compared to {city1}, which has an average AQI of {city1_avg_aqi:.2f}."
            )
        else:
            conclusion = f"Both {city1} and {city2} have similar air quality with an average AQI of {city1_avg_aqi:.2f}."

        # Create the comparison plot
        plt.figure(figsize=(12, 6))
        plt.plot(city1_data['Date'], city1_data['AQI'], label=f"{city1} AQI", color='blue', linewidth=2)
        plt.plot(city2_data['Date'], city2_data['AQI'], label=f"{city2} AQI", color='red', linewidth=2)
        plt.xlabel('Date', fontsize=14)
        plt.ylabel('Air Quality Index (AQI)', fontsize=14)
        plt.title(f"AQI Comparison: {city1} vs {city2} (Last {time_range} Days)", fontsize=16)
        plt.legend()

        # Adjust x-axis ticks for better readability
        plt.xticks(rotation=45)
        plt.tight_layout()

        # Save the plot
        plot_path = os.path.join(app.root_path, 'static', 'comparison_plot.png')
        plt.savefig(plot_path)
        plt.close()

        return render_template(
            'comparison_results.html',
            city1=city1,
            city2=city2,
            plot_url='/static/comparison_plot.png',
            conclusion=conclusion
        )

    return render_template('compare.html')
```

```
209
210
211
212 def determine_air_quality(prediction):
213     if prediction < 50:
214         return 'Air Quality Index is Good', 'The Air Quality Index is excellent. It poses
    little or no risk to human health.'
215     elif 51 <= prediction < 100:
216         return 'Air Quality Index is Satisfactory', 'The Air Quality Index is satisfactory,
    but there may be a risk for sensitive individuals.'
217     elif 101 <= prediction < 200:
218         return 'Air Quality Index is Moderately Polluted', 'Moderate health risk for
    sensitive individuals.'
219     elif 201 <= prediction < 300:
220         return 'Air Quality Index is Poor', 'Health warnings of emergency conditions.'
221     elif 301 <= prediction < 400:
222         return 'Air Quality Index is Very Poor', 'Health alert: everyone may experience more
     serious health effects.'
223     else:
224         return 'Air Quality Index is Severe', 'Health warnings of emergency conditions. The
    entire population is more likely to be affected.'
225
226 if __name__ == '__main__':
227     app.run(debug=True)
```

Listing 1: code implementation of app.py