Link to GitHub Repository

A. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

We planned to work with

1. https://weatherstack.com/dashboard
   d3e23e370dffc1de116d4a28d2e71c41
2. API: Makcorps
3. curl:"https://api.makcorps.com/expedia?hotelid=1450057&currency=USD&rooms=1&adults=2&checkin=2025-12-10&checkout=2025-12-11&api_key=6576a85d9796563d73e34228401"

We wanted to calculate average hotel prices vs weather

B. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

We worked with

1. Weather: **f86bdf89ba9b235fa5ed7e263b6808bd** api.openweathermap.org
2. AQI: Key - B7D7AAE3-FBF1-4349-990F-9302D082B292
   - URL:https://www.airnowapi.org/aq/forecast/zipCode/?format=application/json&zipCode=73008&date=2025-04-15&distance=1000&API_KEY= B7D7AAE3-FBF1-4349-990F-9302D082B292

- The data that we gathered from the weather API was the temperature, humidity, and air pressure in 100 different zip codes.
- The data that we gathered from the air quality index API was the AQI and type of pollutant that occurred in the same group of zip codes.
- From these two API's we calculated the average value of these statistics.

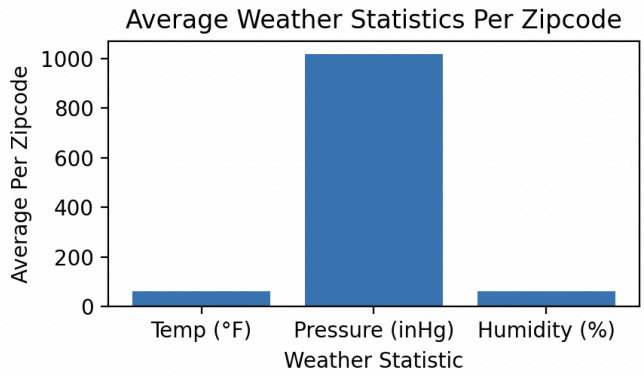
C. The problems that you faced (10 points)
   - We faced a number of issues in our code first we had trouble only accessing 25 rows at a time and then we had trouble building a joint database
   - Had some trouble converting the averages to a text file
   - We had trouble adjusting the margins of the visualizations to fit the screen without being too wide, while also making sure that the values on the axes weren't obstructed

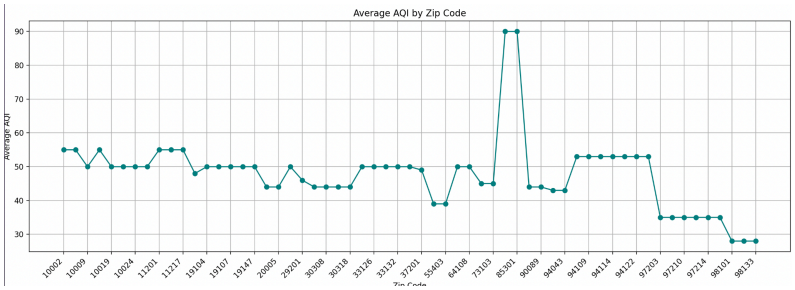
D. The calculations from the data in the database (i.e. a screenshot) (10 points)

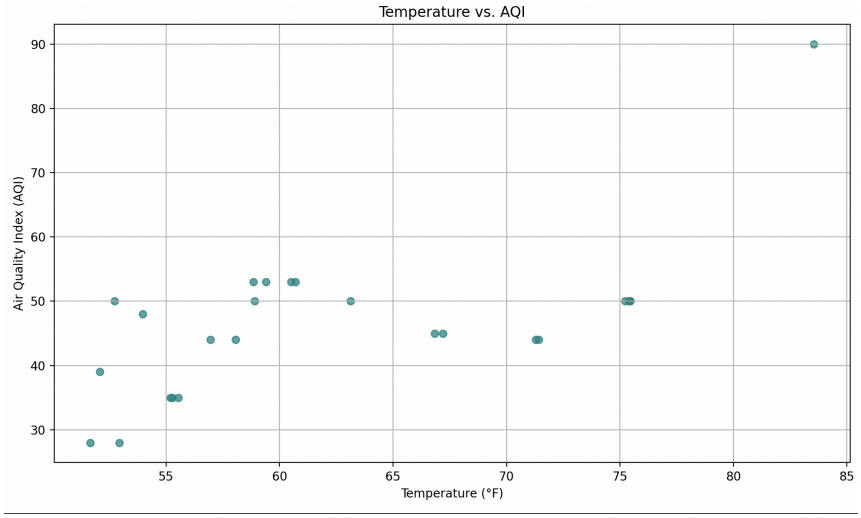
final_project_averages.txt

```
Average AQI and Temperature by Pollutant
Pollutant: O3
  Average AQI: 48.38
  Average Temperature: 62.23 °F

Pollutant: PM2.5
  Average AQI: 43.62
  Average Temperature: 56.97 °F

 Average Temp From Weather: 62.89810126582278
 Avergae Humidity From Weather: 64.9367088607595
 Average Pressure From Weather: 1016.6835443037975

```

E. The visualization that you created (i.e. screenshot or image file) (10 points)







F. Instructions for running your code (10 points)
   - To run the code you will need to run both the Weather1.py file and the AQI.py file until they reach 100 rows in their respective tables. Then you will need to run the joint.py to create the joint table. These tables can be located in a database titled “final_project.db”
   - In order to see our visualizations, they can be found by running the files matplot.py, matplot2.py, and matplot3.py as long as there is data existing in the database.
   - In order to find our calculations from the data, run the file averages.py to open the text file called final_project_averages.txt.

G. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

<u>Weather1.py</u>

1. create_db()
   - Description: Creates the WeatherData table in the SQLite database if it doesn't already exist.
   - Inputs: None
   - Outputs: None
2. fetch_and_clean(zip_code)

- Description: Fetches weather data from the OpenWeather API for a given ZIP code. It extracts temperature, pressure, and humidity values, and returns them as a cleaned list of tuples.
- Inputs: zip_code (string) — a ZIP code to request weather data for.
- Outputs: A list of one tuple: (zip, temp, pressure, humidity). Returns an empty list if the API request fails.

3. insert_weather_data(records)
    - Description: Inserts weather data records into the WeatherData table. It uses INSERT OR IGNORE to prevent duplicate entries for the same ZIP code.
    - Inputs: records (list of tuples) — each tuple contains (zip, temp, pressure, humidity).
    - Outputs: None
4. main()
    - Description: The main function that creates the database table, determines which ZIP codes are left to fetch, gets weather data for 25 of them, inserts the results, and prints how many rows are now in the WeatherData table.
    - Inputs: None
    - Outputs: None

AQI.py

1. create_tables()
    - Description: Creates two tables in the database if they don't already exist: AirQualityData and Pollutants. AirQualityData stores ZIP codes, AQI values, and pollutant references. Pollutants stores unique pollutant names with IDs.
    - Inputs: None
    - Outputs: None
2. fetch_air_quality(zip_code)
    - Description: Sends a request to the AirNow API for a given ZIP code and retrieves the AQI and pollutant type. Cleans the result and formats it into a list of tuples.
    - Inputs: zip_code (string) — the ZIP code to query.
    - Outputs: A list of tuples in the format [(zip, aqi, pollutant)] or an empty list if the API call fails.
3. insert_air_quality(records)
    - Description: Inserts AQI records into the AirQualityData table. If the pollutant does not already exist in the Pollutants table, it inserts it with a new ID. Uses manual ID control to avoid AUTOINCREMENT gaps.
    - Inputs: records (list of tuples) — each tuple includes (zip, aqi, pollutant).
    - Outputs: None
4. main()

- Description: Main function that sets up the database tables, checks which ZIP codes are left to process, fetches new air quality data for 25 random ZIPs, inserts the data, and prints the updated row count.
- Inputs: None
- Outputs: None

Joint.py

1. print_sample_tables()
    - Description: Connects to the database and prints the first 5 rows from both the WeatherData and AirQualityData tables. Used for checking sample entries.
    - Inputs: None
    - Outputs: None (prints output to console)
2. run_join_query()
    - Description: Executes an SQL JOIN between the WeatherData and AirQualityData tables based on ZIP codes. Selects weather and air quality data, displays the first 25 joined results, and saves them into a new table called WeatherAirQualityJoin.
    - Inputs: None
    - Outputs: None (prints results and writes to database)
3. main()
    - Description: The main function that calls print_sample_tables() and run_join_query(). It serves as the entry point for the script.
    - Inputs: None
    - Outputs: None
    -

H. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
4/16/25	Structuring overall code	Discussion	Yes, had help getting ideas for how to structure
4/17/25	Not printing 25 rows in AQI.py	Zoom-OH	Yes, was able to help me resolve the issue
4/17/25	Debugging issues with	Chat GPT	Yes, was able to properly format after using Chat GPT and going to OH

	the 25 rows in AQI.py		
4/21/25	Had duplicate string data	Zoom-OH	Yes, created a new table where duplicate string data could be addressed
4/10/25	Didn't have two working API's	In class	Yes, we got introduced to a second API that ended up working and we used in our project.

Here are the notes that we took during our grading session on 4/17. Since then, we have completed the following tasks that we would have otherwise received points on:

Presentation notes:

- ☐ In the aqi table - replace with the id of the new table
- ☐ Make a new table that can be used for our calculations and visualizations
- ☐ Write out calculations from matplot to a txt file
- ☐ Have a title for the graph
- ☐ Make the x axis more descriptive
- ☐ Use matplotlib to select data from database correctly about the AQI data
- ☐ In the final database need to limit aqi to 25
- ☐ DEAL WITH DUPLICATE STRING DATA