Group Members: Emma Roth, Zumua Yesmin, Grace Maki

SI 206 Final Project Report

Github Link: https://github.com/emmaroth12/SI-206-Project-.git

Goals and Plan

Objective: We aim to find the averages for the UV Indexes, Temperatures, and Sunrise and Sunset times for Ann Arbor between the months of August and December.

APIs/Website:

- **UV Index**: We used the Open UV API (https://www.openuv.io/) in order to calculate the UV index averages.
- **Temperature**: We used Open Meto API (https://open-meteo.com/) in order to calculate the average temperatures.
- **Sunrise and Sunset Times**: We used Sunrise Sunset API (https://sunrisesunset.io/api/) in order to calculate the average sunset and sunrise times, and the hours of daylight over each month, based on geographic coordinates.

Achieved Goals

- **UV Index**: We used an open UV API website (https://www.openuv.io/) in order to gather the information. This website helped us calculate the average UV for each day during the months of August to September. We first gathered the data including the UV max and min. Then, we used that information in order to calculate the average UV for each day.
- **Temperature**: Similarly, we used an open weather api website (https://open-meteo.com/) in order to gather the information. This website helped gather the historical weather data to calculate the daily average temperature(C°). The API provided a longitude and latitude and hourly temperature at 2 meters off the ground for a date range starting from August 1 to Nov 30, 2024.
- **Sunset and Sunrise times**: We aimed to calculate the average sunrise and sunset times for Ann Arbor between the months of August to December. To achieve this goal, we worked with an open sunrise and sunset API website (https://sunrisesunset.io/api/) in order to gather the information. This website provided accurate sunrise and sunset times based on geographic coordinates. The data gathered was the sunrise and sunset times for each day from August 1st to December 31st. We did this by gathering all the dates corresponding to the times collected for cross-referencing and calculations. Then, we used this data to calculate the average sunrise and sunset times for each month. As well as analyzing the hours of daylight over each month.

Problems Faced

- **API Keys**: We had plans to use specific API keys, but they ended up not working out because they were not free. This was challenging because we had to find other sources that gave API information and were free. It was time-consuming to find these other sources, but eventually, we found them.
- **Data Limitation**: We also had trouble limiting the code to only 25 pieces of data at a time. This took some time for us to figure out.
- **Technical Challenges**: For the sunrise and sunset API, we had trouble importing the requests module in Python. We tried to use Stackoverflow to help solve the problem, but we ended up figuring out the problem by looking at past discussion assignments. Additionally, for the sunrise and sunset API, we had trouble in plotting both sunrise and sunset times on a graph because we needed to convert the time data into hours for plotting. We used UM ChatGPT for guidance in starting the code and we were able to convert the sunrise and sunset times into hours for plotting.

Data Calculations

UV Index Calculation - Finding the Average of UV Index
Method:

```
# Function to calculate the average UV index from the stored data
def calculate_average_uv():
    conn = sqlite3.connect('uv_data.db')
    cursor = conn.cursor()

    cursor.execute('''SELECT uv_data.date, uv_data.max_uv, uv_data.min_uv, locations.latitude, locations.longitude
                      FROM uv_data
                      JOIN locations ON uv_data.location_id = locations.id''')
    rows = cursor.fetchall()

    conn.close()

    if rows:
        total_uv = sum([row[1] for row in rows])
        return total_uv / len(rows)
    else:
        return 0


def write_calculation_to_file(average_uv):
    with open('calculation_output.txt', 'w') as f:
        f.write(f'Average UV Index: {average_uv:.2f}\n')
```

```
# Calculate the average UV index
average_uv = calculate_average_uv()
print(f"Average UV index: {average_uv:.2f}")

# Write the calculated data to a file
write_calculation_to_file(average_uv)
```

Output:

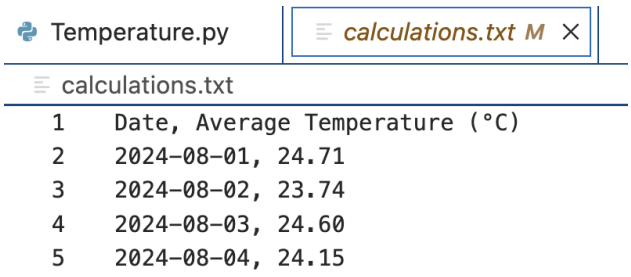
```
calculation_output.txt
1    Average UV Index: 4.50
```

Temperature Calculation - Finding the Average of Temperature
Method:

```
91  # Perform calculations on the data
92  def calculate_daily_averages(db_name):
93      conn = sqlite3.connect(db_name)
94      cursor = conn.cursor()
95      cursor.execute('SELECT date, temps FROM weather')
96      rows = cursor.fetchall()
97      conn.close()
98
99      # Calculate averages
100     results = []
101     for date, temps_str in rows:
102         temps = list(map(float, temps_str.split(",")))  # Convert back to list of floats
103         avg_temp = sum(temps) / len(temps)  # Calculate average
104         results.append({"date": date, "avg_temp": avg_temp})
105
106     return results
107
```

```
# Save calculations to file
def save_calculations_to_file(calculations, file_name="calculations.txt"):
    with open(file_name, "w") as file:
        file.write("Date, Average Temperature (°C)\n")
        for record in calculations:
            file.write(f"{record['date']}, {record['avg_temp']:.2f}\n")
    print(f"Calculations written to {file_name}.")
```

Output:



Sunrise and Sunset Calculation - Finding the Average of Sunrise and Sunset
Method:

```python
# calculate data
def process_and_calculate_data():
    conn = sqlite3.connect('sunrise_sunset.db')
    cur = conn.cursor()

    # join Dates and SunriseSunset tables
    cur.execute('''
    SELECT d.date, s.sunrise, s.sunset
    FROM Dates d
    JOIN SunriseSunset s ON d.id = s.date_id
            ''')

    rows = cur.fetchall()

    # calculations
    day_counts = [0] * 7
    sunrise_times = []
    sunset_times = []
    dates = []

    for row in rows:
        date_str, sunrise_str, sunset_str = row
        date_obj = datetime.strptime(date_str, '%Y-%m-%d')

        # Count each day of the week (Sunday=0, Monday=1, ..., Saturday=6)
        day_of_week = date_obj.weekday()
        day_counts[day_of_week] += 1

        sunrise_time = datetime.strptime(sunrise_str, '%I:%M:%S %p').time()
        sunset_time = datetime.strptime(sunset_str, '%I:%M:%S %p').time()

        dates.append(date_obj)
        sunrise_times.append(sunrise_time)
        sunset_times.append(sunset_time)

    # average sunrise and sunset times
    avg_sunrise_time = average_time(sunrise_times)
    avg_sunset_time = average_time(sunset_times)
```

```python
    # write results to a file
    with open('calculated_data.txt', 'w') as file:
        file.write('Day of the week counts (Sunday=0, ..., Saturday=6):\n')
        file.write(', '.join(f'{day}: {count}' for day, count in enumerate(day_counts)) + '\n')
        file.write(f'Average sunrise time: {avg_sunrise_time}\n')
        file.write(f'Average sunset time: {avg_sunset_time}\n')

    conn.close()
    return day_counts, sunrise_times, sunset_times, dates
```

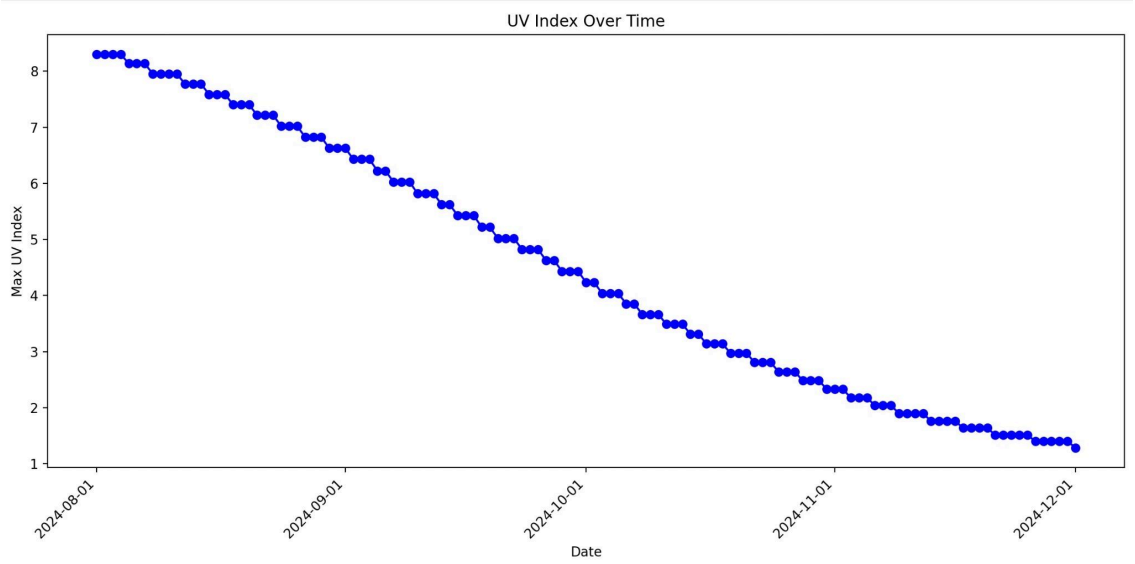
```python
# how to caluclate average time for function above: process_and_calculate_data
def average_time(times):
    total_seconds = sum(t.hour * 3600 + t.minute * 60 + t.second for t in times)
    avg_seconds = total_seconds // len(times)
    return f'{avg_seconds // 3600:02}:{(avg_seconds % 3600) // 60:02}:{avg_seconds % 60:02}'

# Calculate difference between sunset and sunrise times
def calculate_difference(sunrise_times, sunset_times):
    differences = []
    for sunrise_time, sunset_time in zip(sunrise_times, sunset_times):
        sunrise_hour = time_to_hours(sunrise_time)
        sunset_hour = time_to_hours(sunset_time)
        difference = sunset_hour - sunrise_hour
        differences.append(difference)
    return differences

# convert times to hours for plotting
def time_to_hours(t):
    return t.hour + t.minute / 60 + t.second / 3600
```

Output:

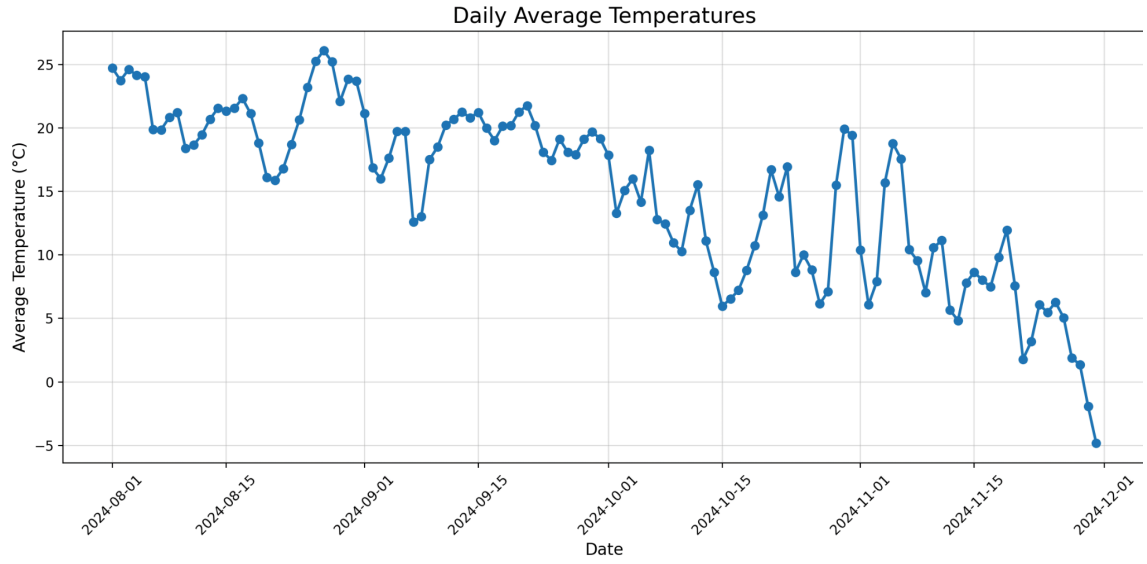
```
Day of the week counts (Sunday=0, ..., Saturday=6):
0: 296, 1: 296, 2: 297, 3: 297, 4: 297, 5: 296, 6: 296
Average sunrise time: 07:08:21
Average sunset time: 19:22:11
```

Visualizations

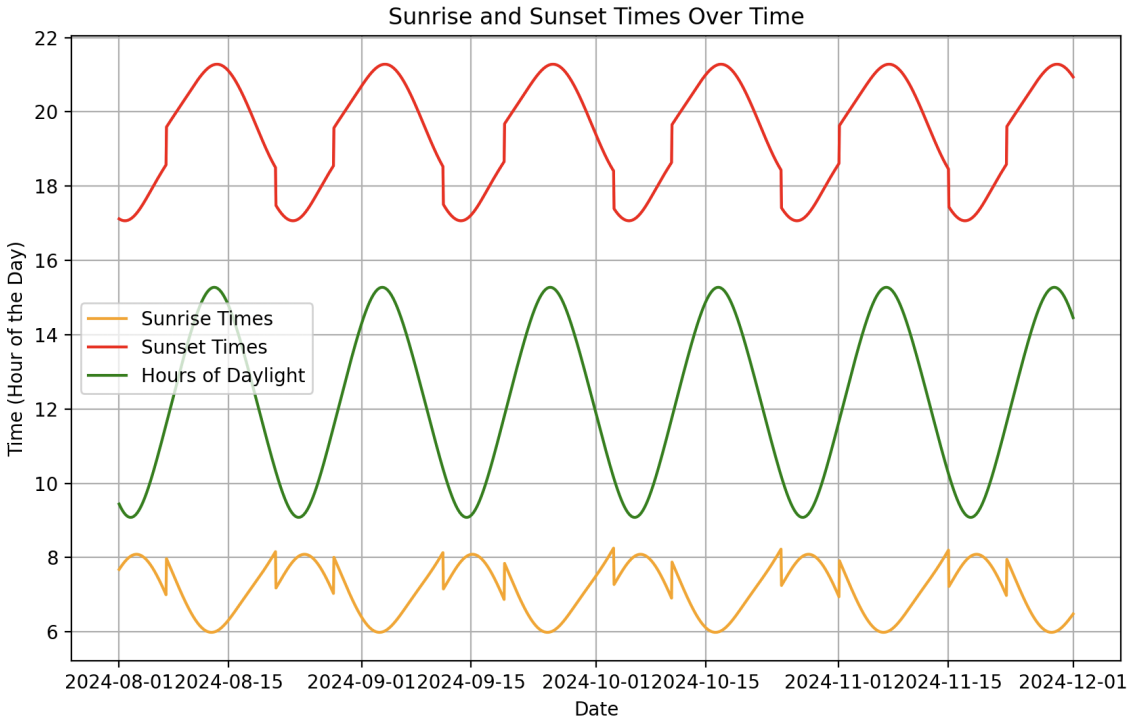
UV Index - Finding the Average of UV Index



Temperature - Finding the Average of Temperature:



Sunrise and Sunset - Finding the Average of Sunrise and Sunset:



Instructions for Running Code

UV Index:

1. Install Required Libraries: Run the following command to install the necessary libraries:
    a. requests for making API requests.
    b. matplotlib for generating plots.
    c. pip install requests matplotlib
2. Obtain OpenUV API Key:
    a. Sign up at OpenUV to get your API key.
    b. In the script, replace the placeholder for UV_API_KEY with your actual API key.
3. Set the Location:
    a. In the script, update the latitude, longitude, and altitude values to match your desired location.
4. Set the Date Range:
    a. Modify the start_date and end_date variables to specify the period for which you want to collect UV data.
5. Run the Script:
    a. Open a terminal or command prompt.
    b. Navigate to the folder where the script is located.
    c. Execute the script
6. What the Script Does:

    a. Creates a database (uv_data.db) to store UV data.
    b. Collects UV data for each day in the specified date range from the OpenUV API.
    c. Calculates the average UV index for the period.
    d. Saves the average UV index in a text file (calculation_output.txt).
    e. generates a plot of UV index over time (saved as a PNG file).
7. Review the Results:
    a. Check the database for stored UV data.
    b. Open the text file for the calculated average UV index.
    c. If generated, view the plot image file showing UV index over time.

Temperature:

1. Install Required Libraries:
    a. Requests for making API requests and receiving their responses.
    b. SQLite3 to store data and connect with the database.
    c. defaultdict dictionary to provide default value for missing keys.
    d. datetime and time delta for working with dates and times.
    e. matplotlib.pyplot for generating plots.
    f. Pandas to manage and process data efficiently.
        i. Install the libraries: ensure you've installed the required libraries by running pip install requests matplotlib pandas
2. Prepare a folder:
    a. Create a file with the provided Python code
3. Set Your API Parameters:
    a. Update the latitude and longitude to match your desired location.
4. Set Date Range:
    a. change the start and end dates in the main_with_visualization()
5. Run code:
    a. Run the Python script by opening a terminal or command prompt.
    b. Execute the script
6. Review Results:
    a. Review the database with a table containing the data
    b. Open text file for average temperature output
    c. View visualization plotting the daily average temperature(C)

Sunset and Sunrise Times:

1. Install Python:
    a. Ensure you have Python 3.6 or above installed on your computer
2. Prepare a Folder:
    a. Create a new folder on your computer where you want to keep the project file
3. Download the Code:

    a. Copy the provided script into a new file called sunrise_sunset_analysis.py and save it in the folder you created
4. Open Command Prompt or Terminal:
    a. Navigate to the folder where you saved sunrise_sunset_analysis.py.
    b. For Windows: Type cmd in the search bar and press Enter to open the Command Prompt.
    c. For MacOS/Linux: Open Terminal from the Applications menu
5. Install Required Packages:
    a. In the same Command Prompt or Terminal, type:
        i. pip install requests pandas matplotlib
            1. requests: This library needs to be installed using pip.
            2. sqlite3: This library is included with Python's standard library, so you typically do not need to install it separately.
            3. datetime: This library is also part of Python's standard library, so no need to install it.
            4. pandas: This needs to be installed using pip.
            5. matplotlib: This needs to be installed using pip.
6. Run the Code:
    a. In the Command Prompt or Terminal, type:
        i. python sunrise_sunset_analysis.py
7. See the Results:
    a. The script will create a database file named sunrise_sunset.db.
    b. It will also generate a file called calculated_data.txt containing the calculated average sunrise and sunset times.
    c. A graph will pop up showing sunrise and sunset times over the specified period.

Documentation for Each Function

UV Index Functions - For average of UV:

1. create_tables()
    a. Purpose: This function sets up the necessary tables in the SQLite database.
    b. Input: None
    c. Output: None
    d. This function creates two tables:
        i. locations: Stores the latitude, longitude, and altitude of the location.
        ii. uv_data: Stores the UV data (max UV, min UV, and time of max UV) for each day, linked to a location.
2. Get_uv_index_for_date (latitude, longitude, altitude, date)
    a. Purpose: Fetches the UV index data from the OpenUV API for a specific date and location.

b. Input:
    i. latitude (float): The latitude of the location
    ii. longitude (float): The longitude of the location.
    iii. altitude (float): The altitude of the location in meters.
    iv. date (string): The date for which UV data is being requested in YYYY-MM-DD format.
c. Output:
    i. Returns a dictionary with the following keys
    ii. date (string): The requested date.
    iii. max_uv (float): The maximum UV index for the day.
    iv. min_uv (float): The minimum UV index for the day.
    v. max_uv_time (string): The time at which the maximum UV occurred.
    vi. If the API request fails or data is unavailable, it returns None.

3. Collect_uv_month_data (latitude, longitude, altitude, start_date, end_date)
    a. Purpose: Collects UV data for a specified date range (month or more) for a given location.
    b. Input:
        i. latitude (float): The latitude of the location.
        ii. longitude (float): The longitude of the location.
        iii. altitude (float): The altitude of the location in meters.
        iv. start_date (string): The start date for the data collection in YYYY-MM-DD format.
        v. end_date (string): The end date for the data collection in YYYY-MM-DD format.
    c. Output:
        i. Returns a list of dictionaries where each dictionary contains:
            1. date (string): The date of the UV data.
            2. max_uv (float): The maximum UV index for that date.
            3. min_uv (float): The minimum UV index for that date.
            4. max_uv_time (string): The time when the maximum UV occurred.
            5. If no data exists for a specific date, it fetches the data from the OpenUV API and appends it to the list.
4. store_uv_data_with_min_max(uv_data, latitude, longitude, altitude)
    a. Purpose: Stores the collected UV data in the SQLite database, linking it to the correct location.
    b. Input:
        i. uv_data (list of dicts): A list of dictionaries, where each dictionary contains UV data for a specific date (e.g., date, max_uv, min_uv, max_uv_time).
        ii. latitude (float): The latitude of the location.

iii. longitude (float): The longitude of the location.
iv. altitude (float): The altitude of the location in meters.

c. Output: None
i. This function stores the provided UV data into the uv_data table of the database. It also ensures that the location is inserted into the locations table if it does not already exist.

5. calculate_average_uv()
   a. Purpose: Calculates the average UV index from the stored data in the database.
   b. Input: None
   c. Output:
      i. Returns a float representing the average maximum UV index calculated from all stored UV data in the database.
      ii. If no data is found, it returns 0.
6. write_calculation_to_file(average_uv)
   a. Purpose: Writes the calculated average UV index to a text file.
   b. Input:
      i. average_uv (float): The average UV index to be written to the file.
   c. Output: None
      i. This function writes the average UV index to a text file (calculation_output.txt) in the format: Average UV Index: <value>.
7. create_visualizations()
   a. Purpose: Creates a plot of the UV index over time and saves it as a PNG file.
   b. Input: None
   c. Output: None
      i. This function generates a plot showing the UV index (max_uv) over time, with dates on the x-axis and UV index values on the y-axis. It then saves the plot as a PNG file (uv_index_plot.png).
8. main()
   a. Purpose: The main driver function that ties everything together. It collects UV data, stores it, calculates the average UV index, and generates the visualizations.
   b. Input: None
   c. Output: None
      i. This function does the following:
         1. Sets the location and date range.
         2. Creates necessary tables in the SQLite database.
         3. Collects and stores UV data.
         4. Calculates and writes the average UV index to a file.
         5. Creates visualizations and saves them as images.

Temperatures - For a daily average of Temperature:

1. Function: fetch_weather_data
    a. Purpose: grabs weather data at hourly temperatures from the API for a specified date range
    b. Input:
        i. Api_url: The base URL of the API
        ii. Params:
            1. Longitude
            2. Latitude
            3. hourly temperature
        iii. Start_date: The start date for fetching data in the format YYYY-MM-DD
        iv. End_date: The end date for fetching data in the format YYYY-MM-DD
    c. Output: Returns a list of dictionaries, each containing:
        i. date (str): The date (YYYY-MM-DD).
        ii. temps (list of floats): A list of hourly temperatures for that date.
        iii. If the request fails: return an empty list []

2. Function: setup_weather_database
    a. Purpose: set up an SQLite database with a weather table containing raw data
    b. Input:
        i. db_name (str): The name of the SQLite database file (weather database)
    c. Output:
        i. Returns conn: database connection object.
        ii. Returns cursor: cursor object to execute queries

3. Function: insert_weather_data
    a. Purpose: Insert data into the weather table in the database.
    b. Input:
        i. cursor: The SQLite cursor object to execute database queries.
        ii. Weather_data: A list of weather data where each dictionary contains:
        iii. date: The date “YYYY-MM-DD”.
        iv. temps: A list of floats for hourly temperatures for that date.
    c. Output: None

4. Function: get_last_inserted_date
    a. Purpose: Retrieve the most recent (latest) date stored in the database.
    b. Input:
        i. Cursor: SQL cursor object
    c. Output:
        i. Returns the latest date “YYYY-MM-DD” if data exists
        ii. If database empty: None

5. Function: store_data_in_batches
    a. Purpose: Access weather data in manageable chunks (up to batch-size rows) and store it in the database.
    b. Input:
        i. db_name (str): The name of the SQLite database file.
        ii. api_url (str): The base URL of the API.
        iii. params (dict): The parameters for the API request.
        iv. batch_size (int): The maximum number of rows to store in the database per run.
    c. Output: None

6. Function: calculate_daily_averages
    a. Purpose: read data from the database to calculate the daily average temperature
    b. Input:
        i. db_name (str): The name of the SQLite database file.
    c. Output:
        i. Returns a list of dictionaries containing date and average temperature.

7. Function:save_calculations_to_file
    a. Purpose: save the calculation to a text file
    b. Input:
        i. calculations (list of dicts): A list of daily averages, where each dictionary contains:
        ii. date (str): The date (YYYY-MM-DD).
        iii. avg_temp (float): The average temperature for that date.
        iv. file_name (str): The name of the file to save the calculations (default: calculations.txt).
    c. Output:
        i. This function writes the average temperature to a text file

8. Function: plot_data
    a. Purpose: Creates a line graph that plots the daily average temperature over time.
    b. Input:
        i. Calculations of the daily temperature averages
    c. Output:
        i. A line graph that charts the daily average temperature over time

9. Function: main_with_visualization
    a. Purpose: Call all the previous functions to execute to fetch and store data, calculate averages, save to a file, and visualize.

b. Input: None
c. Output: None

Sunrise and Sunset Functions - For an average sunset and sunrise times:

1. **Function: setup_database**
    a. Purpose: sets up an SQLite database and creates the necessary tables for storing sunrise and sunset data
    b. Input: None
    c. Output: None (creates a database file named sunrise_sunset.db with two tables: Dates and SunriseSunset)
2. **Function: get_sunrise_sunset**
    a. Purpose: retrieves sunrise and sunset times for a given location and date using the Sunrise Sunset API
    b. Input:
        i. latitude (float): The latitude of the location.
        ii. longitude (float): The longitude of the location.
        iii. date (str): The date in the format YYYY-MM-DD
    c. Output:
        i. results (dict): A dictionary containing the sunrise and sunset times if the request is successful.
        ii. None: If the request fails or the API response is invalid.
3. **Function: get_and_store_data**
    a. Purpose: Collects sunrise and sunset data for the next 25 days from the last stored date and saves it to the database
    b. Input: None
    c. Output: None (updates the database with new sunrise and sunset time for the next 25 days)
4. **Function: process_and_calculate_date**
    a. Purpose: Processes the data from the database to calculate average sunrise and sunset times and the duration of daylight for each day
    b. Input: None
    c. Output:
        i. day_counts (list of int): A list counting occurrences of each day of the week.
        ii. sunrise_times (list of time objects): A list of sunrise times.
        iii. sunset_times (list of time objects): A list of sunset times.
        iv. dates (list of datetime objects): A list of dates corresponding to the sunrise and sunset times.
5. **Function: average_time**
    a. Purpose: Calculates the average time from a list of time objects
    b. Input:

        i. times (list of time objects): A list of time objects to average
    c. Output:
        i. avg_time (str): The average time in HH:MM:SS format
6. **Function: time_to_hours**
    a. Purpose: Converts a time object to a numerical value representing the hours since midnight
    b. Input:
        i. t (time object): A time object to convert
    c. Output:
        i. hours (float): The numerical value of hours since midnight
7. **Function: visualize_data**
    a. Purpose: Generates and displays a graph of sunrise and sunset times over a specified period, along with the duration of daylight
    b. Input:
        i. day_counts (list of int): A list counting occurrences of each day of the week.
        ii. sunrise_times (list of time objects): A list of sunrise times.
        iii. sunset_times (list of time objects): A list of sunset times.
        iv. dates (list of DateTime objects): A list of dates corresponding to the sunrise and sunset times.
    c. Output: A plot showing sunrise and sunset times and the duration of daylight
8. **Function: main**
    a. Purpose: The main function that orchestrates the execution of all other functions to set up the database, collect and store data, process and visualize the results
    b. Input: None
    c. Output: None

Resources Used

- **APIs and Websites**:
    - https://www.openuv.io/
    - https://open-meteo.com/
    - https://sunrisesunset.io/api/
- **Assistance and Tools**:
    - Chat GPT for general debugging, introduced topics such as a map() to help iteration, setting limits, extract date for temperature, helping start code for graphs
    - Additionally, we used Chat GPT to help with merging our files together into one database