SI206 Final Project Report
Lily Wachtel & Ilana Eydelman
GitHub Link: https://github.com/ilana-ey/Lilana-Finalproject.git

Initial Goals and Data Collection

Originally, we wanted to see if there was a correlation between attendance at numerous converts/events during the summer of 2022 and the weather of those same summer months. We were able to find and work with a weather API, but unfortunately could not find a great event API with past concert dates, only future or current, which wouldn't work. We searched endlessly, and numerous concert or event APIs just didn't work and weren't too user friendly - we then switched our idea to collect data on Covid-19 from the summer of 2022 to compare with the weather data we had already collected.

We then planned to use two APIs, one regarding COVID-19 test results in New York the summer of 2022, and the other regarding the weather in New York that same summer of 2022. One of our goals was to see the correlation between the number of positive cases and the weather temperatures for that same month (From May 2022 to August 2022).

Goals Achieved and Actual Data Gathered

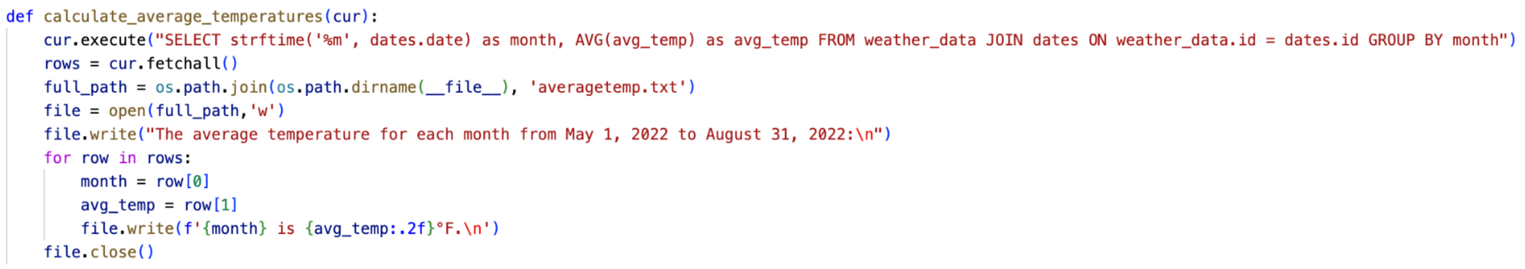
Fortunately, we were able to complete our goal written above, and we collected our Covid data from api.covidactnow.org and our weather data from weatherapi.com. We were able to gather a whole variation of data from the Covid API as well as from the weather API for the calculations. More specifically, as for information from the Covid API: amount of cases (number of cases per month), case density, positivity ratios, new cases weekly, infection rates, beds with covid patients, and finally, weekly covid admissions. As for information from Weather API: temperatures (average temperatures), min/max temperatures, maximum winds, precipitations and conditions.

Problems Faced

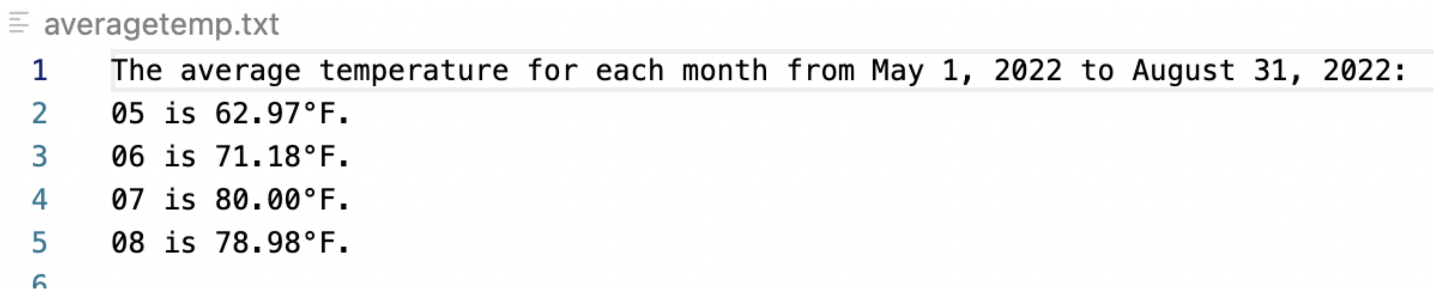
After moving on from the failed attempt to find an events API with past events (not future or current), we then were able to successfully find and work with a flight API (to potentially look for past flights from the dates we wanted), which worked until the API only let us call it a certain amount of times, so we then moved on from that until we found our Covid API, which worked perfectly.

First off (once we were settled with our Covid API), we struggled immensely to find an API/website that would work with us in giving us past data for Covid information (because there were some problems with security and privacy for certain Covid database websites). Second, we struggled with joining data to make visualizations, but we shortly figured that out. Third and lastly, when forming our visualizations, we encountered some difficulties with the neatness, actual visualization part of our graphs - the colors and writing were messed up at some points, but we were able to fix everything and make everything both legible and easy to read!

Calculations

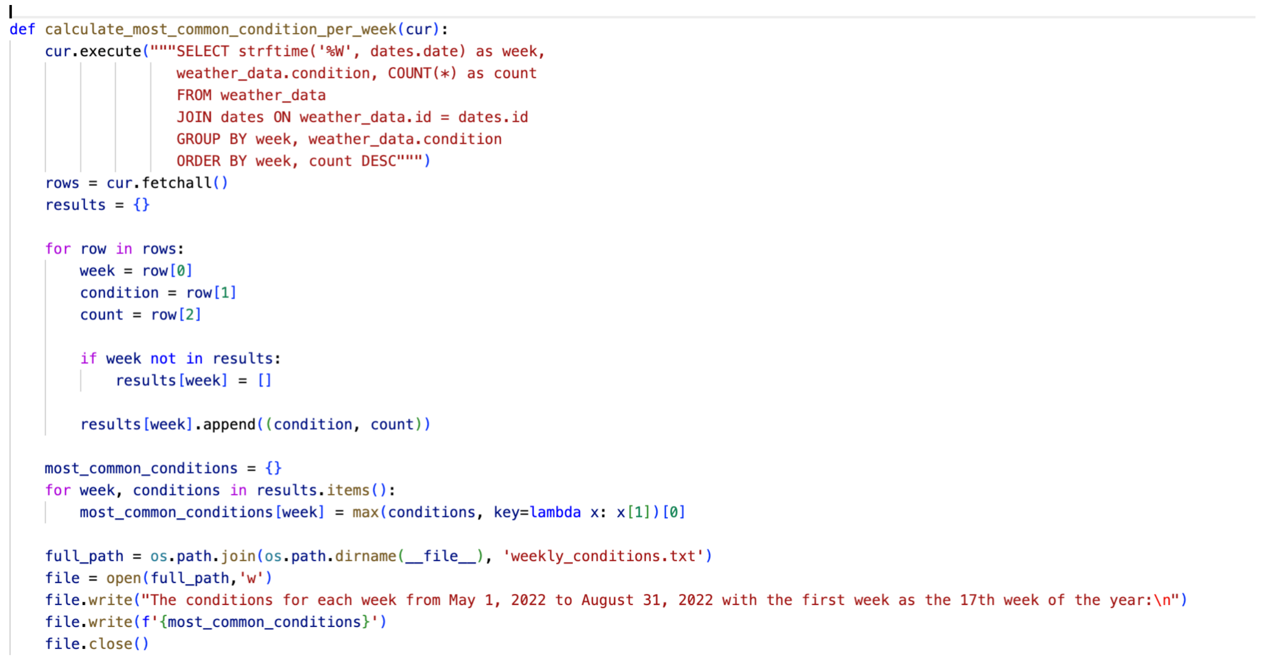


```python
def calculate_average_temperatures(cur):
    cur.execute("SELECT strftime('%m', dates.date) as month, AVG(avg_temp) as avg_temp FROM weather_data JOIN dates ON weather_data.id = dates.id GROUP BY month")
    rows = cur.fetchall()
    full_path = os.path.join(os.path.dirname(__file__), 'averagetemp.txt')
    file = open(full_path,'w')
    file.write("The average temperature for each month from May 1, 2022 to August 31, 2022:\n")
    for row in rows:
        month = row[0]
        avg_temp = row[1]
        file.write(f'{month} is {avg_temp:.2f}°F.\n')
    file.close()
```

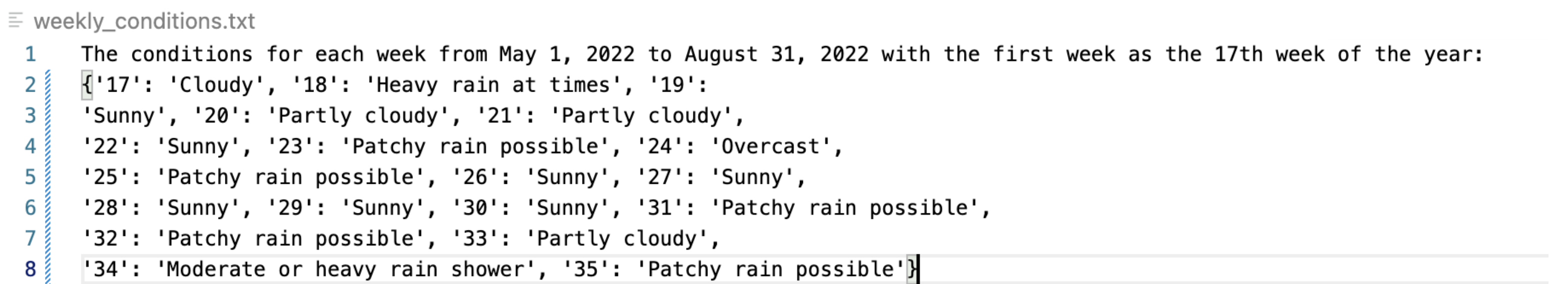


averagetemp.txt

```
The average temperature for each month from May 1, 2022 to August 31, 2022:
05 is 62.97°F.
06 is 71.18°F.
07 is 80.00°F.
08 is 78.98°F.
```

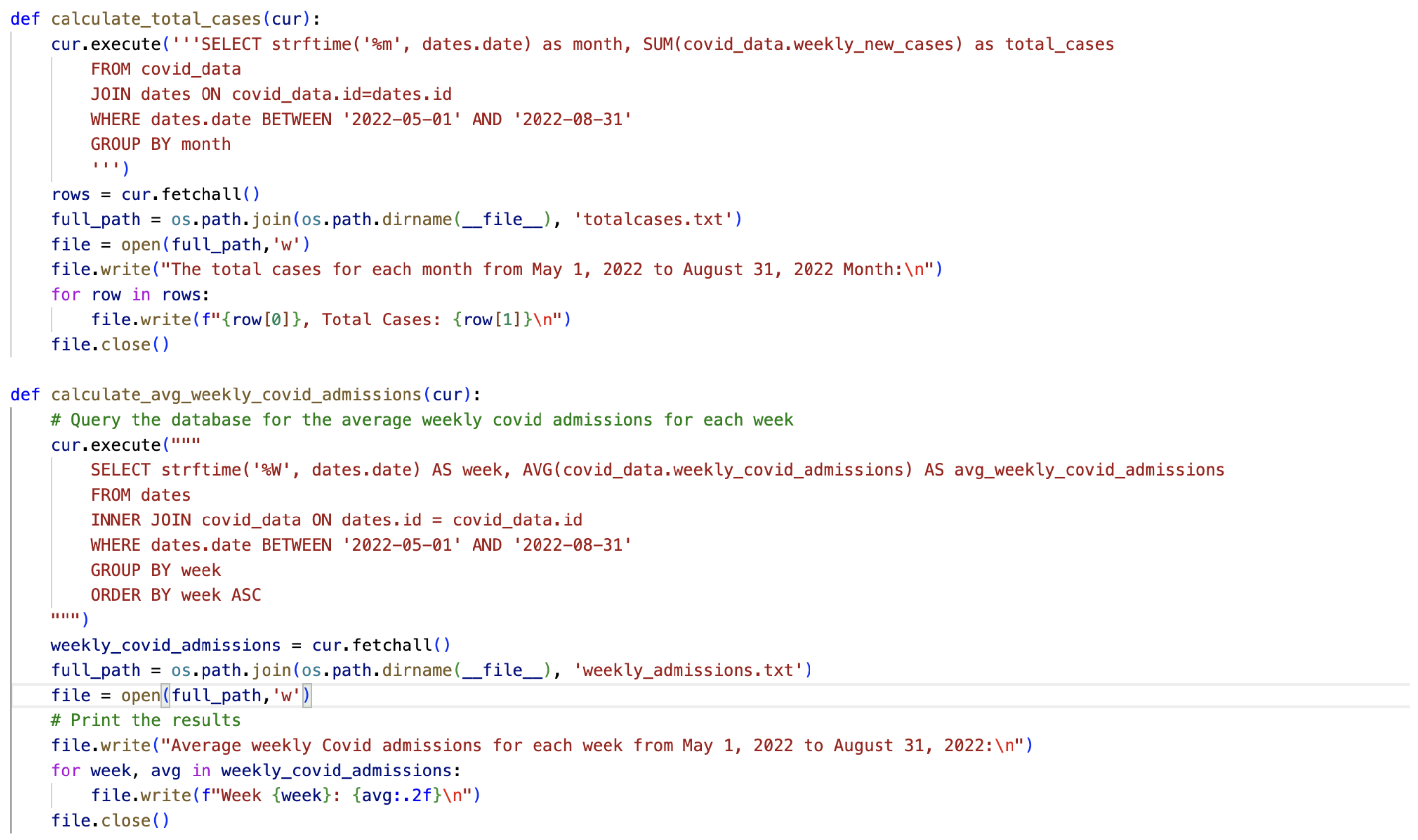


```python
def calculate_most_common_condition_per_week(cur):
    cur.execute("""SELECT strftime('%W', dates.date) as week,
                weather_data.condition, COUNT(*) as count
                FROM weather_data
                JOIN dates ON weather_data.id = dates.id
                GROUP BY week, weather_data.condition
                ORDER BY week, count DESC""")
    rows = cur.fetchall()
    results = {}

    for row in rows:
        week = row[0]
        condition = row[1]
        count = row[2]

        if week not in results:
            results[week] = []

        results[week].append((condition, count))

    most_common_conditions = {}
    for week, conditions in results.items():
        most_common_conditions[week] = max(conditions, key=lambda x: x[1])[0]

    full_path = os.path.join(os.path.dirname(__file__), 'weekly_conditions.txt')
    file = open(full_path,'w')
    file.write("The conditions for each week from May 1, 2022 to August 31, 2022 with the first week as the 17th week of the year:\n")
    file.write(f'{most_common_conditions}')
    file.close()
```

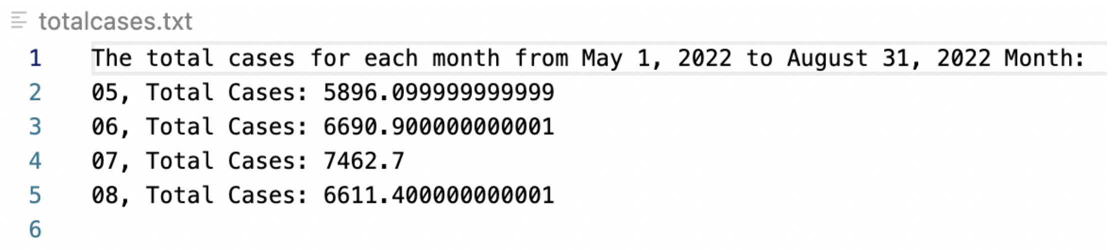


weekly_conditions.txt

```
The conditions for each week from May 1, 2022 to August 31, 2022 with the first week as the 17th week of the year:
{'17': 'Cloudy', '18': 'Heavy rain at times', '19':
'Sunny', '20': 'Partly cloudy', '21': 'Partly cloudy',
'22': 'Sunny', '23': 'Patchy rain possible', '24': 'Overcast',
'25': 'Patchy rain possible', '26': 'Sunny', '27': 'Sunny',
'28': 'Sunny', '29': 'Sunny', '30': 'Sunny', '31': 'Patchy rain possible',
'32': 'Patchy rain possible', '33': 'Partly cloudy',
'34': 'Moderate or heavy rain shower', '35': 'Patchy rain possible'}
```

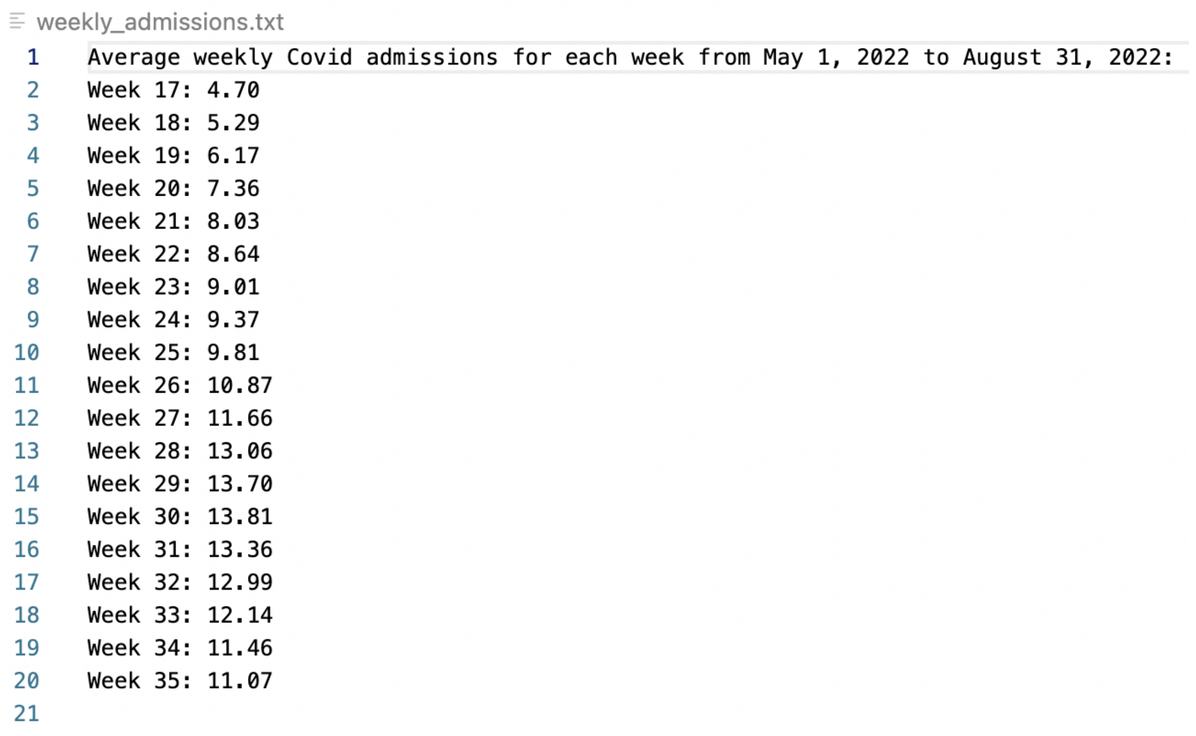


```python
def calculate_total_cases(cur):
    cur.execute('''SELECT strftime('%m', dates.date) as month, SUM(covid_data.weekly_new_cases) as total_cases
        FROM covid_data
        JOIN dates ON covid_data.id=dates.id
        WHERE dates.date BETWEEN '2022-05-01' AND '2022-08-31'
        GROUP BY month
        ''')
    rows = cur.fetchall()
    full_path = os.path.join(os.path.dirname(__file__), 'totalcases.txt')
    file = open(full_path,'w')
    file.write("The total cases for each month from May 1, 2022 to August 31, 2022 Month:\n")
    for row in rows:
        file.write(f"{row[0]}, Total Cases: {row[1]}\n")
    file.close()

def calculate_avg_weekly_covid_admissions(cur):
    # Query the database for the average weekly covid admissions for each week
    cur.execute("""
        SELECT strftime('%W', dates.date) AS week, AVG(covid_data.weekly_covid_admissions) AS avg_weekly_covid_admissions
        FROM dates
        INNER JOIN covid_data ON dates.id = covid_data.id
        WHERE dates.date BETWEEN '2022-05-01' AND '2022-08-31'
        GROUP BY week
        ORDER BY week ASC
    """)
    weekly_covid_admissions = cur.fetchall()
    full_path = os.path.join(os.path.dirname(__file__), 'weekly_admissions.txt')
    file = open(full_path,'w')
    # Print the results
    file.write("Average weekly Covid admissions for each week from May 1, 2022 to August 31, 2022:\n")
    for week, avg in weekly_covid_admissions:
        file.write(f"Week {week}: {avg:.2f}\n")
    file.close()
```



totalcases.txt

```
The total cases for each month from May 1, 2022 to August 31, 2022 Month:
05, Total Cases: 5896.099999999999
06, Total Cases: 6690.900000000001
07, Total Cases: 7462.7
08, Total Cases: 6611.400000000001

```

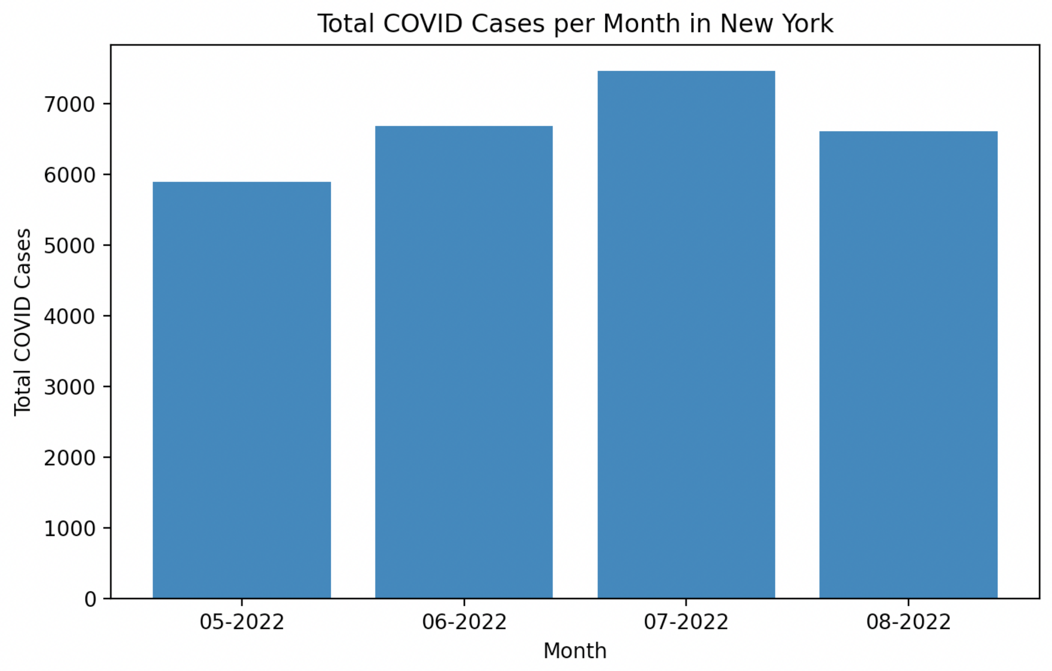


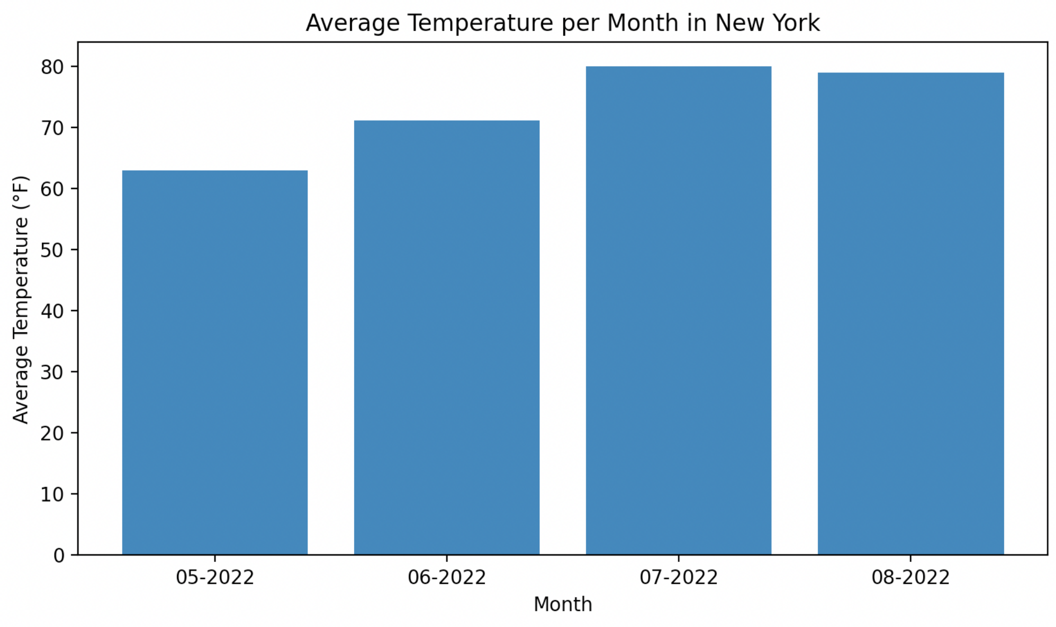
weekly_admissions.txt

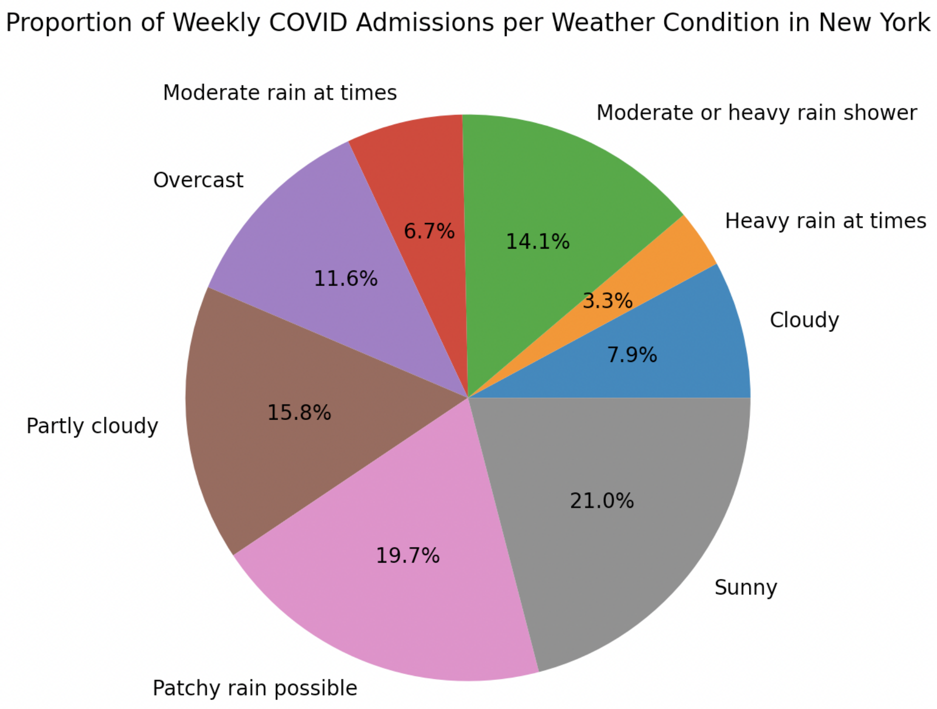
```
Average weekly Covid admissions for each week from May 1, 2022 to August 31, 2022:
Week 17: 4.70
Week 18: 5.29
Week 19: 6.17
Week 20: 7.36
Week 21: 8.03
Week 22: 8.64
Week 23: 9.01
Week 24: 9.37
Week 25: 9.81
Week 26: 10.87
Week 27: 11.66
Week 28: 13.06
Week 29: 13.70
Week 30: 13.81
Week 31: 13.36
Week 32: 12.99
Week 33: 12.14
Week 34: 11.46
Week 35: 11.07

```

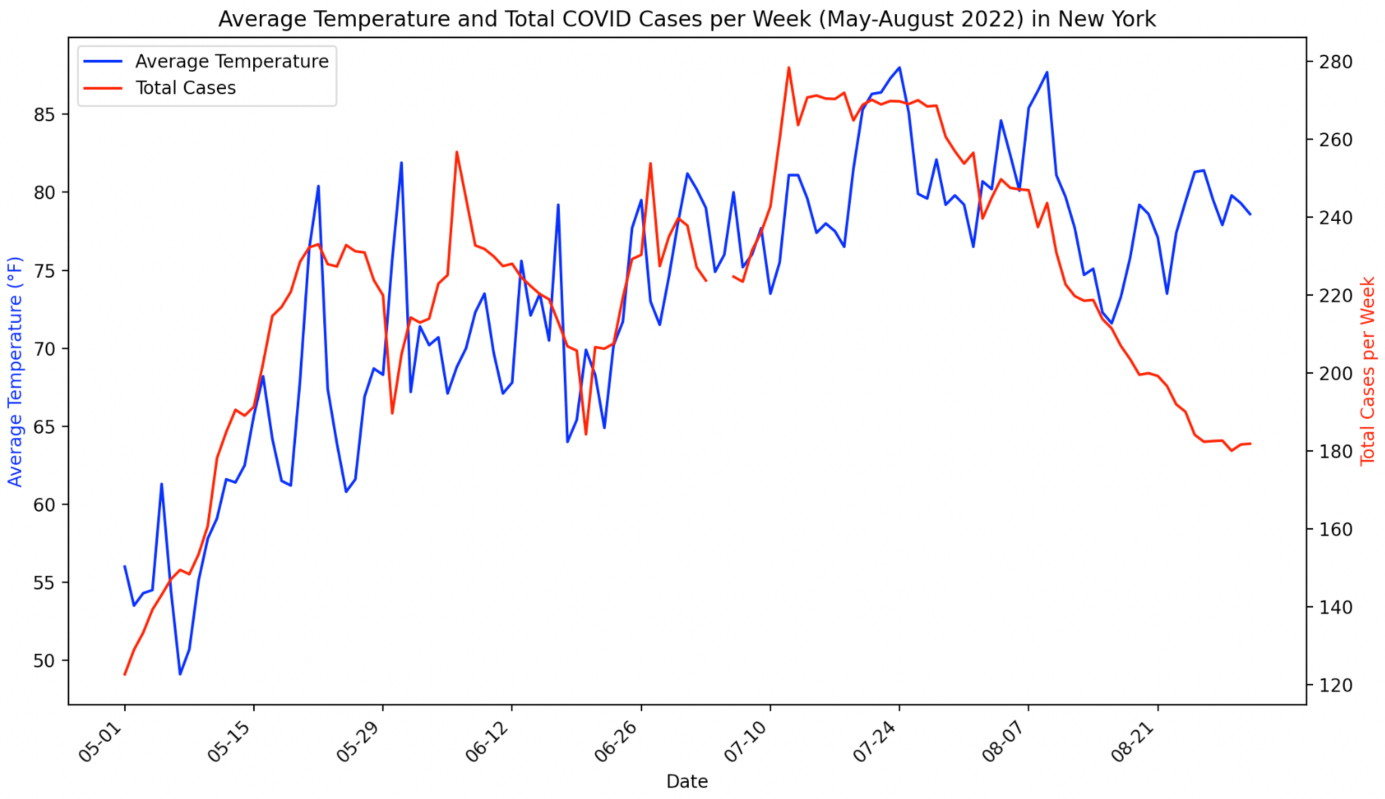
Visualizations

We created four visualizations: The first one is a bar chart showing the total number of covid cases per month, the second one is also a bar chart displaying the average temperature per month, the third one demonstrates weather condition percentages based on weekly covid admissions on a pie chart, and lastly, the fourth visualization is a subplot graph showing the trends between the average temperature and the total number of cases through the time frame.






Proportion of Weekly COVID Admissions per Weather Condition in New York
Moderate rain at times
6.7%
Moderate or heavy rain shower
14.1%
Heavy rain at times
3.3%
Cloudy
7.9%
Overcast
11.6%
Partly cloudy
15.8%
Patchy rain possible
19.7%
Sunny
21.0%


Average Temperature and Total COVID Cases per Week (May-August 2022) in New York
Average Temperature
Total Cases
Average Temperature (°F)
Total Cases per Week
Date
05-01
05-15
05-29
06-12
06-26
07-10
07-24
08-07
08-21
50
55
60
65
70
75
80
85
120
140
160
180
200
220
240
260
280

<u>Instructions for Running Code:</u>

1. Run WeatherAPI.py file first. This should fill the dates table in the database with all the dates and id starting at 1 and incrementing by 1 for each date. In total the Dates table will have 123 rows. This file will also add 25 rows of data from the weather API into the Weather_data table. At the end of each run, it prints out the number of rows in the Weather_data table. You then run the code 4 more times until it prints 123 as the number of total rows. If you run it more times it won't affect the code and will still stay at 123 rows. Once the code is at 123 rows, it makes the calculations based on the weather_data table and dates table and writes them in the "averagetemp.txt" and the "weekly_conditions.txt" files.
2. Next you run the covid.py file. This will also fill the covid_data table with 25 rows of data at a time. It will print the number of rows in the covid_data table after each run. Run it 4 more times again until it prints 123 rows. Once it's at 123 rows all the necessary data is added to the table. Once the row prints 123 rows, it makes calculations based on the dates table and the covid_data table, and writes them in the "totalcases.txt" and "weekly_admissions.txt" files.
3. Now you can run the visualizations.py file. This will create and show the four different visualizations. It first shows the total covid cases per month bar chart. Then it shows the average temperature per month bar chart. Then it shows the subplot graph that has two trend lines to show the total covid cases per week and average temperature per week in order to compare them. Lastly, it shows a pie chart with the proportion of weekly covid admissions for each weather condition.

<u>Code Documentation:</u>

- WeatherAPI.py
  - create_database()
    - Inputs: none
    - Outputs: cursor and connection
    - The purpose of this function is to create a database called "final.database" and create a dates table and a weather_data table
  - insert_dates_table()
    - Inputs: a cursor, a connection, start_date, end_date
    - Outputs: none
    - The purpose of this function is to create a dates table that assignes each day from my start_date to my end_date to an id that starts at 1 and increments by 1 for each day. The table ends up with two columns, an id

        one and a date one with a total of 123 rows. This table will be used to refer dates to an id that will be the primary key in the other tables.

- get_weather_api()
    - Inputs: api key, location, start_date, end_date
    - Outputs: a json dictionary with the weather api data
    - The purpose of this function is to request data from the weather api using the api key and the location as New York. It also sets the time frame from 05-01-2022 to 08-31-2022 and converts it into json and as a dictionary.
- weather_data_table()
    - Inputs: a cursor, a connection, data, id
    - Outputs: data
    - The purpose of this function is to insert data into the weather_data table for each column. It uses the information grabbed by the api within the json dictionary as the variable data. It takes the id from the dates table where the date corresponds, and adds that id into the weather_data table. It then fills in the corresponding columns in the weather_data table. It also only adds 25 rows of data at a time.
- calculate_average_temperatures()
    - Inputs: a cursor
    - Outputs: None
    - The purpose of this function is to calculate the average temperature per month in the given time frame. It uses the dates table and the weather_data table to find the average temperature for each month by averaging the average temperature given for each day. It then writes out the calculations in a file called "averagetemp.txt".
- calculate_most_common_condition_per_week()
    - Inputs: a cursor
    - Outputs: None
    - The purpose of this function is to find the most common weather condition for each week. It uses the dates table and the weather_data table in order to count the most common conditions for each week in the time frame. It then puts it in a dictionary called most_common_conditions and writes in the file called "weekly_conditions.txt". The first week is week 17 and it increments for each week. This is because the first week in my time frame is the 17th week of the year and so on.
- main()
    - Inputs: None
    - Outputs: None
    - The purpose of this function is to run the other functions in this file in a specified order. It also keeps track of the number of rows added to the data

        tables to ensure only 25 are added at a time and that we gather more than 100 rows of data. It also calls the calculations in this file.

- covid.py
    - connect_database()
        - Inputs: None
        - Outputs: a cursor, a connection
        - The purpose of this function is to connect to the same database used for all data tables: “final.database”. It also creates the covid_data table and returns the cursor and connection.
    - get_covid_data()
        - Inputs: api key
        - Outputs: a dictionary with the data grabbed from the covid api and refined to just the dates in our time frame
        - The purpose of this function is to make a call to the covid api using the api key and convert the data into a json dictionary. Then we loop through the dictionary to return another dictionary called covid_dict so that it only has the data for our time frame.
    - insert_covid_data()
        - Inputs: a cursor, a connection, covid_dict
        - Outputs: None
        - The purpose of this function is to retrieve the id from the dates table that corresponds to the same date, then inserts the data from the covid_dict and puts them into the corresponding column within the covid_data table. It also makes sure that the data is limited to only add 25 rows at a time.
    - calculate_total_cases()
        - Inputs: a cursor
        - Outputs: None
        - The purpose of this function is to calculate the total covid cases per month. It uses the dates table and the covid_data table to calculate the total number of cases for each month in the time frame. It then writes out the calculation in a file called “totalcases.txt”.
    - calculate_avg_weekly_covid_admissions()
        - Inputs: a cursor
        - Outputs: None
        - The purpose of this function is to calculate the average weekly covid admissions for each week in the time frame. It uses the dates table and the covid_data table to find the average of each week for covid admissions and writes in a file called “weekly_admissions.txt”.
    - connect_database()
        - Inputs:

        - Outputs:
        - The purpose of this function
    - main()
        - Inputs: None
        - Outputs: None
        - The purpose of this function is to run the other functions in this file in a specified order. It also keeps track of the number of rows added to the data tables to ensure only 25 are added at a time and that we gather more than 100 rows of data. It also calls the calculations in this file.
- visualizations.py
    - bar_chart1()
        - Inputs: a cursor
        - Outputs: a figure
        - The purpose of this function is to view a bar chart of the total number of covid cases per month in New York. The x-axis are the four months and the y- axis are the total covid cases. This chart shows the total number of cases for each month to see which month in that time frame had the most cases.
    - bar_chart2()
        - Inputs: a cursor
        - Outputs: a figure
        - The purpose of this function is to view a bar chart of the average temperature per month in New York. The y-axis is the average temperature in Fahrenheit and the x-axis is the four months in my time frame. This chart can show which month had a higher average temperature.
    - sub_plot()
        - Inputs: a cursor
        - Outputs: a figure
        - The purpose of this function is to view the trends of both the average temperature and the total number of covid cases per week within the time frame in New York. The subplot has two y-axises. One is the total number of cases and the other is the average temperature in Fahrenheit. The x-axis are the weeks in my time frame, but only shows a few so that it does not overcrowd the visual. There is a legend to signify which color line is for which trend. This graph can help you see if there are any similarities in trends between the total number of cases per week and the average temperature for the week.
    - pie_chart()
        - Inputs: a cursor
        - Outputs: a figure

        - The purpose of this function is to view a pie chart that shows how common a weather condition is based on the total of weekly admissions. This graph can help you see the percentage of total weekly admissions based on the type of weather condition that was most common that week. It can help analyze what type of weather condition was present for more weekly covid admissions.
    - main()
        - Inputs: None
        - Outputs: None
        - The purpose of this function is to create a connection to the database and establish the cursor and connection. It also runs all the other functions that create the visualizations in order.

<u>Resource Documentation</u>

Date	**Issue Description**	**Location of Resource**	**Result (did it solve the issue?)**
All of these various resources were used between this 10 day span where we completed our project: 2/10 → 2/20	- Peer tutoring - Past HW assignments - Past lectures instruction - Project 2 - We were also in contact with our amazing IA Cristina and attended office hours when necessary - everyone has been so helpful and patient. - JSONformatter.org	- All on the Canvas :) - JSONformatter.org	- Yes - everything helped!! - Specifically, we used these resources for each part of this project - these past assignments and the general module material from Canvas were just great to have as a reference point to make sure we were on the right track throughout each part of this project and for each function specifically too. - JSON formatter helped us look at the JSON files more closely