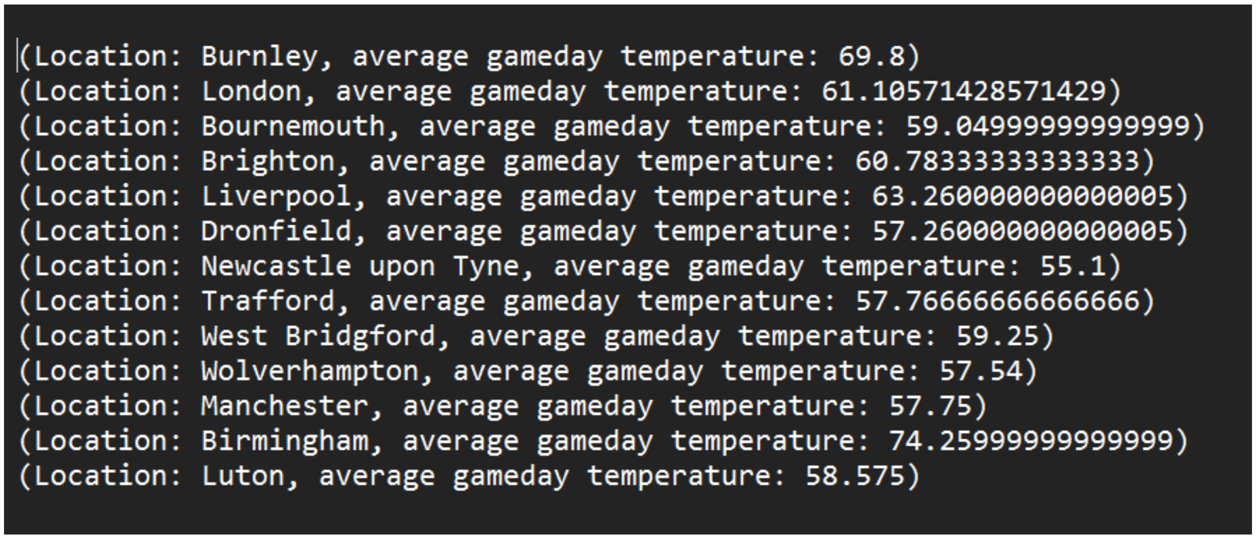
Github: https://github.com/evanaber/SI206FINALPROJECT/tree/main

Report

In addition to your API activity results, you will be creating a report for your overall project.
The report must include:
1.The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)
Our original goal for this project was to get data about football game scores, the weather for that location and place and the betting odds for the game. The original APIs we were going to use were OddsMagnet for the game odds, Apilayer Weatherstck for the weather and Football-Data for the soccer scores. We were going to get the scores and injuries from the football API, from the weather API we were going to get cloud cover, temperature and precipitation from and the odds of the game from OddsMagnet API. With the data we were going to see if the weather affects the odds and who won and how they all interacted with each other.
2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)
We worked with Google Places API, football-data API and open-meteo API. We use football-data to get the information for various football games, including the date and the city the game was played in. We then used this information along with Google Places API to get the coordinates for each city, which then would go into open-meteo to get the weather information for the date and location of each game.
3. The problems that you faced (10 points)
Multiple of the problems that we faced occurred when we first started the project and began to work with our APIs. Originally, we were going to involve betting odds as an API, but we struggled to find odds for the date range we wanted to use. We were simultaneously deciding how to get the correct location for each game from the name of the home team, as that is what our API provided us with. Dr. Ericson recommended that we use an API to find locations, so we decided to replace the betting odds API with Google Places API. Our original API for weather, Weatherstack API, only allowed us to make 100 requests a month, which would not have worked for the amount of times that we tested our project. We decided to switch to a different API with similar data but many more requests. After we got our APIs working, the majority of our issues were resolved. We did run into small problems such as collecting data from the correct competition and figuring out how to only add 25 items at a time, but we worked together to find a way that worked and would make future coding easier for us.
4. The calculations from the data in the database (i.e. a screenshot) (10 points)
calculations.py

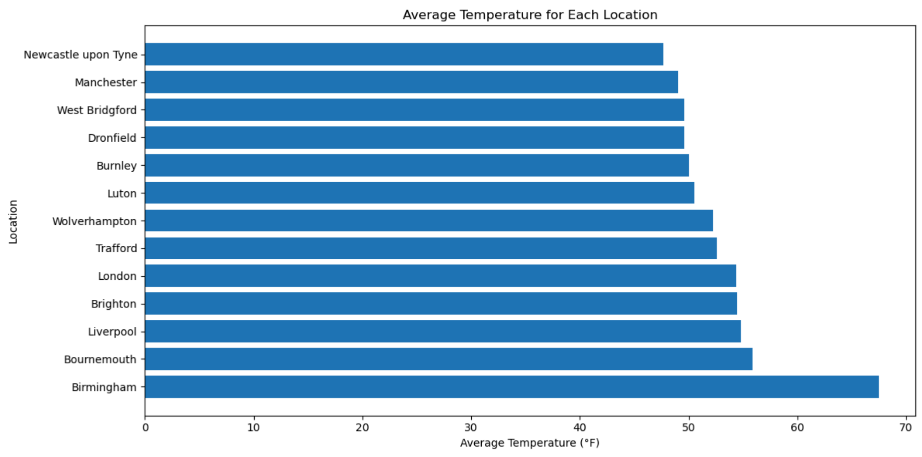
```
import sqlite3
import os
```

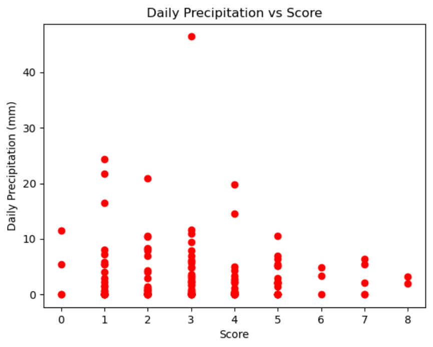
```
path = os.path.dirname(os.path.abspath(__file__))
conn = sqlite3.connect(path + "/" + 'fb_scores.db')
cur = conn.cursor()

cur.execute(
    "SELECT Scores.game_num, Loc_Keys.location, Weather.temperature FROM Scores JOIN
Loc_Keys ON Scores.location = Loc_Keys.id JOIN Weather ON Scores.game_num =
Weather.game_id"
)
games_locations = cur.fetchall()

loc_temps = {}
total_temp = 0
counter = 0
for tup in games_locations:
    if tup[1] not in loc_temps:
        loc_temps[tup[1]] = (0, 0)
    total_temp, counter = loc_temps[tup[1]]
    counter += 1
    total_temp += tup[2]
    loc_temps[tup[1]] = (total_temp, counter)

loc_avgs = []
for location, tup in loc_temps.items():
    loc_avgs.append((location, tup[0]/tup[1]))

with open('calculations.txt', 'w') as writer:
    for tup in loc_avgs:
        writer.write(f'(Location: {tup[0]}, average gameday temperature: {tup[1]}) \n')
writer.close()
```

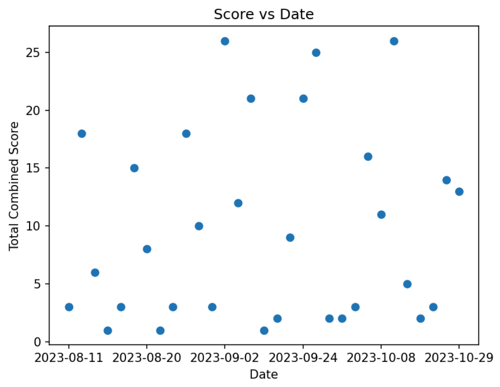


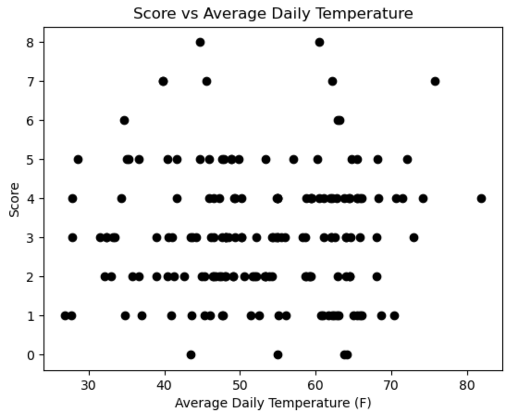
```
(Location: Burnley, average gameday temperature: 69.8)
(Location: London, average gameday temperature: 61.10571428571429)
(Location: Bournemouth, average gameday temperature: 59.04999999999999)
(Location: Brighton, average gameday temperature: 60.78333333333333)
(Location: Liverpool, average gameday temperature: 63.260000000000005)
(Location: Dronfield, average gameday temperature: 57.260000000000005)
(Location: Newcastle upon Tyne, average gameday temperature: 55.1)
(Location: Trafford, average gameday temperature: 57.766666666666666)
(Location: West Bridgford, average gameday temperature: 59.25)
(Location: Wolverhampton, average gameday temperature: 57.54)
(Location: Manchester, average gameday temperature: 57.75)
(Location: Birmingham, average gameday temperature: 74.25999999999999)
(Location: Luton, average gameday temperature: 58.575)
```

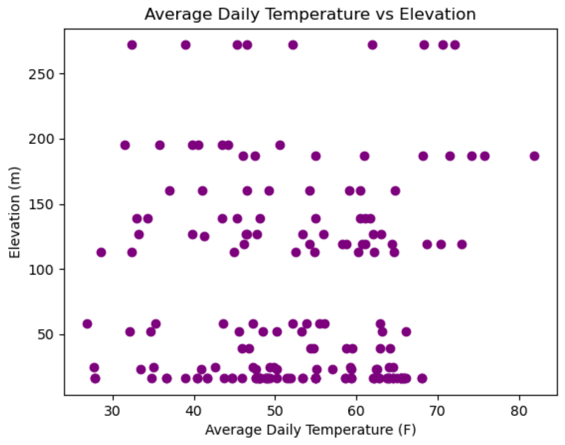
5. The visualization that you created (i.e. screenshot or image file) (10 points)











6. Instructions for running your code (10 points)
Our code includes 6 different files: three for calling APIs, two for creating visualizations, and one for calculations. When running the code, the file *finalproject.py* should be run first, followed by *places.py*, and then *mm_function.py*. This sequence should be repeated four times in order to collect and add 100 items from each API to the database. After the database has been filled, *calculations.py* should be run to calculate the average temperature of each location using the data in the databases. Finally, *visualisations.py* and *avgtempgraph.py* should be run to display the visualizations.
7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)
In visualisations.py, we did not have any functions. The file takes data from the database and creates multiple visualizations using it. In calculations.py, we also did not have any functions. The file goes through the database, collects data, and calculates the average temperature for each location. In finalproject.py, set_up_fb_db() is the first function. This takes in the input of the name of the database and opens a connection to it, and does not return anything. set_api() takes in the link to the API, collects the responses, and returns them. find_matches() takes in the responses as an input. It then goes through the responses and adds them to the database accordingly. It does not return anything as output, but rather commits the changes at the end of the function. main() sets the link, calls set_api(), and calls find_matches().

For the Weather API there is a main function that calls the weather_api function with no inputs the   weather_api function then makes a weather table and a date_id table if they dont already dont exist. It then gets all the data it needs from the database to call the api and calls the api up to 25 times and adds the data it collects into the database after each call then it commits it and is finished.

Places.py documentation:
setup_database():
Creates a new table named Game_Locations if it does not already exist. The table is made to store the game number, date id, and the latitude and longitude. There are no inputs and the function does not return anything, but it prints a message to show the table is ready.

get_coordinates(city_name):
This function uses the Google Places API to get the latitude and longitude coordinates using the city_name string as input. This is a string representing the name of the city whose coordinates are being fetched. It makes a request to the Google Places API with the name and parses the JSON response. If the request is successful, the function returns a tuple with the latitude and longitude as floats. If the request fails, the function returns (None, None) and prints an error message.

process_game_locations():
This function processes the game location data and fills the Game_Locations table with the latitude and longitude of each game location. It fetches up to 25 rows of data at a time by joining the Scores and Loc_Keys tables and filters for games that are not already in Game_Locations. For each row, it uses get_coordinates() to get the coordinates of the city associated with the game. The function does not return anything or take any inputs. Instead, if the coordinates are retrieved, it inserts the data into the Game_Locations table. It also prints progress messages for each city as the table gets populated.

main():
Main calls setup_database() followed by process_game_locations().

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)
ChatGPT for debugging help, matplotlib.org for creating visualizations