Mia Weinberg and Michael Feldman

GitHub Repository: https://github.com/miadw1/206FinalProject

Final Project Report - Football Forecast

1. Goals for the Project

At the beginning of our project, we had aimed to explore the correlation between weather conditions and music streaming habits using data from the Spotify API and the Weather API. We originally planned to gather data on music streaming metrics (genres, moods, tracks, and timestamps) from the Spotify API, and see if there was a correlation between having bad weather and listening to sad music, collecting weather patterns (temperature, humidity, precipitation) from the Weather API. To explore this correlation, our goal was to create four different visualizations, meanwhile uploading increments of at most 25 rows for each dataset. Unfortunately, the Spotify API was too complex, and didn't have the data we were looking for. Additionally, there was an authentication process, including creating developer accounts and scheduling consultations with a Spotify representative. These roadblocks delayed progress significantly, making it infeasible to integrate Spotify data within the given timeframe.

---

2. Goals Achieved

Since our original plan did not work out as we hoped, we had to alter our goals throughout the project. We eventually turned to our mutual love of college football and reset our goals to see if better weather conditions lead to better football (higher scoring games). To do this, we successfully examined data from both the OpenSource WeatherAPI, as we originally had planned, and the CollegeFootballAPI to see if there was a relationship between weather conditions and football game scores. Using the Weather API, we gathered data on temperature, humidity, precipitation, and general weather conditions, such as if it is sunny outside. For the College Football Data API, we gathered data on the football games during the 2024 season that were being played weekly, including teams, scores, venues, and cities. With these datasets, we were able to access at least 100 rows of data with increments of 24 between both of the tables created, "weather" and "games". In addition, we successfully created a calculation function and four visualizations. To calculate our data, we used both the weather and games table to find the average home score, away score, and temperature for a specific city. For our four visualizations, using Matplotlib, we created a bar chart, a stacked bar chart, a scatterplot, and a line plot all demonstrating different aspects of our data to help us come to a conclusion if there is a

correlation between weather and the game score. Ultimately, we were able to successfully meet the criteria for this project and were able to achieve all of the goals that were set for ourselves.

---

3. Problems Faced

Throughout the project, we encountered several challenges. One of the first issues started early on with accessing the SpotifyAPI. Although we had written much of our code, we eventually realized that in order to actually authenticate the API key we had received, we had to go through an approval process. Therefore, instead of waiting for our request to be accepted, we instead decided to start over and find another API key to use, the CollegeFootballAPI. Another challenge that we had was writing code to limit the data to only store a maximum of 25 rows at a time. After a lot of trial and error, we successfully met this project requirement by tracking the number of rows already stored in each table and dynamically appending new rows incrementally to avoid duplication. A third problem we faced was the lack of information provided in the CollegeFootballAPI. While the API listed stadiums, it did not include the cities where these stadiums were located. We researched additional APIs to concert stadiums to cities but we could not find one that satisfied all of the data. Therefore, to address this, we created a custom variable called “stadium_to_city_mapping”, manually hard-coding the city names for each stadium to join the “games” and “weather” tables effectively. Lastly, the fourth problem we faced was a timing issue with running our code. Since the weather data was collected in real time, attempting to run the program at night resulted in an error message saying no data is available when trying to filter for only sunny cities. As a result, in order to see the data and visualization for sunny cities, we could only run the code during the day.

---

4. Calculations from the Database

For our code, we performed the calculations of finding the average home score, away score, and temperature for a specific city.

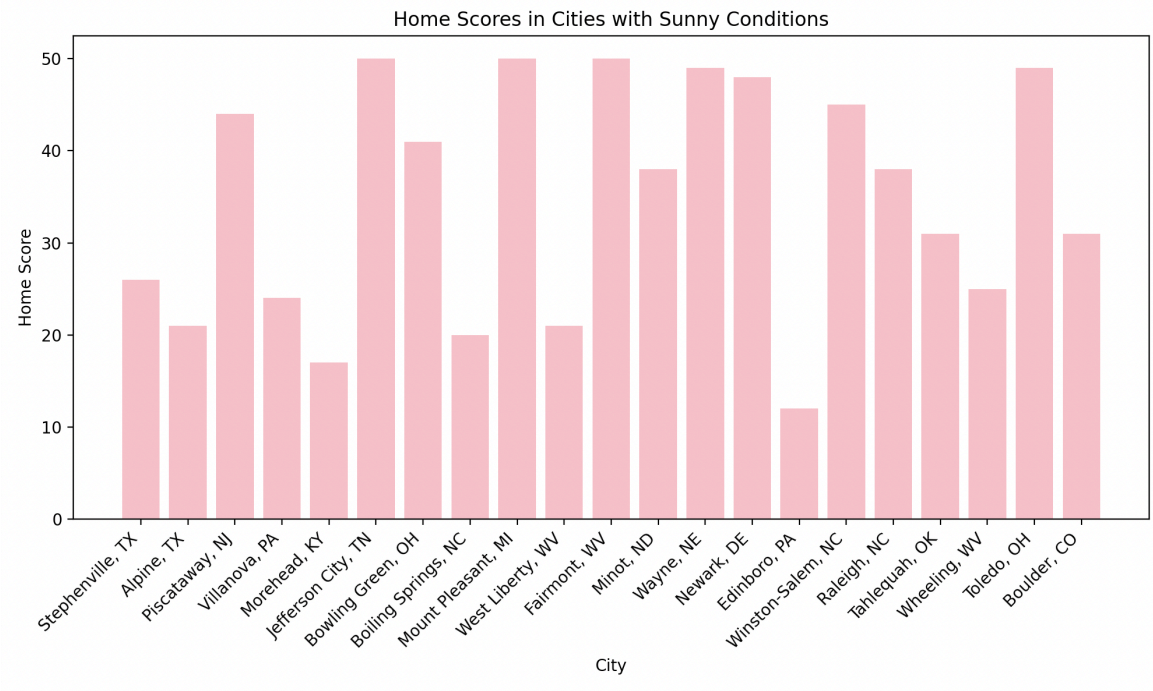
```python
def calculate_city_stats(conn):
    c = conn.cursor()
    c.execute('''
        SELECT c.city_name, COUNT(g.id) AS total_games,
               AVG(g.home_score) AS avg_home_score,
               AVG(g.away_score) AS avg_away_score,
               AVG(w.temperature) AS avg_temperature
        FROM games g
        JOIN cities c ON g.city_id = c.id
        JOIN weather w ON g.city_id = w.city_id
        GROUP BY c.city_name
    ''')
    return c.fetchall()
```

city_stats.txt

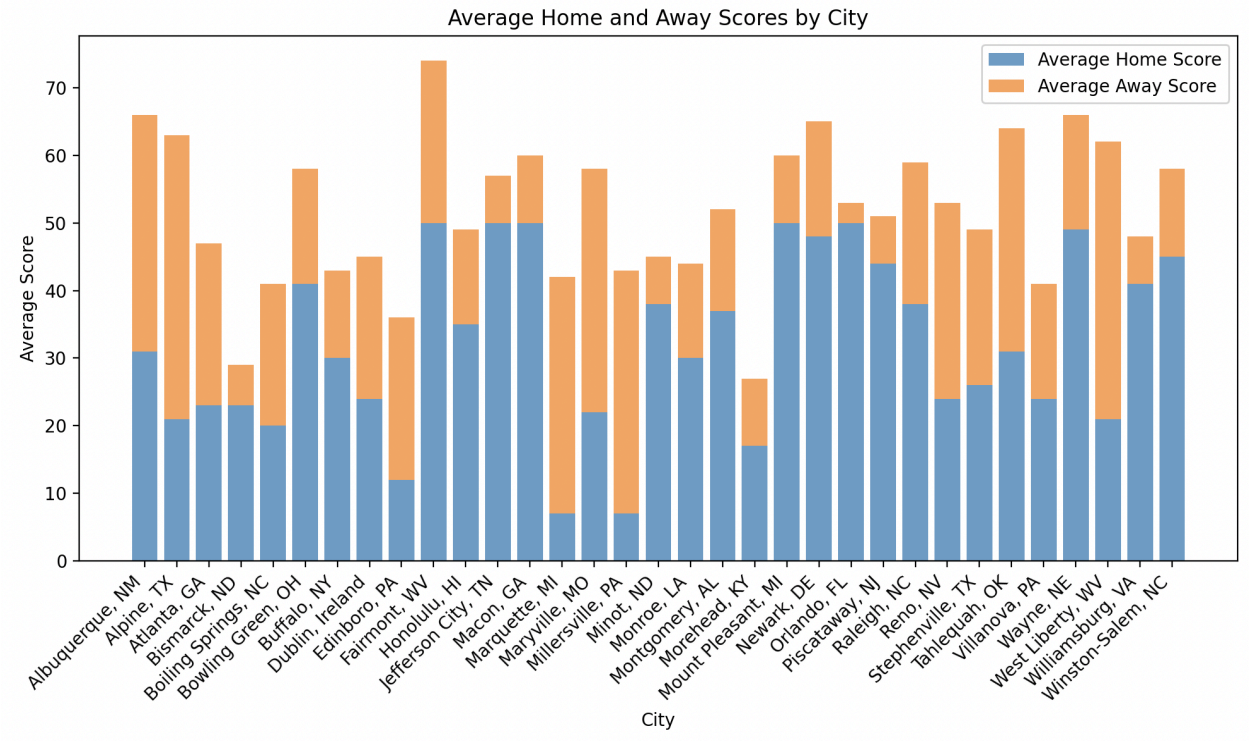
City	Total Games	Average Home Score	Average Away Score	Avg Temp (C)
Albuquerque, NM	1	31.00	35.00	4.40
Alpine, TX	1	21.00	42.00	11.10
Atlanta, GA	1	23.00	24.00	11.10
Bemidji, MN	1	19.00	13.00	0.00
Birmingham, AL	1	41.00	3.00	14.40
Bismarck, ND	1	23.00	6.00	0.00
Boiling Springs, NC	1	20.00	21.00	4.00
Boulder, CO	1	31.00	26.00	4.20
Bowling Green, OH	1	41.00	17.00	2.80
Buffalo, NY	1	30.00	13.00	0.00
Chadron, NE	1	6.00	18.00	0.00
Champaign, IL	1	45.00	0.00	4.40
Columbia, MO	1	50.00	0.00	7.80
Des Moines, IA	1	0.00	0.00	0.60
Dublin, Ireland	1	24.00	21.00	11.30
Edinboro, PA	1	12.00	24.00	0.00
Emporia, KS	1	30.00	14.00	4.40
Fairmont, WV	1	50.00	24.00	2.20
Hays, KS	1	21.00	7.00	1.30
Honolulu, HI	1	35.00	14.00	27.40
Jacksonville, AL	1	27.00	50.00	13.30
Jefferson City, TN	1	50.00	7.00	10.10
Kansas City, KS	1	48.00	3.00	4.40
Kingsville, TX	1	13.00	16.00	23.30
Las Vegas, NM	1	37.00	50.00	3.30
Little Rock, AR	1	50.00	0.00	13.30
Macon, GA	1	50.00	10.00	12.80
Marquette, MI	1	7.00	35.00	0.00
Maryville, MO	1	22.00	36.00	1.30
Millersville, PA	1	7.00	36.00	0.00
Minneapolis, MN	1	17.00	19.00	0.20
Minot, ND	1	38.00	7.00	0.00
Monroe, LA	1	30.00	14.00	17.80
Montgomery, AL	1	37.00	15.00	17.10
Morehead, KY	2	17.00	10.00	8.00
Mount Pleasant, MI	1	50.00	10.00	0.00
Nacogdoches, TX	1	50.00	0.00	18.30
New Orleans, LA	1	50.00	0.00	16.70
Newark, DE	1	48.00	17.00	0.00
Orlando, FL	1	50.00	3.00	18.30
Piscataway, NJ	1	44.00	7.00	0.00
Raleigh, NC	1	38.00	21.00	6.10
Rapid City, SD	1	6.00	35.00	0.00
Reno, NV	1	24.00	29.00	0.40
St. Paul, MN	1	13.00	34.00	0.20
Stephenville, TX	1	26.00	23.00	14.10
Tahlequah, OK	1	31.00	33.00	10.30
Toledo, OH	1	49.00	10.00	1.50
Tulsa, OK	1	50.00	28.00	9.40
Vermillion, SD	1	45.00	3.00	0.00
Villanova, PA	2	24.00	17.00	0.00
Wayne, NE	1	49.00	17.00	0.00
West Liberty, WV	1	21.00	41.00	2.20
Wheeling, WV	1	25.00	28.00	2.20
Wichita Falls, TX	1	18.00	11.00	13.70
Williamsburg, VA	1	41.00	7.00	2.60
Winston-Salem, NC	1	45.00	13.00	2.80

---

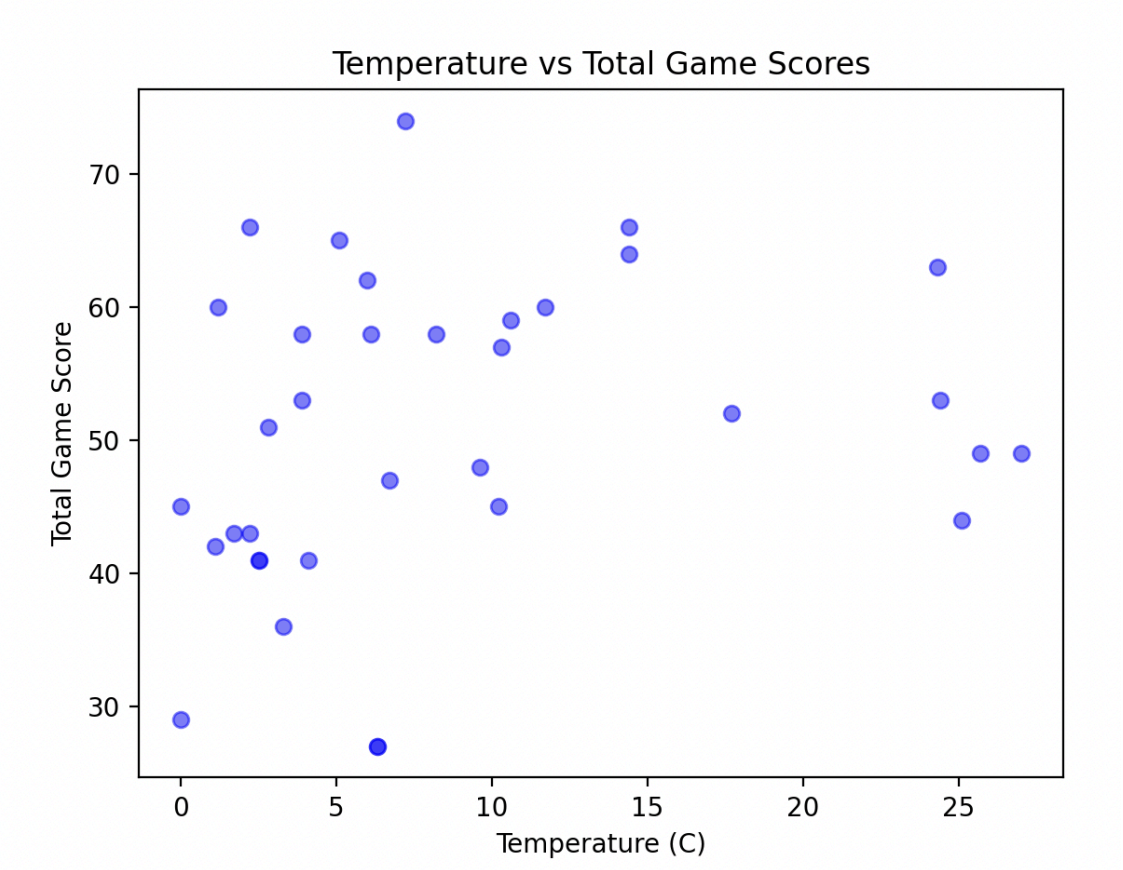
5. Visualizations



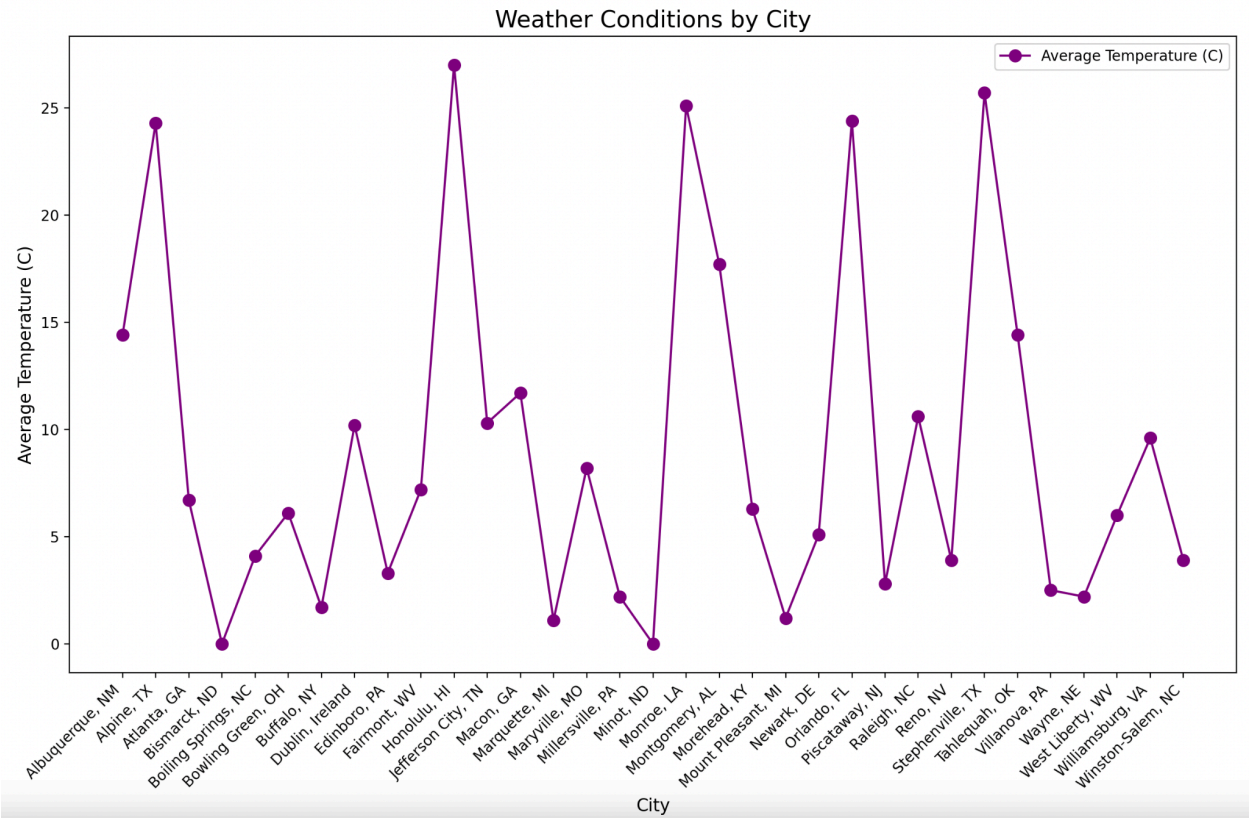
A bar chart using data from both tables, comparing the home scores in cities with currently sunny conditions. We changed this from the example given in lecture through changing the color, specifying the size, and creating a tight layout.



A stacked bar chart using data from the games table. This visualization compares the average home and away scores by city, using our calculation function.



This visualization is a scatter plot using data from both tables through the Join function. It is comparing the temperature versus game score for away and home games. To create this, we set the color and alpha size for the points.



A line plot using data from the table weather. It shows the average temperature for each city. To create this line plot, we set the color, line style, marker style, and the font size.

---

6. Instructions for Running the Code

For our project, we created four different files: “createtables.py” to gather the data and store it in databases, “calculations.py” to select the data and calculate, “visualizations.py” to make the visualizations, and “main.py” to run all of the functions. The three files createtables, calculations, and visualizations are all called within the main file. Therefore, to run the code, all that is needed to do is run the main file.

---

7. Documentation for Each Function

In the first file, “createtables.py”, seven functions were called:

1. def fetch_weather_data(api_key, city): fetches the current weather data for a specified city using the WeatherAPI
    a. Inputs

        i. Api_key (str): the API key required to access the WeatherAPI
        ii. City (str): the name of the city for which weather data is requested
    b. Outputs
        i. A dictionary containing the city (str), temperature (float), humidity (int), precipitation (float), and condition (str)
        ii. It returns None if there is an error in the API call
2. def fetch_cfb_data(api_key, season, max_week=15): fetches college football game data for a specific season from the College Football Data API
    a. Inputs
        i. Api_key (str): the API key required to access the CollegeFootball API
        ii. Season (int): the year of the football season
        iii. Max_week (int): the maximum number of weeks to fetch data for (default is 15)
    b. Outputs
        i. A list of dictionaries, each containing home_team (str), away_team (str), home_score (int), away_score (int), venue (str), and city (str)
3. def update_city_information(db_path, mapping): updates the games table in the database to associate venues with their respective cities
    a. Inputs
        i. Db_path (str): the file path to the SQLite database
        ii. Mapping (dict): a dictionary mapping venue names to city names
    b. Outputs
        i. None: updates the databases in place
4. def create_database(): creates the database for storing sports and weather data
    a. Inputs
        i. None
    b. Outputs
        i. A SQLite connection object
5. def insert_weather_data(conn, weather_data): inserts weather data into the weather table in the database
    a. Inputs
        i. Conn: the SQLite database connection
        ii. Weather_data: a dictionary containing weather data with keys city, temperature, humidity, precipitation, and condition
    b. Outputs
        i. None: inserts data into the database
6. def insert_game_data(conn, game_data): inserts game data in to the games table in the database
    a. Inputs
        i. Conn: the SQLite database connection

        ii. Game_data: a dictionary containing game data with the keys home_team, away_team, home_score, away_score, venue, and city
    b. Outputs
        i. None: inserts data into the database
7. def get_or_create_id(conn, table_name, column_name, value): retrieves the ID of a value from a specified table and column, or inserts the value if it does not exist
    a. Inputs
        i. Conn: the SQLite database connection
        ii. Table_name (str): the name of the table
        iii. Column_name (Str): the column name to check
        iv. Value (Str): the value to look up or insert
    b. Outputs
        i. Id (int): the ID of the value in the database

In the second file, “calculations.py”, four functions were called:

1. def get_combined_data(conn): fetches combined data from the database by joining multiple tables
    a. Inputs
        i. Conn: the SQLite database connection object
    b. Outputs
        i. List: a list of tuples where each tuple contains city_name, home_team, away_team, home_score, away_score, venue_name, temperature, and condition_name
2. def get_sunny_games(conn): fetches game data filtered for games played in cities with sunny weather conditions
    a. Inputs
        i. Conn: the SQLite database connection object
    b. Outputs
        i. List: a list of tuples where the condition is sunny and each tuple contains city_name and home_score
3. def calculate_city_stats(conn): calculates statistics for each city, including the total number of games, average scores for home and away teams, and average temperature
    a. Inputs
        i. Conn: the SQLite database connection object
    b. Outputs
        i. List: a list of tuples where each tuple contains city_name, total_games, avg_home_score, avg_away_score, avg_temperature
4. def write_data_to_file(data, filename): writes calculated city statistics to a text file
    a. Inputs

        i. Data (list): a list of tuples where each tuple contains city_name, total_games, avg_home_score, avg_away_score, avg_temperature
        ii. Filename (str): the name of the file where the data should be written
    b. Outputs
        i. None: writes data to a file

In the third file, “visualizations.py”, four functions were called:

1. def visualize_sunny_scores(data): visualizes home scores in cities where games were played under sunny weather conditions using a bar chart
    a. Inputs
        i. Data (list): a list of tuples where the condition is sunny and each tuple contains city_name and home_score
    b. Outputs
        i. If the input data is empty, it prints a message indicating no data is available
        ii. Plots a bar chart
2. def visualize_average_scores(data):
    a. Inputs
        i. Data (list): a list of tuples where each tuple contains city_name, total_games, avg_home_score, avg_away_score, avg_temperature
    b. Outputs
        i. Plots a stacked bar chart
3. def visualize_temp_vs_scores(conn): visualizes the relationship between temperature and total game scores using a scatter plot
    a. Inputs
        i. Conn: the SQLite database connection object
    b. Outputs
        i. Plot a scatter plot
4. def visualize_weather_conditions(data): plots average temperatures for each city using a line chart to visualize weather conditions
    a. Inputs
        i. Data (list): a list of tuples where each tuple contains city_name, total_games, avg_home_score, avg_away_score, avg_temperature
    b. Outputs
        i. Plots a line chart

In the fourth file, “main.py”, one function is called through importing the other three files:

1. def main(): This function initializes the mapping of stadiums to their corresponding cities as well as calls all of the other functions created to run the code.

a. Inputs
    i. None
b. Outputs
    i. None

---

8. Resources Documented

Date	Issue Description	Location of Resource	Result
2024-11-24	Accessing OpenWeatherAPI key	https://openweathermap.org/appid	Was able to use the documentation to successfully fetch data from and access the weather API key.
2024-12-3	Accessing CollegeFootballAPI Key	https://api.collegefootballdata.com/api/docs/?url=/api-docs.json	Was able to use the documentation to successfully fetch data from and access the games API key.
All throughout	SQL Queries	Class Slides	Referred back to class slides and notes when we were confused about the next steps for our queries. This information helped us complete our code.
All throughout	Debugging	Office Hours (zoom)	Went to office hours to ask questions for help debugging after trying on our own for a while.

All throughout	Debugging	ChatGPT	Used ChatGPT for help debugging throughout the project after trying on our own first.