Project Aims

Our project aimed to collect data from different MLB games throughout the 2023 season and to see how the weather at different stadium locations impacted the number of runs each team scored. The three APIs/websites we planned to use were SportRadar (MLB data API), Geojango (website for MLB stadium locations), and Weatherstack (API for weather). From the SportRadar API, we planned to gather specific statistics about each individual MLB game, such as the number of runs (points) each team had (home/away) in each game. From the Geojango website, we planned to gather the location for each MLB stadium so that we could see how games differed in different parts of the country. Lastly, we planned on using the Weatherstack API in order to obtain the average weather at each location and see how weather conditions affect the different MLB games.

Achieved Goals

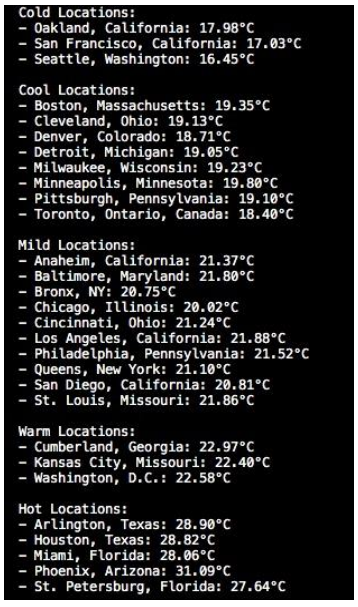
Regarding the goals that were achieved, we successfully utilized both the SportRadar API and the Geojango website. Using both of these sites, we successfully gathered information and statistics about how many runs each team averaged and how teams' performance differed per stadium location. Unfortunately, throughout the project, we began to realize that the Weatherstack API wasn't the proper one for us to use. As a result, we performed research and found a weather API that worked perfectly for us. Our new API came from Oikolab, a great replacement for Weatherstack that allowed us to obtain essentially the same information: the average weather at each stadium location.

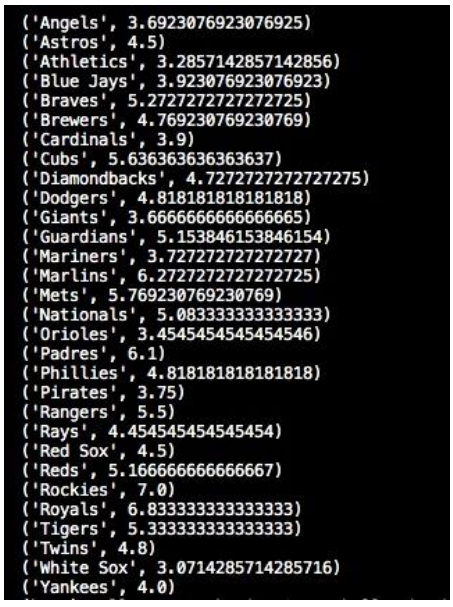
Problems Faced

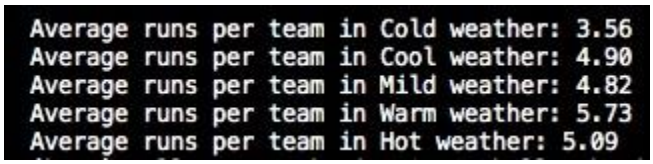
Throughout our project and our code, we faced a multitude of problems. The first problem we ran into was using our SportRadar API. At the start of our use of the SportRadar

API, we noticed that the API trial explicitly states that we are only entitled to "1,000 calls per rolling 30-day period." Because of this trial limit, we were running too many calls practically every time our code was run. As a result, our code wasn't really working. In order to work around this issue, we had to create a couple of new API keys to see what would work. After quickly realizing that we didn't have unlimited access to emails to create APIs, we needed to figure out how to implement code to limit the number of calls we were running. In order to achieve this, we incorporated "time.sleep()" into the code so that the execution of the code would be delayed or would sleep for a period of time. This helped us eliminate the issues we were running into with constantly hitting API call rate limits. Another problem that we encountered was with our databases. When we would run the Python file focused on obtaining the MLB statistics and the other Python file focused on obtaining stadium locations, for some reason, they would result in two different databases. To overcome this issue, we had to attach the databases so that we could have one database with all of the information. While we faced these issues, they made it difficult for us to complete our project along the way, but we found ways to overcome them quickly and efficiently.

Calculations

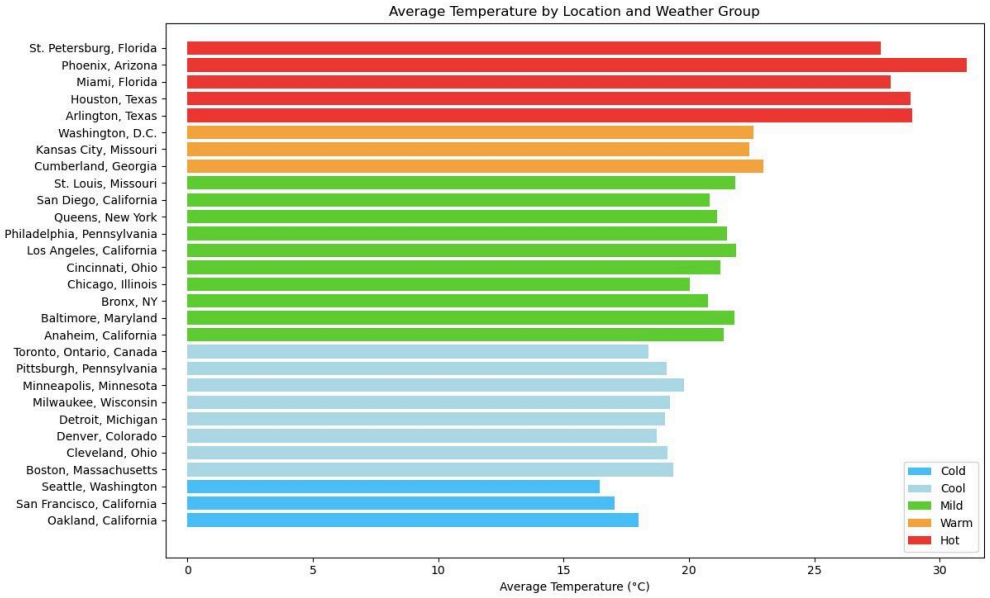




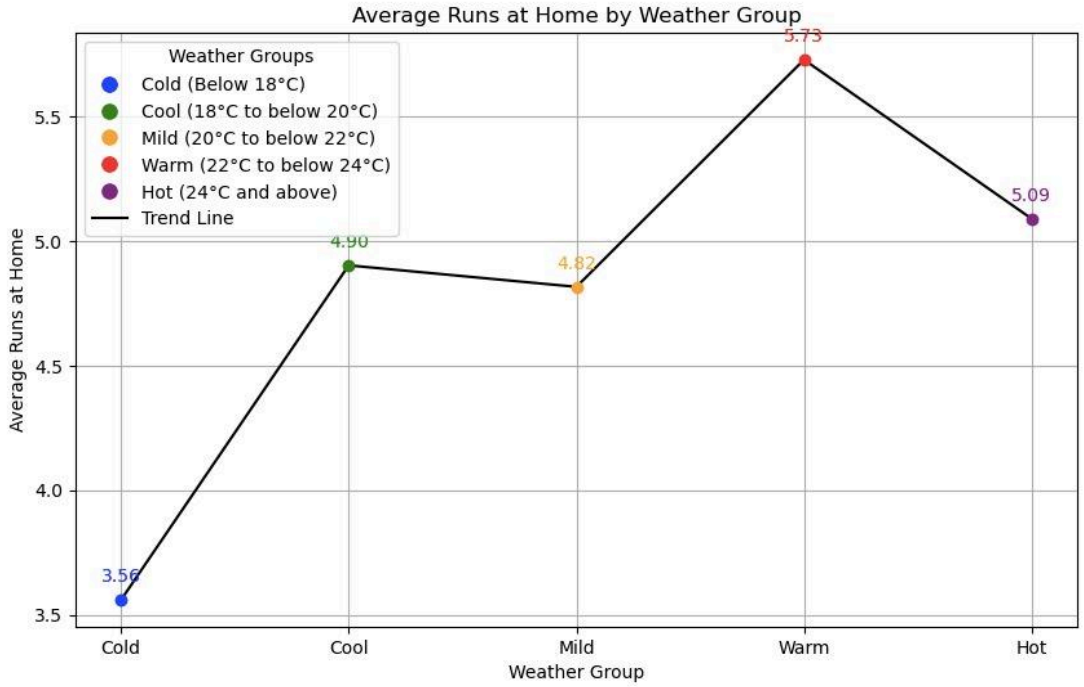


The above screenshots are the calculation output from the code displaying the average number of runs per team, the average number of runs per team based on weather, and the average temperature (in Celsius) for each MLB team's home stadium location.

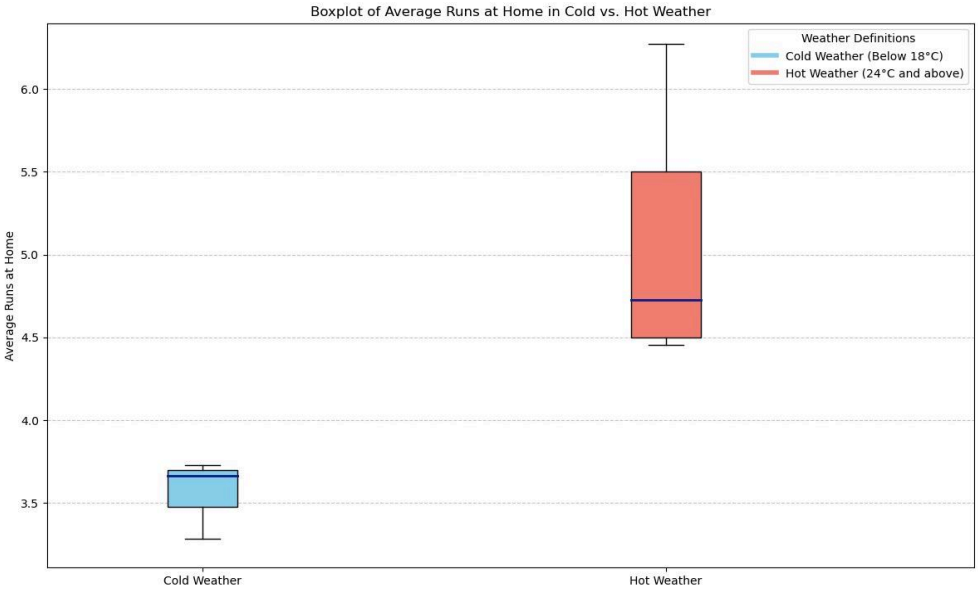
Visualizations



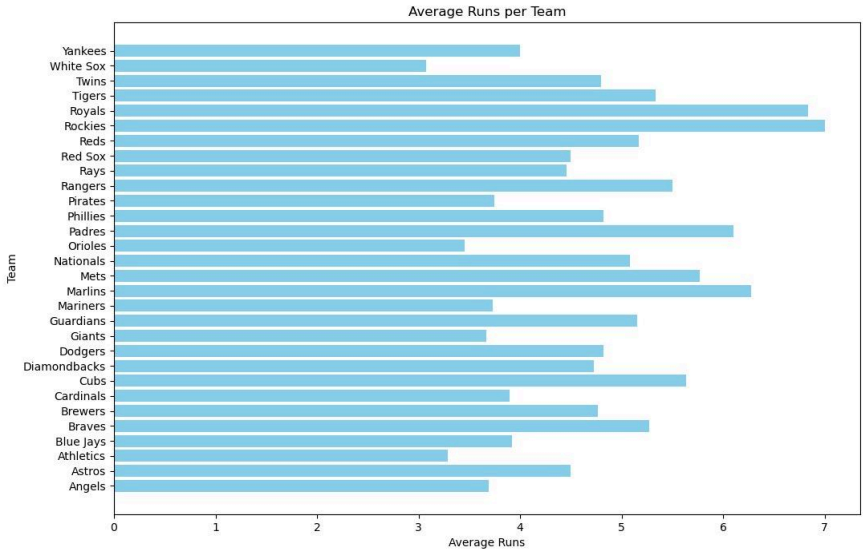
This visualization shows the average temperature per team location and groups them into weather groups (cold, cool, mild, warm, hot). A greater number of MLB teams tend to be located in the cool, mildish, and warm categories. Based on correlations we see in our other visualizations, it makes sense that more teams tend to be in that middle range of runs, whereas there are a few outliers who average way more runs or way fewer runs.



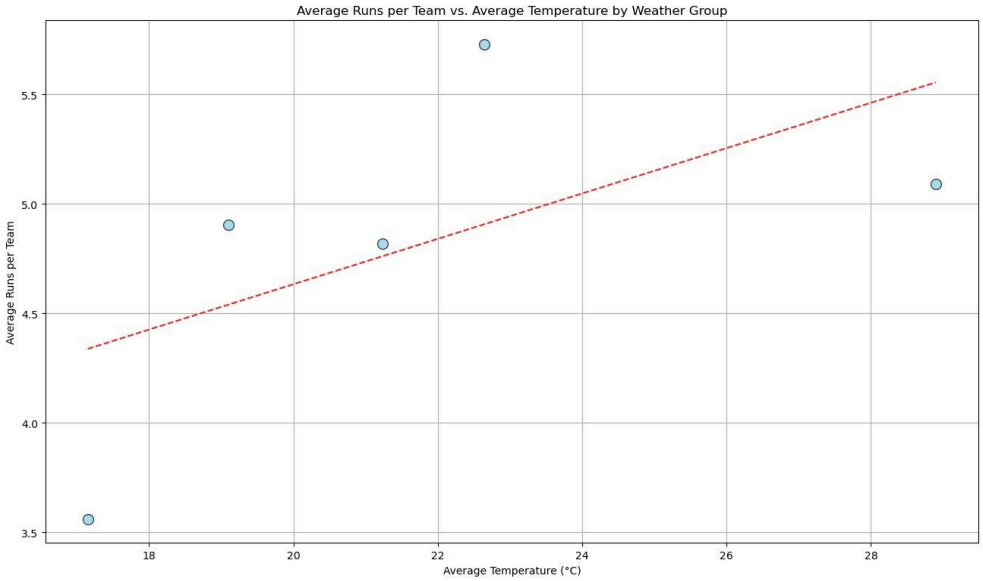
This visualization shows the average runs per weather grouping. In the visualization, the correlation between runs and weather is that warmer temperatures correlate with higher amounts of runs. However, in baseball, there do tend to be confounding variables such as player skill, team skill, team payroll, etc.



This visualization shows a boxplot of runs in cold and hot weather. The boxplots show the median, the different quartiles, and the minimums and maximums.



This visualization shows a bar chart of average team runs per MLB team. The chart isn't meant to show a correlation between anything but is meant to portray the differences between teams based on the confounding variables that are apparent in baseball (player skill, team skill, team payroll)



This visualization shows a scatterplot of average runs per team based on the weather group. It shows that the number of runs drastically changes based on temperature, but it doesn't tend to be linear.

Instructions For Running Code

1. MLB Data: Start by running the mlb.py file. Continue until you see a confirmation message that the database setup is complete.
2. Weather Data: Next, run the weatherdata.py file. Wait for the message that indicates that the database setup for weather data is complete.
3. Location Data: Run the locations.py file and wait until a message confirms that the location data has been successfully inputted into the database.
4. Calculations File: Once all the data from the other files is set up, run the calculations.py file. After executing the calculations file, review the output to properly understand the significance of the calculations.
5. Generating Our Visualizations: Finally, execute the visualization.py file. View the charts and graphs generated to see the correlations between temperatures, runs, and locations within the dataset.

Function Documentation

The MLB Python file was created to fetch MLB game data from the SportRadar API and store it in an SQLite database. Breaking down the code and its functions:

We used the imports: requests (to fetch data from the MLB API), sqlite3 (to store data in the database), time (to create the sleep function to delay the execution of the code), and datetime/timedelta (for using dates in our code).

Moving on to functions: the “setup_database()” function initializes the connection with our “mlb_data.db” database, which incorporates the three tables (Teams, Games, and LastFetched). Teams stores each MLB team’s data; Games stores the runs for each team each game; LastFetched tracks the last time the data was fetched. The “get_last_fetched_date(conn)” function gets the last fetched date from the LastFetched table in the database and converts it into a datetime object. The “update_last_fetched_date(conn, date)” function updates the LastFetched table when the data fetch was last completed. The “fetch_and_store_data(conn, max_items=25)” function is the most complex function within the MLB Python file. It loops through each day, making API requests and obtaining data from the last fetched date until the specified end date. The function inserts new teams into the Teams table and game data into the Games table. We included a delay between API calls within the function using “time.sleep()” to avoid going over the call limit.

The Weather Python file was created in order to scrape MLB team locations from a webpage, fetch weather data, and store both locations and weather in an SQLite database.

We used imports: sqlite3 (to store data in the database), requests (to fetch data), BeautifulSoup (to parse through the geojango location website), datetime/delta (for using dates in our code), and JSON (to process JSON data).

Moving on to functions: the “setup_database(db_name)” function sets up a database that stores tables based on teams (team names), games (game details), locations (team stadium locations), weather (weather data), and metadata (tracks last index). The “scrape_mlb_data(url)” function extracts MLB team names and stadium locations. The “insert_into_database(conn, mlb_data)” function inserts data into the Locations table of the database. The “get_average_temperature(location, start_time, end_time)” function fetches the average

temperature for the locations using our weather API. The “insert_weather_into_database(conn, weather_data)” function inserts location, date, and temperature into the Weather table in the database. The “fetch_and_store_weather(conn, locations, weeks, max_items=25)” function calculates weather data for a series of weeks, limiting the amount of data being processed. It fetches the average temperature for each location and week and stores it in the database.

The Locations Python file scrapes MLB team names and stadium locations from a webpage and stores the data in our database. We used imports: requests (to fetch the data), BeautifulSoup (to parse through websites), and sqlite3 (to interact with the database).

Moving on to functions: the “setup_database” function connects to our SQLite database and creates locations and metadata tables. We also dropped three tables (MLBTeams, MLBLocations, and a table we accidentally misspelled). The “scrape_mlb_data” function fetches the data needed for stadium locations. The code iterates over rows in a table and gets team names and stadium locations. The “insert_into_database” function connects to our database.

The Calculations Python file uses functions to analyze data about MLB teams and weather conditions and observe a relationship with team runs. We import SQLite3 in order to interact with our database.

Moving on to functions: the “calculate_average_temperature_and_classify(db_name)” function connects to our database and calculates the average temperature for each location. It categorizes the average temperatures into groups (Cold, Cool, Mild, Warm, Hot) based on temperature ranges. The “get_average_runs_per_team(database_name)” function connects to our database and calculates the team's average home score.

Resource Documentation

Date	**Issue Description**	**Location of Resource**	**Result (did it solve the issue?)**

4/10/23	We were having issues setting up the documentation for our SportRadar API and Weatherstack API	GSI/IA assistance in class during the open help session in the lecture	Yes, it solved the issue, and we were able to start properly using the API keys and implementing data into our code
4/23/23	We were having issues with our API keys and were running out of calls to use as a result	W3 Schools tutorials	Kind of: W3 schools sort of solved our issue. We used W3 schools to teach us how to implement a function that would delay the execution of our code: the sleep() function
4/23	Struggling to write a part of our code	ChatGPT	Yes, GPT helped us learn how to write a part of our code that we had trouble understanding.
4/23	Still struggling with the sleep() function	Youtube	Yes, in addition to W3 school tutorials, the Youtube video we found on the sleep() function helped further our understanding of how to use it so we could implement it into

			the code.
4/23	Difficulty understanding how to as well as the purpose of attaching databases	ChatGPT	Yes, ChatGPT helped us gain an understanding of what attaching databases means and how to do so properly.
4/25 - 4/29	Struggling with certain code throughout the project	Previous Homework Assignments	We used previous homework assignments to help us figure out how to create certain parts of our code that we struggled to write.
Throughout Project	Needing Assistance	ChatGPT	Helped with small code issues and debugging.