SI 206 - Final Project Report

Nicholas Karns, Faraz Ali, Evan Marks

Github Repository Link - https://github.com/nskarns/si206finalproject.git

Overall Project Goals

Our team individually found a dataset that we found interesting and further dug into each of those areas. Those areas being police jobs held in UK neighborhoods, average goals per team in the soccer premier league, and the correlation of weight and height for players in California basketball teams. This report will dive further into each of these three areas and analyze the data.

Police Jobs Held In UK Neighborhoods

The goal of this portion of the project is to find out more about how many jobs are held within UK neighborhoods and more specifically the total number of people in a certain position. To figure this out, I used an API that showed this data for police workers in UK neighborhoods. While working on this project, the biggest problem I ran into was limiting the API calls to 25 look-ups per function run and making sure my data output was correct based on what was in the database. I solved the first API problem by asking ChatGPT for an answer and solved the correct amount in data output by looking through the values in the database.

police_job_data.txt

```
The data below shows each police jobs in UK neighborhoods and how many people are within those positions.
It first shows the total number of jobs in the database, then how many people held specific positions.
The 'Other' category is the total number of jobs minus all the jobs shown below.

Police Job Data (job title - total):
Total - 392
Sergeant - 31
PCSO - 143
Police Constable - 89
PC - 70
Inspector - 22
SGT - 14
Other - 23
```

Figure #1 - Shows Data And Explain What This Data Is Showing

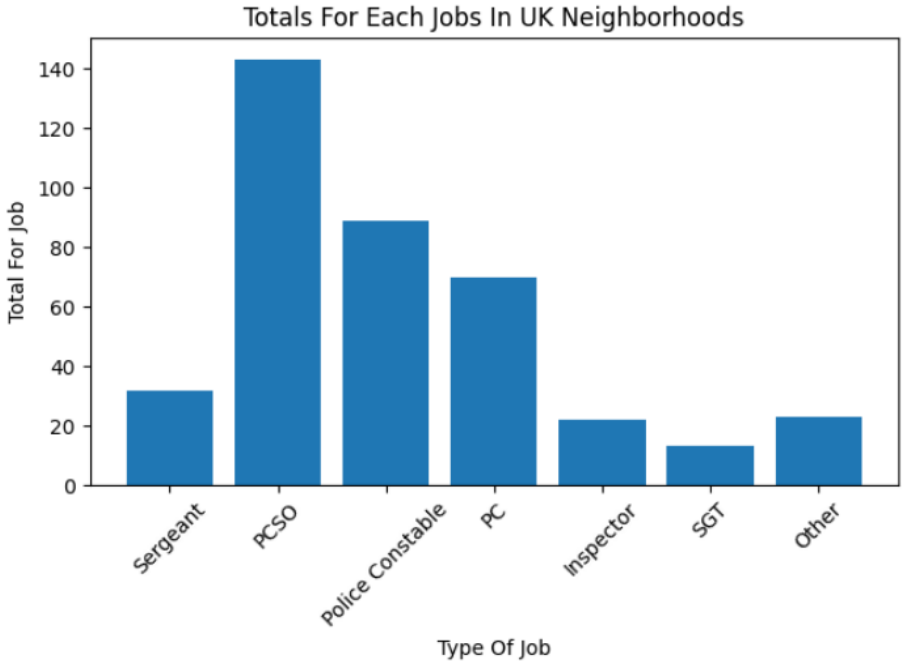


Figure #2 - A bar chart showing types of jobs and how many people are in those positions.

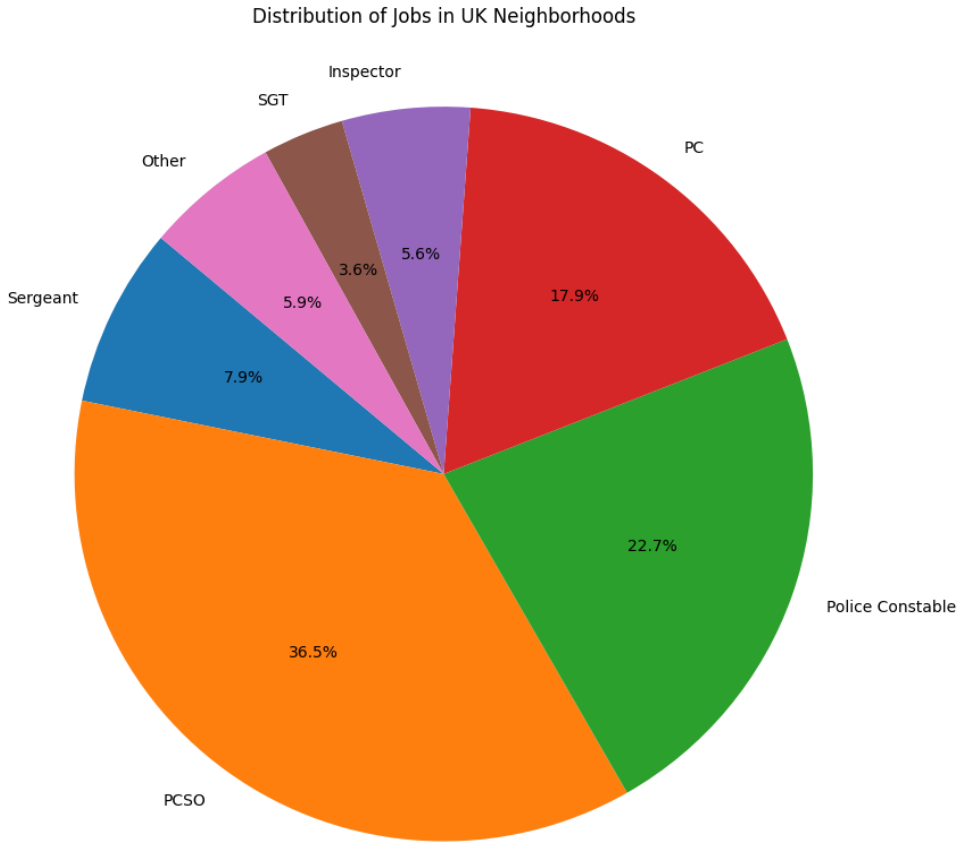


Figure 3 - A pie chart showing the percentage of each position that is held.

My code was split into two different files. The first file would call the API and make sure that the most number of people being grabbed at the same time was capped at 25. When a person was grabbed from the API, the data being inserted to the database would include the city_id the person worked at, the name of that person, and the position of that person. In the second file, the calculations for showing how many people had certain positions were done. In order to do this, I made a list of jobs then found how many people worked in each of those jobs. Once finished, I subtracted the total number of people by the total number of jobs found in the database call to see how many other jobs there were in the database. Once done, I created a bar graph showing the job position in the x-axis and the total number of jobs in that position in the y-axis. On top of that, a pie chart was created showing the distribution of the jobs in UK neighborhoods.

grab_police_data(conn, cur, city_id):
Input - conn is a connection to the database. cur lets us make calls to and from that database.
Output - No output.
Description - Grabs data from the API for a certain city_id and limits the amount of people it grabs to 25. Once it grabs a certain amount, it sends that data (city_id, full name, and job position) to the database.

calculate_police_avg(conn, cur):
Input - conn is a connection to the database. cur lets us make calls to and from that database.
Output - Returned a list of all jobs and how many people were in that job.
Description - Made calls to the database to determine how many people were in each position.

create_police_bar_graph(results):
Input - results is a tuple with two lists, one of the names of the job and one of the total people in that job.
Output - No output.
Description - Creates a bar graph showing the results of the database calls with position in the x-axis and the total number of positions in the y-axis.

create_police_pie_graph(results):
Input - results is a tuple with two lists, one of the names of the job and one of the total people in that job.
Output - No output.
Description - Creates a pie chart showing the distribution of jobs within UK neighborhoods.

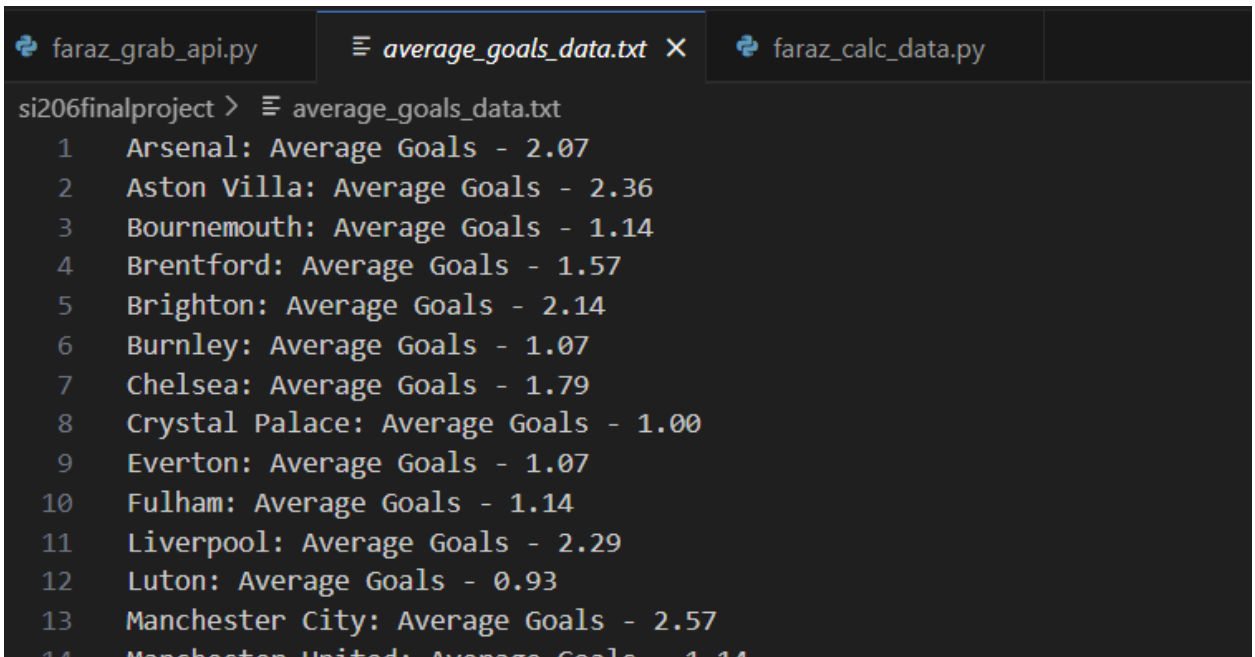
Date	Issue Description	Location Resources	Result
12/05/23	Issue with how to gather information from API at a limit of 25 grabs.	ChatGPT	page=1, pageSize=25 ChatGPT showed that this was the way to only grab 25 people or things at a time.

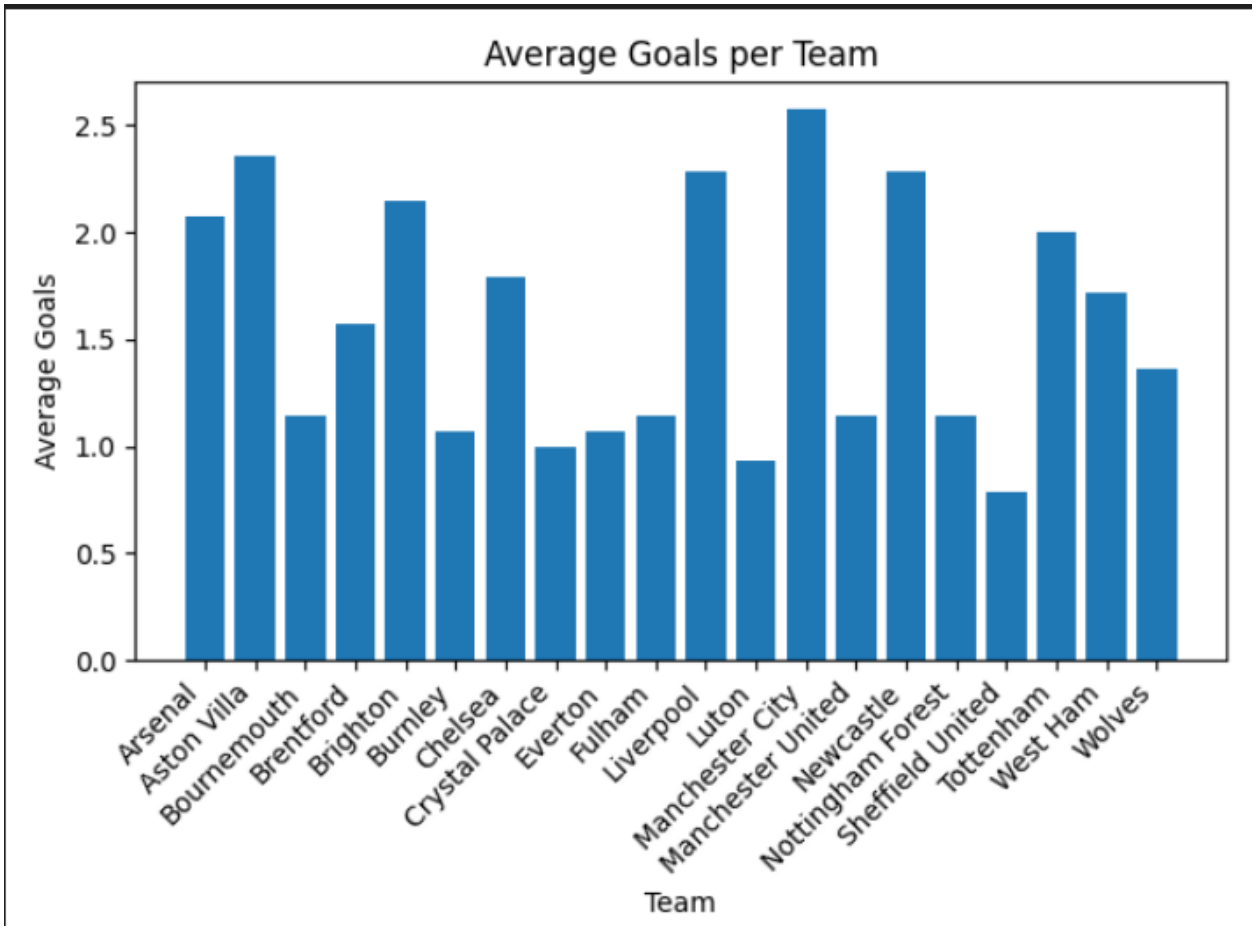
Faraz Ali

The goal of this project was to gather information about Premier League soccer teams from every matchday of the season this far. From there, I want to calculate the average goals per game from every team and visualize the data to understand the offensive strength of each team. To do this, I needed to successfully access a Premier League Live Scores API on RapidAPI.

I was able to achieve these goals by successfully accessing the API, creating a database through SQL, and finally using the data to calculate the average goals per game for every Premier League team.

There were several problems I faced when pursuing the project's goals. Firstly, my initial project plan was to look at individual player statistics on Premier League teams. However, to be able to access advanced statistics, RapidAPI charges a fee. Instead, I was able to look at Premier League team statistics from every game, and focused my research on that topic. When accessing the data and populating my database with the winners of each game, I initially had a logical error because my code did not know how to deal with games that were a draw. To fix this, I essentially said that if neither scores were greater than each other, than the game was a "Draw".





The instructions for using the code consist of three functions. The first function grabs the data from the API, establishes a new data structure using SQL, and then populates the table with the corresponding data from the API. When running the first function, you must change the matchday every time the file is run. The function pulls all 10 games from each matchday and populates the table with the data. To get all the data, the function must be run 14 times. The column names are: (matchday, team1_name, team2_name, team1_score, team2_score, winner). The first five variables are pulled directly from the API. However, the winner column is done by comparing the scores of each team. If a team has a greater score than their opponent, then they are declared the winner. If the scores are the same, the winner is declared as “Draw”. The second function calculates the average goals per game of every team throughout the season and writes to a text file with this data. It groups each team with their respective score each matchday, and then adds the scores of each team and divides them by 14 which is the number of match days thus far. The last function creates a bar chart comparing all 20 teams and their average goals per game.

grab_PL_team_data(database_path):
Input - Desired database path
Output - No output.

Description - Creates a table within a given database and populates with data from API.

calculate_average_goals(database_path):
Input - Desired database path. Must be the same database path used in the last function.
Output - Returns a dictionary with the total goals and match count for every team in the league.
Description - Calculates the average goals per game for every team in the Premier League. After doing this, it writes the results into a text file.
create_average_goals_chart(database_path):
Input - Desired database path. Must be the same database path used in the last two functions.
Output - No output.
Description - Calls the previous function to get the dictionary of each team's totals goals and matches played. From there, it calculates the average goals per game for each team and creates a bar chart. The bar chart plots the teams on the x-axis and the average goals on the y-axis. It also saves the graph as a png file.

Date	Issue Description	Location Resources	Result
12/3/23	Issue with formatting the average goals per game in a text file in second function.	ChatGPT	{average_goals:.2f}\n ChatGPT showed me a way to format the average goals so that it goes to two decimal places and also adds a newline character at the end.

Evan Marks

1) This project aimed to collect information about players from California's four NBA teams, specifically their heights and weights. Using this data, I planned to calculate the average weight per height group (5'9-6'3)(6'3-6'7)(6'7-7'0). Using RapidAI's API-NBA, I planned first to get a list of all NBA teams and their team IDs, filter to just California's teams, and create lists of player dictionaries for each team that included their Player ID, full name, weight, and height. Using this data, I developed two databases with players and their heights and players and their weights. I created a bar graph of the average weight per height group using this data.
2) I achieved all of my goals successfully. Using the Teams endpoint of the API, I broke down the API by team and filtered it only to teams within the NBA franchise. I manually reviewed the remaining teams and used my prior basketball knowledge to isolate the 30

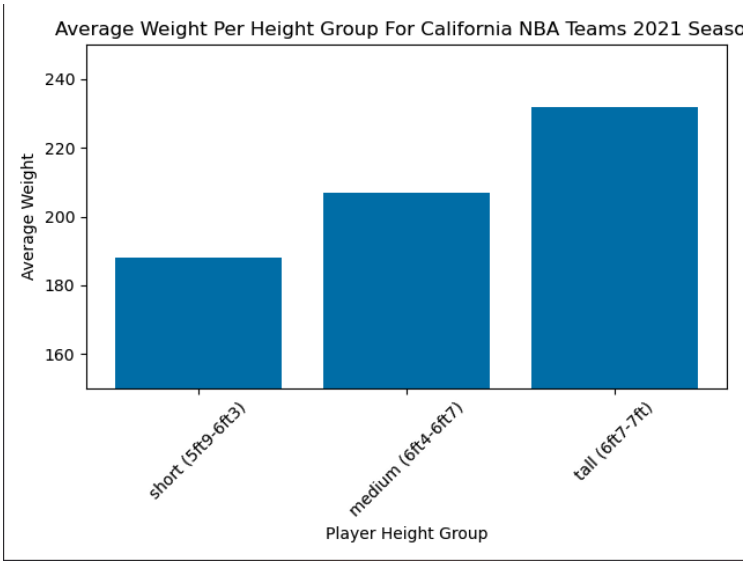
current NBA teams (2021). I isolated California's four teams (Lakers, Clippers, Kings, Warriors) using the API's team IDs. To confirm that the number of players in my player list was equal to the number of players on the four teams, I ensured that the length of the four players_data lists was equal to the number of players on the all_players_list. I then went through the individual player dictionaries, putting my desired data into the database and creating a graph with calculations.

3) One problem I faced was that some of the player weights and heights on the API were missing. I accounted for these players by making a "none" category in my database and withholding those players from my final calculations. Another problem I faced was that my code often struggled to run such a large quantity of API data simultaneously, so I added pause times. A third problem I faced was not limiting the number of items stored in the database to below 25 each time the code ran. I fixed this by adding a for loop to iterate 19 times for each time the code ran so that it ultimately had all 114 players after six times.
4) Below are the average weight per height group calculations, accounting for the "None" category.

```
This file contains the average weight per height group for California NBA teams during the 2021 season.

Average Weight Data (Height Range (ft. in.) - Average Weight (pds)):
None - (0.0, 'None')
medium (6ft4-6ft7) - (207.0, 'medium (6ft4-6ft7)')
short (5ft9-6ft3) - (188.0, 'short (5ft9-6ft3)')
tall (6ft7-7ft) - (232.0, 'tall (6ft7-7ft)')
```

5) Below is the visualization I created.



6) INSTRUCTIONS: Run the evan_grab_api code 6 times so that the database fills up with all 114 California players. Then run the evan_calculate_data code once to calculate the average weight per height group and view the bar chart.
7) Documentation For Functions:

1. create_player_dict(player)
Input:
player: A dictionary containing player information retrieved from the NBA API.
Output:
Returns a dictionary containing specific player details, including id, full name, team, weight, height, and height group.
2. set_up_tables(cursor, conn)
Input:
cursor: SQLite cursor object.
conn: SQLite connection object.
Output:
Creates two tables in the SQLite database: weights and heights, if they don't already exist.
3. fill_database(cursor, conn)
Input:
cursor: SQLite cursor object.
conn: SQLite connection object.
Output:
Fills the weights and heights tables with player data from the all_players_list. It inserts player ID, full name, weight, height, and height group into the respective tables.
4. visual_inputs(cursor, conn)
Input:
cursor: SQLite cursor object.
conn: SQLite connection object.
Output:
Retrieves and processes data from the SQLite database, calculates the average weight per height group, and creates a text file (nba_weights_heights.txt) containing the results. Also, generates a bar chart (nba_weights_heights.png) displaying the average weight per height group.
5. calculate_and_graph(cursor, conn)
Input:
cursor: SQLite cursor object.
conn: SQLite connection object.
Output:
Calculates the average weight per height group and generates a bar chart (nba_weights_heights.png) displaying the results. Additionally, creates a text file (nba_weights_heights.txt) containing the average weight data.