University of Michigan

SI 206 Final Project Report

Github Link:

https://github.com/zfewkes/SI-206-Final-Project/tree/master

By Zachary Fewkes, Abhay Shakapur, and Alex Choi

SI 206

Professor Barbara Erickson

4/27/2021

I. Goals

The original purpose of this project was to:

1. Create data tables for a soccer, basketball, and football league for the 2019-2020 season
2. “Normalize” the data for each sport and plot them against each other
3. See which teams in the Premier League tended to score more goals for home games
4. Use two API’s and Beautiful soup to get the data into the database

II. Achieved Goals

1. Getting the data from balldontlie API, API-FOOTBALL, API-HOCKEY. The data was created into a table from each API and was put into a database. The main table for each sport contained the column names: id, game_id, home_team_score, away_team_score, agg_score, hometeam_id, away_team_id. The last two columns were used to join on the second table associated with each sport that had the columns ‘team’ and ‘stadium’ so that we could avoid repeating these strings in the larger table. This data was from the 2019 - 2020 season
2. Used Beautiful soup to read the stadiums for each team from https://en.wikipedia.org/wiki/List_of_National_Basketball_Association_arenas
3. Plotting the distribution of the scores of the three sports with a Kernel Density Estimation and a bar plot using seaborn
4. Getting “Normalized” scores based on how many standard deviations a team’s mean was away from the mean of all the teams scores with the formula:

$$\frac{(team\ mean) - (all\ teams\ of\ the\ sports\ mean)}{(all\ teams\ of\ the\ sport\ standard\ deviation)}$$

5. Plotting the top 10 best teams across the three different sports with the normalization applied in comparison to each other using plotly
6. Writing the Json data of the mean, median, and standard deviation of the scores for each team in the database. Also wrote these stats for the entire league
7. Getting every 2019-2020 Premier League match score using Beautiful Soup and getting the team names and stadiums using API-FOOTBALL
8. Collected data from database to calculate average number of goals made at each stadium using a SQL function and SQL commands
9. Used the calculation to create a bar graph that compares average goals made for each Premier League team when playing at their own stadium

III. Problems Faced

1. There was some difficulty in getting the stadiums and game scores from the APIs. Often either one of these results had to be supplemented with Beautiful soup to scrape them off of Wikipedia.
2. There was some initial confusion on whether we could store the team names in our main tables for each sport. This was of course not allowed, as repeated strings take up too much memory in a database.

3. A few of our original ideas for the project could not have involved using any calculations, such as plotting all the points from a data table in order to create a heatmap that compares players' scores from normalized data.
4. Seaborn ended up being finicky with the intensity adjustments for the Kernel Density Estimation. It took some sifting through documentation to figure this out
5. Difficulty was encountered plotting the axis titles with plotly. This was also rectified by looking at the excellent documentation.
6. There was a lot of confusion on how to store the data when reading it in from the database, such that we could calculate the mean, median, and standard deviation and store them. It was eventually decided to use dictionaries of dictionaries to solve the issue. This is well documented later in this paper.
7. There was some confusion on how to efficiently use the join statement to get the team names from the supplementary table for each sport. This question was answered in office hours.

IV. Calculation File

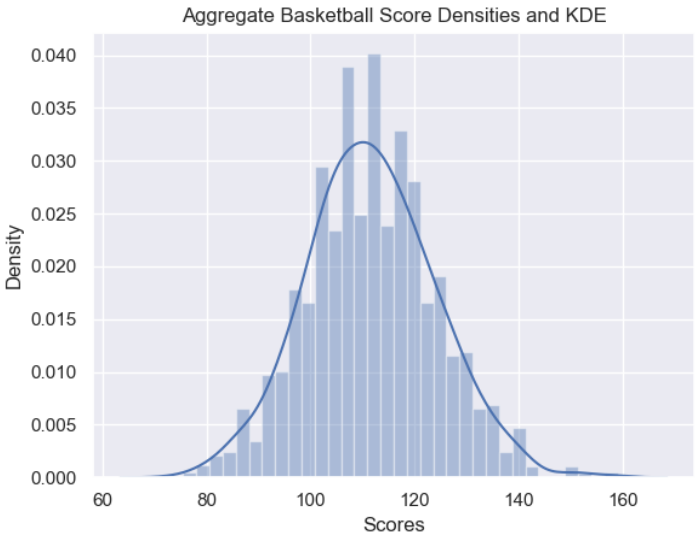
These can be retrieved from the following links:

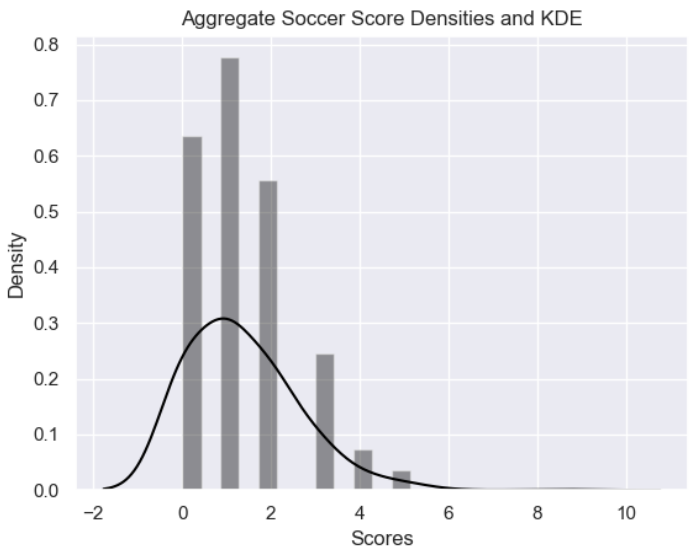
-https://github.com/zfewkes/SI-206-Final-Project/blob/master/data%20processing.py

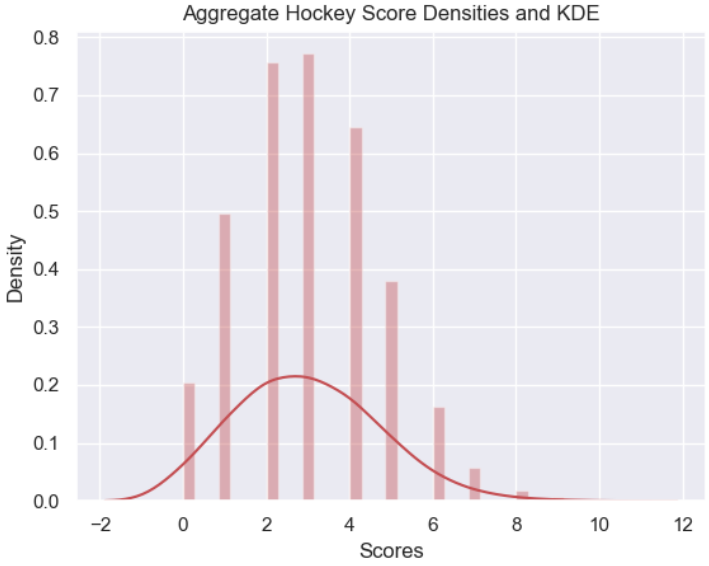
-**SI-206-Final-Project/soccer_calculations.py at master · zfewkes/SI-206-Final-Project (github.com)**

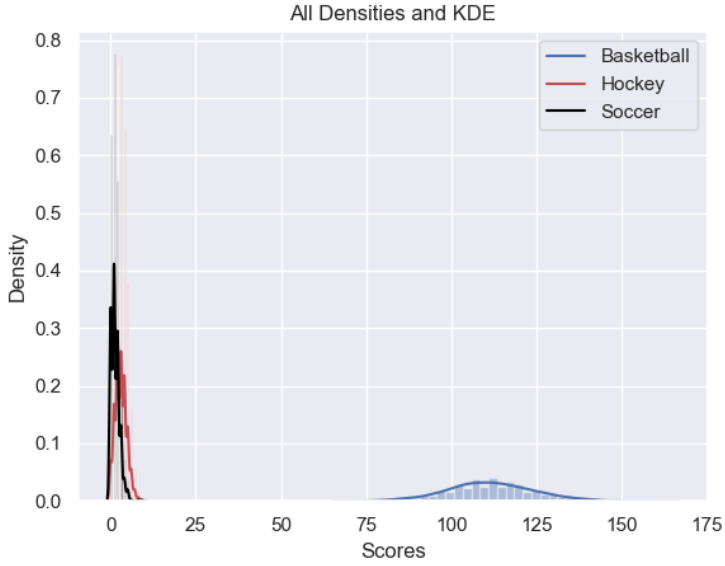
- https://github.com/zfewkes/SI-206-Final-Project/blob/master/norm%2Bscatter.py

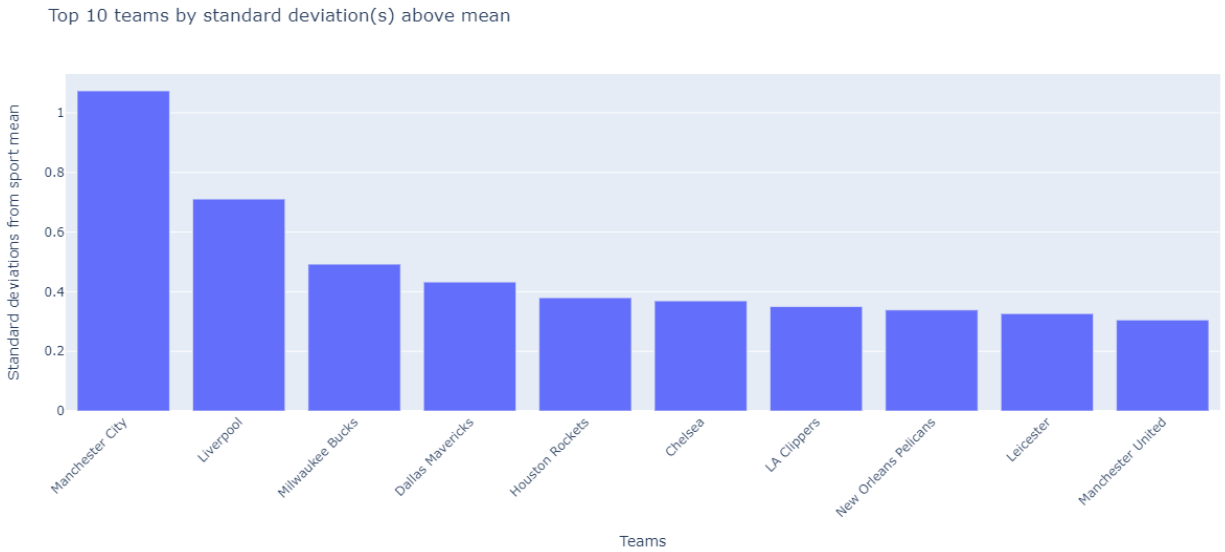
V. Visualizations



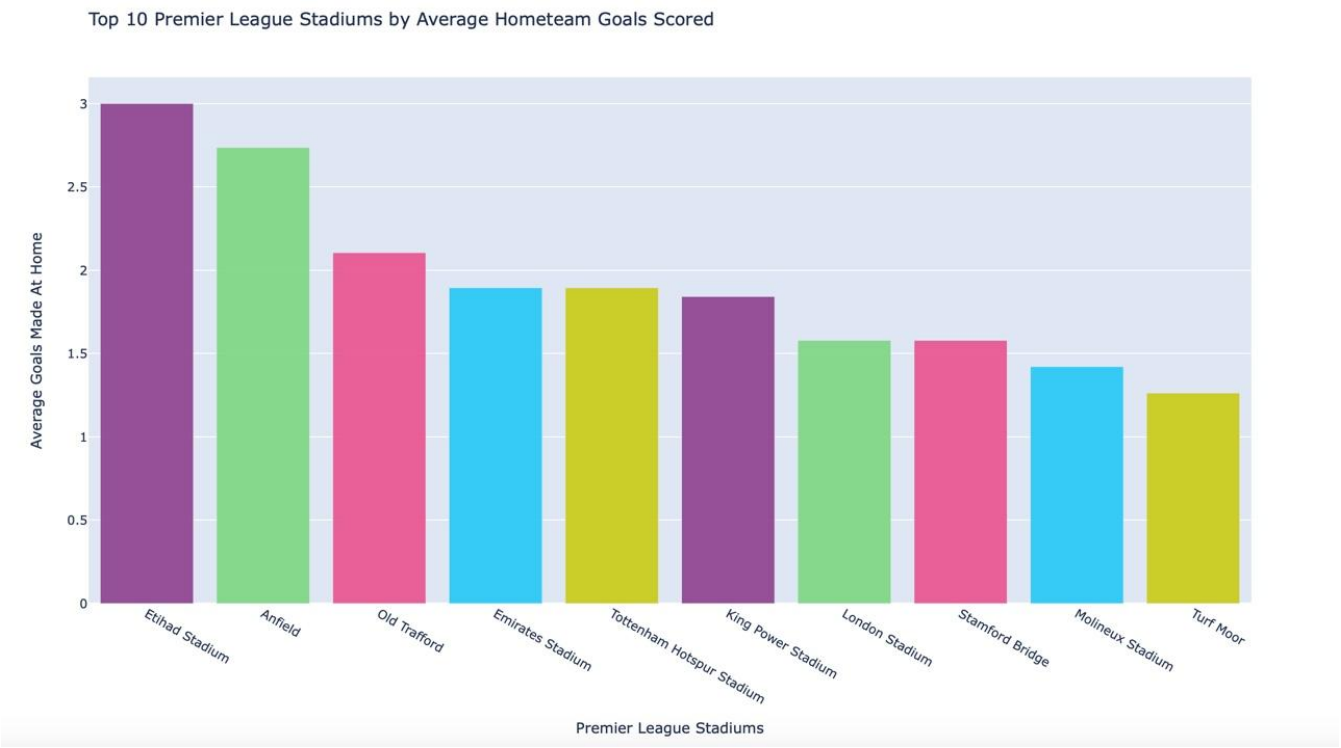




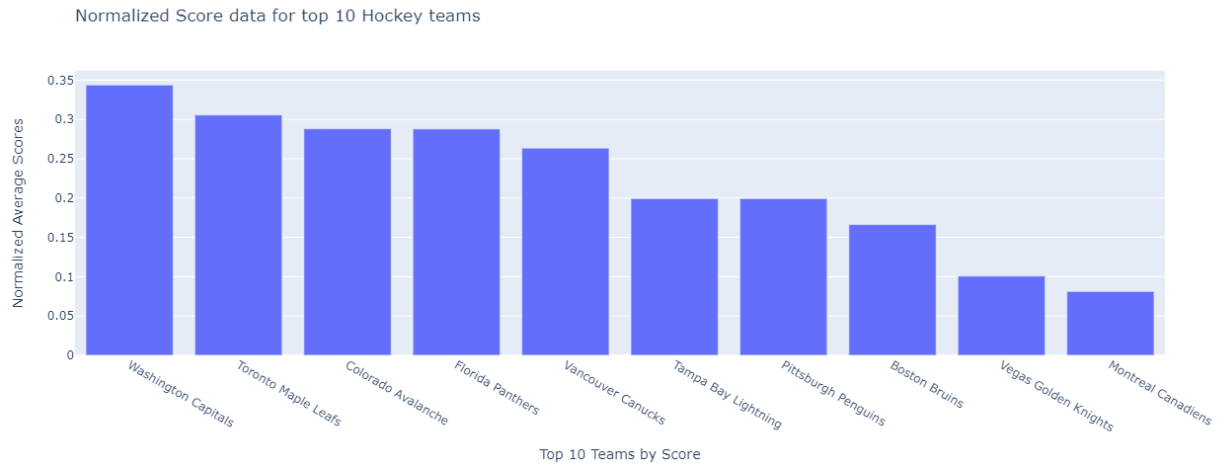


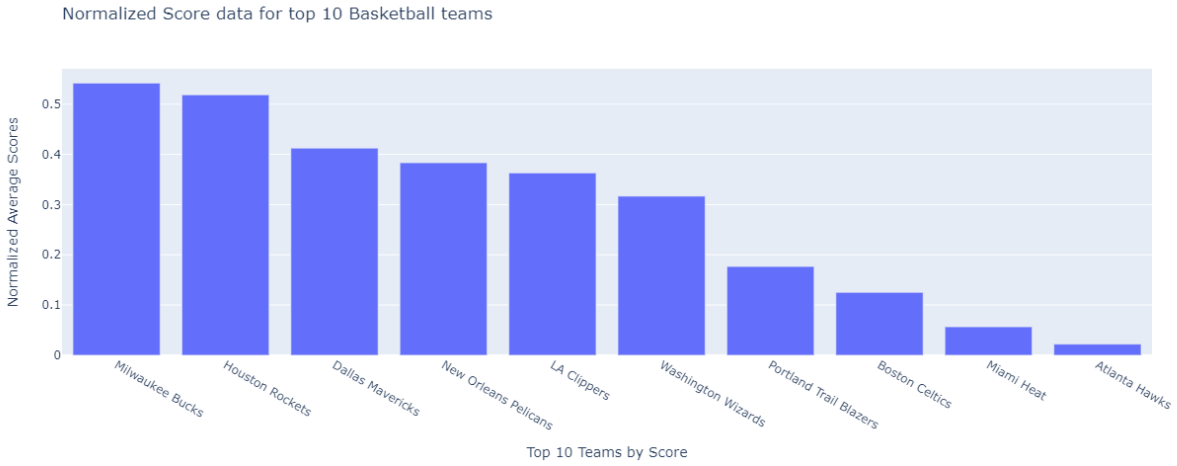


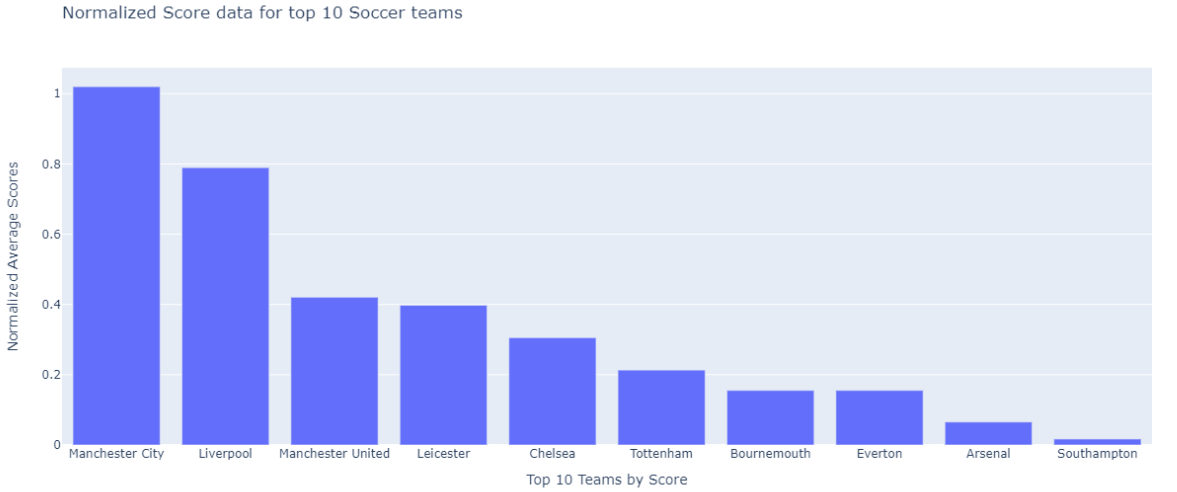
Bar Chart for Soccer Data



Bar Chart for Normalized Top ten Teams data







VI. Instructions for running code

1. Install packages: for plotly, matplotlib, seaborn, and sqlite3

1. Create a database called Sports.db in DB browser

2. For basket_ball.py simply press run the program 45 times until the data table fills up (1142 entries) or until you are satisfied with the amount of entries.

3. Run soccer_data.py 15 times until the data table fills up to 380 rows or run the file until a preferred amount of entries are met
4. Run hockey.py at least 15 times or until satisfied
5. For data processing.py, simply run the program with a few values in the Sports.db database for each sport/table. The program should automatically create the graphs and calculate the data that it's supposed to display. The plotly graph will open in the browser, but the other two will open as their own files
6. For norm+scatter.py, simply make sure that the database has entries for each file and then run. It should generate all three graphs automatically in your browser
7. For soccer_calculations.py, as above make sure that the database has a few entries for each sport and then simply click run. The plotly graph will appear in your browser

VII. Code Documentation

Code shared by basket_ball.py, hockey.py, and soccer_data.py:

1. def setUpDatabase(db_name):

Input:

- db_name, a variable giving the name of the database to connect to

Output:

- Nothing

Description:

- Connects to the database specified by db_name and returns the database cursor and connection.

2. def setUpSportsTable(name, cur, conn):

Input:

- File name for the data table, database cursor and connection

Output:

- Column names for the data table

Description:

- Creates datatable of information for every match in the 2019-2020 season if it doesn't already exist.

3. def setUpLocation_TeamTable(name, cur, conn):

Input:

- File name for the data table, database cursor and connection

Output:

- Column names for the data table

Description:

- Creates datatable for every team name and its stadium if it doesn't already exist

Code Exclusive to basket_ball.py:

1. def read_all_team_ids():

Input:

- Nothing

Output

- data, a dictionary gotten directly from the API that contains every team and its ID or prints “something’s wrong” if the request throws an error

Description:

- Makes a request to the balldontlie api and reads all the teams and their ids (determined by the api) and stores them in data (along with a bunch of other data)

2. def read_all_stadium_ids():

Input:

- Nothing

Output:

- A dictionary, team_dict, that contains the team names as keys and the stadium the team plays at as a value.

Description:

- Uses Beautiful Soup to read the wikipedia page https://en.wikipedia.org/wiki/List_of_National_Basketball_Association_arenas, and find all the team names and where they play. Writes this to a dictionary to later be written to the basketball_teams_stadium table

3. def read_25_ball_dont_lie_api(page):

Input:

- The page to read from the API’s data

Output:

- A dictionary, data, that has the information retrieved from the balldontlie api in JSON format with exactly 25 games.

Description:

- Makes a request to the balldontlie api, then takes the JSON returned and writes it to a python object. This will later be cherry picked and stored in the function write_to_bball_db. Because of the way the api works, we can request only 25 games data, making it really easy to limit ourselves to that number per run.

4. def write_to_bball_db(data, cur, conn):

Input:

- data(from read_25_ball_dont_lie_api)
- cur (the database cursor)
- conn(the database connection)

Output:

- Writes the items in data to the Basketball table, and creates the columns game_id, home_team_score, away_team_score, agg_score, home_team_id, visitor_team_id

Description:

- Takes the data returned from read_25_ball_dont_lie_api and writes it to the database table "Basketball"

5. def write_to_basketball_teams_stadiums(data, cur, conn, stadiums):

Input:

- data(dictionary returned from read_all_team_ids()),
- cur (the database cursor),
- conn(the database connection),
- and stadiums(dictionary returned from read_all_stadium_ids())

Output:

- Writes id, team, and stadium to Basketball_teams_stadiums

Description:

- Takes the stadiums collected from read_all_stadium_ids() and the teams and ids collected from read_all_team_ids() and writes them to the supplemental basketball table, Basketball_teams_stadiums

6. def main():

Input:

- None

Output:

- 25 rows to the sport

Description:

- Runs many all the functions above and also calculates the page to read data from in the balldontlie api to get 25 games and put them in the database.

Code Exclusive to soccer_data.py

1. def get_team_ids():

Input:

- None

Output:

- A dictionary in which the key is the string ‘teams’ and the value is a list of dictionaries that contains facts for each Premier League team, such as ‘name’, ‘venue_name’, ‘code’, and ‘founded’.

Description:

- Gets data from API-FOOTBALL by using requests.get to put the 2019-2020 season code into the URL and by using the given API key. The JSON string from the requests.get is converted into a dictionary. An exception is made if the API call does not work.

2. def get_game_results():

Input:

- None

Output:

- List of tuples that contains the score of each match and the row and column where the match was located in the Wikipedia table. Also, a list of all team names and the row that the team name was located on the table.

Description:

- Creates a soup object from Wikipedia page of the 2019-2020 Premier League season to extract all the scores from a results table(2019–20 Premier League - Wikipedia). Web scrapes the data by looping through the row tags and getting the column tags and team

name for each row. A few team names from the page did not exactly match the team names from the API, so a few adjustments had to be made to the tags that identified the team name. These adjustments needed to be made for creating the team_stadium_dict (described in the main() function).

3. def write_to_soccer_db(data, team_list, team_stadium_dict, cur, conn):

Input:

- data(tup_list from get_game_results() function)
- team_list(team_list variable from get_game_results() function)
- team_stadium_dict(created in main() function)
- Database cursor and connection

Output:

- Utilizes the data given to insert the game_id, home_team_score, away_team_score, agg_score, stadium_hometeam_id, and visitor_team_id into the Soccer data table.

Description:

- Initializes the game id and row number to 1. Loops through each of the tup_list. If the row value in each tuple is in 'team_stadium_dict' and if the row count is less than 26, then the values are inserted into the database and a new row is created. Each time that a row is affected, by inserted values, the row count increments by 1. The score values are inserted by getting the last two items from the tup_list. The home and away team IDs are obtained by using the first two items from the tup_list as the index to the team_stadium_dict.

4. def write_to_soccer_teams_stadiums(data, cur, conn):

Input:

- data(used stadium_list, which is created in main() function)
- Database cursor and connection

Output:

- Writes the team_id, team name, and stadium name to the Soccer_teams_stadiums datatable

Description:

- Loops through stadium_list, which a list of tuples, in order to take each item in each tuple and insert it into Soccer_teams_stadiums data table

5. def main():

Input:

- None

Output:

- 25 rows to the soccer game results table after each run
- 20 rows of the soccer teams and stadiums data table

Description:

- Creates the stadium_list and team_stadium_dict in order to use these two variables as input when running the two functions above. Stadium_list and team_stadium_dict use the dictionary returned from the get_team_ids() function and loops through each team dictionary. Team_stadium_dict uses the stadium_list and team_list variables to connect the team name's Wikipedia table row number with the stadium_id from stadium_list

variable. Main() function runs many of the functions above to put data from 25 games into the database.

Code Exclusive to hockey.py

1. def getScores():

Input:

- None

Output:

- Returns data about the games played in the 2019-2020 hockey season from a hockey api in a json format

Description:

- This code pulls the game data from the api

2. def getArenas():

Input:

- None

Output:

- Returns data about the games played in the 2019-2020 hockey season from a hockey api in a json format

Description:

- This code pulls the game data from the api

3. def parseScores():

Input:

- Takes in data from the getScores function in a dictionary format

Output:

- Returns a dictionary about game data with the unnecessary information about each game filtered out

Description:

- This function is used to pull the necessary data from the list of game data dictionaries which included the home team id, home team score, away team score, total score, and the stadium id

4. def parseArenas():

Input:

- Takes in data from the getArenas function in a dictionary format

Output:

- Returns a dictionary that contains the necessary information about each arena in the hockey leagues

Description:

- The function takes in game data about the hockey games and pulls out data about each arena and team from the hockey leagues.

5. def writing_arenadata():

Input:

- Takes in the cur, conn for the db

Output:

- The function writes the arena data from the getArenas function that was saved locally in the code

Description:

- The function writes up the data that was selected from the getArenas function into the Hockey stadiums db

6. def writing_gamedata():

Input:

- Takes in the cur, conn for the db

Output:

- The function writes the game data from the getScores function that was saved locally in the code

Description:

- The code uses the game data and writes it into the hockey_games.db

norm+scatter.py code

1. def pullBasketball(cur, conn):

Input:

- Takes in the cur, conn

Output:

- Returns the home team score and team from the basketball db

Description:

- This function pulls the data about each home team score and team for a game from the db

2. def pullHockey(cur, conn):

Input:

- Takes in the cur, conn

Output:

- Returns the home team score and team from the hockey db

Description:

- This function pulls the data about each home team score and team for a game from the db

3. def pullSoccer(cur conn):

Input:

- Takes in the cur, conn

Output:

- Returns the home team score and team from the soccer db

Description:

- This function pulls the data about each home team score and team for a game from the db

4. def sortScores(dic):

Input:

- Takes in a dictionary

Output:

- Returns a dictionary sorted by values

Description:

- This function takes in the given dictionary and returns a dictionary that is sorted by the values

5. def normalize(xlist, ylist):

Input:

- Takes in two dictionaries

Output:

- Returns a dictionary of team names and normalized values

Description:

- This function normalizes the given data so that later on it can be graphed.

6. def visualize(sorted_team, normalized):

Input:

- Takes in the sorted and normalized data

Output:

- Returns a visual representation of the data for the top ten teams

Description:

- This function creates a visualization of the normalized data for the top ten teams

data processing.py code

1. def working_agg_scores(names, cur, conn):

Input:

- names(a list of the sport names)
- cur(database cursor)
- conn(database connector)

Output:

- A dictionary of all the aggregate scores for every game played during the season in the format {'sport1' : [agg_score1, agg_score2, …], 'sport2': [agg_score1, agg_score2, …], ... }

Description:

- This is just collecting all the aggregate scores from the database. It's a bit vestigial; this data doesn't get used later but could be useful for a couple of reasons

2. def working_home_scores(names, cur, conn):

Input:

- names(a list of the sport names)
- cur(database cursor)
- conn(database connector)

Output:

- A dictionary of all the team scores for every home game played during the season in the format:

{‘sport1’ : [home_score1, home_score2, …], ‘sport2’: [home_score1, home_score2, …],

...}

Description:

- This collects all the home team scores from each sports database

3. def working_away_scores(names, cur, conn):

Input:

- names(a list of the sport names)
- cur(database cursor)
- conn(database connector)

Output:

- A dictionary of all the team scores for every away game played during the season in the format:

  {‘sport1’ : [home_score1, home_score2, …],’ sport2’: [home_score1, home_score2, …],

  ...}

Description:

- This collects all the away team scores from each sports database

4. def game_score_plot(all_scores, cur, conn):

Input:

- all_scores (A dictionary that contains all the scores for each team in each sport, formatted as: all_scores{‘basketball’: [score1, score2, score3, …], ‘soccer’: [score1, score2,

score3, …], ‘hockey’: [score1, score2, score3, …]} that’s retrieved from appending the results from working_home_scores() and working_away_scores().

Output:

- Makes three graphs using seaborn. These are the density functions (kde and barplots) of each sports game score.

Description:

- These are the seaborn plots that are earlier in the graph. For more information of what these graphs mean (and what the code is doing) take a look at the documentation provided later in this document.

5. def calc_agg_mean_median_std(all_scores, cur, conn):

Input:

- all_scores (A dictionary that contains all the scores for each team in each sport, formatted as: all_scores{‘basketball’: [score1, score2, score3, …], ‘soccer’: [score1, score2, score3, …], ‘hockey’: [score1, score2, score3, …]} that’s retrieved from appending the results from working_home_scores() and working_away_scores().

Output:

- A dictionary that contains the mean, median and standard deviation of all of the games played over the season. It is stored as:

  {‘sport1’: {‘mean’: mean_val, ‘median’ : median_val, ‘std’ : std_val }, ‘sport2’: {}... }

Description:

- Uses numpy arrays to calculate the mean, median, and the standard deviation of all the games for each sport.

6. def calc_med_mean_std_per_team(cur, conn):

Input:

- cur(database cursor)

Output:

- Returns a dictionary that contains the mean, median and standard deviation of each team in the following format:

  {'sport1': [{'team': team_name1, 'mean': mean_val1, 'median' : median_val1, 'std', std_val1 }, {'team': team_name1, 'mean': mean_val1, 'median' : median_val1, 'std', std_val1 }, …], 'sport2': [{'team': team_name1, 'mean': mean_val1, 'median' : median_val1, 'std', std_val1 }, {'team': team_name1, 'mean': mean_val1, 'median' : median_val1, 'std', std_val1 }, …], ... }

Description:

- This is a huge function that accomplishes a lot. It first reads all the game and team data from the database using a join statement. It does this in two parts: first getting the home games and then the away games. It stores these values in an intermediary dictionary that it uses this to calculate the stats. It does this for all three sports and finally returns.

7. def write_out_json(filename, team_stats):

Input:

- filename(name of the file to write out to)
- team_stats (the dictionary returned from calc_med_mean_std_per_team())

Output:

- A json file with mean, median, and mode for each team in basketball, hockey, and soccer. It does the is the format:

  {'sport1': [{'team': team_name1, 'mean': mean_val1, 'median' : median_val1, 'std', std_val1 }, {'team': team_name1, 'mean': mean_val1, 'median' : median_val1, 'std', std_val1 }, …], 'sport2': [{'team': team_name1, 'mean': mean_val1, 'median' : median_val1, 'std', std_val1 }, {'team': team_name1, 'mean': mean_val1, 'median' : median_val1, 'std', std_val1 }, …], ... }

Description:

- This just writes calc_med_mean_std_per_team() to JSON, which was pretty easy since it was already pretty much in that format

8. def find_top_teams(agg_stats, team_stats):

Input:

- agg_stats(the dictionary returned from calc_agg_mean_median_std() in the format: {'sport1': {'mean': mean_val, 'median' : median_val, 'std' : std_val }, 'sport2': {}... })
- team_stats(the dictionary returned from calc_med_mean_std_per_team(). This has been described a few times, so refer to earlier documentation if you need a refresher)

Output:

- A sorted tuple list of the teams and the calculated value $\frac{(team\ mean) - (all\ teams\ of\ the\ sports\ mean)}{(all\ teams\ of\ the\ sport\ standard\ deviation)}$ in the form [('team_name', calculated_val), …]

Description:

- This function builds on what we found earlier to calculate the number of standard deviations away from the sport means each team's mean is, and judges their strength based on this.

9. def plot_top_teams(top_teams):

Input:

- top_teams(the list of tuples from find_top_teams())

Output:

- Creates a plotly graph of the calculated_value v.s. the team name

Description:

- Makes the plotly graph seen earlier on this document. See the documentation down below for more information about what the code is actually doing.

10. def main():

Input:

- None

Output:

- Creates 5 graphs - 3 distribution graphs of each of the sports and one graph with them all plotted against each other. Also creates a plotly graph of all the teams against each other, based on the number of standard deviations of the team’s mean from the sports mean.

Description:

- Runs all the functions described above in the correct order with the correct calls.

soccer_calculations.py code

1. def calculation(conn, cur):

Input:

- Database cursor and connection

Output:

- Dictionary for each stadium and their average home team score

Description:

- Creates a dictionary by fetching all of the rows of the query result that selected the stadium and average home team score. These are selected from the two data tables that were joined based on stadium table ids that were equal to the other table's hometeam_id. To get a value for each stadium, we had to group the final result by hometeam_id. The key value pairs are created for each of the rows.

2. def write_file(stadium_dict):

Input:

- The dictionary returned from the previous function

Output:

- JSON file of the dictionary of average home goals scored in each stadium

Description:

- Writes the calculation for average home goals scored for each stadium into a JSON file.

3. def sort_tuples(stadium_dict, number):

Input:

- stadium_dict from the calculation function and the number of stadiums we want to restrict in the output

Output:

- list of tuples that is sorted from highest to lowest based on the average home goal scored value

Description:

- Change stadium_dict from a dictionary into a list of tuples in the form of (stadium, average goals)

4. def create_graph(stadium_goal_sorted):

Input:

- Tuple list of stadium, average goals returned from the sort_tuples function

Output:

- Creates a plotly bar graph

Description:

- Creates a bar graph that shows the top ten Premier League stadiums by average home team goals scored. The x axis consisted of the ten stadium names, and the y axis had the values of average goals scored at home. The bar chart had different colors for every five stadiums.
-

VIII. Documentation

Date	Issue Description	Location of Resource	Result
4/23/2021	Needed to plot axis titles on plotly graph	https://plotly.com/python/figure-labels/	Resolved
4/23/2021	Needed basketball data	https://www.balldontlie.io/#get-all-games	Great API that really helped!
4/23/2021	Wanted to plot distributions but was unsatisfied with what Matplotlib had to offer	https://seaborn.pydata.org/tutorial/distributions.html	Resolved
4/7/2021	Searching for APIs	**API-Football® - Best Football (Soccer) API [For Developers] (rapidapi.com)**	Discovered good API that contained much soccer data
4/22/2021	How to add colors to the bar graph	RGBA Color Picker	Chose which colors to create based on customizing own RGBA values
4/22/2021	Setting titles in plotly charts	**Setting the Font, Title, Legend Entries, and Axis Titles | Python | Plotly**	Knew how to add text to the bar graph
4/8/2021	Problems with API data	**https://dashboard.api-football.com/**	Was able to pull all of the wanted data
4/15/2021	Needed to grab stadium names	**https://en.wikipedia.org/wiki/List_of_National_Basketball_Association_arenas**	Had a very nice table that made grabbing the data very easy
4/15/2021	Had to get scores, home team, and away team for every game	**2019–20 Premier League - Wikipedia**	Although it was somewhat difficult, we got it to work