SI 206 Final Project

Team Name: NCAA Basketball Analytics
Group Members: Adrian Berrigan and Brandon Wortman
Contact Info: adberrig@umich.edu and bwortman@umich.edu
Repository: https://github.com/brandonw4/NCAA-Basketball-Analytics

Goals For Our Project: We wanted to see what statistics had the greatest effect on AP rank and winning percentage. Originally these statistics were going to include things like player seniority, scoring, average player height, etc. Our goal was to create graphs of the data we collected to visually demonstrate clear trends in various statistics. This data would ideally be able to clearly demonstrate what statistics are most important for winning in the NCAA.

Goals That We Achieved: We created a database that holds over 300 ranked teams when fully stocked. We found an API that provided statistics including team points for home/ road games, average points scored, and average points allowed. We found a website that provided AP ranking data that we then scraped. Using this data we calculated and visualized multiple pieces of data such as the effect points scored have on team wins and the effect home points scored have on AP ranking.

Problems That We Faced: We couldn't use the ESPN API as we originally planned. It changed from being public to being a private API. NCAA sports data is a tough market with essentially no free public APIs. Luckily we found API-BASKETBALL and they gave us a key since we are students. However, it didn't have the data (player seniority, 3-point shooting, etc) we wanted originally so we had to change statistics to what we had available. Also, we switched the website we were scraping from so that the names would match up, meaning we could use the team names as a key to join our tables between the API and website.

Files Containing Our Calculations From Database:

ranking_vs_home_points.json


```
{"1": 2109.0, "2": 1841.0, "3": 2214.0, "4": 2035.0, "5": 1683.0, "6": 1449.0, "7": 1724.0,
 "8": 1773.0, "9": 1794.0, "10": 2198.0, "11": 1627.0, "12": 2027.0, "13": 1814.0, "14": 1920.0,
 "15": 1438.0, "16": 1458.0, "17": 1771.0, "20": 1527.0, "21": 1919.0, "22": 1376.0, "24": 1609.0,
 "25": 1413.0, "26": 1475.0, "27": 1343.0, "28": 1423.0, "31": 1340.0, "32": 1575.0, "33": 1262.0,
 "36": 1425.0, "37": 1445.0, "39": 1192.0, "42": 1733.0, "43": 1319.0, "44": 1300.0, "46": 1308.0,
 "47": 1391.0, "48": 1406.0, "49": 1744.0, "50": 1274.0, "51": 1361.0, "52": 1332.0, "55": 1203.0,
 "56": 1293.0, "58": 1390.0, "60": 1503.0, "61": 1437.0, "62": 1365.0, "67": 1672.0, "68": 1072.0,
 "70": 1531.0, "71": 1082.0, "73": 1392.0, "74": 1517.0, "75": 1222.0, "76": 1462.0, "79": 1347.0,
 "80": 1327.0, "81": 1404.0, "82": 1268.0, "83": 1530.0, "84": 1296.0, "85": 1616.0, "86": 1380.0,
 "89": 1461.0, "90": 1398.0, "91": 1431.0, "92": 1399.0, "93": 1474.0, "94": 1228.0, "96": 1455.0,
 "97": 1441.0, "101": 1345.0, "102": 1026.0, "103": 1414.0, "106": 1244.0, "107": 1375.0, "108": 1238.0,
 "109": 1123.0, "110": 1453.0, "111": 1156.0, "112": 1214.0, "113": 1440.0, "114": 1037.0, "117": 1108.0,
 "118": 1254.0, "119": 1456.0, "120": 1071.0, "121": 1431.0, "124": 1228.0, "125": 1279.0, "126": 1198.0,
 "130": 1209.0, "131": 1616.0, "134": 1294.0, "135": 976.0, "136": 807.0, "139": 1459.0, "140": 1134.0, "141": 1162.0,
 "142": 1263.0, "143": 1065.0, "145": 1392.0, "147": 1262.0, "149": 1233.0, "150": 1250.0, "151": 1316.0, "152": 1371.0,
 "154": 1101.0, "156": 1208.0, "157": 1369.0, "160": 1410.0, "161": 1283.0, "163": 986.0, "164": 1219.0, "165": 1519.0,
 "168": 1121.0}
```

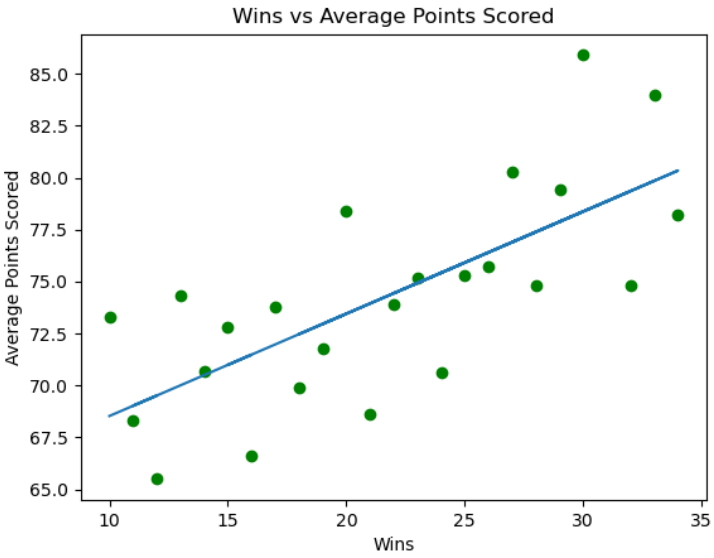

wins_vs_points_scored.json

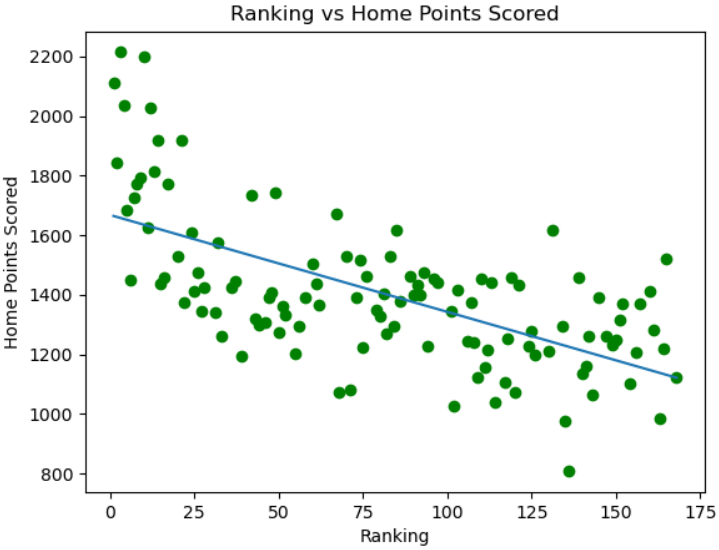

```
{"33": 84.0, "28": 74.8, "27": 80.3, "32": 74.8,
 "23": 75.2, "26": 75.7, "34": 78.2, "22": 73.9, "29": 79.4,
 "30": 85.9, "19": 71.8, "21": 68.6, "25": 75.3, "24": 70.6,
 "18": 69.9, "20": 78.4, "17": 73.8, "15": 72.8, "16": 66.6,
 "14": 70.7, "13": 74.3, "11": 68.3, "12": 65.5, "10": 73.3}
```

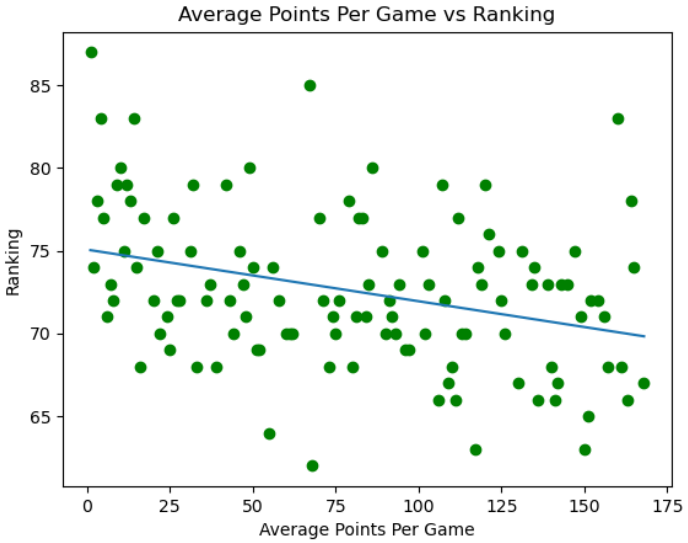

average_points_per_game_vs_ranking.json


```
{"1": 87, "2": 74, "3": 78, "4": 83, "5": 77, "6": 71, "7": 73, "8": 72, "9": 79, "10": 80, "11": 75,
 "12": 79, "13": 78, "14": 83, "15": 74, "16": 68, "17": 77, "20": 72, "21": 75, "22": 70, "24": 71,
 "25": 69, "26": 77, "27": 72, "28": 72, "31": 75, "32": 79, "33": 68, "36": 72, "37": 73, "39": 68,
 "42": 79, "43": 72, "44": 70, "46": 75, "47": 73, "48": 71, "49": 80, "50": 74, "51": 69, "52": 69,
 "55": 64, "56": 74, "58": 72, "60": 70, "61": 70, "62": 70, "67": 85, "68": 62, "70": 77, "71": 72,
 "73": 68, "74": 71, "75": 70, "76": 72, "79": 78, "80": 68, "81": 71, "82": 77, "83": 77, "84": 71,
 "85": 73, "86": 80, "89": 75, "90": 70, "91": 72, "92": 71, "93": 70, "94": 73, "96": 69, "97": 69,
 "101": 75, "102": 70, "103": 73, "106": 66, "107": 79, "108": 72, "109": 67, "110": 68, "111": 66,
 "112": 77, "113": 70, "114": 70, "117": 63, "118": 74, "119": 73, "120": 79, "121": 76, "124": 75,
 "125": 72, "126": 70, "130": 67, "131": 75, "134": 73, "135": 74, "136": 66, "139": 73, "140": 68,
 "141": 66, "142": 67, "143": 73, "145": 73, "147": 75, "149": 71, "150": 63, "151": 65, "152": 72,
 "154": 72, "156": 71, "157": 68, "160": 83, "161": 68, "163": 66, "164": 78, "165": 74, "168": 67}
```

Our Visualizations:







Instructions for Running Code:
Run NCAA Basketball Analytics.py and select **n** when asked if you want to use cache (cache doesn't take data from the API and is there for testing purposes ONLY). After selecting not to use the cache the database will be updated. Run NCAA Basketball Analytics.py 5 more times to get over 100 results in each table, again selecting **n** each time when prompted. During peak hours the API may experience high latency and take a while to be called. After the database has been filled to the desired amount locate the viz file and run it. Visualizations will appear one at a time, simply hit the x to exit the visualization and the next will appear.

Documentation of Our Functions:

NCAA Basketball Analytics.py Functions

def create_table():

- Creates the Points table used in our database. Holds Names, Wins, Losses, average points scored, and average points allowed.

def data_entry():

- Writes the data for Names, Wins, Losses, average points scored, and average points allowed. It does this via a for loop that iterates through a list of tuples containing all the required data for each team. The teams in the list (list3) are appended there in code outside of our functions.

def create_table_name():

- Creates the Names table used in our database. Holds team IDs (used later to call the API to search for team statistics), names, total home points, and total away points.

def data_entry_name():

- Writes the data for the Names table for team IDs and team names. It does this via a for loop that iterates through a list of tuples containing all the required data for each team.

def extra_data():

- This adds the total home points scored and total away points scored statistic to the Names table. This had to be done in a separate function from data_entry_name() because the IDs are needed to be able to call the API endpoint to get statistics like total away points scored.

get_team_rankings():

- This function is used to web scrape NCAA AP rankings using beautiful soup. It returns data_dict which is used in the bulk of our code. Data_dict is a dictionary of dictionaries so we iterate through it using the .items() for loop method and then append the data to a list, testlst. Testlst is then used in our code to append to another list, namelst while also limiting the amount of data in namelst to below 25.

create_table_rank():

- Creates the Ranking table used in our database. Holds Name and Rank as columns.

def count_columns():

- This function counts the number of rows in the Ranking table. Returns the variable data6 an integer that shows how many rows we have. This data is used later on as the first number in a slice. This is so our program always writes new data only and each run picks up where the last run left off.

def ranking_data():

- This function writes the data for our rank and limits the entries to less than 25. This in turn limits the data recorded in the other tables since Names only records teams in the Rankings table and Points only has teams in Names.

def read_data():

- This function appends the data2 global variable. It writes all the IDs stored in Names to the data2 list. These IDs are then used in an f string of calling API endpoints. Basically, the statistics we want are all held in separate API endpoints for each team. We use a for loop and this list of IDs to get all the statistics for all of the teams.

Viz.py Functions

def ranking_vs_home_points():

- This function takes the cursor and connection object as an input since it utilizes our SQL table. It uses the SQL join command to link the Names and Rankings tables. It uses Matplotlib to visualize AP ranking vs how many homes points a team scores. We used a scatterplot.

def wins_vs_points_scored():

- This function takes the cursor and connection object as an input since it utilizes our SQL table. It uses Matplotlib to visualize the effect Average points scored for a team has on total team wins. We used a scatterplot.

def average_points_per_game_vs_ranking():

- This function takes the cursor and connection object as an input since it utilizes our SQL table. It uses Matplotlib to visualize the effect that average points scored per game have on AP ranking. Since average points per game is not a statistic we have it had to be calculated. We added the wins and losses columns together to get

the total games played. We joined the Points, Ranking, and Names tables using join statements. We added total away and home points together to get total points. We did total points / the prior total games value to get average points per game. It uses Matplotlib to visualize the effect between this average points per game value and ranking.

Resources:

Date	Issue Description	Location of Resource(s)	Result
4/19/2022	Adrian was in charge of the API data and needed to learn API endpoint notation for the team's API.	https://rapidapi.com/api-sports/api/api-basketball/	Learned everything needed to continue
4/23/2022	Adrian confused about json.dumps vs json.dump.	https://www.geeksforgeeks.org/write-a-dictionary-to-a-file-in-python/ https://www.geeksforgeeks.org/json-dump-in-python/ https://stackoverflow.com/questions/54755971/saving-multiple-api-calls-in-a-json-file-in-python https://stackoverflow.com/questions/12309269/how-do-i-write-js	Successfully wrote dictionary to a file as needed.

		on-data-to-a-file	
4/23/2022	Adrian had an issue with JSON file saving with backslashes	https://stackoverflow.com/questions/67442928/saving-json-adds-backslashes	No more backslashes! Was able to create a cache file.
4/20/2022	Adrian had an issue iterating through nested dictionary	https://stackoverflow.com/questions/43752962/how-to-iterate-through-a-nested-dict https://www.w3schools.com/python/ref_dictionary_items.asp	Found the .items() function through this and then that fixed the issue
4/21/2022	Adrian needed a refresher on .fetchall() and how to find the number of rows in the table.	https://stackoverflow.com/questions/21829266/how-to-get-the-numbers-of-data-rows-from-sqlite-table-in-python	Was successfully able to implement a function that counts the rows in our table.
4/21/2022	Adrian needed to compare two lists and needed ideas	https://datagy.io/python-subtract-two-lists/	Was able to essentially subtract two lists with an “if in” statment
4/23/2022	Adrian was having issues updating Names table. Insert didn’t work	https://www.w3schools.com/sql/sql_update.asp	Found update function to work

		https://pynative.com/python-sqlite-update-table/	perfectly
4/19/2022	Adrian needed help with json.loads and json.load to use data from API	https://www.geeksforgeeks.org/read-json-file-using-python/	Was able to successfully load API data
4/19/2022	Adrian troubleshooting LONG runtime issues. Needed to know how long the program was running.	https://stackoverflow.com/questions/1557571/how-do-i-get-time-of-a-python-programs-execution	Used import time and code to get runtime duration.
4/20/2022	Adrian needed an f string refresher	https://realpython.com/python-f-strings/	Used f string successfully in API endpoint
4/20/2022	Adrian needed an SQL refresher for using data	https://www.sqlitetutorial.net/sqlite-python/sqlite-python-select/ https://www.sqlitetutorial.net/sqlite-python/sqlite-python-select/	Successfully queried all required data.
4/20/2022	Adrian needed a SQL refresher for creating tables/reading data	https://www.youtube.com/watch?v=NCc5r7Wr7gg	Successfully learned all the SQL syntax needed from this video and the textbook.
4/19/2022	Adrian needed a refresher on dictionaries and pulling keys	https://www.geeksforgeeks.org/python-get	Successfully remembered

	and items from them.	-key-from-value-in-dictionary/	everything needed
4/26/2022	Brandon needed to figure out how to create a line of best fit using numpy and matplotlib	How to Plot Line of Best Fit in Python (With Examples) - https://www.statology.org/line-of-best-fit-python/	Figured out the functions needed in order to calculate line of best fit
4/25/2022	Brandon needed to figure out how to use the SUM function in SQL to calculate multiple columns and how to combine more than one join to calculate across multiple tables.	https://learnsql.com/blog/how-to-join-3-tables-or-more-in-sql/	Figured out how to use the SUM and JOIN functions in SQL
4/22/2022	Brandon needed to figure out how to use different string formatting with sql execute to add data in more than one spot	https://realpython.com/python-f-strings/	Figured out to use an f string instead of the SQL ?, ? method in this case
4/21/2022	Brandon was confused on how to use SQL to only insert when the data is not already there	https://dba.stackexchange.com/questions/189058/how-do-i-insert-record-only-if-the-record-doesnt-exist	Figured out that selecting the data first and then checking to see if it was there before adding it worked
4/21/2022	Brandon was confused on	https://stackoverflow.	Figured out

	how to gather tr and td tags specifically in Beautiful Soup	com/questions/33181706/how-to-grab-a-specific-td-within-a-tr-with-beautifulsoup	how to loop through beautifulsoup tags and use the find methods to get more specific tags
4/19/2022	Brandon was confused on the requests function in Python	https://docs.python-requests.org/en/latest/	Figured out the correct format
4/24/2022	Brandon was confused on some of the functions in matplotlib and needed a refresher on plt	https://matplotlib.org/cheatsheets/	Remembered how to implement scatter plots