Andrew Bonnah, Chloe Powell, Yessenia Santana Pérez
Prof. Barbara Ericson
SI 206 - Data Oriented Programming
Dec. 12, 2023

Team CALL ME SALAD Final Report

Summary

As basketball fanatics, we wanted to focus on the APIs and websites that had NBA statistics. Our primary objective was to understand who the best teams are, who the best players in the league are, and the specific statistics behind it. The APIs we used are NBA Stats and balldontlie.io to find the data we were looking for. As for the websites, we only used ESPN for 2023 player statistics. The data we were hoping to gather from both the APIs and the websites was player points scored, assists by a specific team, and winners/losers for every team in the NBA. Luckily, we were able to use all of the APIs we planned on using with little issues. For the NBA Stats API, there were many different ways we could summon the data. One of the difficulties we had was deciding which statistics to use and what calculations to make. However, after we formulated a research question, it was much easier to pinpoint which data to use and which to disregard. We ultimately were able to get all the required information and answer the following research questions

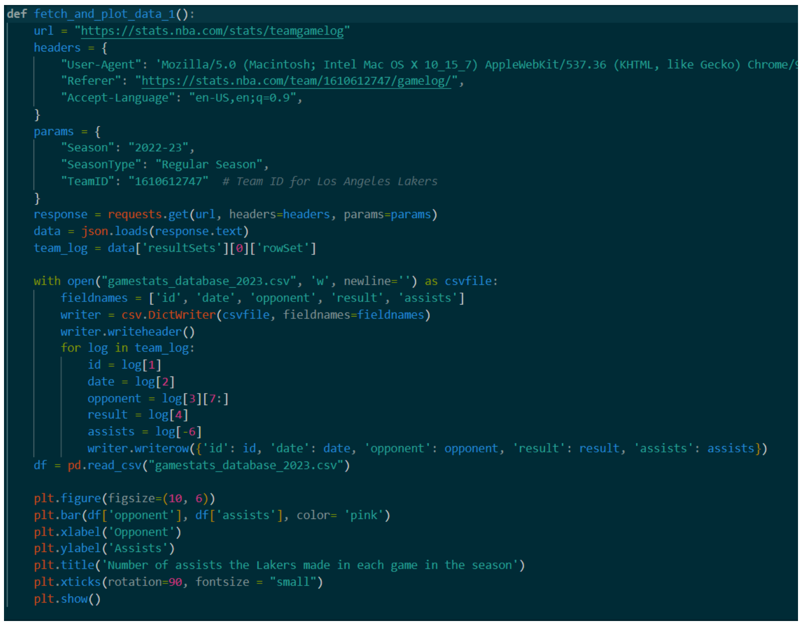
Research Questions

RQ1: The driving research question was as follows: How do certain basketball statistics such as assists impact the outcome of the game?

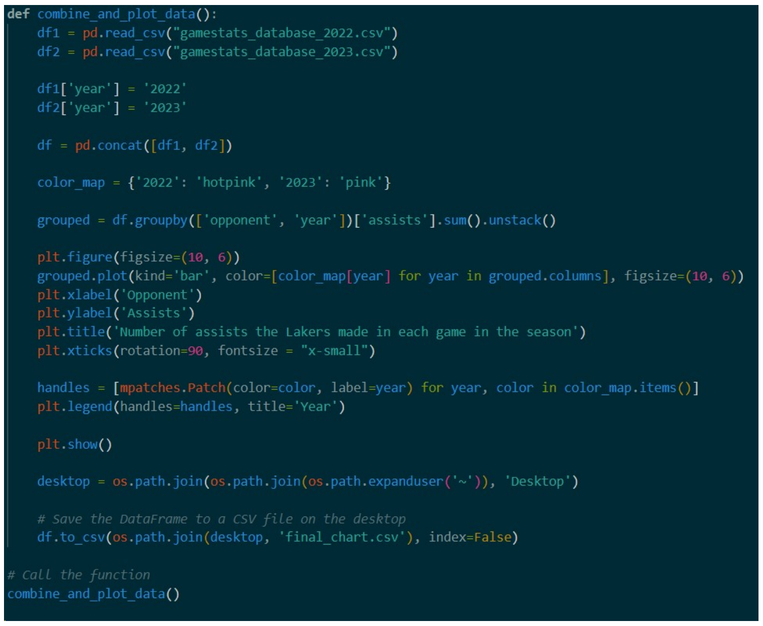
Challenges

One of the biggest challenges we faced was combining our code into one working code. Each of us created databases in a slightly different way, making it difficult to create the two way integer table when it was time. Continually, since one group member had an HP and the other two had Apple computers, some code had to be formatted differently so it would work on the new interface. Finally, ensuring the database was loading 25 entries at a time was difficult to complete, especially considering each API/website had its own information that needed to be formatted slightly differently. Ultimately, we were able to overcome every challenge we faced but this was definitely a time consuming process.

Documentations



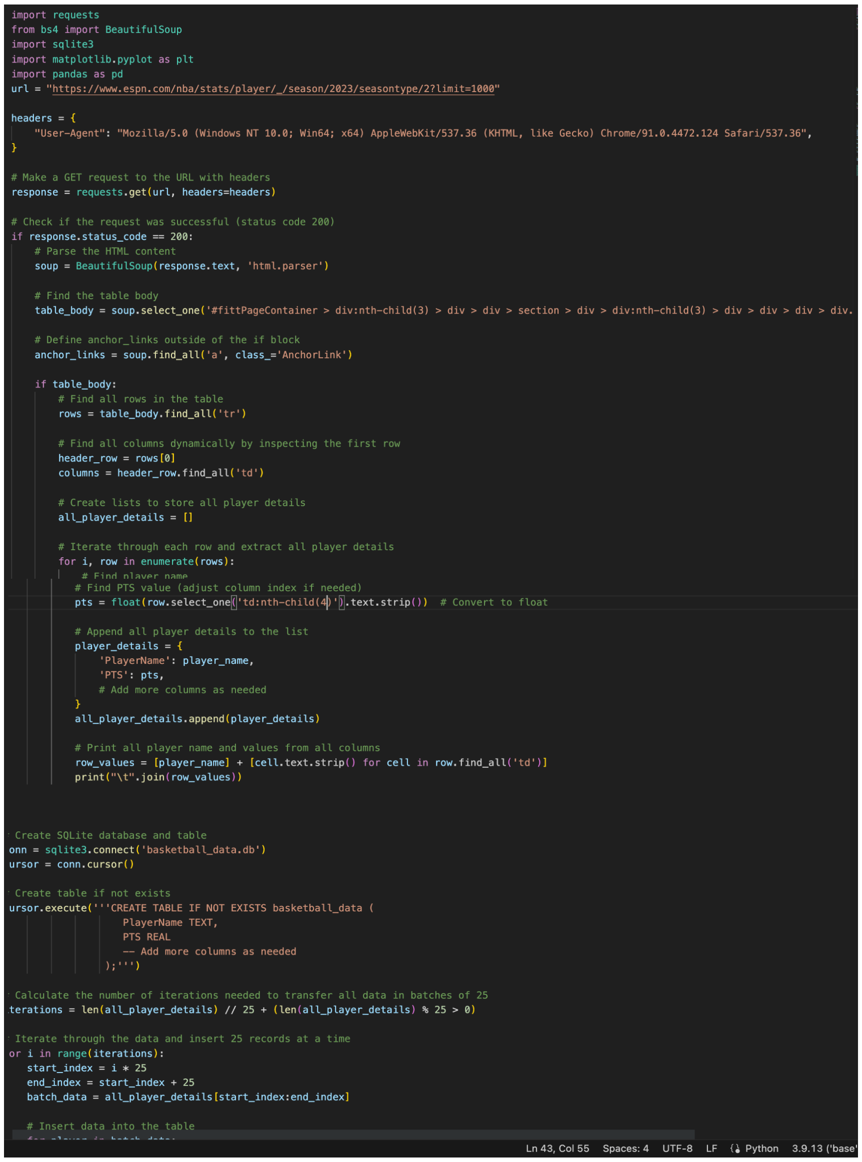
```python
def fetch_and_plot_data_1():
    url = "https://stats.nba.com/stats/teamgamelog"
    headers = {
        "User-Agent": 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
        "Referer": "https://stats.nba.com/team/1610612747/gamelog/",
        "Accept-Language": "en-US,en;q=0.9",
    }
    params = {
        "Season": "2022-23",
        "SeasonType": "Regular Season",
        "TeamID": "1610612747"  # Team ID for Los Angeles Lakers
    }
    response = requests.get(url, headers=headers, params=params)
    data = json.loads(response.text)
    team_log = data['resultSets'][0]['rowSet']

    with open("gamestats_database_2023.csv", 'w', newline='') as csvfile:
        fieldnames = ['id', 'date', 'opponent', 'result', 'assists']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for log in team_log:
            id = log[1]
            date = log[2]
            opponent = log[3][7:]
            result = log[4]
            assists = log[-6]
            writer.writerow({'id': id, 'date': date, 'opponent': opponent, 'result': result, 'assists': assists})
    df = pd.read_csv("gamestats_database_2023.csv")

    plt.figure(figsize=(10, 6))
    plt.bar(df['opponent'], df['assists'], color= 'pink')
    plt.xlabel('Opponent')
    plt.ylabel('Assists')
    plt.title('Number of assists the Lakers made in each game in the season')
    plt.xticks(rotation=90, fontsize = "small")
    plt.show()
```



```python
def combine_and_plot_data():
    df1 = pd.read_csv("gamestats_database_2022.csv")
    df2 = pd.read_csv("gamestats_database_2023.csv")

    df1['year'] = '2022'
    df2['year'] = '2023'

    df = pd.concat([df1, df2])

    color_map = {'2022': 'hotpink', '2023': 'pink'}

    grouped = df.groupby(['opponent', 'year'])['assists'].sum().unstack()

    plt.figure(figsize=(10, 6))
    grouped.plot(kind='bar', color=[color_map[year] for year in grouped.columns], figsize=(10, 6))
    plt.xlabel('Opponent')
    plt.ylabel('Assists')
    plt.title('Number of assists the Lakers made in each game in the season')
    plt.xticks(rotation=90, fontsize = "x-small")

    handles = [mpatches.Patch(color=color, label=year) for year, color in color_map.items()]
    plt.legend(handles=handles, title='Year')

    plt.show()

    desktop = os.path.join(os.path.join(os.path.expanduser('~')), 'Desktop')

    # Save the DataFrame to a CSV file on the desktop
    df.to_csv(os.path.join(desktop, 'final_chart.csv'), index=False)

# Call the function
combine_and_plot_data()
```

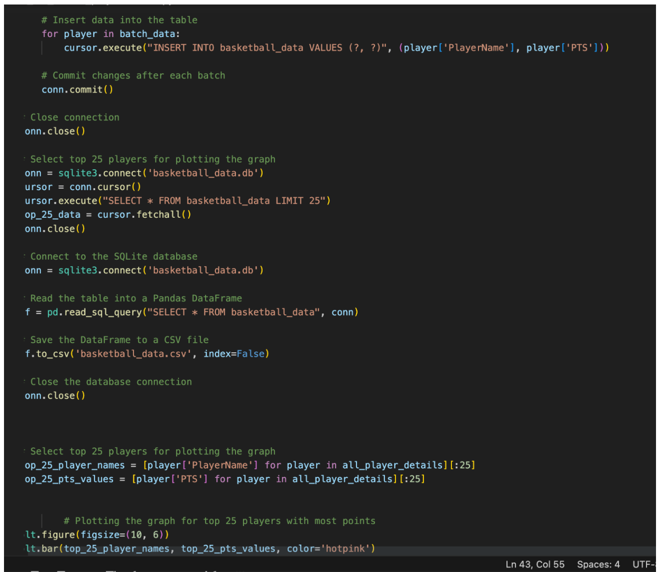
This is the code used to create Visualization IV. The function fetch_and_plot_data_1 takes no arguments and sends a request to NBA Stats API to fetch game logs for a certain team depending on the parameters given (in this case, the Lakers). Then, it stores the game logs for the team selected in the team_log variable. After, the function creates a CSV file named gamestats_database_2023.csv with the fields id, date, opponent, result, and assists. For each game log in team_log, the id, date, opponent, result, and assist are extracted and written into the CSV file. Once this happens, the CSV file is written into a Dataframe where the x-axis is the opponents and the y-axis is the # of assists. Finally, a bar plot is created from that DataFrame.

This process is repeated twice under the functions fetch_and_plot_data_1() and fetch_and_plot_data_2(). Both create a bar plot for their respective years (in this case, 2022 and 2023). The function combine_and_plot then combines the two bar plots into one graph, depicted as Visualization iv. It does this by first reading in the two existing Data Frames and concatenating them into one. The new DataFrame is grouped by opponent and year and the sum of assists is calculated for each group. The bar plot is then created from this grouped DataFrame. Finally, a legend for the bar plot is created and then the whole thing is saved to a CSV file called final_chart.csv. This is the code responsible for Visualization iv.



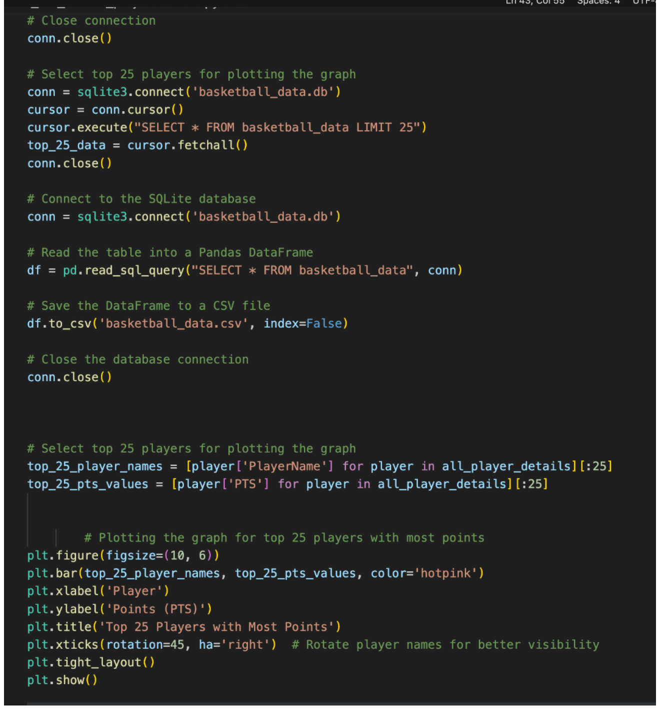
```python
import requests
from bs4 import BeautifulSoup
import sqlite3
import matplotlib.pyplot as plt
import pandas as pd
url = "https://www.espn.com/nba/stats/player/_/season/2023/seasontype/2?limit=1000"

headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
}

# Make a GET request to the URL with headers
response = requests.get(url, headers=headers)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the HTML content
    soup = BeautifulSoup(response.text, 'html.parser')

    # Find the table body
    table_body = soup.select_one('#fittPageContainer > div:nth-child(3) > div > div > section > div > div:nth-child(3) > div > div > div > div.

    # Define anchor_links outside of the if block
    anchor_links = soup.find_all('a', class_='AnchorLink')

    if table_body:
        # Find all rows in the table
        rows = table_body.find_all('tr')

        # Find all columns dynamically by inspecting the first row
        header_row = rows[0]
        columns = header_row.find_all('td')

        # Create lists to store all player details
        all_player_details = []

        # Iterate through each row and extract all player details
        for i, row in enumerate(rows):
            # Find player name
            # Find PTS value (adjust column index if needed)
            pts = float(row.select_one('td:nth-child(4)').text.strip())  # Convert to float

            # Append all player details to the list
            player_details = {
                'PlayerName': player_name,
                'PTS': pts,
                # Add more columns as needed
            }
            all_player_details.append(player_details)

            # Print all player name and values from all columns
            row_values = [player_name] + [cell.text.strip() for cell in row.find_all('td')]
            print("\t".join(row_values))



 Create SQLite database and table
onn = sqlite3.connect('basketball_data.db')
ursor = conn.cursor()

 Create table if not exists
ursor.execute('''CREATE TABLE IF NOT EXISTS basketball_data (
                    PlayerName TEXT,
                    PTS REAL
                    -- Add more columns as needed
                );''')

 Calculate the number of iterations needed to transfer all data in batches of 25
terations = len(all_player_details) // 25 + (len(all_player_details) % 25 > 0)

 Iterate through the data and insert 25 records at a time
or i in range(iterations):
   start_index = i * 25
   end_index = start_index + 25
   batch_data = all_player_details[start_index:end_index]

   # Insert data into the table
```

```
Ln 43, Col 55   Spaces: 4   UTF-8   LF   Python   3.9.13 ('base'
```



```python
    # Insert data into the table
    for player in batch_data:
        cursor.execute("INSERT INTO basketball_data VALUES (?, ?)", (player['PlayerName'], player['PTS']))

    # Commit changes after each batch
    conn.commit()

 Close connection
onn.close()

 Select top 25 players for plotting the graph
onn = sqlite3.connect('basketball_data.db')
ursor = conn.cursor()
ursor.execute("SELECT * FROM basketball_data LIMIT 25")
op_25_data = cursor.fetchall()
onn.close()

 Connect to the SQLite database
onn = sqlite3.connect('basketball_data.db')

 Read the table into a Pandas DataFrame
f = pd.read_sql_query("SELECT * FROM basketball_data", conn)

 Save the DataFrame to a CSV file
f.to_csv('basketball_data.csv', index=False)

 Close the database connection
onn.close()



 Select top 25 players for plotting the graph
op_25_player_names = [player['PlayerName'] for player in all_player_details][:25]
op_25_pts_values = [player['PTS'] for player in all_player_details][:25]


        # Plotting the graph for top 25 players with most points
lt.figure(figsize=(10, 6))
lt.bar(top_25_player_names, top_25_pts_values, color='hotpink')
```

```
Ln 43, Col 55   Spaces: 4   UTF-
```



```python
# Close connection
conn.close()

# Select top 25 players for plotting the graph
conn = sqlite3.connect('basketball_data.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM basketball_data LIMIT 25")
top_25_data = cursor.fetchall()
conn.close()

# Connect to the SQLite database
conn = sqlite3.connect('basketball_data.db')

# Read the table into a Pandas DataFrame
df = pd.read_sql_query("SELECT * FROM basketball_data", conn)

# Save the DataFrame to a CSV file
df.to_csv('basketball_data.csv', index=False)

# Close the database connection
conn.close()



# Select top 25 players for plotting the graph
top_25_player_names = [player['PlayerName'] for player in all_player_details][:25]
top_25_pts_values = [player['PTS'] for player in all_player_details][:25]


        # Plotting the graph for top 25 players with most points
plt.figure(figsize=(10, 6))
plt.bar(top_25_player_names, top_25_pts_values, color='hotpink')
plt.xlabel('Player')
plt.ylabel('Points (PTS)')
plt.title('Top 25 Players with Most Points')
plt.xticks(rotation=45, ha='right')  # Rotate player names for better visibility
plt.tight_layout()
plt.show()
```

This script is a web scraping example that extracts NBA player statistics from the ESPN website, stores the data in an SQLite database, and then generates a bar graph using matplotlib to visualize the top 25 players with the most points.

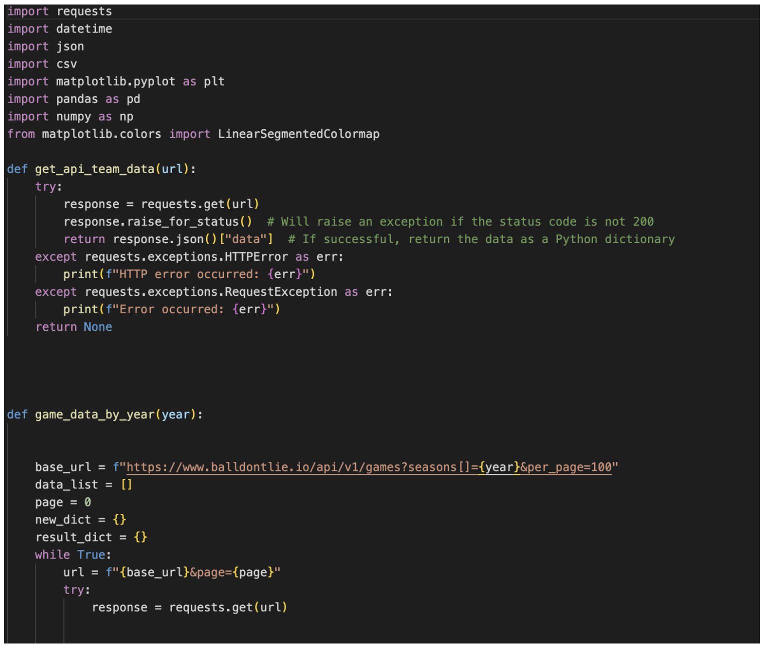
Here's an overview of how the code works:

- Web Scraping:

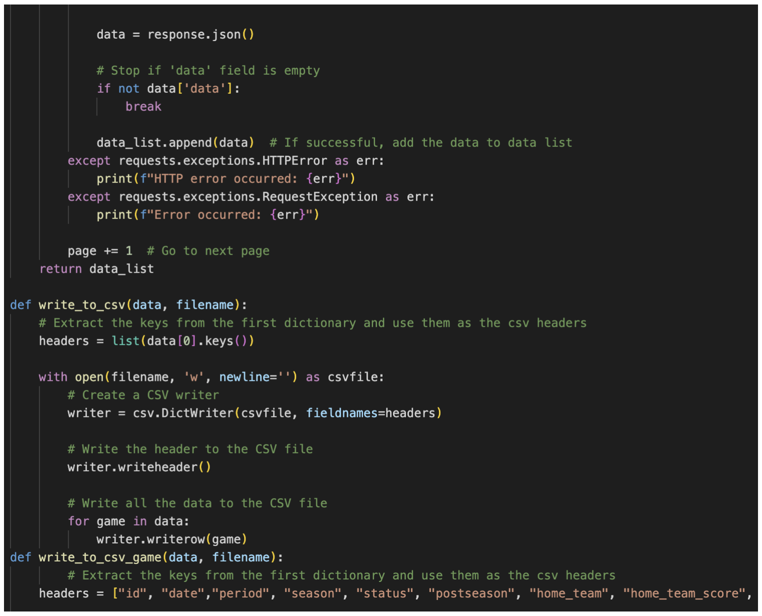
  - The script uses the requests library to make a GET request to the specified URL, simulating a web browser request.
  - The BeautifulSoup library is used to parse the HTML content of the page.
  - It identifies the table containing player statistics by selecting the appropriate HTML elements using CSS selectors.
- Data Extraction:
  - Player details, such as name and points (PTS), are extracted from the HTML table.
  - The extracted data is stored in a list of dictionaries named all_player_details.
- Database Creation and Data Insertion:
  - An SQLite database (player_database_y.db) is created, and a table (player_database_y) is defined to store player details.
  - The script then calculates the number of iterations needed to insert data in batches of 25 (to avoid SQLite's maximum variable number constraint).
  - It iterates through the data and inserts 25 records at a time into the SQLite database.
- Data Retrieval for Plotting:
  - The script connects to the SQLite database again to retrieve the top 25 players' data.
  - It uses this data to create a Pandas DataFrame and saves it to a CSV file (player_data_y.csv).
- Plotting:
  - Finally, the script uses matplotlib to create a bar graph that visualizes the top 25 players with the most points.
- Display Output:
  - The graph is displayed using plt.show().

Notes:

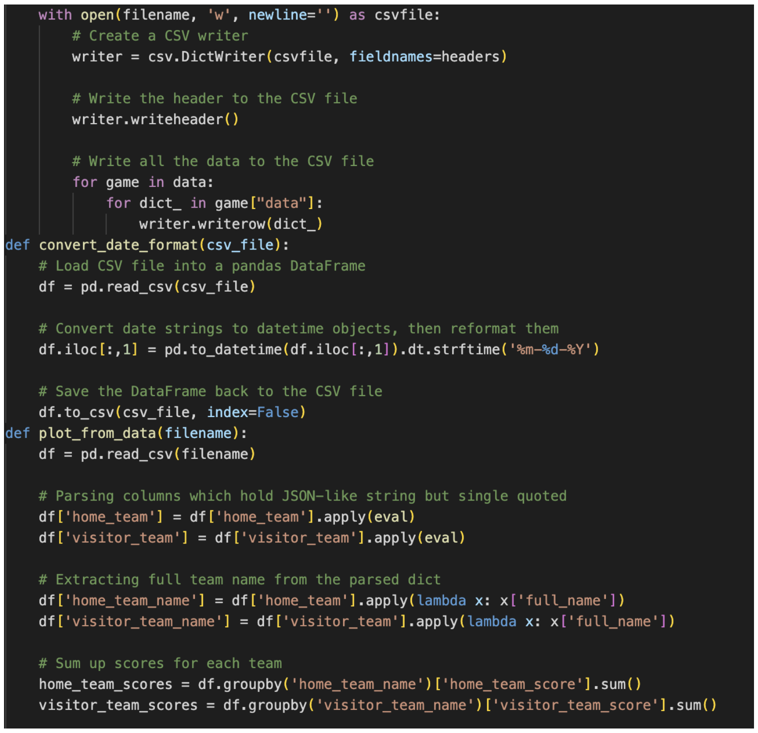
The script assumes that the structure of the HTML page and the class names used in the script remain unchanged. If the ESPN website structure changes, the script may need adjustments.The script may need to be adapted for different seasons or additional statistics. Also, if we want to grab something else from the website, another row the only we will have to do it's to go to the website count which column we want and change the number in this function: pts = float(row.select_one(‘td:nth-child(4)’).text.strip()) for the number of the column we want to print in the grab. This will be based on the top scoring players. As well we can change the graph to print as many player with the most point we want by changing the value in top_25_players_names= [player['Player_Name'] for player in all_player_details][:25] we can change that for how many player we want the chart to present in the graph.



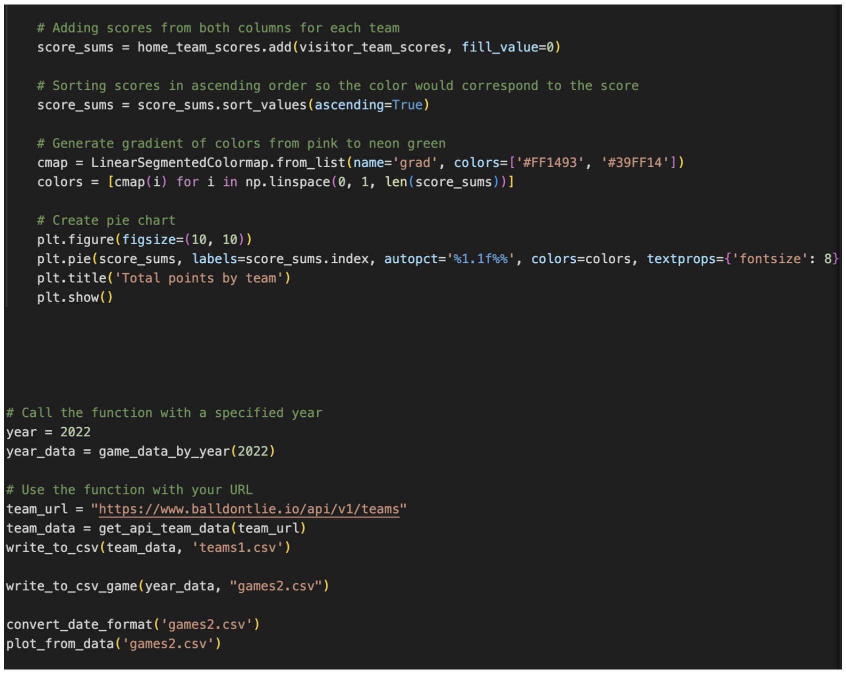
```python
import requests
import datetime
import json
import csv
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from matplotlib.colors import LinearSegmentedColormap

def get_api_team_data(url):
    try:
        response = requests.get(url)
        response.raise_for_status()  # Will raise an exception if the status code is not 200
        return response.json()["data"]  # If successful, return the data as a Python dictionary
    except requests.exceptions.HTTPError as err:
        print(f"HTTP error occurred: {err}")
    except requests.exceptions.RequestException as err:
        print(f"Error occurred: {err}")
    return None



def game_data_by_year(year):


    base_url = f"https://www.balldontlie.io/api/v1/games?seasons[]={year}&per_page=100"
    data_list = []
    page = 0
    new_dict = {}
    result_dict = {}
    while True:
        url = f"{base_url}&page={page}"
        try:
            response = requests.get(url)
```



```python
            data = response.json()

            # Stop if 'data' field is empty
            if not data['data']:
                break

            data_list.append(data)  # If successful, add the data to data list
        except requests.exceptions.HTTPError as err:
            print(f"HTTP error occurred: {err}")
        except requests.exceptions.RequestException as err:
            print(f"Error occurred: {err}")

        page += 1  # Go to next page
    return data_list

def write_to_csv(data, filename):
    # Extract the keys from the first dictionary and use them as the csv headers
    headers = list(data[0].keys())

    with open(filename, 'w', newline='') as csvfile:
        # Create a CSV writer
        writer = csv.DictWriter(csvfile, fieldnames=headers)

        # Write the header to the CSV file
        writer.writeheader()

        # Write all the data to the CSV file
        for game in data:
            writer.writerow(game)
def write_to_csv_game(data, filename):
        # Extract the keys from the first dictionary and use them as the csv headers
    headers = ["id", "date","period", "season", "status", "postseason", "home_team", "home_team_score",
```



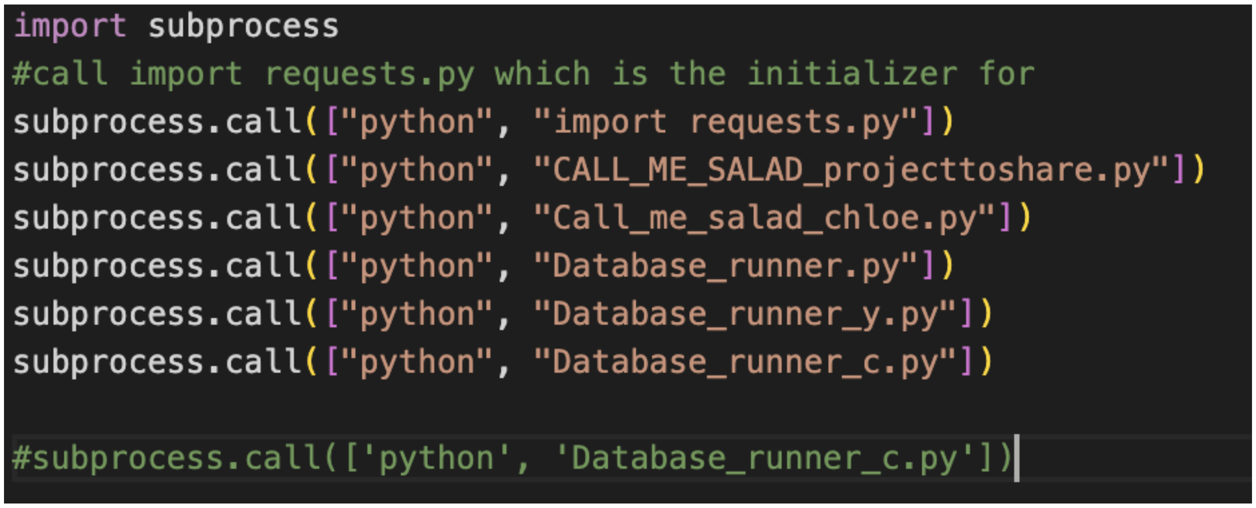
```python
    with open(filename, 'w', newline='') as csvfile:
        # Create a CSV writer
        writer = csv.DictWriter(csvfile, fieldnames=headers)

        # Write the header to the CSV file
        writer.writeheader()

        # Write all the data to the CSV file
        for game in data:
            for dict_ in game["data"]:
                writer.writerow(dict_)
def convert_date_format(csv_file):
    # Load CSV file into a pandas DataFrame
    df = pd.read_csv(csv_file)

    # Convert date strings to datetime objects, then reformat them
    df.iloc[:,1] = pd.to_datetime(df.iloc[:,1]).dt.strftime('%m-%d-%Y')

    # Save the DataFrame back to the CSV file
    df.to_csv(csv_file, index=False)
def plot_from_data(filename):
    df = pd.read_csv(filename)

    # Parsing columns which hold JSON-like string but single quoted
    df['home_team'] = df['home_team'].apply(eval)
    df['visitor_team'] = df['visitor_team'].apply(eval)

    # Extracting full team name from the parsed dict
    df['home_team_name'] = df['home_team'].apply(lambda x: x['full_name'])
    df['visitor_team_name'] = df['visitor_team'].apply(lambda x: x['full_name'])

    # Sum up scores for each team
    home_team_scores = df.groupby('home_team_name')['home_team_score'].sum()
    visitor_team_scores = df.groupby('visitor_team_name')['visitor_team_score'].sum()
```



```python
    # Adding scores from both columns for each team
    score_sums = home_team_scores.add(visitor_team_scores, fill_value=0)

    # Sorting scores in ascending order so the color would correspond to the score
    score_sums = score_sums.sort_values(ascending=True)

    # Generate gradient of colors from pink to neon green
    cmap = LinearSegmentedColormap.from_list(name='grad', colors=['#FF1493', '#39FF14'])
    colors = [cmap(i) for i in np.linspace(0, 1, len(score_sums))]

    # Create pie chart
    plt.figure(figsize=(10, 10))
    plt.pie(score_sums, labels=score_sums.index, autopct='%1.1f%%', colors=colors, textprops={'fontsize': 8}
    plt.title('Total points by team')
    plt.show()




# Call the function with a specified year
year = 2022
year_data = game_data_by_year(2022)

# Use the function with your URL
team_url = "https://www.balldontlie.io/api/v1/teams"
team_data = get_api_team_data(team_url)
write_to_csv(team_data, 'teams1.csv')

write_to_csv_game(year_data, "games2.csv")

convert_date_format('games2.csv')
plot_from_data('games2.csv')
```

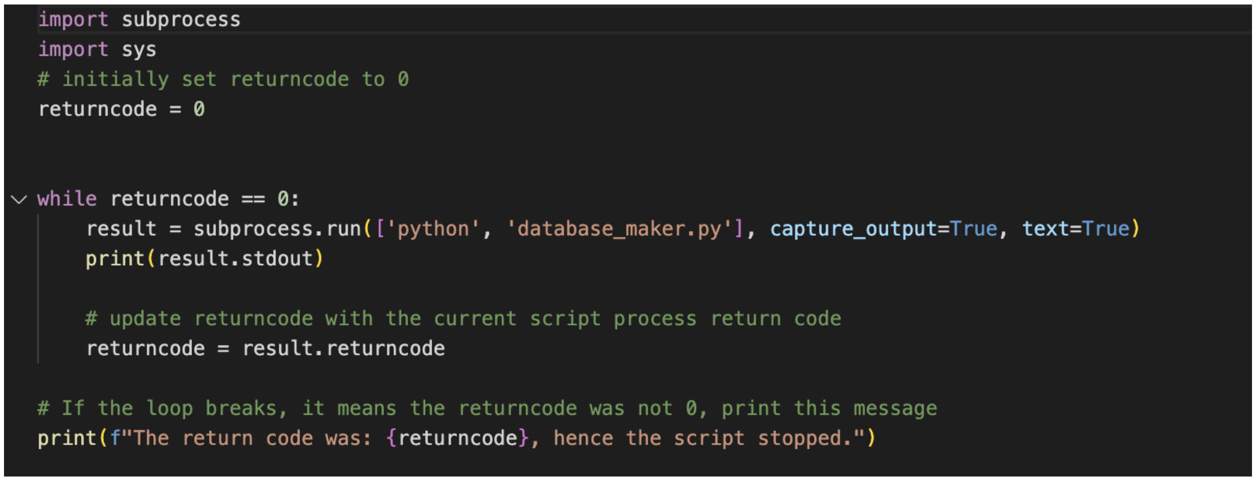
In this file, called “import requests.py” the game_data_by_year function calls the balldontlie.api when it is passed a year, the code calls the api and collects every single game that was played in the season, and collects the teams’ scores as well as whether or not the teams played at home or away and the date the games played and stores them in a list of dictionary. This code also calls the convert_data_format function which is passed that csv file that was previously collected. The function converts the date format that the balldontlie.api returns to the american standard datetime format of month-day-year. The write_to_csv function writes the team details that are returned from the get_api_team_data funcion, while the write_to_csv_game function writes the games from the game_data_by_year function to a spreadsheet. Finally this code section used the LinearSegmentationColorMap, the Matplotlib, and the numpy packages to graph the total percentage of points out of all points scored and find the correct gradient color.

That concludes the parts of this code that creates the data and graphs the majority of the data. The other part of the codebase creates the sql databases, as well as the combined database too. To achieve this uploading of data from the split databases, and have the code update the databases by 25 at a time it took a few different files (we discussed this part with our graders on our presentation day to confirm that this was a valid way to implement and get credit for this requirement.) The codebase 'main' file is the file named Runner Runner.py:



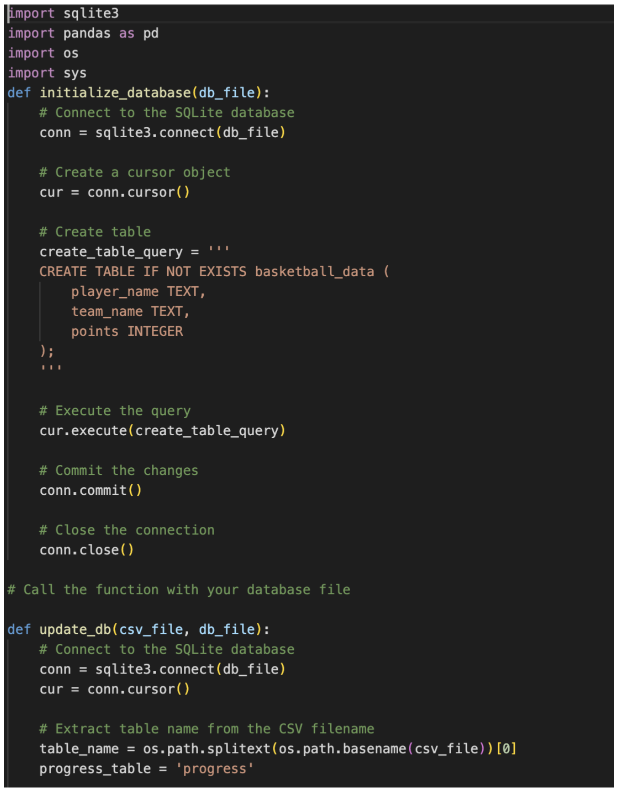
```python
import subprocess
#call import requests.py which is the initializer for
subprocess.call(["python", "import requests.py"])
subprocess.call(["python", "CALL_ME_SALAD_projecttoshare.py"])
subprocess.call(["python", "Call_me_salad_chloe.py"])
subprocess.call(["python", "Database_runner.py"])
subprocess.call(["python", "Database_runner_y.py"])
subprocess.call(["python", "Database_runner_c.py"])

#subprocess.call(['python', 'Database_runner_c.py'])
```

Runner Runner.py calls the other files in the codebase to be run, it achieves this by first running main files, import requests.py, call_me_salad_projecttoshare.py, and call_me_salad_chloe.py, which create the graphs and spreadsheets, and then it runs a runner for each database called 'database_runner.'

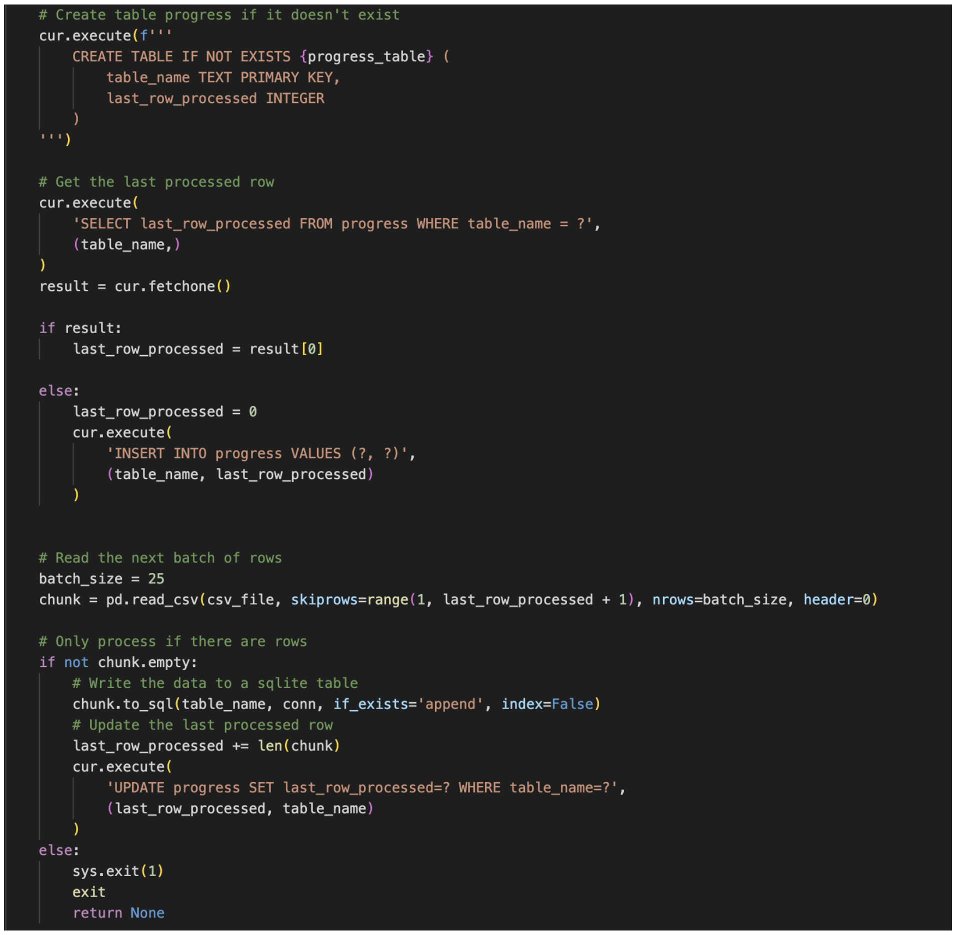


```python
import subprocess
import sys
# initially set returncode to 0
returncode = 0


while returncode == 0:
    result = subprocess.run(['python', 'database_maker.py'], capture_output=True, text=True)
    print(result.stdout)

    # update returncode with the current script process return code
    returncode = result.returncode

# If the loop breaks, it means the returncode was not 0, print this message
print(f"The return code was: {returncode}, hence the script stopped.")
```

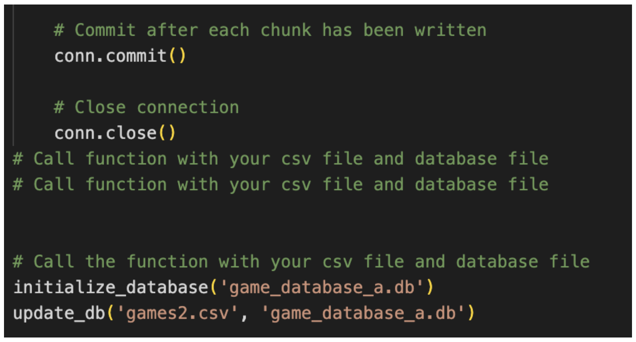
This is so that code "each time the code is run it updates 25 entries into the database." These runner files then run a while loop until their respective return code is changed from 0 to 1. In the loop the database_maker.py is called which is what actually looks at the spreadsheet and the database to see which entry needs to be updated next (the database is given a separate page to mark its last entry for easy troubleshooting.)



```python
import sqlite3
import pandas as pd
import os
import sys
def initialize_database(db_file):
    # Connect to the SQLite database
    conn = sqlite3.connect(db_file)

    # Create a cursor object
    cur = conn.cursor()

    # Create table
    create_table_query = '''
    CREATE TABLE IF NOT EXISTS basketball_data (
        player_name TEXT,
        team_name TEXT,
        points INTEGER
    );
    '''

    # Execute the query
    cur.execute(create_table_query)

    # Commit the changes
    conn.commit()

    # Close the connection
    conn.close()

# Call the function with your database file

def update_db(csv_file, db_file):
    # Connect to the SQLite database
    conn = sqlite3.connect(db_file)
    cur = conn.cursor()

    # Extract table name from the CSV filename
    table_name = os.path.splitext(os.path.basename(csv_file))[0]
    progress_table = 'progress'
```



```
# Create table progress if it doesn't exist
cur.execute(f'''
    CREATE TABLE IF NOT EXISTS {progress_table} (
        table_name TEXT PRIMARY KEY,
        last_row_processed INTEGER
    )
''')

# Get the last processed row
cur.execute(
    'SELECT last_row_processed FROM progress WHERE table_name = ?',
    (table_name,)
)
result = cur.fetchone()

if result:
    last_row_processed = result[0]

else:
    last_row_processed = 0
    cur.execute(
        'INSERT INTO progress VALUES (?, ?)',
        (table_name, last_row_processed)
    )


# Read the next batch of rows
batch_size = 25
chunk = pd.read_csv(csv_file, skiprows=range(1, last_row_processed + 1), nrows=batch_size, header=0)

# Only process if there are rows
if not chunk.empty:
    # Write the data to a sqlite table
    chunk.to_sql(table_name, conn, if_exists='append', index=False)
    # Update the last processed row
    last_row_processed += len(chunk)
    cur.execute(
        'UPDATE progress SET last_row_processed=? WHERE table_name=?',
        (last_row_processed, table_name)
    )
else:
    sys.exit(1)
    exit
    return None
```



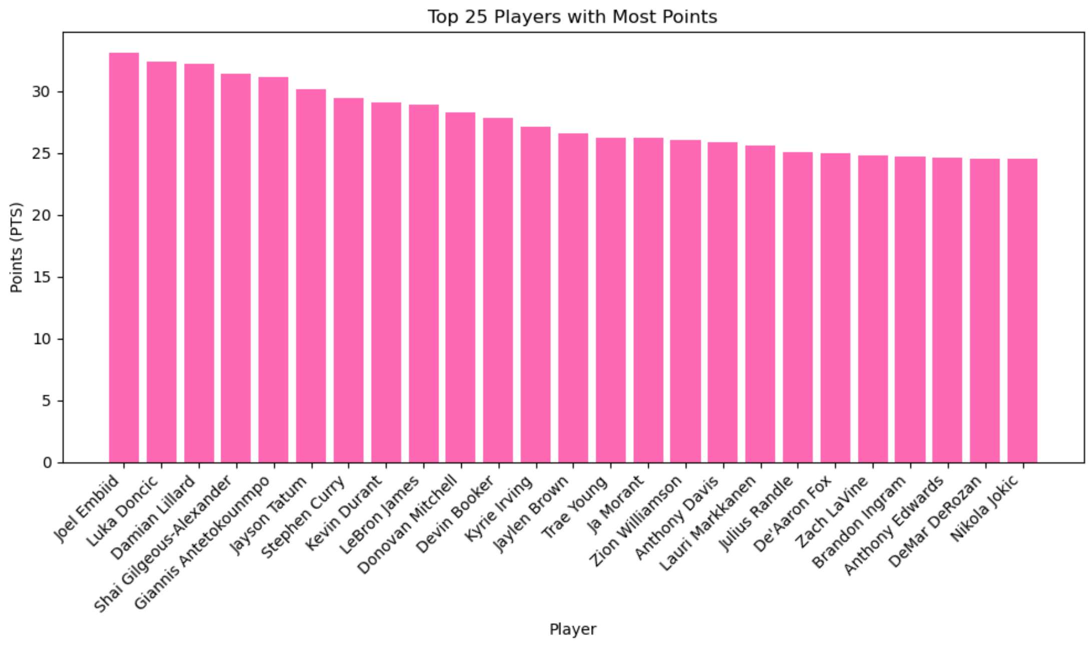
```
    # Commit after each chunk has been written
    conn.commit()

    # Close connection
    conn.close()
# Call function with your csv file and database file
# Call function with your csv file and database file


# Call the function with your csv file and database file
initialize_database('game_database_a.db')
update_db('games2.csv', 'game_database_a.db')
```

This is one of the database_makers which call the initialize database function to make a blank database with these three headers, because all of the data has these three components. Then the update_db function is passed the csv and the empty database and updates the next 25 entries until an empty chunk is found then it changes the return code and the database_runner will stop running the file, and the next subprocess will then begin.

There are two final python script called Final_CombinedDB_Graph.py, that takes the database that is combined with the Database_Linker on the key of the date which combines from the API BallDontlie.api and the FreeNBAStats API the points scored by the selected team with the number of assists scored by the selected team as well.

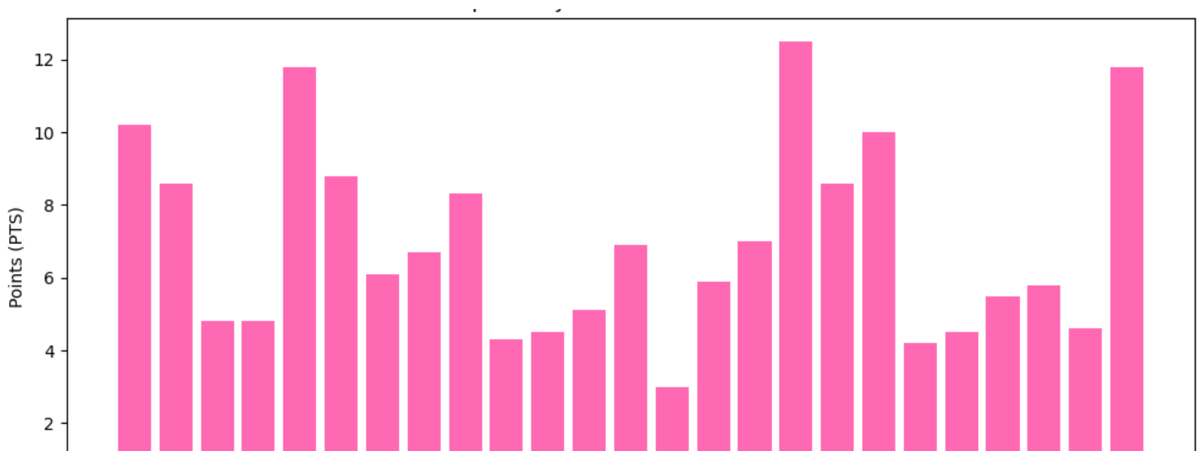
Visualizations

Top 25 Players with Most Points from 2022-23 Season.



i.Visualization Impact: This graph shows the top 25 players with most points from the 2022-23 season. The impact of such a visualization goes beyond mere data representation. It influences fan engagement, team strategies, player recognition, and various aspects of the broader sports ecosystem. Visualizations are powerful tools for conveying complex information in an accessible and meaningful way.

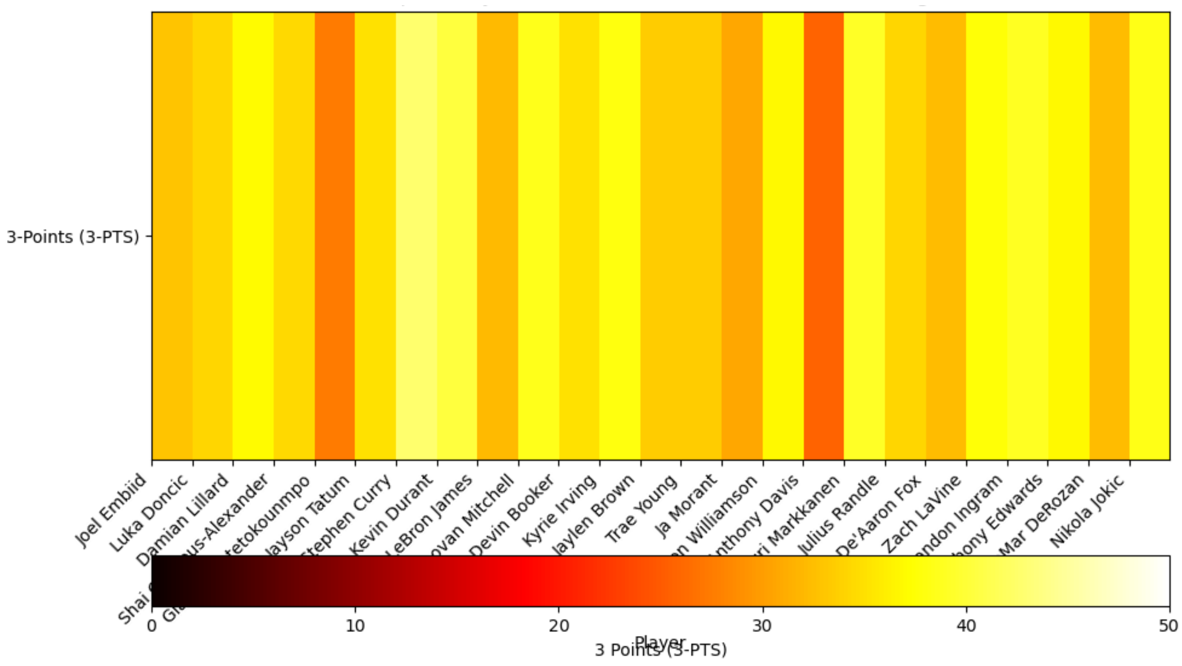
Top 25 Scoring Players' Rebounds from 2022-23 Season.



ii. Visualization Impact: Visualizing the rebounds of the top 25 scoring players enhances the depth of analysis and provides a more holistic perspective on player contributions. It influences various aspects of team strategy, fan engagement, fantasy sports, and media reporting within the basketball ecosystem.

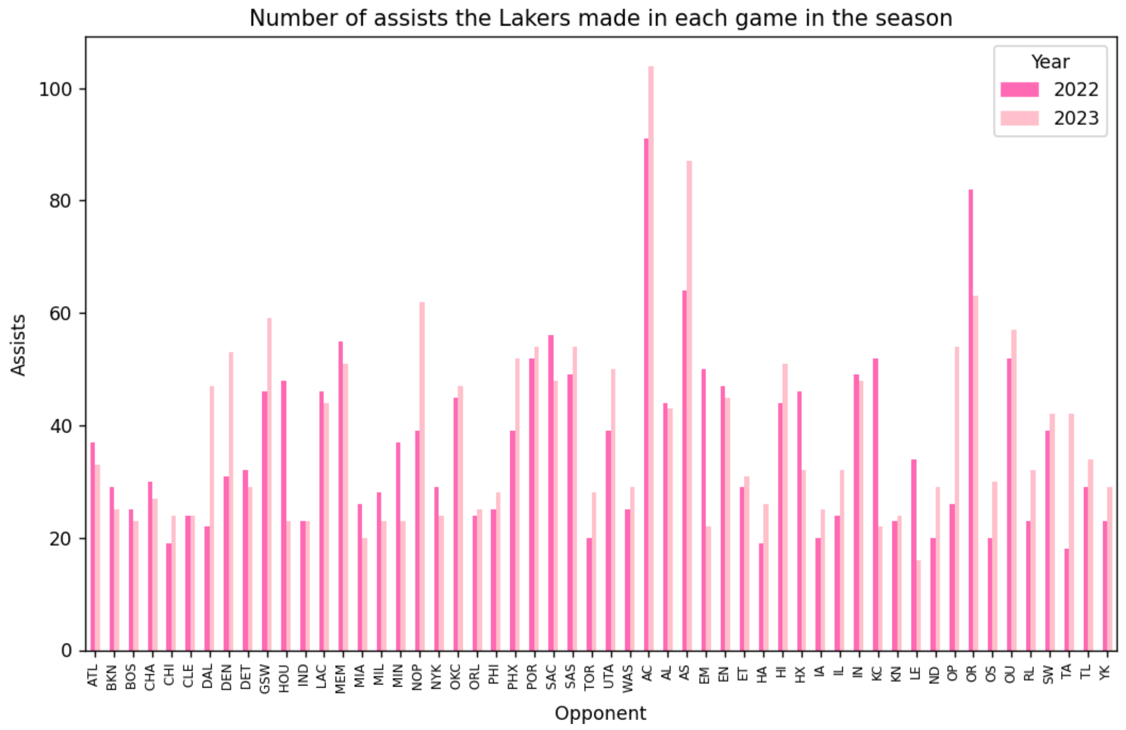
Top 25 Scoring Players' 3-Points Percentage from 2022-23 Season

(From 0-50% made)



iii. Visualization Impact: This heat map illustrating the 3-Points Percentage of the Top 25 Scoring Players from the 2022-23 Season from 0-50%, this provides a multifaceted tool for coaches, fans, analysts, and fantasy basketball enthusiasts alike. Its visual impact extends beyond numerical data, offering strategic insights and enhancing the overall appreciation of player performances.

Number of Assists the Lakers Made in Each Game in Season 2022-23

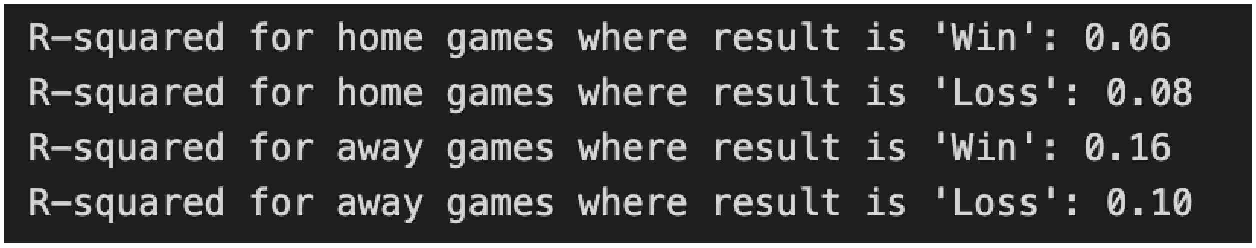


iv. Visualization Impact: This bar chart shows the amount of assists the Lakers made against each team in the regular season for both 2022 and 2023. It is interesting to note how the Lakers perform similarly against each team regardless of the year, with a few exceptions.

Calculations

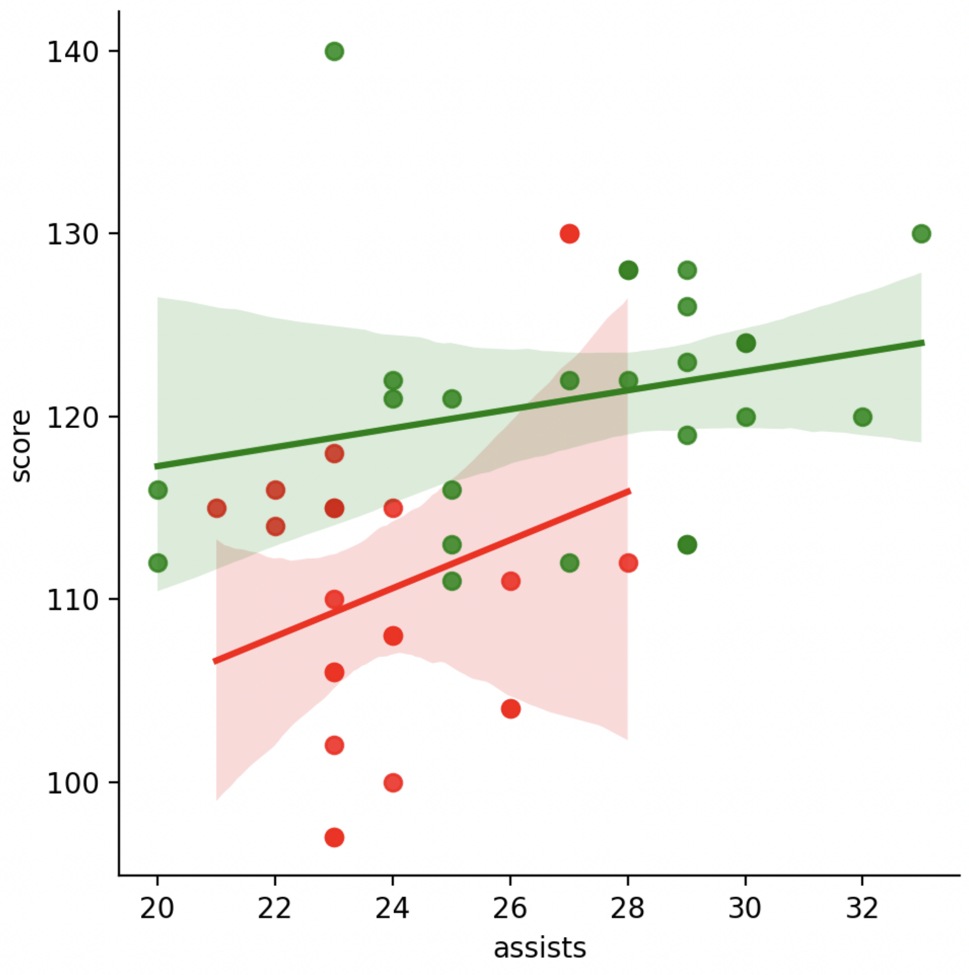
We ran several calculations with our data, such as plotting the assists and points regression, and finding every teams total percentage of points out of all points scored in a season and then graphing that on a pie chart. For the r-squared calculations we used scipy, and also used the seaborn to obtain a graph with standard error bands in the linear regression scatterplot visualizations.

In running our code with the Lakers, we found a result that showed a very weak correlation between assists and points, for home games where the team won the r-squared was only 0.06 while for games where they loss the r-squared was 0.08, while for away games there was a noticeably stronger correlation of 0.16 and when it was a loss the correlation was a little lower at 0.10.



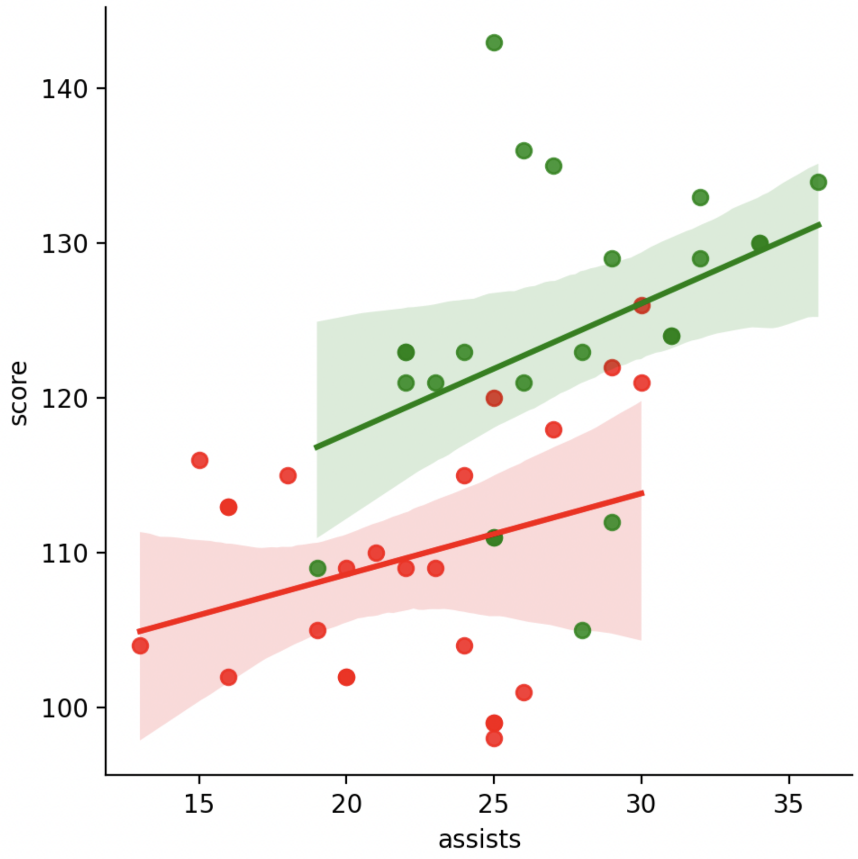
```
R-squared for home games where result is 'Win': 0.06
R-squared for home games where result is 'Loss': 0.08
R-squared for away games where result is 'Win': 0.16
R-squared for away games where result is 'Loss': 0.10
```

Total Team Assists and Total Team Points per Game, for Away Games, Sorted By Wins and Loses with Lines of Best Fit and Standard Error for the Lakers in Season 2023-24.



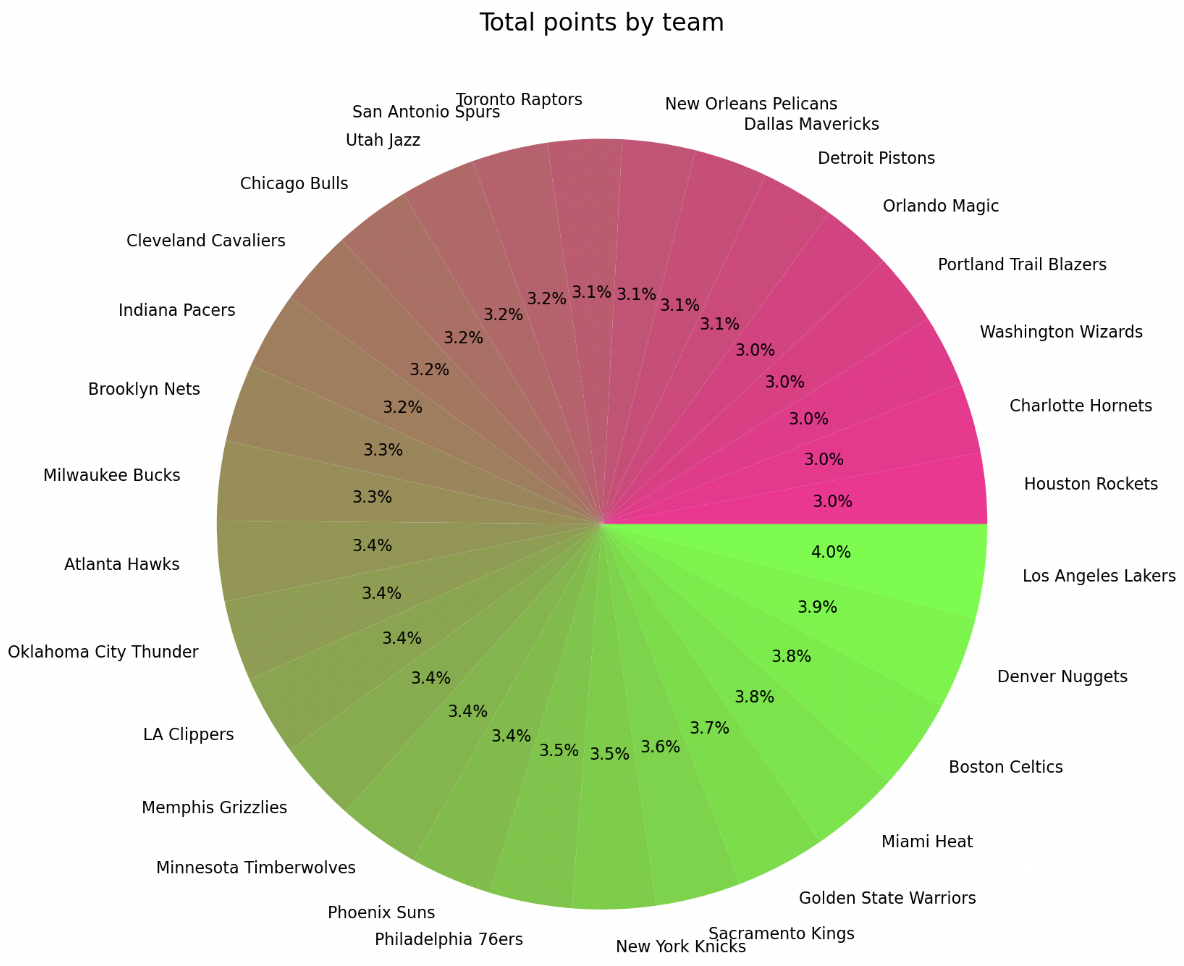
v. Visualization Impact: This graph shows the team assists against team points for away games and shows the very weak correlation that was found between team assists and team points. Also because the lines have different slopes it reveals an interesting feature of the data that the more points are scored the less assist's impact the score when the Lakers lose at away games, but for wins the more points are scored the assists aren't correlated with a higher score like they are for losses.

Total Team Assists and Total Team Points per Game, for Home Games, Sorted By Wins and Loses with Lines of Best Fit and Standard Error for the Lakers in Season 2023-24 Total .



vi. Visualization Impact: This graph shows the team assists against team points for home games and shows the very weak correlation that was found between team assists and team points. Also because the graphs have different slopes it reveals an interesting feature, contrary to away games the opposite is true for home games the correlations directions are swapped and a high number of assists impacts the score more for wins then it does for losses.

Total Points Per Team in Season 2023-24



vii. Visualization Impact: This is a table that shows what percentage of points were scored by what teams in the most recent season of NBA. From the table, we can tell that the Los Angeles Lakers are winning with 4.0% of all points scored for that season.

Conclusion

Ultimately, for our report we knew we wanted to focus on Basketball Statistics. During our grading session, we received feedback that we needed a heavier focus on the calculations and to make sure we had two tables that share an integer key. Since then, we have created Visualizations V and VI that share an integer key and show what our calculations represent visually.

We also were able to answer our research question. It is clear there is a weak correlation between points and assists. However, it was fascinating to see how the correlation at away games was stronger than at home games. This means that the number of points scored is not heavily influenced by the number of assists and that there are other factors that influence the difference in points. Overall, although this project was very time consuming, it was super interesting to use real world applications with our code. Also, being able to see our result reflected into a graph was very satisfying.

Resources

As a group we presented on December 8th, by the time we did not have our project finished, thus the IA's guided us with how we should proceed for the next steps missing in our project. Also, I went back to the slides provided in class for clarification in the Beautiful Soup topic to be able to do web scraping and also the HTML slides for better understanding in order to be able to make the code part for the website.

For the function fetch_and_plot_data1(), I used ChatGpt on December 10 to help me code the headers and subheaders (user-agent, referrers, and accept-language). I kept on getting odd request errors, and the addition of the header stopped these errors from continuing. It definitely helped me get my code to run properly.