Brandon Monge & Paul Chafetz
SI 206 Final Report

Github Repository: https://github.com/pjchafetz/206_project

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

For the financial-related portion of the project, we planned on using AlphaVantage, a popular API for collecting and analyzing market data. With AlphaVantage, we planned on collecting data from the S&P 500 and comparing it to data with Bitcoin such as the open price and close price on various days across 2020 and 2021; this allows the calculation of the return on investment of time periods and average price.

For the basketball portion, we planned to use Ball Don't Lie, a straightforward API for NBA-related data. We planned to collect performance data for a couple of prominent basketball stars. This would include team-specific plays, points per game or across games, and total performance over seasons. This would allow us to calculate their scoring potential and game impact.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

For the financial-related portion of the project, we did complete the goal on collecting both Bitcoin and S&P 500 data. However, we decided to use two different APIs for this portion, one for S&P 500 and another for Bitcoin. We used the Alpha Vantage API to collect S&P 500 data across 2020 and 2021 such as the dates and prices associated with those dates. For Bitcoin, we decided to use the CoinGecko API to gather similar data, such as the closing prices on various similar dates across 2020 and 2021.

For the basketball portion, the goal was changed slightly. While we did focus on particular basketball players, the actual data collected was narrowed down. In particular, we indexed Michael Jordan's, LeBron James's, and Kobe Bryant's free throw, field goal, and 3-point field goal percentages over a subset of their NBA career. This data allowed us to more easily visualize the players' accuracy and shooting skills as a measure of their performance.

3. The problems that you faced (10 points)

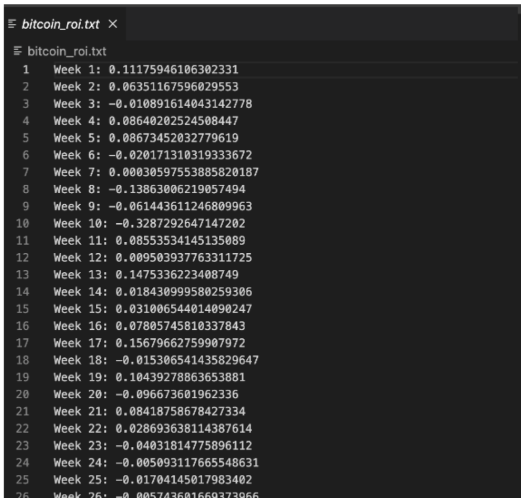
During the financial portion of the project, we faced challenges such as dealing with API rate limits along with handling missing or inconsistent data across time periods. Additionally, we had to ensure that the data collection process for both the S&P 500 and Bitcoin was synchronized and comparable across different time periods.

During the basketball portion of the project, the biggest challenge faced was understanding the API documentation. While the multiple API endpoints are documented, the returned JSON keys are not. When querying by page, it was not explained how the data was ordered. This meant that you could receive two adjacent data points from 1980 and 2022 with no indication of how or why. It took a lot of trial and error to uncover the undocumented behavior of the API.

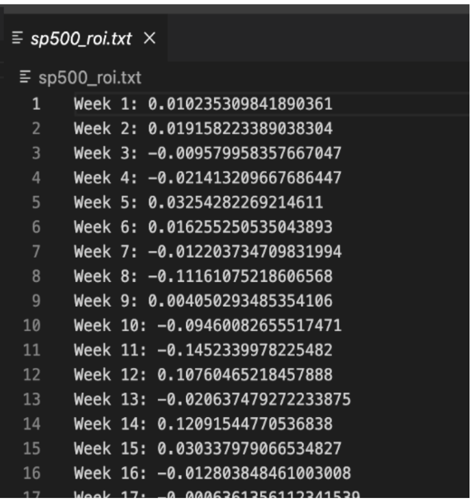
4. The calculations from the data in the database (i.e. a screen shot) (10 points)

Financial portion:

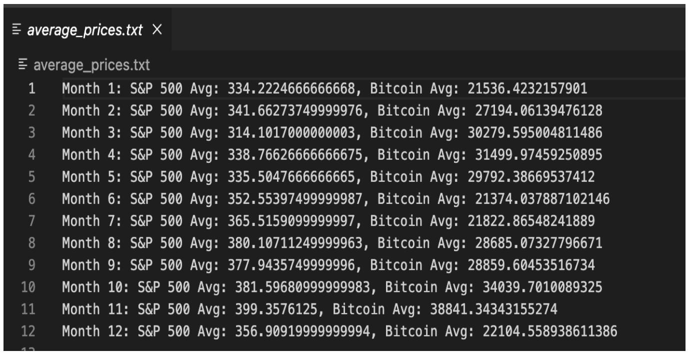
(key code provided plus snippets of the calculations file, one text file on the GitHub repo)



```
Week 1: 0.11175946106302331
Week 2: 0.06351167596029553
Week 3: -0.010891614043142778
Week 4: 0.08640202524508447
Week 5: 0.08673452032779619
Week 6: -0.020171310319333672
Week 7: 0.00030597553885820187
Week 8: -0.13863006219057494
Week 9: -0.061443611246809963
Week 10: -0.3287292647147202
Week 11: 0.08553534145135089
Week 12: 0.009503937763311725
Week 13: 0.1475336223408749
Week 14: 0.018430999580259306
Week 15: 0.031006544014090247
Week 16: 0.07805745810337843
Week 17: 0.15679662759907972
Week 18: -0.015306541435829647
Week 19: 0.10439278863653881
Week 20: -0.096673601962336
Week 21: 0.08418758678427334
Week 22: 0.028693638114387614
Week 23: -0.04031814775896112
Week 24: -0.005093117665548631
Week 25: -0.01704145017983402
Week 26: -0.005743601669373966
```

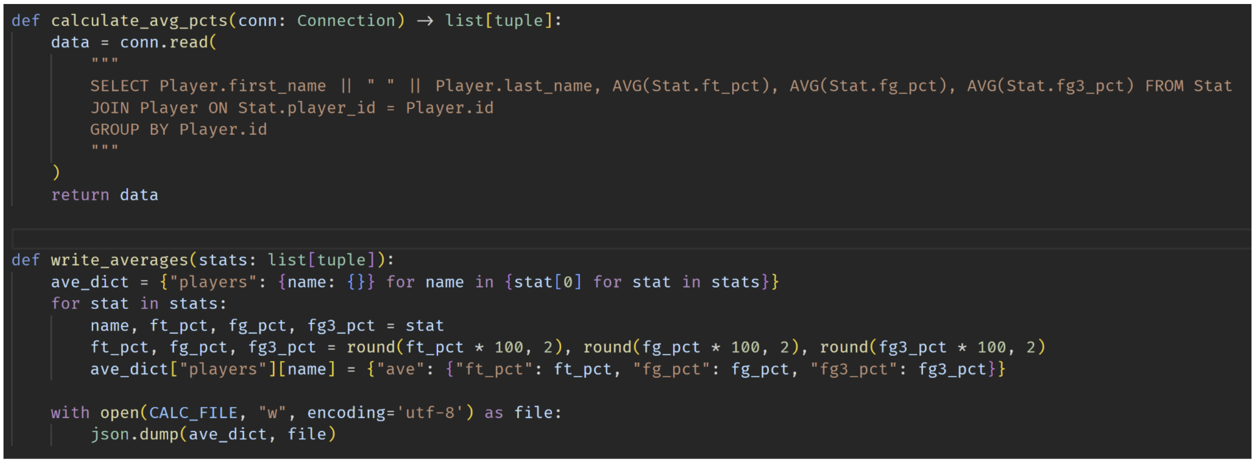


```
Week 1: 0.010235309841890361
Week 2: 0.019158223389038304
Week 3: -0.009579958357667047
Week 4: -0.021413209667686447
Week 5: 0.03254282269214611
Week 6: 0.016255250535043893
Week 7: -0.012203734709831994
Week 8: -0.11161075218606568
Week 9: 0.004050293485354106
Week 10: -0.09460082655517471
Week 11: -0.1452339978225482
Week 12: 0.10760465218457888
Week 13: -0.020637479272233875
Week 14: 0.12091544770536838
Week 15: 0.030337979066534827
Week 16: -0.012803848461003008
Week 17: -0.0006361356112341539
```

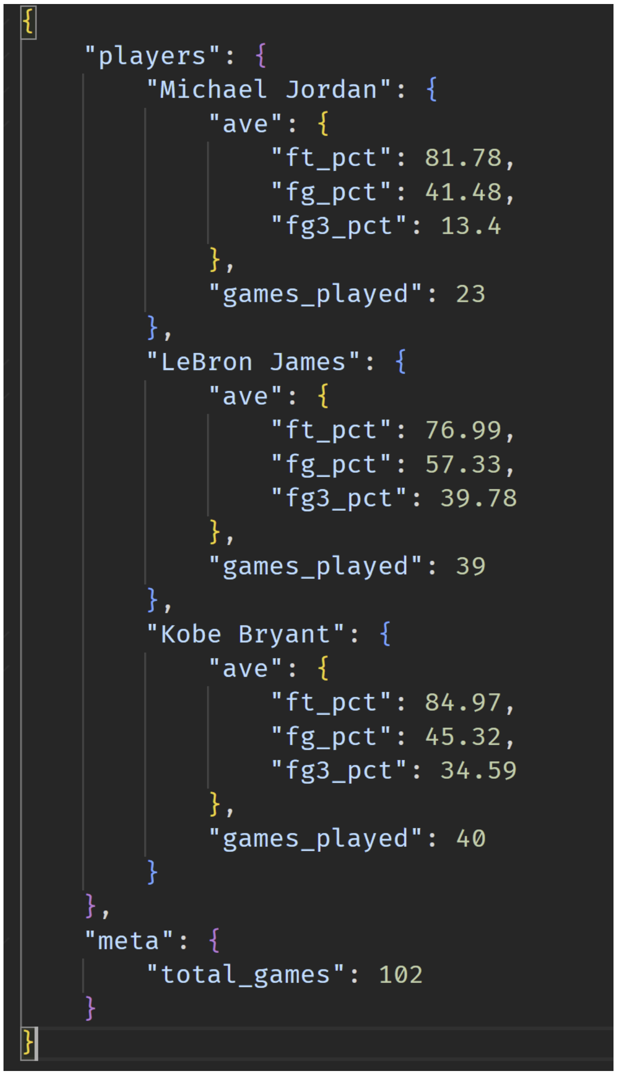


```
Month 1: S&P 500 Avg: 334.2224666666668, Bitcoin Avg: 21536.4232157901
Month 2: S&P 500 Avg: 341.66273749999976, Bitcoin Avg: 27194.06139476128
Month 3: S&P 500 Avg: 314.1017000000003, Bitcoin Avg: 30279.595004811486
Month 4: S&P 500 Avg: 338.76626666666675, Bitcoin Avg: 31499.97459250895
Month 5: S&P 500 Avg: 335.5047666666665, Bitcoin Avg: 29792.38669537412
Month 6: S&P 500 Avg: 352.55397499999987, Bitcoin Avg: 21374.037887102146
Month 7: S&P 500 Avg: 365.5159099999997, Bitcoin Avg: 21822.86548241889
Month 8: S&P 500 Avg: 380.10711249999963, Bitcoin Avg: 28685.07327796671
Month 9: S&P 500 Avg: 377.9435749999996, Bitcoin Avg: 28859.60453516734
Month 10: S&P 500 Avg: 381.59680999999983, Bitcoin Avg: 34039.7010089325
Month 11: S&P 500 Avg: 399.3576125, Bitcoin Avg: 38841.34343155274
Month 12: S&P 500 Avg: 356.90919999999994, Bitcoin Avg: 22104.558938611386
```

Basketball portion:



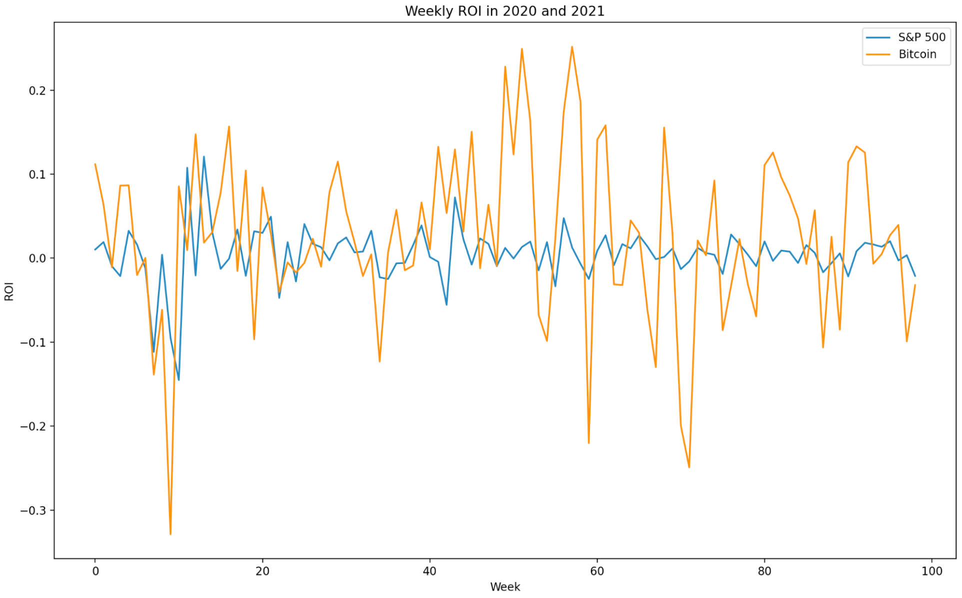
```python
def calculate_avg_pcts(conn: Connection) -> list[tuple]:
    data = conn.read(
        """
        SELECT Player.first_name || " " || Player.last_name, AVG(Stat.ft_pct), AVG(Stat.fg_pct), AVG(Stat.fg3_pct) FROM Stat
        JOIN Player ON Stat.player_id = Player.id
        GROUP BY Player.id
        """
    )
    return data


def write_averages(stats: list[tuple]):
    ave_dict = {"players": {name: {}} for name in {stat[0] for stat in stats}}
    for stat in stats:
        name, ft_pct, fg_pct, fg3_pct = stat
        ft_pct, fg_pct, fg3_pct = round(ft_pct * 100, 2), round(fg_pct * 100, 2), round(fg3_pct * 100, 2)
        ave_dict["players"][name] = {"ave": {"ft_pct": ft_pct, "fg_pct": fg_pct, "fg3_pct": fg3_pct}}

    with open(CALC_FILE, "w", encoding='utf-8') as file:
        json.dump(ave_dict, file)
```

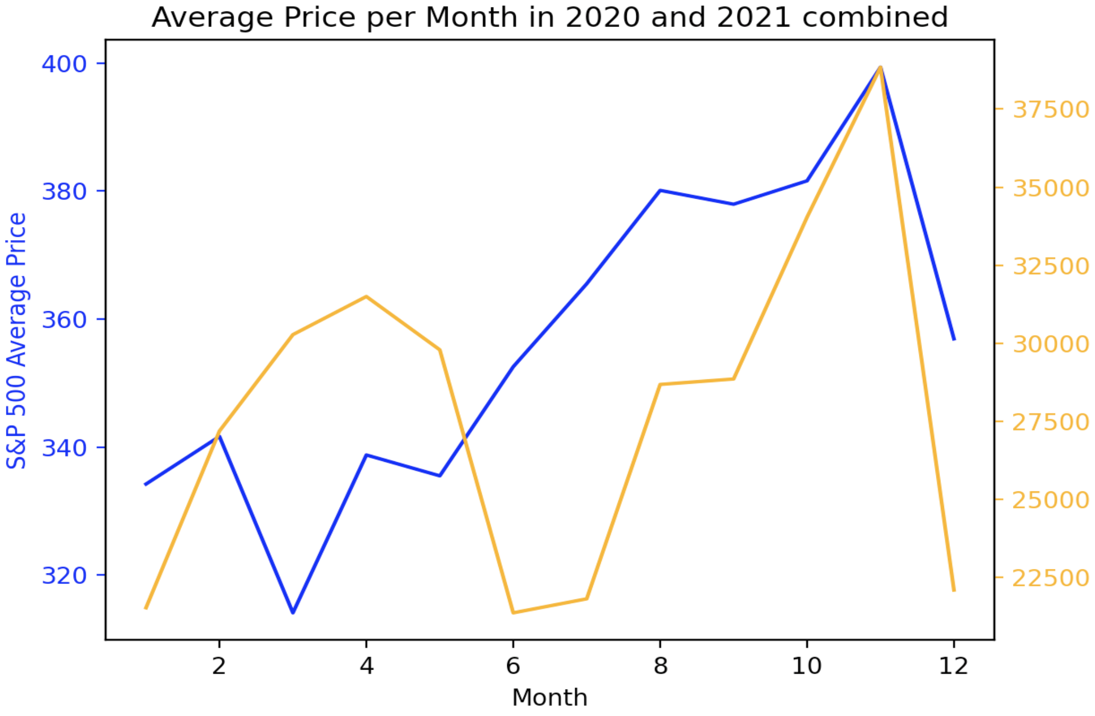


```json
{
    "players": {
        "Michael Jordan": {
            "ave": {
                "ft_pct": 81.78,
                "fg_pct": 41.48,
                "fg3_pct": 13.4
            },
            "games_played": 23
        },
        "LeBron James": {
            "ave": {
                "ft_pct": 76.99,
                "fg_pct": 57.33,
                "fg3_pct": 39.78
            },
            "games_played": 39
        },
        "Kobe Bryant": {
            "ave": {
                "ft_pct": 84.97,
                "fg_pct": 45.32,
                "fg3_pct": 34.59
            },
            "games_played": 40
        }
    },
    "meta": {
        "total_games": 102
    }
}
```

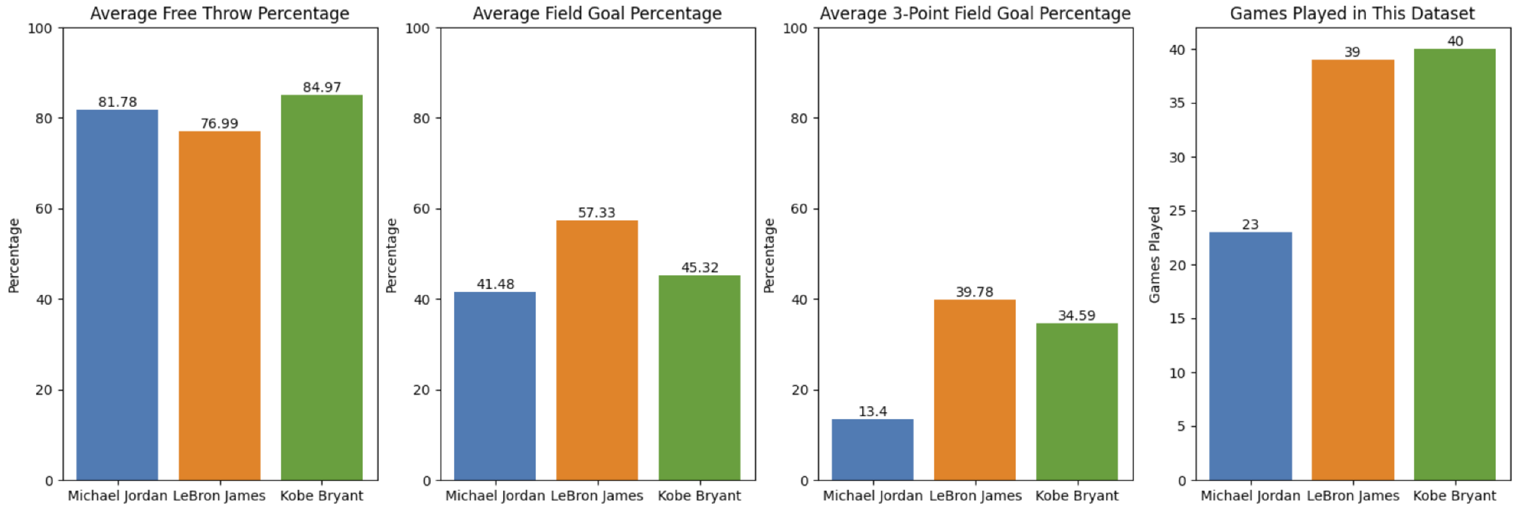
5. The visualization that you created (i.e. screen shot or image file) (10 points)

Financial portion:





Basketball portion:



6. Instructions for running your code (10 points)

Please refer to our README.md file in our GitHub repository.

After downloading a copy of the GitHub codebase, set up your development environment

1. Create a Python virtual environment (**python -m venv env && source env/bin/activate**)
2. Install the required package (**pip install -r requirements.txt**)

You can either run the files directly with **python**, or use the Makefile.

- Stocks
  - **stocks.py** - the API fetching program
  - **stocks_visualization.py** - the data calculation and visualization script for **stocks.py**
  - **stocks_data.txt** - the calculated stocks data in text format
- Basketball
  - **balldontlie.py** - the API fetching program
  - **balldontlie_visualization.py** - the data calculation and visualization script for **balldontlie.py**
  - **balldontlie_data.json** - the calculated NBA data

The Makefile supports **make** commands to run the program.
See the README or Makefile for more information.

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

Financial Portion:

stocks.py:

DataFetcher.__init__:

Input: sp500_symbol, bitcoin_symbol

Output: None

Initializes the class with the given symbols for the API calls.

DataFetcher.main:

Input: None.

Output: None.

Executes the main workflow for fetching and storing data. This includes creating SQL Tables, fetching data, and printing summary information

DataFetcher.create_table:

Input: table_name, data_type.

Output: None.

Creates a table in the database based on the given table_name and data_type.

DataFetcher.clean_data:

Input: data, data_type.

Output: Returns cleaned data (i.e. column names) based on the given data and data_type.

DataFetcher.get_bitcoin_weekly_data:

Input: start_date.

Output: Returns a Pandas DataFrame with weekly Bitcoin data starting from the given start_date.

DataFetcher.fetch_and_clean_data:

Input: symbol, data_type, start_date, end_date.

Output: Fetches and returns the cleaned data from the Alpha Vantage or CoinGecko APIs based on the given inputs.

DataFetcher.fetch_and_clean_data_with_retry:

Input: symbol, data_type, start_date, end_date.

Output: Fetches and cleans data with 3 retries in case of failure.

store_data:
Input: data, table_name, data_type.
Output: None.
Stores the given data in the database under the specified table_name and data_type.

store_and_update:
Input: symbol, data_type, table_name.
Output: The total data points collected, the start_date, and the end_date.
Fetches, cleans, and stores the data into the table based on the symbol and data_type.

get_latest_date:
Input: table_name.
Output: Returns the latest date in the given table_name.

get_data_points_count:
Input: table_name.
Output: Returns the total number of data points in the given table_name.

print_summary:
Input: sp500_data_points, bitcoin_data_points, sp500_start, sp500_end, bitcoin_start, bitcoin_end.
Output: None.
Prints a summary of the data points fetched, stored, and remaining.

main:
Input: None.
Output: None.
Entry point of file - creates a DataFetcher class that collects the data

stocks_visuals.py:
execute_query:
Input: query.
Output: Executes the given SQL query and returns the results as a list of tuples.

calculate_roi:
Input: prices.
Output: Returns a list of ROI (return on investment) for the given prices.

output_to_file:
Input: data, file_name.
Output: None.
Writes the given data to a file with the specified file_name.

output_average_prices_to_file:
Input: data, file_name.
Output: None.
Writes the average prices data to a file with the specified file_name.

plot_roi:
Input: sp500_roi, bitcoin_roi.
Output: None.
Plots the weekly ROI for S&P 500 and Bitcoin (2020 and 2021).

plot_average_prices:
Input: data.
Output: None.
Plots the average prices per month for S&P 500 and Bitcoin (2020 and 2021).

main:
Input: None.
Output: Executes the main workflow for fetching, calculating, and visualizing the data.

Basketball Portion:

utils.py:
Connection.__init__:
Input: filename of the database.
Output: None.
Creates a sqlite3 connection to the given filename.

Connection.__del__:
Input: None.
Output: None.
Destructor of the sqlite3 connection.

Connection.__str__:
Input: None.
Output: Returns the customized string representation of the Connection.

Connection.read:
Input: query, inputs.
Output: list of fetched data from the DB given the query and inputs.

Connection.write:
Input: query, inputs.
Output: None.
Executes a query onto the DB connection and commits any data.

fetch_json:
Input: url, params.
Output: JSON dictionary data from a request made to the url with params.

balldontlie.py
create_tables:
Input: Connection.
Output: None.
Creates the four tables necessary for the database.

get_and_update_page:
Input: Connection, player_id, per_page.
Output: the correct page to use in the API request.
Reads the last-read page for a player from the database increments that page.

get_and_store_player_by_name:
Input: Connection, name.
Output: the player's ID.
Queries the API for a specific player's name, storing the resulting API entry in the DB and returning it.

store_stat:
Input: Connection, statistical data.
Output: None.
Inserts the statistical data into the Stat table of the DB.

store_team:
Input: Connection, team data.
Output: None.
Inserts the team data into the Team table of the DB.

store_game:
Input: Connection, game data.
Output: None.
Inserts the game data into the Game table of the DB.

store_all_stats_for_player:
Input: Connection, per_page, player_id.
Output: count of data stored.
Fetches the API for data of the given player and stores it into the DB.

main:
Input: None.
Output: None.
Main workflow that prepares the DB, collects the data, and calculates the information.

balldontlie_visualization.py:
calculate_avg_pcts:
Input: Connection.
Output: Average free throw %, field goal %, and 3-point field goal % for each player.

write_averages:
Input: statistical data.
Output: None.
Writes the calculated statistical averages into a JSON file.

calculate_games_played:
Input: Connection.
Output: Number of games played by each player in the DB dataset.

write_games_played:
Input: statistical data.
Output: None.
Writes the total number of games and each players' game count into the JSON file.

plot:
Input: data.
Output: None.
Creates four bar graph visualizations of the calculated input data.

main:
Input: None.
Output: None.
Opens the calculated data file and creates the visualizations.

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result
April 9, 2023	Working with dates and times	datetime module documentation: https://docs.python.org/3/library/datetime.html	Successfully manipulated and formatted dates
April 9, 2023	Database operations with SQLite	sqlite3 module documentation: https://docs.python.org/3/library/sqlite3.html	Successfully created and managed SQLite database
April 9, 2023	Introducing delay between API calls	time module documentation: https://docs.python.org/3/library/time.html	Successfully added delays to prevent API rate limiting
April 9, 2023	Data manipulation using pandas	pandas documentation: https://pandas.pydata.org/pandas-docs/stable/index.html	Successfully cleaned and processed data using pandas

April 9, 2023	Fetching data from APIs	requests documentation: https://docs.python-requests.org/en/latest/	Successfully get requests working in the virtual environment
April 9, 2023	Fetching Stock data	Alpha Vantage API documentation: https://www.alphavantage.co/documentation/	Successfully fetched data from Alpha Vantage API
April 12, 2023	Fetching data from API	Ball Don’t Lie API documentation: https://www.balldontlie.io/home.html#introduction	Successfully fetched Ball Don’t Lie API data
April 13, 2023	Fetching Bitcoin data	CoinGecko API (pycoingecko) documentation: https://github.com/man-c/pycoingecko	Successfully fetched Bitcoin data from CoinGecko API
April 13, 2023	Plotting data with matplotlib	matplotlib documentation: https://matplotlib.org/stable/contents.html	Successfully created and displayed plots using matplotlib