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

The goals of this project were to compare average playtime to average pro player playtime and other player engagement metrics between different MOBA (Multiplayer Online Battle Arena) games, with League of Legends and DOTA 2 providing two specific case studies to compare between.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you planned to gather

The goals that we managed to achieve with this project was being able to collect data from multiple API's, about multiple MOBAs, and we were able to highlight the data we wanted to look at (game time, kills, deaths, player info, etc), and compile it into a database and graph it, so we were able to highlight and compare trends relating to the attributes we were interested in.

The APIs we worked with were the RAWG Video Games Database, the official League of Legends (LOL) competitive data API for the Asia server, and the OpenDota API. The data we gathered included game ID, game name, and average playtime from the MOBA tag category in the RAWG API; match data such as game duration from specific competitive matches in the Asia server of League of Legends from the LOL API; and for the OpenDota API, match duration, kills, deaths, assists, wins, and losses for Pro Players was recorded, as well as math durations for Pro and Public Matches.

3. The problems that you faced

One of the problems we faced was the games_database db not taking in any data from the RAWG API even though everything else in the code was functioning correctly. This turned out to be a simple capitalization problem, as the tag should've been "moba" and not "MOBA". Another problem we faced was figuring out how to make all these different APIs make sense together, as they all collected and formatted their data differently. We ended up settling on using playtime measured in hours as a common metric between the different APIs, as they all measured playtime in one way or another whether through game durations or through user-reported average playtimes.

Another issue encountered was working around the free daily rate limits imposed by free api programs, as the 'free tier' for some website offered a small amount of requests per day, and so for the instances where I had to call a lot of data at a time to parse through, to work around this limit I had to incorporate .sleep into the code (time.sleep), adding a delay between api requests.

And another issue was trying to work all together consistently, as we were all busy at various times with other class work and exams, and as a result it became difficult at times to be updated and to update the whole group, and it led to delays in getting work done together.

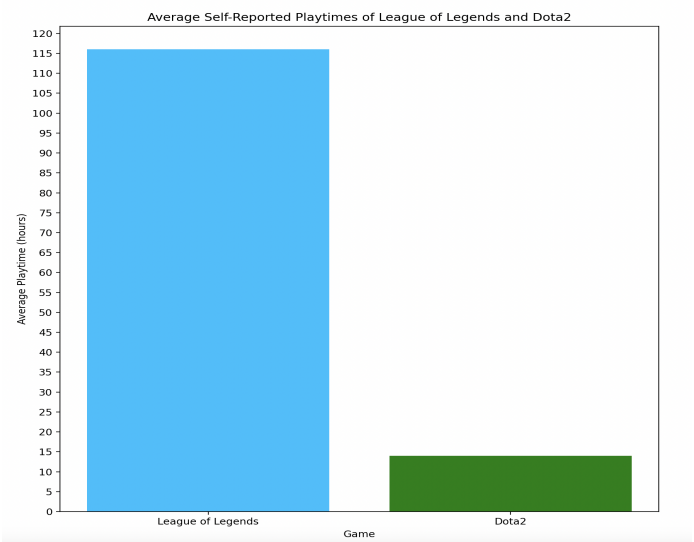
One other issue encountered was when generating graphs using Matplotlib for the Dota2 data, as all the histograms generated were all part of a subplot, and when graphed, showed a messy overlapping of titles and axis labels getting cut off by the webpage. To fix this, instead of making them all part of a subplot, each histogram was individually as a figure, so it got its own page so all the axis labels and titles can be seen clearly for each graph now.

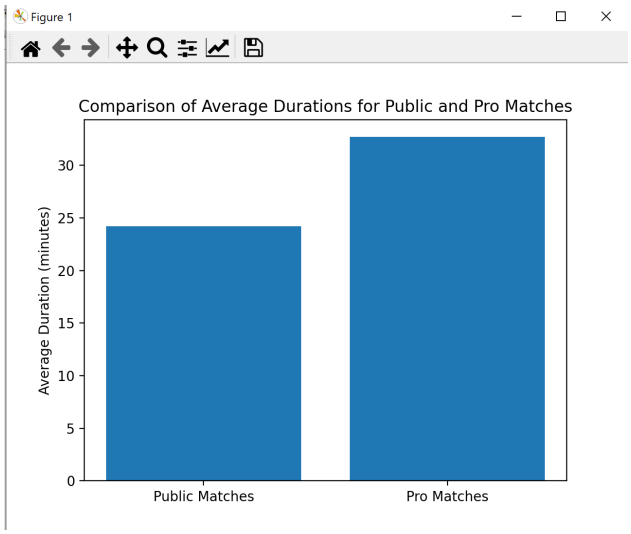
4. *The calculations from the data in the database (i.e. a screenshot) (10 points)*

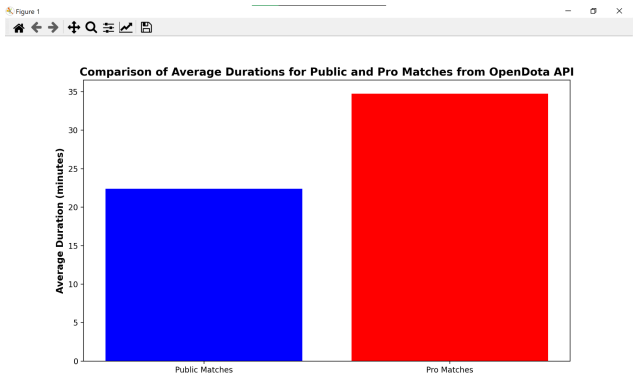
calculations.txt

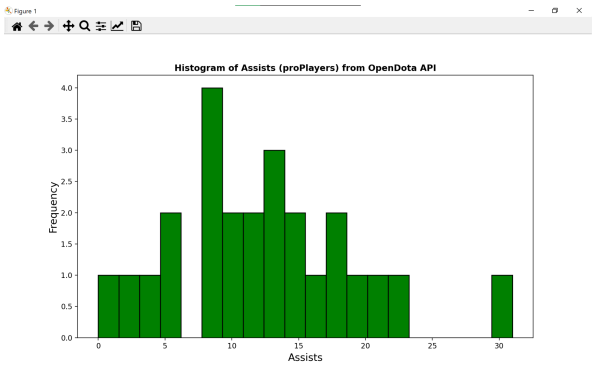
```
Final Calculations

Calculations # 1: Average number of games LOL

Average game duration for League of Legends: 23.76 minutes
        -from the Faker_matches table
Total playtime for League of Legends: 116 hours
        -from the JoinedData
total_playtime * 60 / average_game_duration
      (* 60 to convert hours to minutes)
Average number of games played: 292.0

Calculations # 2: Game Duration vs k/d/a
    -data from the Faker_matches table
    -Grouped League of Legend Matches by games >10 minutes, 10-20, 20-30, 30-40 and 40+
    -the average k/d/a was calculated by using the formula (kills + assists)/deaths
    -the k/d/a was then averaged by the number of games in that categorie
    -the table shown below displays that information
Time Slot    Average K/D/A
< 10         0.00
10 - 20      3.67
20 - 30      4.86
30 - 40      3.76
40+          3.36
```

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

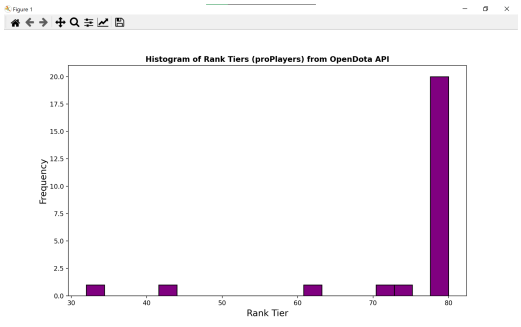


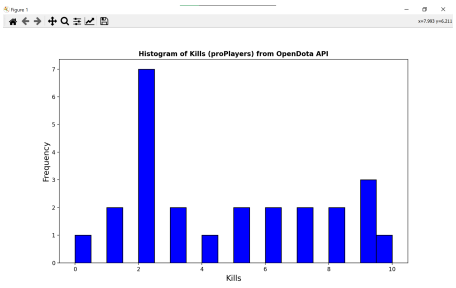



Comparison of Average Durations for Public and Pro Matches from OpenDota API
Average Duration (minutes)
Public Matches
Pro Matches

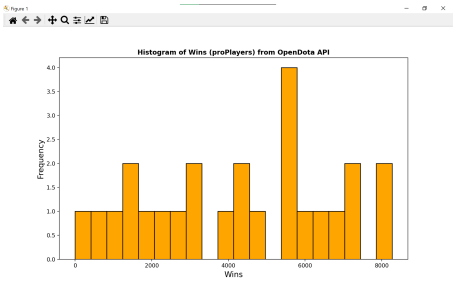

Histogram of Assists (proPlayers) from OpenDota API
Frequency
Assists

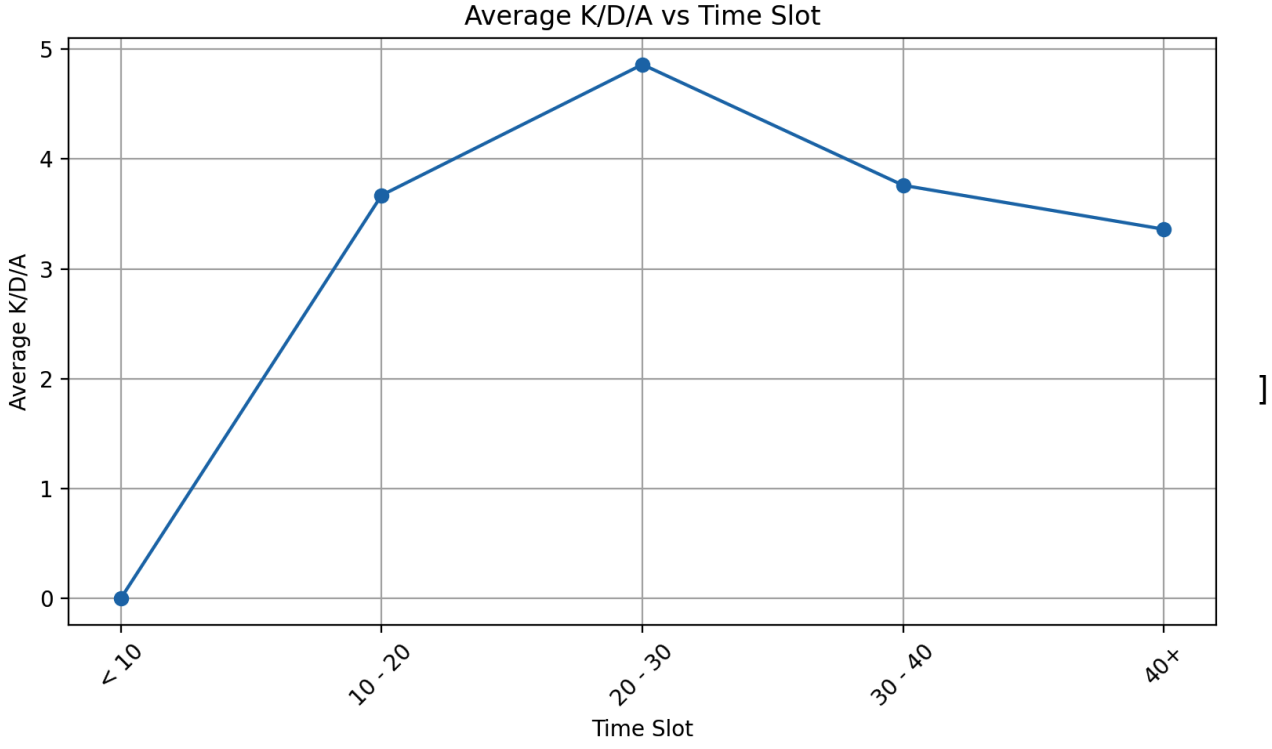

Histogram of Deaths (proPlayers) from OpenDota API
Frequency
Deaths


Histogram of Rank Tiers (proPlayers) from OpenDota API
Frequency
Rank Tier


Histogram of Kills (proPlayers) from OpenDota API
Frequency
Kills


Histogram of Losses (proPlayers) from OpenDota API
Frequency
Losses


Histogram of Wins (proPlayers) from OpenDota API
Frequency
Wins


Average K/D/A vs Time Slot
Average K/D/A
< 10
10 - 20
20 - 30
30 - 40
40+
Time Slot

]

6. Instructions for running code

You'll need to have requests, pandas, sqlite3, and matplotlib installed before running these programs. You also may need API keys for RAWG, the Riot Games League of Legends API, and the OpenDota2 API(not required, but there is a daily rate limit). The rawg.py, league_of_legends.py, and SI206Dota2OpenDotaAPI-newest.py should be run at least 4-5 times each first to load the games_database.db with the corresponding data. Then, the db_join_and_calculations.py should be run to join the GameNames and GamePlaytimes tables together in the database and run calculations. Finally, the data_visualizations.py should be run to create visualizations of the data and calculations in matplotlib.

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

Functions in RAWG.py:
get_games_info takes inputs of api_url, tag, ordering, current_page, and page_size, and it makes an API request with the current page number, parses the JSON response, and returns the results field of the JSON response.
update_database takes inputs of cursor and games_info and, if the game's name and playtime don't already exist, inserts them into the GameNames and GamePlaytimes tables respectively. Returns None.
main() takes no inputs, specifies the page and page size being retrieved from the API url, creates the PageInfo, GameNames, and GamePlaytimes tables if they don't already exist, executes the get_games_info and update_database functions, updates the page number, commits to the database and then closes. Returns None.

Functions in League_of_Legends.py:
get_match_data takes inputs of region, match_id, and api_key for a specific match from the Riot Games API and returns a dictionary containing all data about the match.
did_win takes inputs of PUUID (player ID) and match_data (output of get_match_data) and returns a boolean for whether or not the player won the match.
game_duration takes the input of match_data and returns the duration of the map in seconds.
get_champion takes inputs of PUUID and match data and returns the name of the champion used by a specific player in a match.
get_kill takes inputs of PUUID and match data and returns the number of kills a specific player got in a match.
get_death takes inputs of PUUID and match data and returns the number of deaths a specific player had in a match.

Get_assist takes inputs of PUUID and match data and returns the number of assists that a specific player had in a match.

Functions for SI206Dota2OpenDotaAPI-newest.py:
make_api_request(api_endpoint, params=None): makes requests to the api, given the list of api_endpoints taken from OpenDota API, so it basically gets a request to each API endpoint, and if successful, parses the JSON response and prints out the API endpoint, but if not, will print an error message for debugging
init_db(): takes in nothing, initializes the database by first connecting to the SQlite database, then using the cursor to execute SQlite commands, creates tables for pro players and game matches , and then commits the changes and closes the connections
main():
Executes the functions pertaining to the data collection, database storage, and matplotlib graph plotting. Here, it first initializes and creates a database, then first iterates through every pro player found, looks for specific categories like wins, losses, kills, deaths, etc. and lists it out, then stores it into the tables created from the database. Next, it does the same for Pro and Public matches, and then takes these stats and graphs it using matplotlib, creating some bar graphs and histograms from the data collected.

db_join_and_calculations.py:
Joins the GameNames and GamePlaytimes tables together, creates a new table JoinedData. Then it fetches the game durations and playtimes from the League of Legends data (both from the League API and the RAWG API) in the database and calculates the average number of games played. Then it retrieves K/D/A ratios data, categorizes games based on duration, calculates K/D/A per game, and calculates average K/D/A for each time slot.

data_visualizations.py:
Creates a bar plot comparing the playtime data for League of Legends and DOTA2 from the JoinedData table. Retrieves Dota2 match data from the publicMatches and proMatches tables in the database, calculates average match durations, and loads player data into a DataFrame. Creates a bar chart comparing average durations for public and pro matches, and histograms showing distributions of K/D/A, rank tiers, wins, and losses for pro players in Dota2. Creates a line graph showing average K/D/A versus time slot for League of Legends games.

8. Resource documentation:

Date	Issue Description	Location of	Result (did it

		Resource	solve the issue?)
12/10/23	Documentation of tags	https://api.rawg.io/docs/#operation/games_list	Seeing the example given for calling tags led to the capitalization fix
12/10/23	ValueError Inhomogenous Array Shape	Claude.ai	Helped explain what that error meant for creating a bar chart; I managed to fix the error by myself
12/10/23-12/11/23	Unsure about Format for requesting Apis	Canvas (discussion 8)	Having worked on code that already utilizes apis helped me to fully figure out how to request and handle APis
12/10/23-12/11/23	Unsure about format for utilizing databases	Canvas(discussion 12)	Helped me to frame my code around initializing, creating, and adding data to my databases
12/10/23-12/11/23	Unsure about matplotlib	Canvas (discussion 12, Matplotlib Lecture Slides, Runestone)	Helped me to understand how to use Matplotlib to visualize my data collected
12/10/23-12/11/23	Unsure how to go about rate limiting issues	Chatgpt	Helped me to understand how to incorporate datetime to create delays to avoid exceeding the daily rate limit for api requests
12/10/23-12/11/23	Unsure about how	UM-gpt	Helped me to

	to calculate average duration for Pro and Public matches		discover a function - deltatime that helped me to convert the data I received in seconds to an average time in minutes:seconds format instead

Github repo link:
https://github.com/JosephYM37/SI-206-Final-Project-Game.git

Slides:
https://docs.google.com/presentation/d/1UnC5miKUgs6UFMzkFBiCHxEBSnvLrZmv5g9_nRr9ToQ/edit?usp=sharing