Savin Dasanayaka & Ananthu Nair

SI 206 Final Report

A. **Goal of the project**

Our goal for this project is to see if there's a correlation between a Pokémon's average stats and its competitive tier. Pokémon are categorized into different tiers based on their "perceived viability in the current metagame". A Pokémon's stats determine its performance in different statistical categories like HP and attack. Higher stats indicate better performance in those categories, so Pokémon with higher stats are considered stronger. However, we're curious if this means that the Pokémon with the highest average stats are all used enough in competitive play to be considered OU (Overused) or Ubers (overpowered Pokémon, usually banned).

For this goal, we planned to gather data on the Pokémon's tier from Smogon.com, a comprehensive online resource for Pokémon battling created by community fans. We also planned on getting stat data from Pokeapi to calculate an average stat score for each Pokémon we wanted to look at.

B. Achieved goals

We achieved our goal of seeing if there is a correlation between a Pokémon's average stats and its competitive tier and found that there was not a strong correlation between the two. We first used a random Pokémon generator to generate over 100 random Pokémon from generation 7 or earlier, and compiled them into one large master list. We carried through using Smogon's website to get the Pokémon's tiers using BeautifulSoup and Playwright to scrape the dynamic JavaScript site. Playwright is a Python module that renders in dynamic JavaScript from a website and gives it time to load, which we then also used to get a specific section of the HTML code. We then used BeautifulSoup to search for the specific text we needed (corresponding to a Pokémon's specific tier) and save that data. For Pokémon-specific stats data, we used PokeAPI, which is an open source API that allowed us to get all of a specific Pokémon's stats. We looped through our master list and made API calls for each Pokémon, and stored each stat in a dictionary. We were also able to successfully add all this data into two tables in our database. One table was an id table for the tiers and the other was our main stats table, which contained the Pokémon, their tier, and their individual stats. Lastly, we selected data from both tables in the database using a join and made two visualizations of our collected data.

We ended up finding that there is somewhat of a correlation between the Pokémon's average stats and its tier, with the highest average stats having Uber stats and the lowest having ZU status (ZeroUsed). However, a lot of midrange pokemon hovering around 80-90 are in tiers from ZU to OU (OverUsed), so it does not look like there's a definitive connection between a Pokémon's average stats and its tier.

C. Problems faced

We faced many problems during this project. Initially, we wanted to look at how movie release dates impacted streaming numbers for songs in the movies using OMDB API, a website with soundtrack names, and Spotipy, but Spotipy did not offer streaming numbers, so we moved on to this idea.

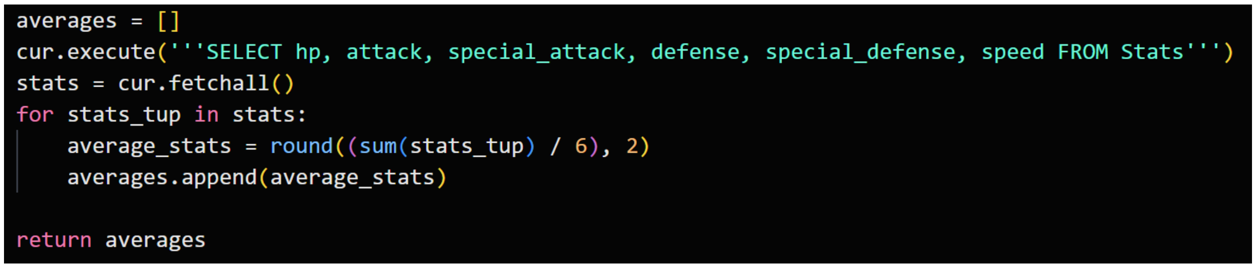
This idea had its own problems. Namely, scraping the website was difficult at first because it used JavaScript for animations, so it was dynamic, and we were pulling the source code instead of the HTML structure. This was our biggest challenge to get by since we had not learned how to scrape dynamic websites, but using Playwright, we were able to. After using Playwright, another major problem was that each iteration would take up to 20 seconds, which would take a ton of time to store 100 items in the database. We got by this by doing some research on Playwright arguments on the internet.

Regarding the API, we had issues where the Pokeapi would not have a Pokémon's information in the database i.e. Giratina. To solve this, we used a try-and-except which would only work when the API returned something.

When adding to the database, we had some problems limiting the entries per code run to 25. Our original code would count how many Pokémon were inserted and stop when the number hit 25. However, it would keep iterating from the beginning of the list, and if a Pokémon was not accepted for whatever reason, it would consider that Pokémon again and take a lot of unnecessary time. To fix this, we used the COUNT method in SQL to get the number of Pokémon at the beginning of the iteration and only iterate until 25 were added to that number. Then, it would restart the iteration where the previous one left off.

D. Calculations from data in database

Our primary calculation was finding the average base stat value for each Pokémon.

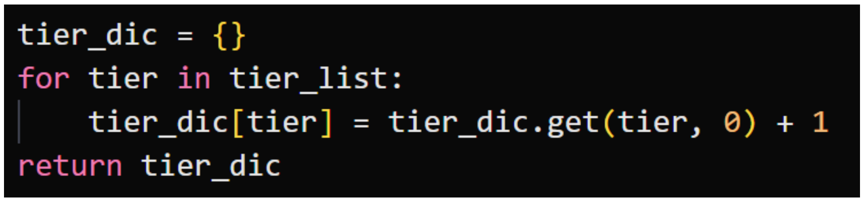


```
averages = []
cur.execute('''SELECT hp, attack, special_attack, defense, special_defense, speed FROM Stats''')
stats = cur.fetchall()
for stats_tup in stats:
    average_stats = round((sum(stats_tup) / 6), 2)
    averages.append(average_stats)

return averages
```

The code above shows our process for making these calculations for each Pokémon. We SELECTed all 6 stats from our Stats table for each Pokémon, and then summed the tuple that was returned by the fetchall(). We then divided the sum by 6 and rounded to 2 decimals to get a clean average, and added these averages to a list to pass into matplot.

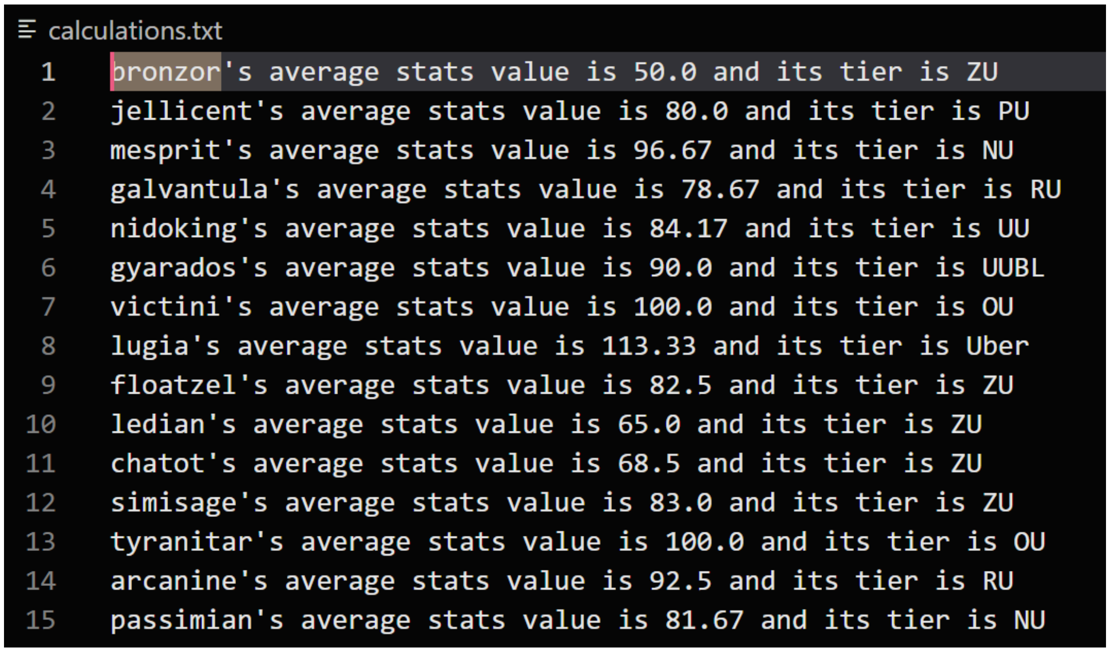
We also found the total number of Pokémon for each tier to visualize the distribution of Pokémon across the 8 tiers.



```python
tier_dic = {}
for tier in tier_list:
    tier_dic[tier] = tier_dic.get(tier, 0) + 1
return tier_dic
```

This code simply gets the totals for each tier by looping through the list of all tier values and incrementing the dictionary value.

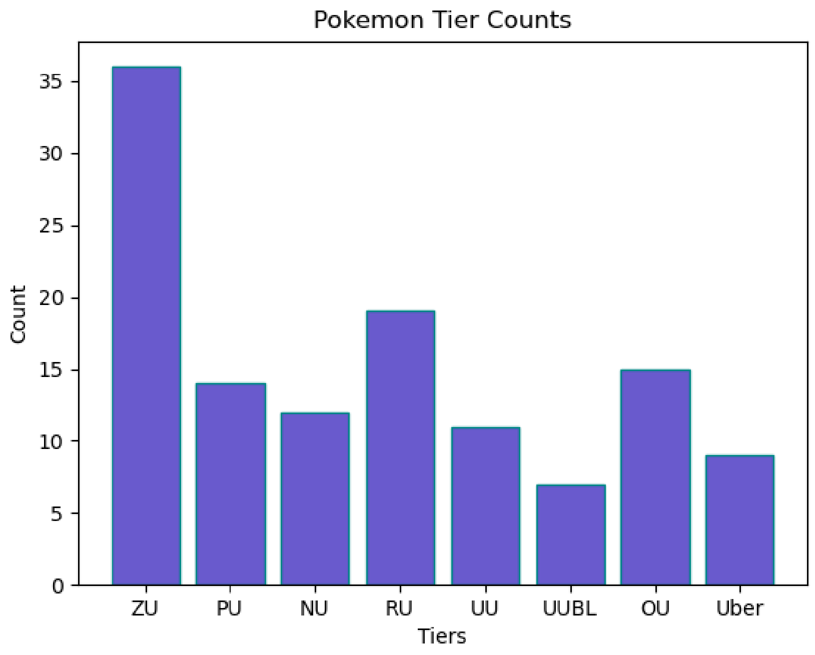
Lastly, we compiled the average stats and tier values into a simple text file:



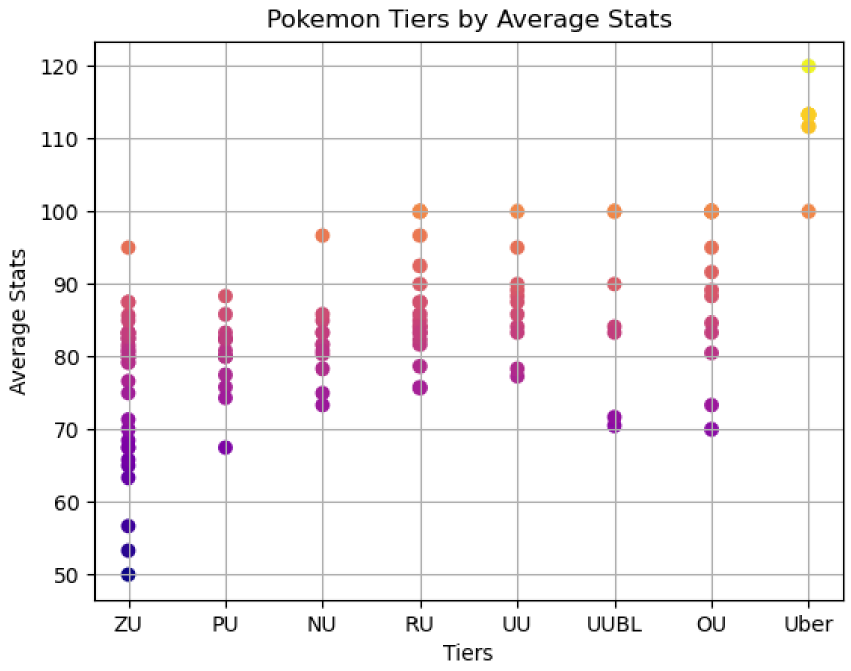
```
bronzor's average stats value is 50.0 and its tier is ZU
jellicent's average stats value is 80.0 and its tier is PU
mesprit's average stats value is 96.67 and its tier is NU
galvantula's average stats value is 78.67 and its tier is RU
nidoking's average stats value is 84.17 and its tier is UU
gyarados's average stats value is 90.0 and its tier is UUBL
victini's average stats value is 100.0 and its tier is OU
lugia's average stats value is 113.33 and its tier is Uber
floatzel's average stats value is 82.5 and its tier is ZU
ledian's average stats value is 65.0 and its tier is ZU
chatot's average stats value is 68.5 and its tier is ZU
simisage's average stats value is 83.0 and its tier is ZU
tyranitar's average stats value is 100.0 and its tier is OU
arcanine's average stats value is 92.5 and its tier is RU
passimian's average stats value is 81.67 and its tier is NU
```

E. **The visualizations we created**

We made two visualizations from this data. One was the bar graph that displayed the counts of Pokémon in each tier:



The second visualization plotted tiers against average stat values, which allowed us to see if, as the tier increased, the average stat value did too:



F. **Instructions for running the code**

When you run APIs_and_Soup.py, it will attempt to add the first 25 Pokémon in the master list to the database. To get at least 100 Pokémon in the database, run the code in this file at least 4 to 5 times. You will end up with 2 tables in your database, one for the tiers and their IDs, and one for each Pokémon and its stats.

When you run calculations_and_matplot.py, two graphs will be output: a graph for the count of Pokémon by tier and a graph with Pokémon average stats by tier. These graphs will show you the relationship between tiers and Pokémon in the tiers. A .txt file with the calculations of each Pokémon's stats and its tier will also be output.

G. **Documentation for each function (input, output)**

APIs_and_Soup.py			
Function Name	**Function Operation**	**Input**	**Output**
get_pokemon_tier	Gets the Pokémon's tier from the Smogon website	Pokémon name	String of Pokémon's tier from Smogon (i.e. RU or Uber)
get_pokemon_stats	Gets the stats of the given Pokémon	Pokémon name	A dictionary of Pokémon stats from PokeAPI
setup_db	Sets up database connection	None	Database cursor and connection (cur, conn)
setup_tier_table	Creates and updates the Tiers table in the database with tiers and assigns an id to each tier	Database cursor and connection (cur, conn)	None
update_stats_table	Uses Pokémon tiersand stats dictionary to insert names, stats, and tier_id into a table in the database	Pokémon name, Database cursor and connection (cur, conn)	None

calculations_and_matplot.py			
Function Name	**Function Operation**	**Input**	**Output**
make_averages_list	Uses the database info to calculate each Pokémon's average stats and store them in a list	Database cursor and connection (cur, conn)	A list of Pokémon average stats (in database row order)
make_tier_list	Joins the stats and tiers table from the database to select each Pokémon's tier and store them in a list	Database cursor and connection (cur, conn)	A list of Pokémon tiers (in database row order)
write_calculations	Writes each Pokémon's average stats and tier to a text file	Average stats list, tiers list, database cursor and connection (cur, conn)	.txt file, returns None
tier_counts	Uses the list returned from make_tier_list to create a dictionary with tiers as keys and counts as values	tier_list (from make_tier_list)	A dictionary of Pokémon tiers and their counts
graph_tier_counts	Uses the dictionary returned from tier_counts to create a bar graph of tiers and their corresponding frequency	tier_dict (from tier_counts)	A bar graph, returns None
graph_tier_by_avgstat	Uses lists from make_averages_list and make_tier_list to create a scatter plot of tiers vs average stats by tier	Averages list, tiers list (from make_averages_list and make_tier_list)	A scatter plot, returns None

H. Document all resources used

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
4/2/25	Figuring out the autoincrement function in sqlite	https://www.tutorialspoint.com/sqlite/sqlite_using_autoincrement.htm	Yes, this website helped us understand autoincrement and how to make adaptive tables
4/8/25	Getting randomized Pokémon	https://randomPokémon.com/	Yes, the website generated random Pokémon.
4/8/25	Learning what tiers are, how they are determined	https://bulbapedia.bulbagarden.net/wiki/Tier	Yes, the website described the difference in tiers
4/9/25	Learning how to use the Selenium library to webscrape a dynamic website (JavaScript)	https://www.youtube.com/watch?v=onlQ7fL4ey8&ab_channel=JohnWatsonRooney	Yes, this video helped us get the html structure instead of the js source code. However, selenium did not end up working for us because of how long it took to run one instance.
4/10/25	Learning how to use the Playwright library to webscrape a dynamic website (js)	https://www.youtube.com/watch?v=H2-5ecFwHHQ&ab_channel=JohnWatsonRooney	Yes this video helped us understand Playwright which was much better for reducing the runtime of scraping smogon's website
4/10/25	Figuring out why page.goto from playwright was taking so long to render the page and return something	https://www.youtube.com/watch?v=qvlfbHFxqnI&ab_channel=Checkly	This video helped a lot with shortening our runtimes

4/11/25	Learning how to get the number of rows in a sqlite database to index from the last row inserted	https://www.sqlitetutorial.net/sqlite-count-function/	Yes, this helped us shorten our code in main by a lot
4/11/25	Learning how to order unique xticks in matplot	https://www.reddit.com/r/learnpython/comments/kfjrfu/how_to_change_the_order_of_xticks_in_a_matplotlib/	No, this code did not end up working

Link to Github Repository: **https://github.com/nairanan/si206Final.git**