# **Pokemon, Anime, and Mario**

SI 206 Final Report - Fall 2019

By SoJung Ham and Dylan Yono

GitHub: <https://github.com/sjham16/206finalproject.git>

**Goals for the Project**

Our project changed substantially from our initial project plan. Initially we were planning on comparing data from users of anime tracking websites across APIs for those trackers, using Seaborn for visualizations, etc. In the end, we went with the goals of comparing average base stats for each Pokemon type from the Pokemon API, and comparing the average user rating of Mario games across each decade from the RAWG (video game database) API. We wanted to visualize them with Matplotlib. We took a different direction with the anime API and made it our extra credit.

For all of our APIs, we wanted to accomplish all the required objectives — creating a database, writing to a text file, calculating information, making the visualizations, etc.

* **Pokemon API:** Pokemon all have different types (fire, water, grass, etc.) and base stats that determine their different strengths (attack, defense, special attack, special defense, hp, speed). We wanted to calculate the average base stats for each Pokemon type and compare across type using bar graphs for each base stat. Therefore we could see things like: Which Pokemon type has the Pokemon with the highest speed, or which Pokemon type has the Pokemon with the lowest defense, etc. We wanted to utilize a Pokemon API to collect data on all original 151 Pokemon’s base stats to accomplish this.
* **RAWG API (Mario):** Nintendo’s “Mario” franchise is enormous — it has well over 200 games when you account for all the series’ based on side characters (we didn’t include all the Yoshi games or Luigi games for example), and still over 100 that are just about Mario. The goal was to collect data on all of the Mario games including their release date and average rating from users on a game tracking website, RAWG. Utilizing the RAWG API, we wanted to visualize the average rating of Mario games by the decade. This could answer questions like, what was the best and worst decade for Mario games? Is the Mario series getting better or worse?
* **Jikan API (Anime):** Anime is aired on a seasonal basis (winter/spring/summer/fall each year), and most anime are adapted from a certain source material (such as manga, light novels, etc.). For the most recent season of anime, fall 2019, we wanted to utilize the Jikan API to see which types of adaptations were being rated the highest. This could answer questions like, on average, what are fall 2019 visual novel adaptations being rated? Which type of source material has the highest-rated adaptations in fall 2019?

**Goals that were Achieved**

We achieved our goals for both APIs. On the Pokemon API, we created bar charts for each base stat, showing each Pokemon type’s average base stat for the selected stat (speed, attack, etc.) You could effectively answer the questions asked in our goals (Electric types have the highest speed, ghost types have the lowest defense). On the RAWG API, we created a line plot that shows the average rating of Mario games released in a given decade and how the average rating has changed over the decades. You could also effectively answer the questions asked in our goals (2000s were the best, 2010s were the worst, and the Mario series is getting worse according to RAWG users). On the Jikan API, we created a bar chart that shows the average rating of anime adaptations’ source material. You could effectively answer the questions asked in our goals (visual novel adaptations were rated 6.895/10 on average, manga adaptations were rated highest).

**The Problems Faced**

Most issues we ran into were easily and quickly googleable/solvable — e.g., our code was not running in the VSCode terminal, so we just ran it from Anaconda Prompt. The only roadblock that we really had to figure out on our own was how to limit our code to collecting only 20 items of data from the API at a time. We didn’t know how to do this without manually changing the code each time we ran to go over the next 20 items on the API. We couldn’t figure out the methods presented in the lecture slides so we resolved it through our own testing and practice.

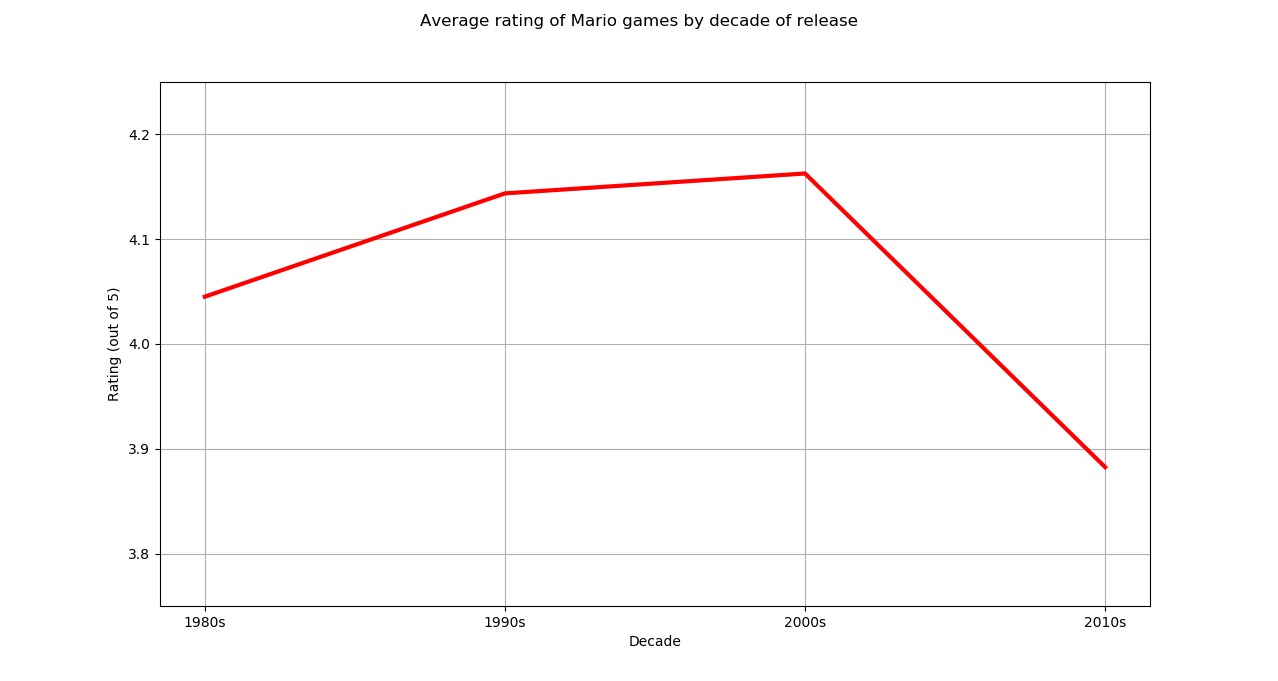
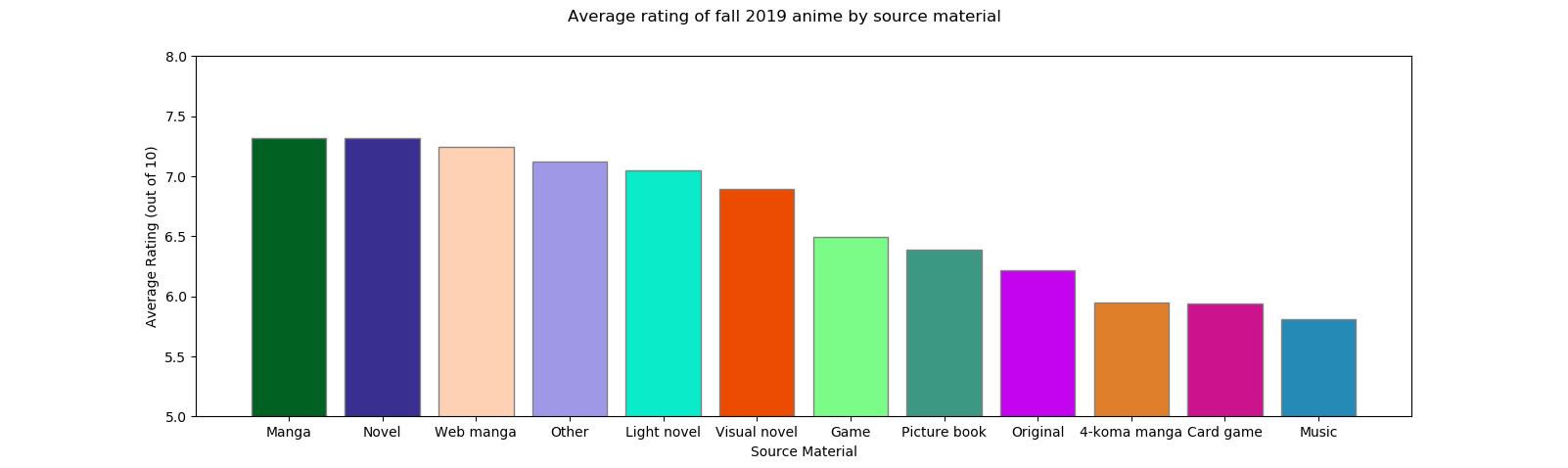
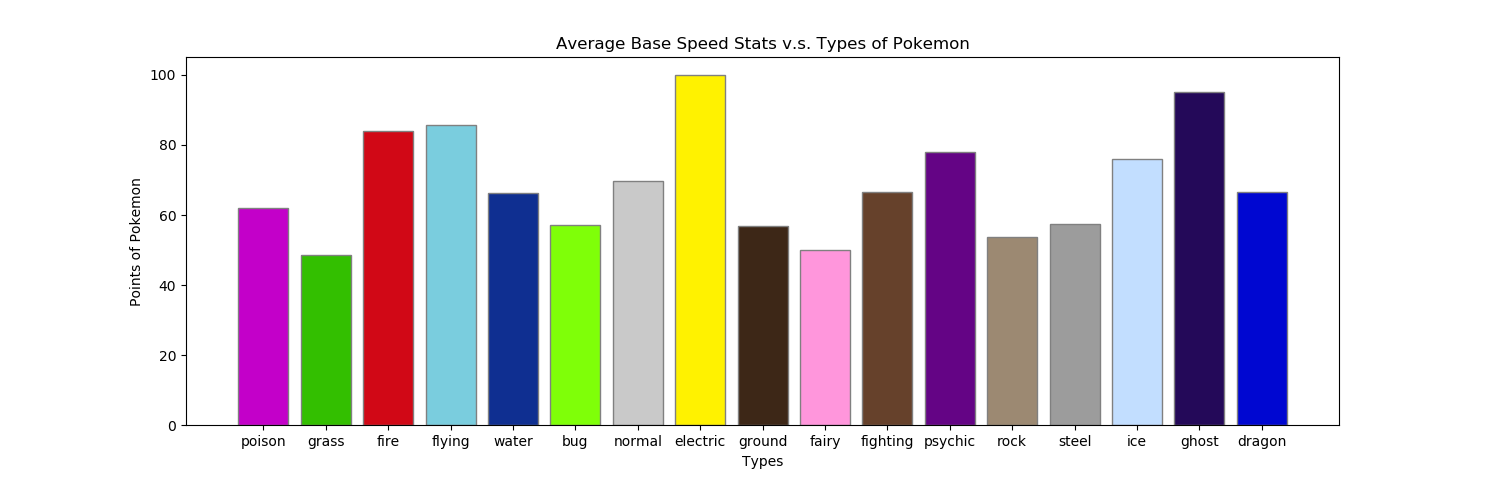
At first we found that the solution was pretty simple: Let the program collect all the data so far from the cache, and then if it has to call the API, stop it after collecting 20 items and writing those to the cache and then quit the program. Then the next time we ran the code, it would collect everything from the cache thus far, and then collect the next 20 items from the API. Once this was solved the rest of our project was smooth sailing, or so we thought: The requirement wasn’t about how many times the API was called, but how many items were *saved to the database*.

We ended up heavily restructuring our code to achieve this. For the RAWG API, we used a placeholder row while adding data to the database that tracked which page we needed to call. This was complicated, but it worked fine once we got it figured it out. But by the time we did the Jikan API, we figured out a much smoother method of calling len() to see how many rows were in the database already, and then slicing the dictionary of data based on that. We ended up doing a lot of unnecessary work, but we managed to satisfy the requirements on the rubric.

**File that Contains Calculations from the Database**

* pokemon\_calc.csv
* mario\_data.txt
* anime\_data.txt

**Visualizations Created**



**Instructions for Running The Code**

* **Pokemon API:** Run “pokemans.py” from the terminal. We included a cache file with all the data, “pokemon\_cache.json”. If you delete the cache file, the API will be called. If “videogame.db” is not present or does not have all the necessary data, the code may have to be run multiple times to fill up the database. Follow the instructions and input a number to see a visualization or write data to a text file.
* **RAWG API:** Run “marios.py” from the terminal. We included a cache file with all the data, “mario\_cache.json”. If you delete the cache file, the API will be called. If “videogame.db” is not present or does not have all the necessary data, the code may have to be run multiple times to fill up the database. Follow the instructions and input a number to see a visualization or write data to a text file.
* **Jikan API:** Run “anime.py” from the terminal. We included a cache file with all the data, “jikan\_cache.json”. If you delete the cache file, the API will be called. If “videogame.db” is not present or does not have all the necessary data, the code may have to be run multiple times to fill up the database. Follow the instructions and input a number to see a visualization or write data to a text file.

**Documentation of Functions (including input and output)**

**Pokemon.py** documentation

* setUpDataBase()
  + This function sets up a database. It takes a filename to name the database and returns a cursor and connection.
* readCache(CACHE\_FNAME)
  + This function reads from the JSON cache file and returns a dictionary from the cached data. If the file doesn’t exist, it returns an empty dictionary.
* write\_cache(cache\_file, cache\_dict)
  + This function encodes the cache dictionary (cache\_dict) into JSON format and writes the contents in the cache file (cache\_file) to save the search results.
* get\_data\_with\_caching(cur, conn)
  + This function loads up to 20 items from the cache or API into the database. It takes a cursor and connection as input.
* calculate\_data(cur, conn)
  + This function selects all the relevant data from the tables using a database join. It takes a cursor and connection as input. It returns the data it selects to be used in the stat calculation functions.
* getAverageSpeedStats(data, pokemon\_type) -- repeated for special defense, special attack, defense, attack, and HP
  + This function calculates the average speed/sp.def/etc. stat for a type. It takes the table data and a pokemon type as input. It returns the average speed/sp.def/etc. stat for pokemon of that type.
* createAverageSpeedGraph() -- repeated for special defense, special attack, defense, attack, and HP
  + This function shows a graph of the average base speed/sp.def/etc. stat for all pokemon types. It calls the getAverage(Speed/SpecialDefense/etc.)Stats function on every type. It visualizes that data using matplotlib.

**marios.py** documentation

* setUpDatabase(db\_name)
  + This function sets up a database. It takes a filename to name the database and returns a cursor and connection.
* read\_cache(CACHE\_FNAME)
  + This function reads from the JSON cache file and returns a dictionary from the cached data. If the file doesn’t exist, it returns an empty dictionary.
* write\_cache(cache\_file, cache\_dict)
  + This function encodes the cache dictionary (cache\_dict) into JSON format and writes the contents in the cache file (cache\_file) to save the search results.
* get\_mario\_data(cur, conn)
  + This function loads up to 20 items from the cache or API into the database. It also adds a placeholder item to the database to track which page of data to collect each time the code runs. It takes a cursor and connection as input.
* decade\_rate(cur, conn)
  + This function does all the data calculation. It takes a cursor and connection as input. It converts each release date into a release decade and then creates a table of decades. Using the game id, a database join selects the game name, rating, and decade of release. It returns a list of tuples with each decade and the average Mario game rating.
* decade\_rating\_chart(data)
  + This function makes a line chart showing the average rating of Mario games by decade of release. It takes the data from decade\_rate as input and shows a graph of the data using MatPlotLib.
* write\_to\_txt(data)
  + This function takes the data from decade\_rate as input. It creates a text file of the data with each item on a new line.

**anime.py** documentation

* setUpDatabase(db\_name)
  + This function sets up a database. It takes a filename to name the database and returns a cursor and connection.
* read\_cache(CACHE\_FNAME)
  + This function reads from the JSON cache file and returns a dictionary from the cached data. If the file doesn’t exist, it returns an empty dictionary.
* write\_cache(cache\_file, cache\_dict)
  + This function encodes the cache dictionary (cache\_dict) into JSON format and writes the contents in the cache file (cache\_file) to save the search results.
* get\_anime\_data(cur, conn)
  + This function loads up to 20 items from the cache or API into the database. It takes a cursor and connection as input.
* best\_source(cur, conn)
  + This function does all the data calculation. It takes a cursor and connection as input. It cleans all the data and inputs the clean data into new tables. Using the anime id, a database join selects the anime title, score, and source material. It returns a list of tuples with each source and the average rating.
* make\_chart(data)
  + This function makes a bar chart showing the average rating of anime by source material. It takes the data from best\_source as input and shows a graph of the data using MatPlotLib.
* write\_to\_txt(data)
  + This function takes the data from best\_source as input. It creates a text file of the data with each item on a new line.

**Documentation of all resources used**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Issue Description** | **Location of Resource** | **Result** |
| 12/9/2019 | Reading json data into a table | Stack Overflow | Success. .items() |
| 12/9/2019 | Sqlite module not working, even though it’s already installed | Stack Overflow | Turns out I had to just restart my computer |
| 12/11/2019 | Tab key stopped working in VS code | Internet<https://stackoverflow.com/questions/35519538/visual-studio-code-tab-key-does-not-insert-a-tab> | Fixed |
| 12/11/2019 | Forgot how to insert list of tuples into a database | Internet<https://stackoverflow.com/questions/34596967/python-inserting-list-of-tuples-into-sqlite> | Fixed |
| 12/12/2019 | Don’t know how to change bar chart colors | Internet  <https://python-graph-gallery.com/3-control-color-of-barplots/> | Fixed |
| 12/12/2019 | Don’t know how to change scale of y-axis | Internet  <https://stackoverflow.com/questions/3777861/setting-y-axis-limit-in-matplotlib> | Fixed |
| 12/12/2019 | Forgot how to write to a text file | Internet  <https://www.geeksforgeeks.org/reading-writing-text-files-python/> | FIxed |