Megan Fan, Katelyn Ma, Sabrina Yu

Prof. Ericson

SI 206

21 April 2023

Final Project Report

Goals/Data We Planned to Work With

The motivation behind selecting the particular APIs for Animal Crossing, Genshin Impact, and Monster Hunter was because the three of us have a collective interest in video games, and wanted to see if we find any similarities between three games with three very different playstyles. In particular, we wanted to see if there was anything in common between item rarity and base stats in all three games so we could break down the traits of what each game counts as something of value.

For each game, we gathered data on valuable items. In Animal Crossing, fish and sea creatures are some of the items that are worth value to the player, as the player can catch these fish in exchange for “bells”, which are the in-game currency. Some fish are worth more bells than others and those are marked as more “rare” to catch because of their higher value. Monster Hunter and Genshin Impact have similar systems in which weapon rarities denote the value, and each weapon has a base attack number which calculates into how much damage the weapons do. From the three APIs, we planned to gather the information of item name, rarity, and value indicator, which is price in bells for Animal Crossing and attack for Monster Hunter and Genshin Impact.

Goals/Data We Gathered and Achieved

After completing the project, we were able to end up gathering the data on the values and rarities for the items in each game. We placed all of the data into a database which we

could easily access in DB Browser for SQLite, and this way we could easily access the data to use for comparison.

Additionally, we achieved our goals of being able to make reasonable connections between the rarity and the value of each item in their respective games. We made visuals for each game to chart the correlations between the rarities and the values, and when we run our code, it will produce these bar graphs in order to view the comparisons we made.

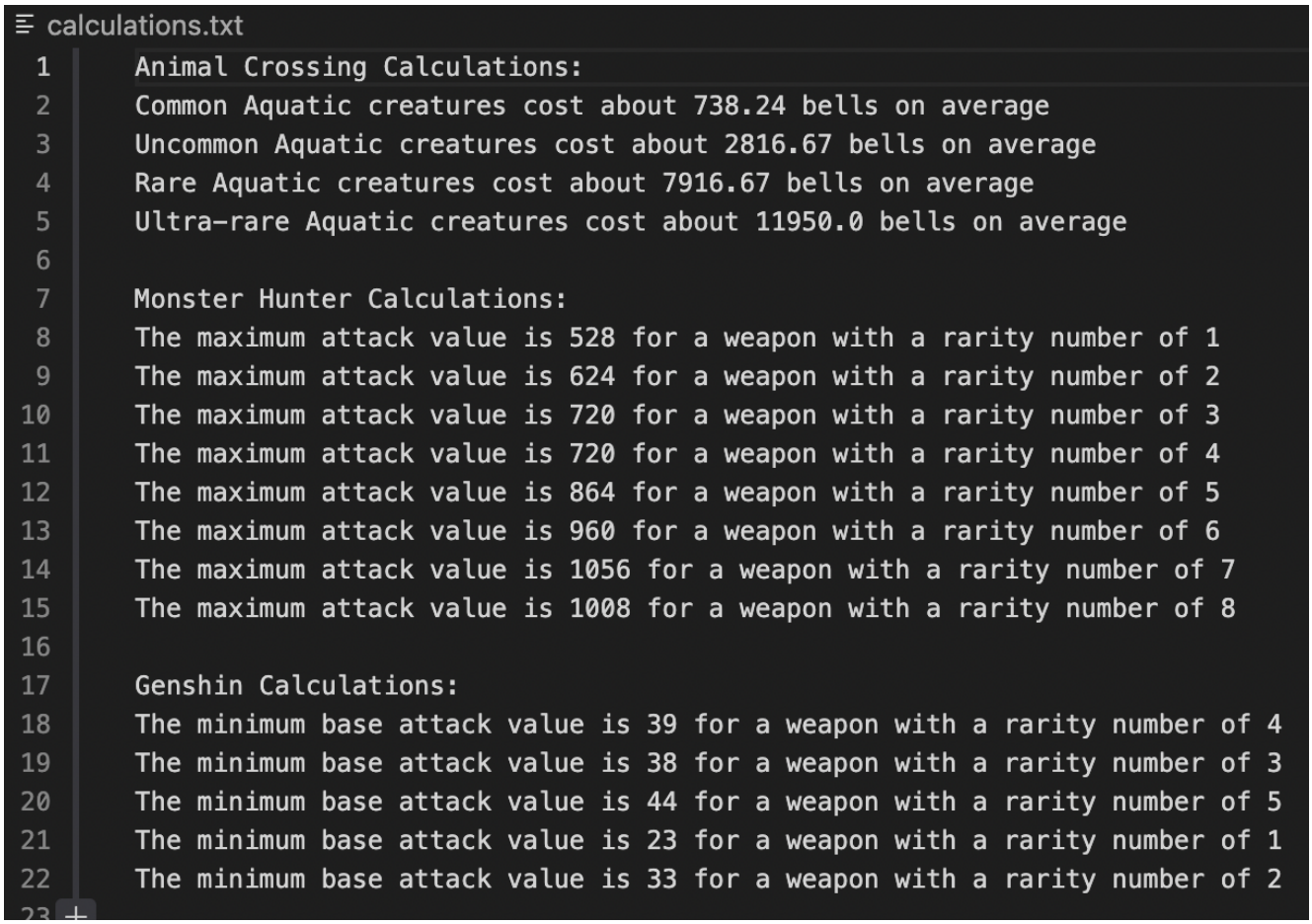
Problems and Troubleshooting

We had problems with figuring out how to get the code to only retrieve 25 items without the function restarting after the code runs - whenever we ran it, it just started over from the first ID instead of going onto the next 25 items. However, with help from a GSI, we got a lot of clarification on the instructions and rubric, and were able to troubleshoot and solve this issue. Afterwards, when we ran the code, it would allow us to parse through all of the data in the database in groups of 25, rather than restarting each time the function ran.

Another minor problem was getting the databases to load in for everyone. Since we were all working on one file, we had trouble figuring out how to all view the same updated databases. Even after pushing on GitHub, we didn't all have access to the same things, and we only got on the same page after deleting and then redownloading our VSCode files. Although having one file for everything caused a few troubles, it ultimately allowed us to be able to work in collaboration more efficiently and help with debugging each other's functions, and we used a liveshare to see everyone's updates live. We also made our own files to copy into the main file through GitHub if we were having trouble with liveshare.

Calculations from the Data (Calculation File Screenshot)

Attached is a screenshot of the calculations on our file from the data in the database.

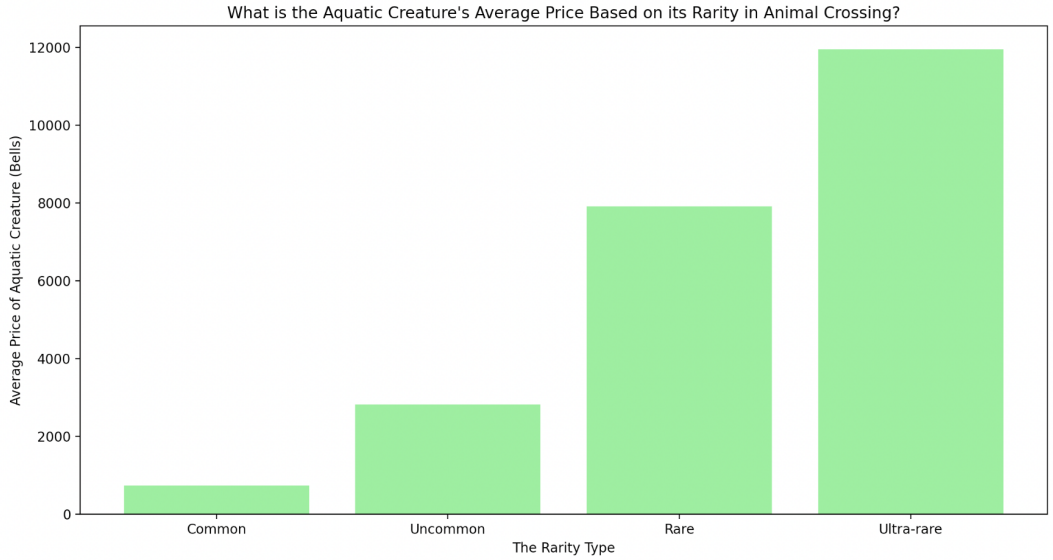


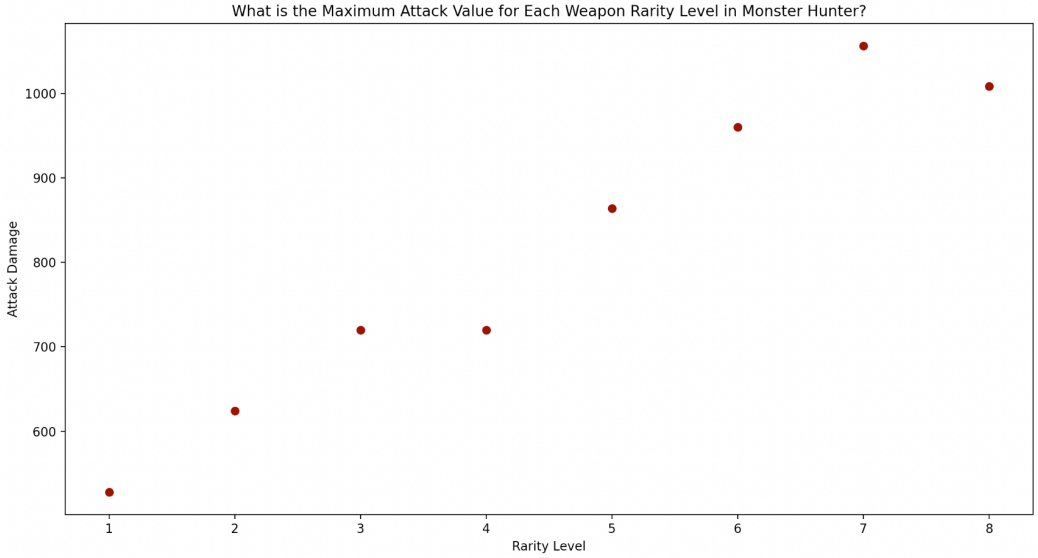
calculations.txt

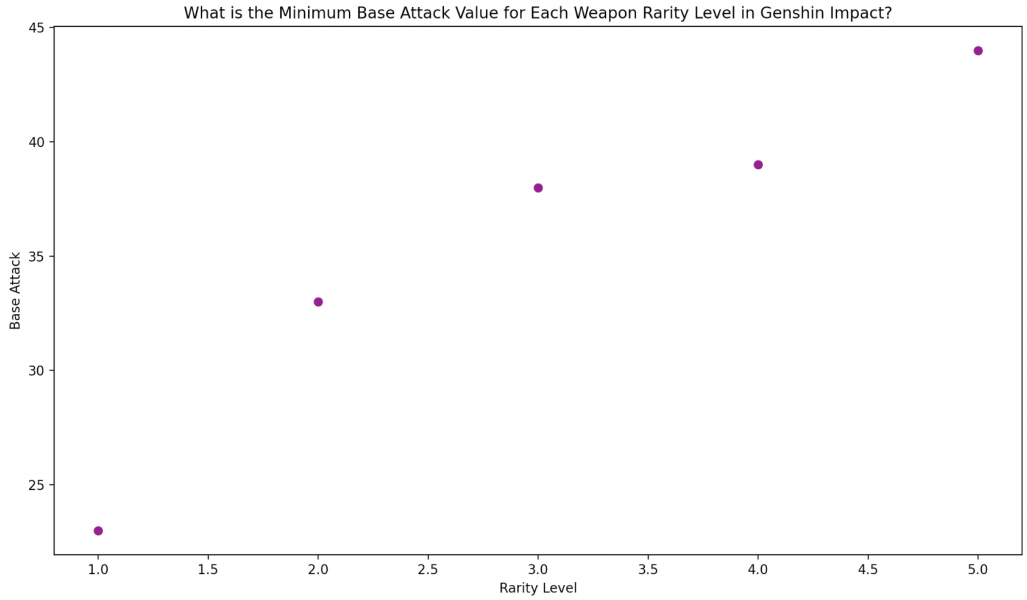
```
Animal Crossing Calculations:
Common Aquatic creatures cost about 738.24 bells on average
Uncommon Aquatic creatures cost about 2816.67 bells on average
Rare Aquatic creatures cost about 7916.67 bells on average
Ultra-rare Aquatic creatures cost about 11950.0 bells on average

Monster Hunter Calculations:
The maximum attack value is 528 for a weapon with a rarity number of 1
The maximum attack value is 624 for a weapon with a rarity number of 2
The maximum attack value is 720 for a weapon with a rarity number of 3
The maximum attack value is 720 for a weapon with a rarity number of 4
The maximum attack value is 864 for a weapon with a rarity number of 5
The maximum attack value is 960 for a weapon with a rarity number of 6
The maximum attack value is 1056 for a weapon with a rarity number of 7
The maximum attack value is 1008 for a weapon with a rarity number of 8

Genshin Calculations:
The minimum base attack value is 39 for a weapon with a rarity number of 4
The minimum base attack value is 38 for a weapon with a rarity number of 3
The minimum base attack value is 44 for a weapon with a rarity number of 5
The minimum base attack value is 23 for a weapon with a rarity number of 1
The minimum base attack value is 33 for a weapon with a rarity number of 2
```

Visualizations Created (Screenshot)

Attached are screenshots of the bar graphs that are produced when our code runs.







How to Run the Code

Here are steps to run our code!

1. Download the .zip file
2. Unzip the file and open the folder in a program that can run code (we used VSCode)
3. Open getdata.py
4. Run getdata.py at least 6 times
5. Open processandvisualizedata.py
6. Run processandvisualizedata.py once

Documentation of Functions

File 1 - getdata.py: Gathers the data from APIs and creates tables in the database.

- def load_json(filename):
    - Takes in a filename as an input and returns a Python dictionary with the data that we want. The function itself reads the json files and then puts the contents into a dictionary.

- def write_json(filename, dict):
    - Takes in a Python dictionary and then returns it as a json object.
- def get_json_info(url, params = None):
    - The function first checks if the url has any params. Then, it checks if the url works, and then takes a working url and turns it into a json object using load_json, which the function returns.
- def genshin_cache(filename, url):
    - Writes a json object from the inputted url - it does not return anything, but it does create a json object.
- def cache_json(filename, url):
    - Turns the information from the API url into a json object by calling get_json_info and write_json. Does not return anything, but creates a json object from the url.
- def setUpdatabase(db_name):
    - This function takes in a name that will be used for the database, and then creates a database file - it returns the cursor and the connection.
- def create_acnh_dictionary(fishfile, seafile):
    - Takes the data from fishfile and seafile and then puts them into categories of name, price, availability, and rarity - this is done with both files so that they can eventually be compiled into one dictionary. This function also converts the “speed” of the fish into “rarity”, so that they can be evaluated in the same way as the other functions. Calls load_json to access the data as a Python object, and then returns a dictionary with all of the information in it - acnhdict, which is the overall organized dictionary for everything in Animal crossing.
- def create_acnh_name_table(cur, conn):

    - This calls create_acnh_dictionary and then creates a table. The function then puts acnhdict into the table with four columns, and inserts 25 rows into the table at a time.
- def create_genshin_dictionary(genshinfile):
    - Takes the data from genshinfile and then puts them into categories of name, rarity, passive name, and base attack - this is done with both files so that they can eventually be compiled into one dictionary. If there is no passive name, the function returns “None”. Calls load_json to access the data as a Python object, and then returns a dictionary with all of the information in it - weapondict, and initializes the dictionary so that the weapon names are the keys and everything else are the values.
- def create_genshin_table(cur, conn):
    - This calls create_genshin_dictionary and then creates a table. The function then puts weapondict into the table with four columns, and inserts 25 rows into the table at a time.
- def create_monster_dict(monster_file):
    - Takes the data from monster_file and then puts them into categories of id, name, attack, and rarity - this is done with both files so that they can eventually be compiled into one dictionary. Calls load_json to access the data as a Python object, and then returns a dictionary with all of the information in it - monster_dict, and initializes the dictionary so that the names are the keys and everything else are the values.
- def create_monster_table(cur, conn):
    - This calls create_monster_dict and then creates a table. The function then puts monsterdict into the table with four columns, and inserts 25 rows into the table at a time.

File 2 - processandvisualizedata.py: Processes the data, creates graphs, and writes the calculations txt file.

- def process_acnh_data(db):
    - This function sorts through the variable types and then finds the average of its prices. This function can then find the average price for each variable type, and initializes a dictionary called new_dict where the keys are the variable types and the values are the average prices.
- def create_acnh_graph(new_dict):
    - Takes in the dictionary from the previous function and then sets the keys as the x-axis and the values as the y-axis. This function also sets a color for the graph, names the x and y axes, and a title as the name of the graph.
- def process_monster_data(db):
    - This function will go through the database and find the attack and rarity of each weapon, and then it will find the highest attack for each level of rarity. Then the function makes a dictionary called monster_dict where the keys are the rarities and the values are the attack.
- def create_monster_graph(monster_dict):
    - Takes in the dictionary from the previous function and then sets the keys as the x-axis and the values as the y-axis. This function also sets a color for the graph, names the x and y axes, and a title as the name of the graph.
- def process_genshin_data(db):
    - This function will go through the database and find the base attack and rarity of each weapon, and then it will find the lowest base attack for each level of rarity. Then the function makes a dictionary called genshin_dict where the keys are the rarities and the values are the attack.
- def create_genshin_graph(genshin_dict):

    - Takes in the dictionary from the previous function and then sets the keys as the x-axis and the values as the y-axis. This function also sets a color for the graph, names the x and y axes, and a title as the name of the graph.
- def write_txt(filename, acnh_dict, monster_dict, and genshin_dict):
    - Creates a txt file and writes the information from the dictionaries into this file that were created from the previous functions that processed the data.
- def main():
    - Calls all of the functions to process the data, create graphs, and then write the txt file.

Documentation of Resources

Date	Issue Description	Location of Resource	Result (Did it solve the issue?)
4/19/23	Trouble figuring out how to find the number of rows in a table.	https://www.freecodecamp.org/news/sql-distinct-statement-how-to-query-select-and-count/#:~:text=Conclusion-,In%20SQL%2C%20you%20can%20make%20a%20database%20query%20and%20use,as%20part%20of%20the%20count.	Yes
4/19/23	Finding a way to get all the keys and values from a dictionary.	https://www.tutorialspoint.com/How-to-get-a-list-of-all-the-keys-from-a-Python-dictionary	Yes
4/20/23	Trouble understanding how joining two tables works.	https://www.w3schools.com/sql/sql_join.asp	Yes
4/21/23	Trouble figuring out	https://www.geeksfor	Yes

	how to make a scatterplot.	geeks.org/graph-plotting-in-python-set-1/	