Final Project Report

SI 201

Gamer Soups: Corey Armstrong, Jason Bachi, LooLu Wiltse

https://github.com/csamich/201FP.git

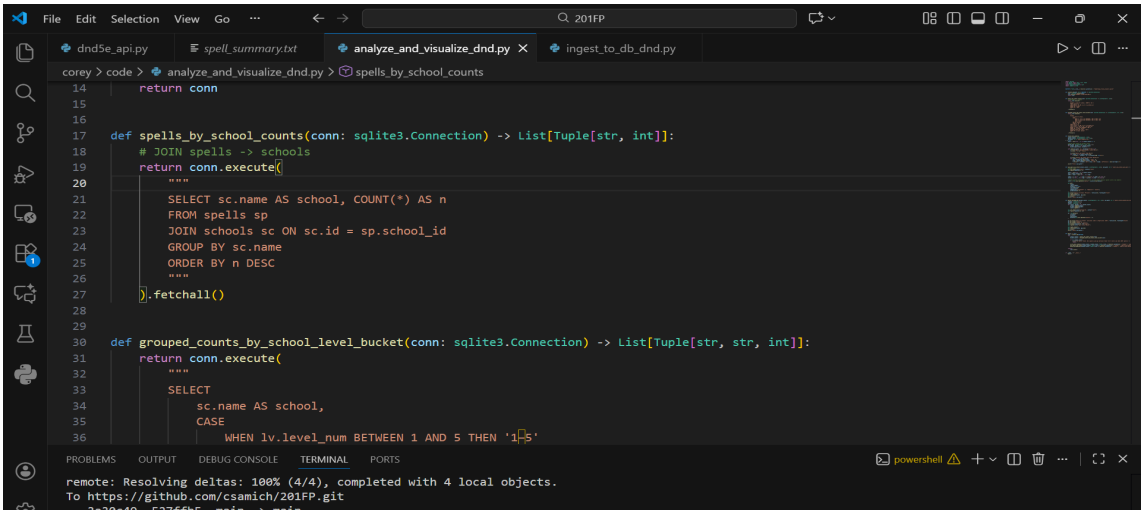
For this report we planned to investigate the prevalence of certain types of gaming and food statistics in our chosen APIs for open5e, fruityvice, pokeapi, and jelly-belly-wiki. We ended up investigating 5e-bits, fruityvice, pokeapi, and jelly-belly-wiki.

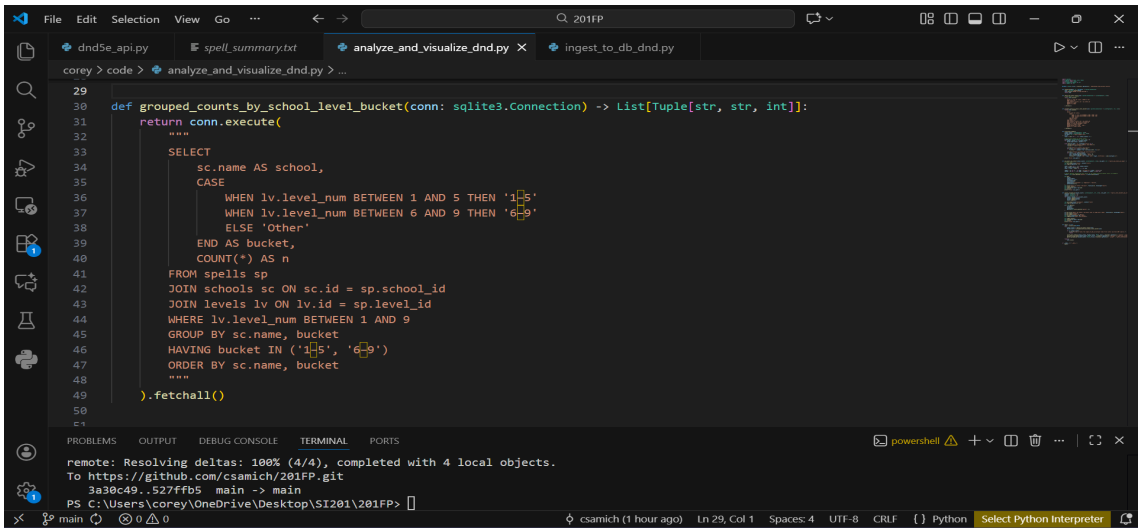
Corey:
Planned Goal: For open5e we were originally going to investigate the distribution of spells according to school of magic and the average range of spells by school of magic. This was to see what the preferred schools of magic and spell ranges were by the creators.

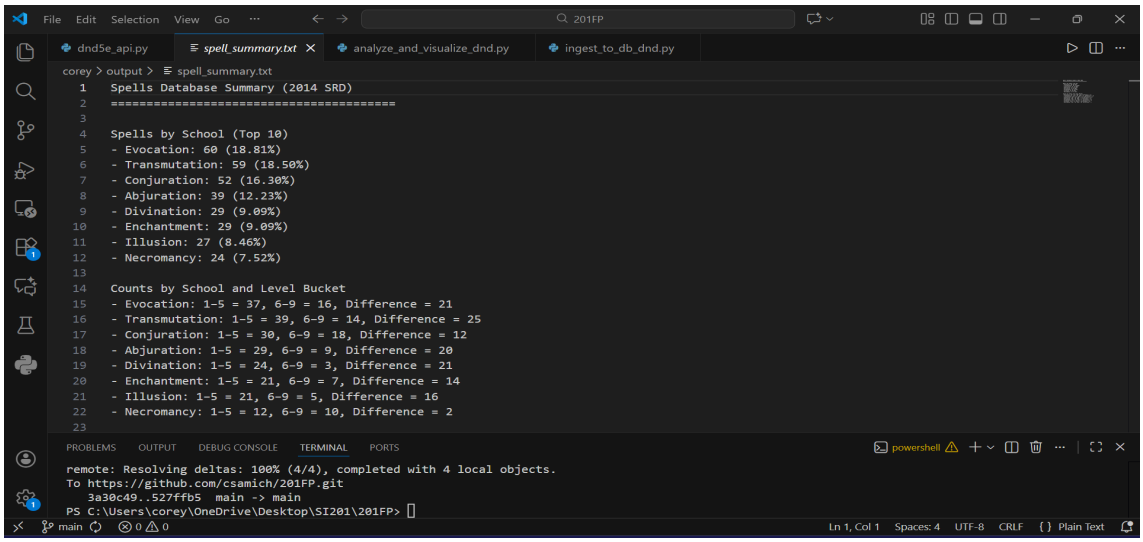
Result Goal: Using 5e-bits, we successfully investigated the distribution of spells according to school of magic and the difference between the number of high-level and low-level spells by school of magic. This was to see what the preferred schools of magic and proliference of spell levels according to the school of magic are. This would highlight which schools of magic are underrepresented and at what levels of play they are most prevalent.

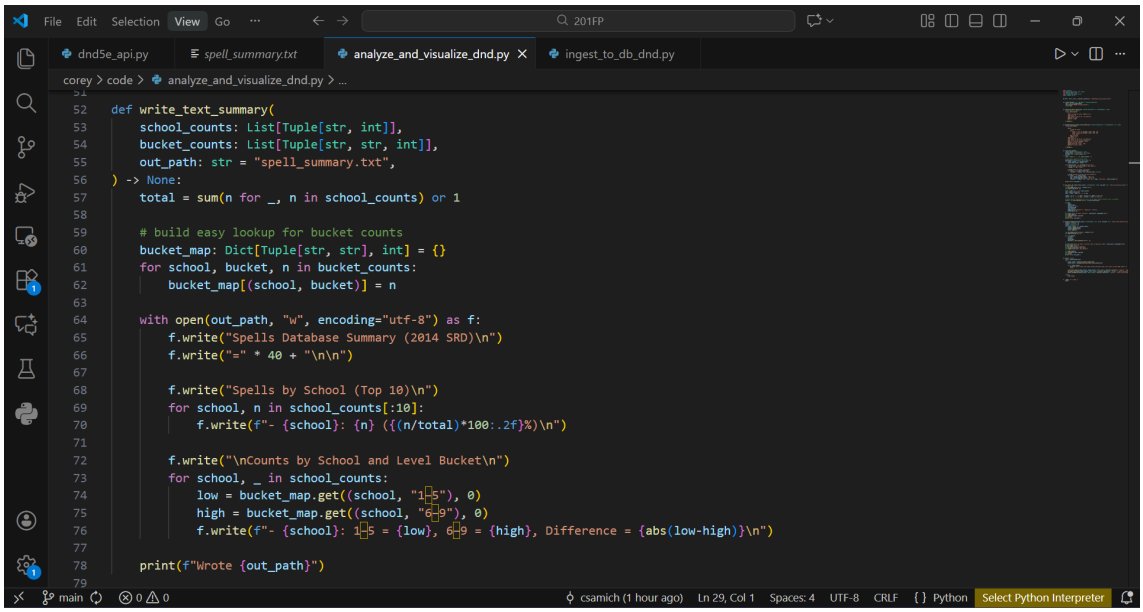
Problems: When investigating I realized that the ranges for spells were written as text and had a wide range of values which would make it more difficult to visualize. So I changed plans and decided to analyze the difference between the number of high and low-level spells for each school of magic instead. Another issue I ran into earlier on was the source material the spells were from. I only wanted to analyze spells from official Wizards of the Coast materials, but open5e had many third party spells integrated. So I decided to use 5e-bits which only relies on the official 5e SRD rules. I also had trouble using Seaborn instead of Matplotlib, so I used the documentation from the web and ChatGPT to help me. I also ran into issues with timeouts when trying to get my data, so I used ChatGPT to help me develop a plan for preventing timeout errors and debug my solutions.

Calculations:

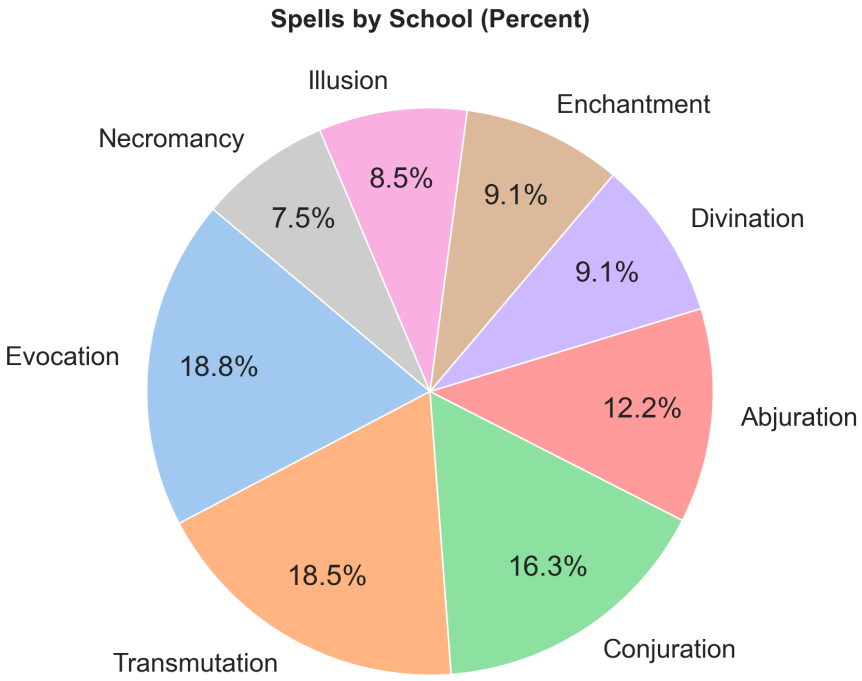


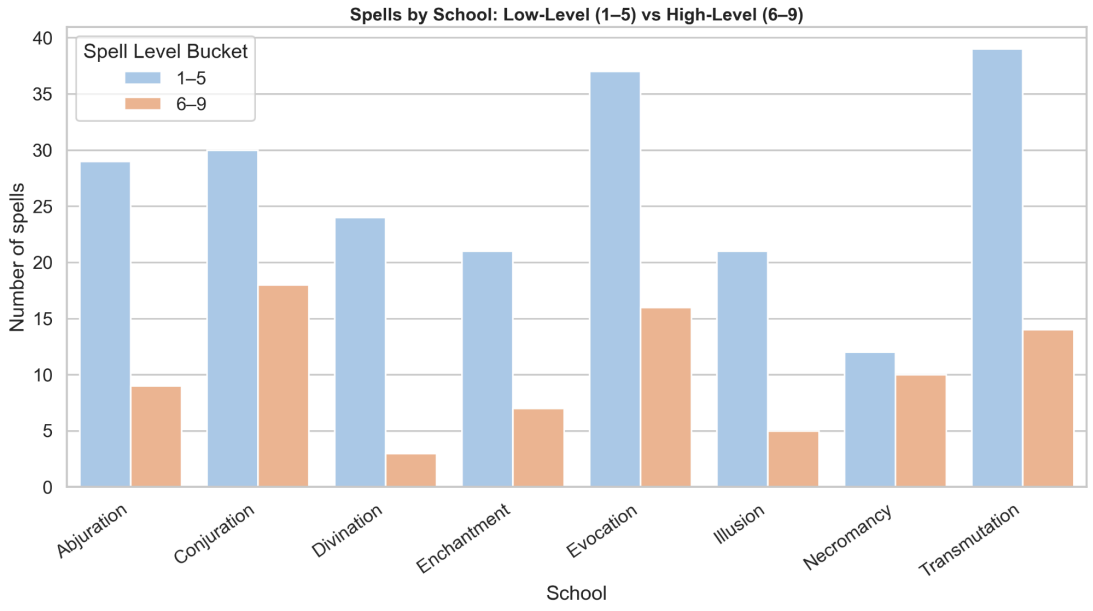

File Edit Selection View Go
201FP
dnd5e_api.py
spell_summary.txt
analyze_and_visualize_dnd.py
ingest_to_db_dnd.py
corey > code > analyze_and_visualize_dnd.py > ...
def grouped_counts_by_school_level_bucket(conn: sqlite3.Connection) -> List[Tuple[str, str, int]]:
return conn.execute(
"""
SELECT
sc.name AS school,
CASE
WHEN lv.level_num BETWEEN 1 AND 5 THEN '1–5'
WHEN lv.level_num BETWEEN 6 AND 9 THEN '6–9'
ELSE 'Other'
END AS bucket,
COUNT(*) AS n
FROM spells sp
JOIN schools sc ON sc.id = sp.school_id
JOIN levels lv ON lv.id = sp.level_id
WHERE lv.level_num BETWEEN 1 AND 9
GROUP BY sc.name, bucket
HAVING bucket IN ('1–5', '6–9')
ORDER BY sc.name, bucket
"""
).fetchall()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/csamich/201FP.git
3a30c49..527ffb5 main -> main
PS C:\Users\corey\OneDrive\Desktop\SI201\201FP>
main
csamich (1 hour ago)
Ln 29, Col 1
Spaces: 4
UTF-8
CRLF
Python
Select Python Interpreter


File Edit Selection View Go
201FP
dnd5e_api.py
spell_summary.txt
analyze_and_visualize_dnd.py
ingest_to_db_dnd.py
corey > output > spell_summary.txt
Spells Database Summary (2014 SRD)
Spells by School (Top 10)
- Evocation: 60 (18.81%)
- Transmutation: 59 (18.50%)
- Conjuration: 52 (16.30%)
- Abjuration: 39 (12.23%)
- Divination: 29 (9.09%)
- Enchantment: 29 (9.09%)
- Illusion: 27 (8.46%)
- Necromancy: 24 (7.52%)
Counts by School and Level Bucket
- Evocation: 1–5 = 37, 6–9 = 16, Difference = 21
- Transmutation: 1–5 = 39, 6–9 = 14, Difference = 25
- Conjuration: 1–5 = 30, 6–9 = 18, Difference = 12
- Abjuration: 1–5 = 29, 6–9 = 9, Difference = 20
- Divination: 1–5 = 24, 6–9 = 3, Difference = 21
- Enchantment: 1–5 = 21, 6–9 = 7, Difference = 14
- Illusion: 1–5 = 21, 6–9 = 5, Difference = 16
- Necromancy: 1–5 = 12, 6–9 = 10, Difference = 2
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/csamich/201FP.git
3a30c49..527ffb5 main -> main
PS C:\Users\corey\OneDrive\Desktop\SI201\201FP>
main
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
Plain Text


File Edit Selection View Go
201FP
dnd5e_api.py
spell_summary.txt
analyze_and_visualize_dnd.py
ingest_to_db_dnd.py
corey > code > analyze_and_visualize_dnd.py > ...
def write_text_summary(
school_counts: List[Tuple[str, int]],
bucket_counts: List[Tuple[str, str, int]],
out_path: str = "spell_summary.txt",
) -> None:
total = sum(n for _, n in school_counts) or 1
# build easy lookup for bucket counts
bucket_map: Dict[Tuple[str, str], int] = {}
for school, bucket, n in bucket_counts:
bucket_map[(school, bucket)] = n
with open(out_path, "w", encoding="utf-8") as f:
f.write("Spells Database Summary (2014 SRD)\n")
f.write("=" * 40 + "\n\n")
f.write("Spells by School (Top 10)\n")
for school, n in school_counts[:10]:
f.write(f"- {school}: {n} ({(n/total)*100:.2f}%)\n")
f.write("\nCounts by School and Level Bucket\n")
for school, _ in school_counts:
low = bucket_map.get((school, "1–5"), 0)
high = bucket_map.get((school, "6–9"), 0)
f.write(f"- {school}: 1–5 = {low}, 6–9 = {high}, Difference = {abs(low-high)}\n")
print(f"Wrote {out_path}")
main
csamich (1 hour ago)
Ln 29, Col 1
Spaces: 4
UTF-8
CRLF
Python
Select Python Interpreter

Visualizations:

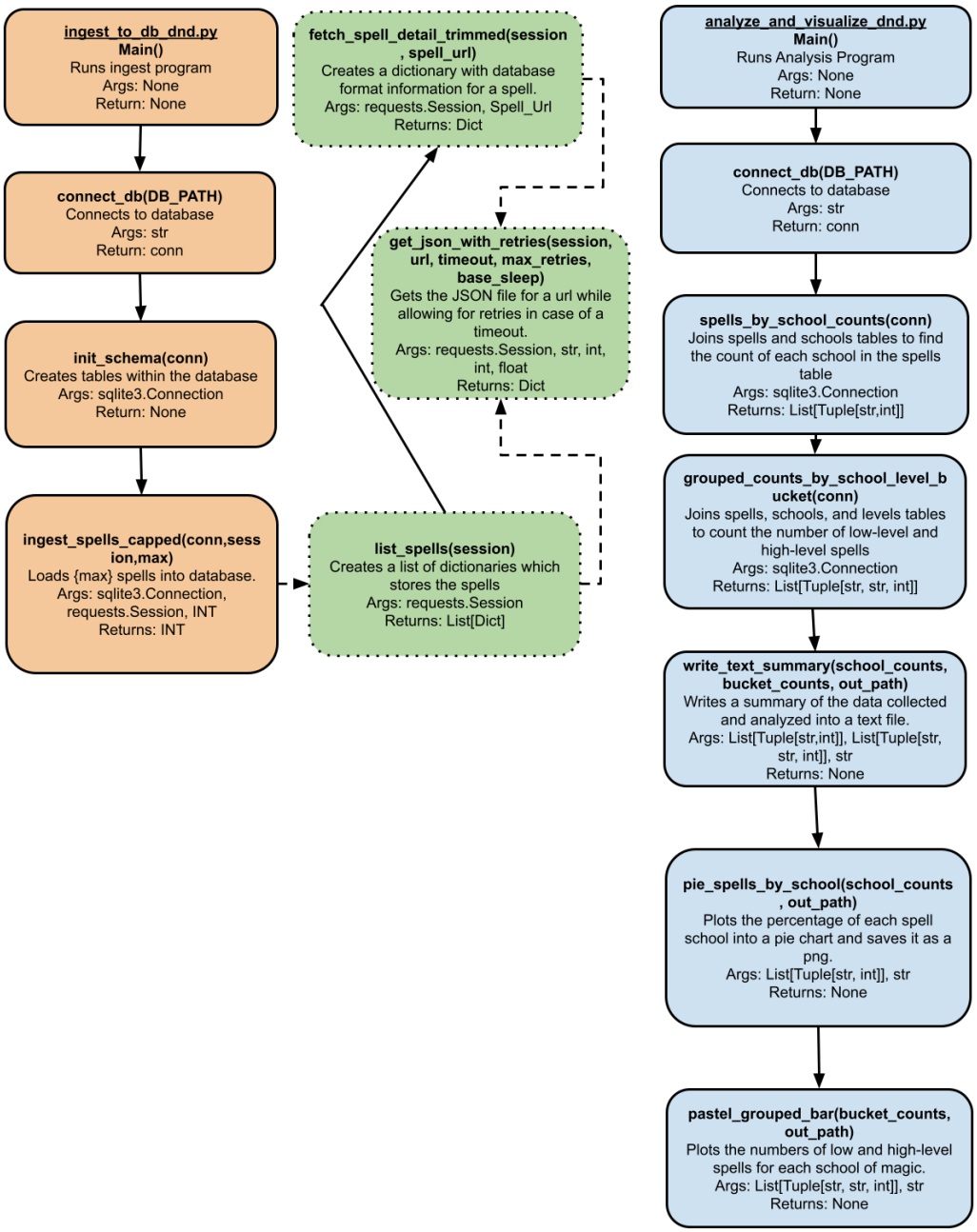




Instructions:
Simply unpack and run my ingest_to_db_dnd.py until it gives 0 NEW spells inserted, and then run analyze_and_visualize_dnd.py to create the visuals. They require 2 parent directories, the

names of which don’t matter, and a subdirectory on the same level as the folder the code is in by the name of “output”.

Function Diagram:



Documentation:

Date	Issue Description	Location of Resource	Result
12/14/25	Parameter for Spell Range is not what I need.	https://api.open5e.com/	Replaced- I chose to use a different parameter after analyzing my options and chose levels.
12/14/25	Timeout errors keep popping up.	ChatGPT	Fixed- I used ChatGPT to help me debug and plan a function to prevent timeout errors
12/14/25	The amount of third-party spells in the data makes it difficult to parse to find official content.	https://5e-bits.github.io/docs/introduction	Replaced- Chose to use 5e-bits instead of open5e since it uses only official material
12/15/25	Trying to change the colors of the plot to be more pleasing	ChatGPT, https://seaborn.pydata.org/tutorial/color_palettes.html	Fixed- Using ChatGPT and the online documentation allowed me to replace matplotlib with seaborn.

Jason: 1. For my portion of the project the plan was to pull data from two APIs: PokeAPI and Fruityvice, and store the results in a shared sqlite database. The goal was to gather pokemon information such as types, weights, and baste stats from PokeAPI, and to gather fruit names and nutrition data from Fruityvice. Using the data the plan was to compute summaries like average Pokemon strength by type and relationships like weight vs total stats and to generate a small set of graphs that highlight the patterns found in the data.

2. I implemented the PokeAPI part so the database stores Pokemon information in a way that lets us connect related facts together. I also got the Fruityvice part working so the database stores fruit names and nutrition information. Since Fruityvice only provides 49 fruits total, I also had to reorganize the nutrition data into a format where each fruit contributes multiple nutrition entries. That way we still meet the requirement of having at least 100 stored items from that source, and it also makes it easier to calculate averages and rankings. For the website requirement and extra credit, I scraped Pokemon data from PokemonDB while using BeautifulSoup and stored those results in the database too. After the data was in the database, I

wrote a script that calculates results and saves them into output files, and another script that creates charts and saves them as image files for the report.

3. One problem I ran into was that Fruityvice's full list is only 49 fruits, which is below the required 100 items for a source. To fix that without making up data, I stored the nutrition information in a one row per fruit per nutrient style table which gives well over 100 real entries derived from the Fruityvice API. Another problem was that some fruit list pages were blocked so I couldn't use them for the website portion. I solved that by switching the website part to PokemonDB which worked and then I scraped a table of Pokemon data from that site. I also made sure that the scripts follow the rule that each run inserts at most 25 new items and that running them again doesn't cause duplicates.

4.

```
(.venv) [jason] ~/201FP/ $ ls -l jason/outputs
total 40
-rw-r--r-- 1 jason jason 13283 Dec 15 18:45 fruit_avg_nutrients_by_family.json
-rw-r--r-- 1 jason jason   800 Dec 15 18:45 fruit_top10_sugar.json
-rw-r--r-- 1 jason jason  1050 Dec 15 18:45 pokemon_avg_total_stats_by_type.json
-rw-r--r-- 1 jason jason  9608 Dec 15 18:45 pokemon_weight_vs_total_stats.json
-rw-r--r-- 1 jason jason   140 Dec 15 18:45 web_primary_type_counts.csv
(.venv) [jason] ~/201FP/ $ head -n 20 jason/outputs/pokemon_avg_total_stats_by_type.json
[
  {
    "type": "fire",
    "avg_total_stats": 429.6666666666667,
    "count": 9
  },
  {
    "type": "water",
    "avg_total_stats": 408.27777777777777,
    "count": 18
  },
  {
    "type": "ghost",
    "avg_total_stats": 405.0,
    "count": 3
  },
  {
    "type": "psychic",
    "avg_total_stats": 404.2,
    "count": 5
```

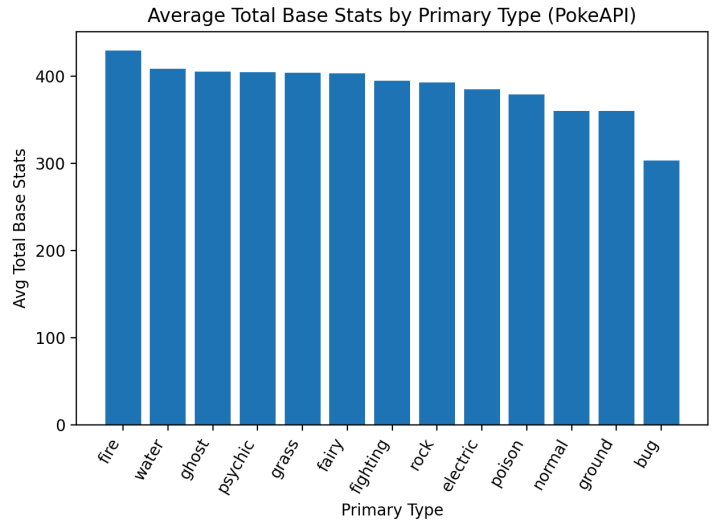
```
(.venv) [jason] ~/201FP/ $ head -n 20 jason/outputs/fruit_top10_sugar.json
[
  {
    "fruit": "Jackfruit",
    "family": "Moraceae",
    "sugar": 19.1
  },
  {
    "fruit": "Persimmon",
    "family": "Ebenaceae",
    "sugar": 18.0
  },
  {
    "fruit": "Banana",
    "family": "Musaceae",
    "sugar": 17.2
  },
  {
    "fruit": "Mangosteen",
    "family": "Clusiaceae",
    "sugar": 16.11
```

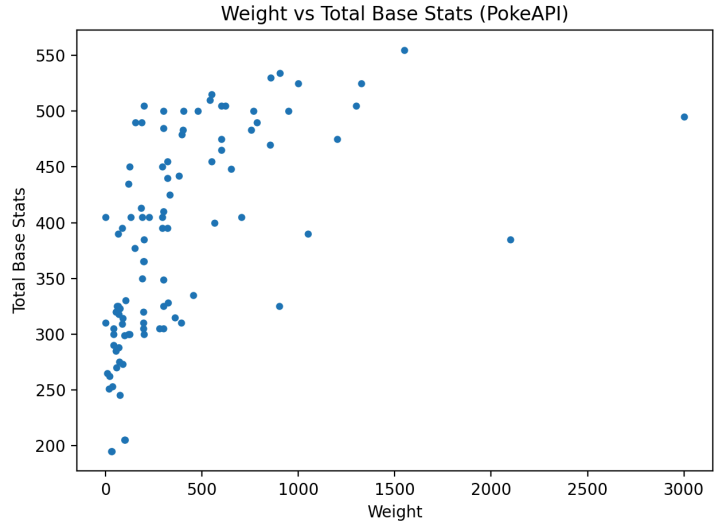
The first screenshot shows the results of running jason/scripts/calc_outputs.py. The first part

lists the generated calculation output files in jason/outputs/. The second part shows pokemon_avg_total_stats_by_type.json, which contains the average total pokemon base stats for each primary type, calculated from the sqlite database by summing base stats and connecting each Pokemon to its primary type.

The second screenshot shows fruit_top10_sugar.json which lists the top fruits ranked by sugar value. The ranking was computed from fruityvice nutrition data stored in the database, combined with fruit and family information.

5.


Average Total Base Stats by Primary Type (PokeAPI)
Avg Total Base Stats
0
100
200
300
400
fire
water
ghost
psychic
grass
fairy
fighting
rock
electric
poison
normal
ground
bug
Primary Type


Weight vs Total Base Stats (PokeAPI)
Total Base Stats
200
250
300
350
400
450
500
550
0
500
1000
1500
2000
2500
3000
Weight

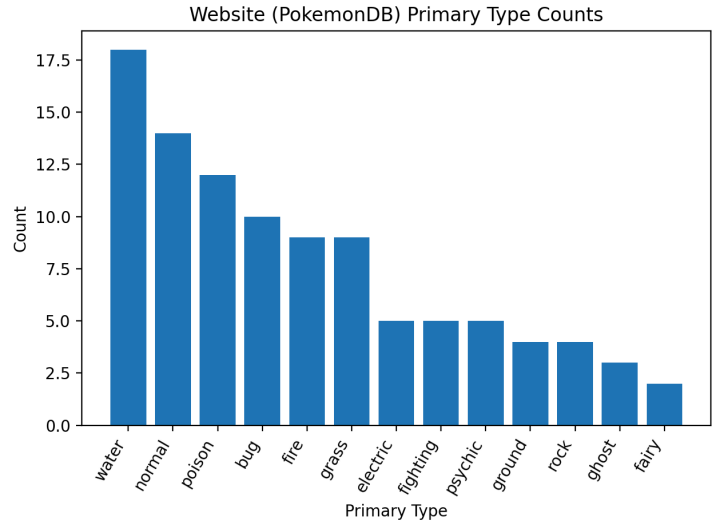

Website (PokemonDB) Primary Type Counts
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
water
normal
poison
bug
fire
grass
electric
fighting
psychic
ground
rock
ghost
fairy
Primary Type

Image 1 is a bar chart which shows the average total base stats for Pokemon grouped by their primary type, using PokeAPI data stored in our sqlite database. It summarizes which primary types tend to have higher or lower overall base stats on average (fire and water higher in this sample, while bug is lower).

Image 2 is a scatter plot which compares each Pokemon's weight to its total base stats using Poke API data from the database. It shows the overall relationship between weight and strength in this sample, with most points clustering at lower weights with a few heavier outliers, rather than a strict linear trend.

Image 3 is a bar chart which shows how many Pokemon in the scraped PokemonDB dataset fall into each primary type. It summarizes which primary types appear most frequently in the subset we collected (water and normal appear most often in this sample).

6. Run these from the repo root with required packages installed (requests, beautifulsoup4, pandas, matplotlib):

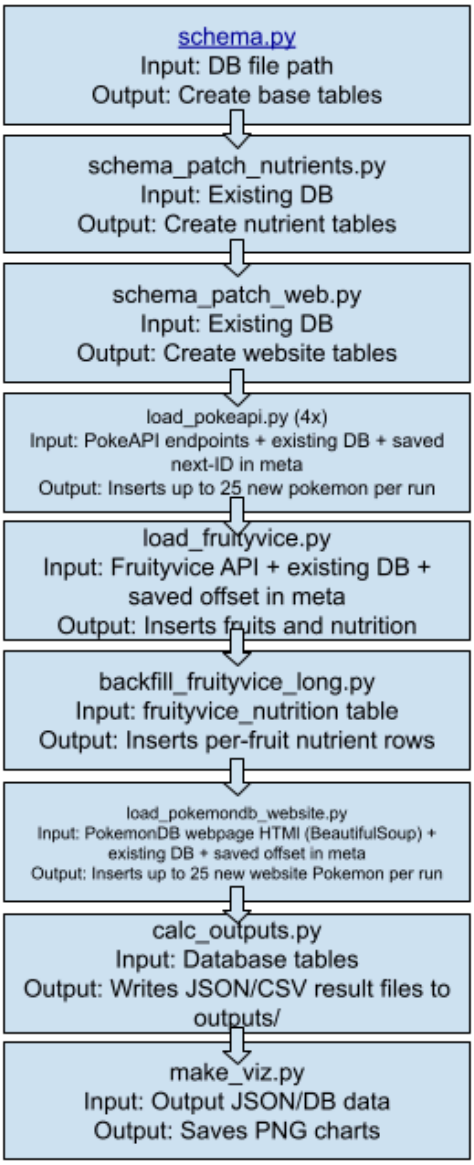
Create/ensure tables exist: python jason/scripts/schema.py
python jason/scripts/schema_patch_nutritients.py
python jason/scripts/schema_patch_web.py

Load data for PokeAPI: python jason/scripts/load_pokeapi.py (4x)
For Fruityvice: python jason/scripts/load_fruityvice.py (4x)
Create long format nutrient rows (to reach 100+ rows for Fruityvice): python jason/scripts/backfill_fruityvice_long.py
PokemonDB website scrape (4x): python jason/scripts/load_pokemondb_website.py

Run calculations: python jason/scripts/calc_outputs.py

Generate visualizations: python jason/scripts/make_viz.py

7.



8.

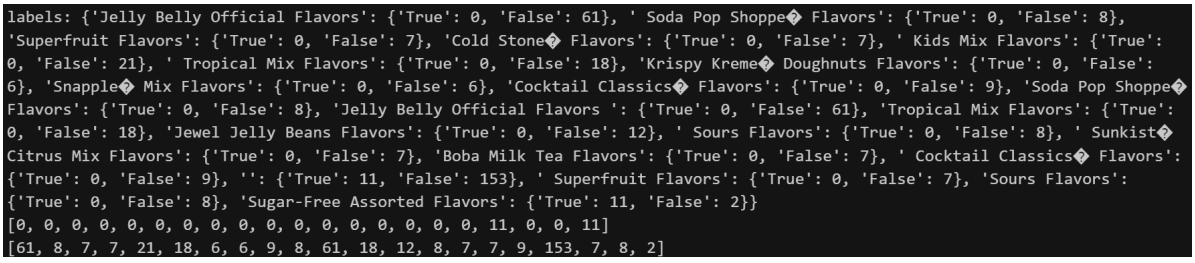
Date	Issue Description	Location of Resource	Result (did it solve the issue?)
Dec 15	Needed PokeAPI endpoint format and fields	PokeAPI documentation (pokeapi.co)	Yes - Confirmed endpoints/JSON fields and implemented PokeAPI loading

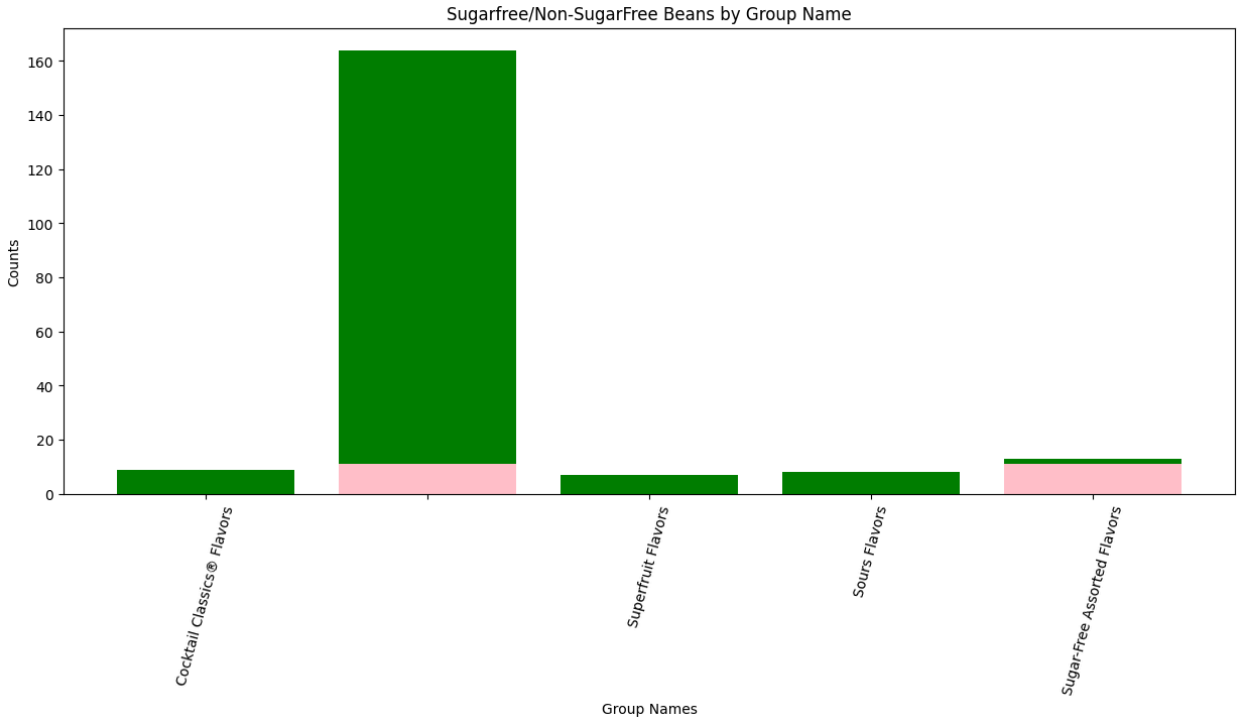
Dec 15	Needed Fruityvice endpoint and nutrition field names	Fruityvice API documentation (fruityvice.com)	Yes- Implemented Fruityvice loading and stored nutrition data correctly
Dec 15	Website source needed for BeautifulSoup scraping and 100+ items	PokemonDB “Pokedex All” page (pokemondb.net/pokedex/all)	Yes - Scraped table data with BeautifulSoup and inserted into DB
Dec 15	Designing SQLite schema, 25-per-run inserts, JOIN queries, and plot generation	ChatGPT	Yes - Helped structure/debug scripts and queries, final code was tested locally and produced outputs/plots
Dec 15	Verifying SQL JOIN outputs / checking table counts	DB Browser for SQLite	Yes - Verified tables, row counts, and JOIN-based results matched expectations

Bonus A: We used more than the minimum number of APIs, and we also included a BeautifulSoup website source (PokemonDB) so our extra-credit work was not API-only. This includes 100+ items and the calculations as well as the outputs.

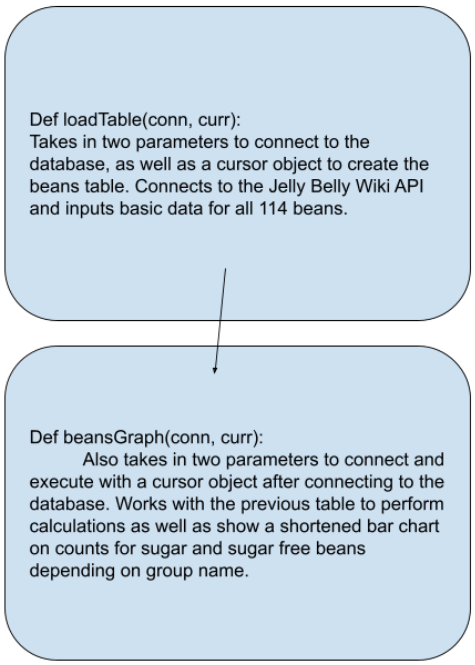
Bonus B: We included at least two additional visualizations beyond the minimum required, and the extra charts are included in the report as image files.

LooLu:

1. For my portion of the project, I simply decided to work with just one API, that being the Jelly Belly Wiki created by Moses Atia Poston. Ultimately, I changed my goal to try to break down all the beans in the API by their “group name,” being those like classical flavors, special edition flavors, sugar free flavors, etc. I ultimately decided to divide them into counting the amount of beans per flavor, dividing them into two categories being sugarfree and non-sugarfree beans. To visualize this, I made use of Matplotlib to create a stacked bar chart to show a visual distribution of all the counts of beans divided by group name and further into their sugarfree status.
2. I essentially ended up revamping what I wrote about calculating in my initial plan, and I improvised and tried to stick with simply calculating the number of beans that are sugar free and not sugar free, and counting them for all the named categories that they belong to as labeled in the Jelly Belly Wiki API.
3. The core problem I encountered was with time management. I started my portion of the project quite late and scrambled to create a table and perform a calculation with the data. Another major problem I also encountered was trying to format my graph to be visually readable. I still need to work on scaling and other proportions in order to make it not appear so visually spread out. In order to fix this, I decided to mainly create a final graph that includes the last five group names, as they do have some more visually-interesting data as to the proportions of sugar free to non-sugar free beans. One category included beans that did not have any specific label, so it goes into a “nameless” group name.
4. In the screenshot below, this is the raw output of a nested dictionary I created called “labels,” as well as one list of counted sugar free beans, and the last list is the counts for the sugar beans. The nested dictionary essentially breaks down all the beans into their group names if they have one, and then it further counts the beans into all the groups that they belong to, depending on whether they are sugar free or not.
    a. 



5.
6. Instructions: Essentially download and run .py file SI201FinalProjectAttempt.py, as well as with the required packages imported.
7. I wrote all the code in SI201FinalProjectAttempt.py, including the below two functions of which it is mostly comprised of:



8.

Date	Issue Description	Location of Resource	Result
12/14	I needed help figuring out how to set up the API so that I could access it remotely for the project.	https://jelly-belly-wiki.netlify.app/api	Fixed - helped explain how to load the API and make use of certain endpoints and on how to access the data
12/14 through 12/15	I needed help on debugging, formatting, and explanations on how to correctly implement my code, also helped explain the project instructions to me in different wording	UM-GPT	Fixed - helped explain to me how to debug and how to format some of my figure and plots
12/15	I needed to use a quick resource to help me display my plot, as well as brief debugging tips using the built-in debugging explanation tools.	Google Colab	Fixed - was able to lively update my graph and format it to make it somewhat visible
12/14 to 12/15	I needed help for refreshers and brief explanations on the formatting of SQL as a reminder, and also for explanations on other concepts I needed refreshers on	Google's AI Overview in search results	Fixed - was able to smoothly use SQLite and was reminded on how to format the language used to work with table creation

9.