1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

We planned to use OpenWeatherMap API, (https://openweathermap.org/api) AirVisual API (https://www.iqair.com/air-pollution-data-api), and TMDb (https://www.themoviedb.org/). From the WeatherMap API, we planned to collect Current Weather Data, ZIP codes within a state, .From the AirVisual API, Hourly Forecast 4 days, we planned to collect data about city names, coordinates, real-time Air Quality Index (AQI), and real-time weather data. From the TMDb API, we planned to collect movie titles,movie popularity, ratings, release date, revenue and genre of the 100 most recent movies. However, two of these APIS required payment for data collection over time and were not paginated, so we had to re-evaluate and replace some of them.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

We did use the TMDb API, and from this API we collected data such as titles, movie popularity, ratings, release date, revenue and genre of the 100 most recent movies. We used the genre data to create a new table so that we could identify each movie's genre without repeating string data in the table.

We used the PokeAPI, and from this API we collected data on the first 100 pokemon in the database including their pokedex number (id), name, height, weight, type, attack, defense, and speed. We used the type data to create a new table so that we could identify each Pokemon's type without repeating string data.

The last API we used was the Dogs API from API Ninja, and from this api we collected data on 100 dog breeds including the breed name, energy level (on a scale of 0-5), barking level (scale of 0-5), weight, and height.

3. The problems that you faced (10 points)

We faced a few issues. At first, we had trouble finding APIs to use that were both free and paginated. We also ran into the issue with Pokemon where type was string data that repeated so we had to create a new table and assign each type a numerical identifier to not violate that rule. We also realized on the day of presentations that we had misunderstood the instructions. Our main.py had a for loop to only call 20 items from each table at a time. We did not realize that we were meant to have to run the file multiple times so that only a max of 25 items were taken on each run. Changing that file logic proved to be difficult for us as we had to retool different parts of the code to fit that structure.

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

calc.txt

```
Average Rating of Each Genre

Average rating for Action films:  6.3
Average rating for Adventure films:  6.4
Average rating for Animation films:  7.3
Average rating for Comedy films:  6.7
Average rating for Crime films:  5.9
Average rating for Documentary films:  5.8
Average rating for Drama films:  6.3
Average rating for Family films:  7.0
Average rating for Fantasy films:  6.9
Average rating for Horror films:  6.5
Average rating for Music films:  7.2
Average rating for Science Fiction films:  7.0
Average rating for Thriller films:  6.6
Average rating for War films:  7.1


Average Stats for Each Pokemon Type

Average stats for Bug pokemon: Attack 53.00, Defense 50.50, Speed 49.50
Average stats for Electric pokemon: Attack 54.00, Defense 62.00, Speed 83.00
Average stats for Fairy pokemon: Attack 57.50, Defense 60.50, Speed 47.50
Average stats for Fighting pokemon: Attack 99.00, Defense 59.00, Speed 60.00
Average stats for Fire pokemon: Attack 75.78, Defense 60.44, Speed 84.44
Average stats for Ghost pokemon: Attack 50.00, Defense 45.00, Speed 95.00
Average stats for Grass pokemon: Attack 73.11, Defense 61.67, Speed 52.22
Average stats for Ground pokemon: Attack 82.50, Defense 67.50, Speed 80.00
Average stats for Normal pokemon: Attack 70.50, Defense 49.29, Speed 77.29
Average stats for Poison pokemon: Attack 74.75, Defense 60.25, Speed 60.67
Average stats for Psychic pokemon: Attack 45.20, Defense 41.00, Speed 84.80
Average stats for Rock pokemon: Attack 85.00, Defense 126.25, Speed 42.50
Average stats for Water pokemon: Attack 72.11, Defense 81.44, Speed 63.00


Average Weight for Each Dog Energy Levl

Average weight for dogs with energy 0: 73.75
Average weight for dogs with energy 2: 157.50
Average weight for dogs with energy 3: 106.33
Average weight for dogs with energy 4: 80.20
Average weight for dogs with energy 5: 75.91

```

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

https://imgur.com/a/Y6hIuEZ

6. Instructions for running your code (10 points)

Running our code is very simple.

To write all the data to the database, you first need to ensure that database.sqlite3 is deleted. Then, you continually run the main.py file until it outputs “no data to retrieve.” In those 17 runs, 15 of those are adding 20 items to each respective main table (pokemon, dogs, and

movies in that order), and 2 of those are creating the type and genre tables, which store the ID #s for each dataset, so there is no repeating string data.

To output our calculations and visualizations, you simply run main2.py once. If calc.txt does not yet exist, it will create the file and output our chosen calculations there, and then output all 5 visualizations. If calc.txt already exists when main2.py is run, then it will just output all 5 calculations.

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

File Name	Function Name	Input	Output	Description
config.py	getdb()	None	Database connection	It sets up a connection to the SQL database and returns it.
config.py	closedb()	conn	none	It commits the changes to the database and closes the connection.
pokemon.py	getpokemon()	cur, st, end	Prints a message that lets users know how many pokemon were added to the table.	Takes in the starting Pokemon id #, and the ending Pokemon id# and loops through the range of all the Pokemon id #s to send a request to the Poke API for all the relevant data for each Pokemon. Then it prints a message indicating how many pokemon were added.
pokemon.py	gettypes()	cur	Prints a message that lets users know the table was created.	The function executes a SQL “INSERT” statement on the type table passing in the ID and name of the type using the TYPES dictionary.
dog.py	get_dogs()	cur, offset	Prints a message that lets users know how many dogs were added to the table.	It takes in the minimum height and the offset (which tells the dog function how many entries it needs to skip). It returns the data as a json object and then inserts it into the SQL database using the cursor item. Then it prints a message indicating how many items were added to the table.
movie.py	moviedata()	cur, page	Prints a message that lets	It uses the API of themoviedb.org to get the data of popular movies and store them in a database. It sends a

			users know how many movies were added to the table.	GET request to the API (specifying which page it wants data from with the page input) and receives the response in JSON format. It then inserts 20 movies at a time into the SQL database using the cursor item. Then it prints a message to indicate that 20 movies have been added.
movie.py	get_genres()	cur	Prints a message that lets users know the genre table was created.	This function extracts the genre data from the JSON response and uses a for loop to insert each genre into the genres table in the database, with the genre ID and name as column values.
main.py	main()	None	None	It opens a connection to a SQLite database then creates a cursor object so that the SQL queries can be executed. It opens the schema.sql file to create the necessary tables then executes all the SQL queries at once. It calls the helper() function to populate the database with data then closes the database connection.
main.py	helper()	cur	Prints message “No data to retrieve” (Under certain conditions)	It calculates the total number of rows already in each relevant table (pokemon, dogs, and movie) then divides that number by 20 to see which page number it needs to start collecting data from to insert into the table. It also inserts data into the type and genres table after their respective tables are full. When every table is full, it prints “No data left to retrieve”
main2.py	avg_rating_by_genre()	None	Calculation file	Creates a file that contains calculations for the average rating by genre of movie.
main2.py	avg_stats_by_type()	None	Calculation file	Calculates the average stats(attack, defense, and speed) for each pokemon type and creates a file representing these results.

main2.py	avg_weight_by_energy()	None	Calculation file	Makes a file that contains the calculations for the average weights for each dog's energy level.
main2.py	pokemon_pie()	None	Pie chart of Pokemon types	Retrieves the pokemon types from the database and uses the colors for each type to create a pie chart that represents the pokemon types.
main2.py	pokemon_radar()	None	Radar plot of average Attack, Defense, and Speed stats for each Pokemon type	Retrieves the average attack, defense, and speed stats of each Pokmeon type and uses the color for each type to create a radar plot displaying the data.
main2.py	genre_rat()	None	Bar graph of the average movie rating by movie genre	Creates a bar graph of the average movie rating by movie genre by extracting the genre and rating data from the database and using the corresponding genre colors to make the graph.
main2.py	pop_vs_rating()	None	Scatter plot of movie popularity by rating	Takes the movie genre and movie popularity (and corresponding color) to create a scatter plot that represents the correlation between movie rating and movie popularity.
main2.py	weight_vs_energy()	None	Boxplot of dog weights by energy level.	Creates a box plot that represents dog weights by energy level (by using the data from the database).

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Problem	Resource	Result
4/8	Python how to install requests with pip because it was	https://www.activestate.com/resources/quick-reads/how-to-pip-install-requests-python-package/	Yes, this did solve our problem. The directions were relatively

	broken and matplotlib with pip because it was new	https://matplotlib.org/stable/users/installing/index.html	straightforward for both installations
4/8	Downloading WSL Ubuntu for windows to make the Apple to Windows coding crossover easier	https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-10#1-overview https://apps.microsoft.com/store/detail/ubuntu/9PDXGNCFSCZV	Yes, this solved the problem and again was relatively straightforward.
4/11	auto increment used in SQLite (needed a way to auto increment the values in the table)	https://www.tutorialspoint.com/sqlite/sqlite_using_autoincrement.htm#:~:text=SQLite%20AUTOINCREMENT%20is%20a%20keyword,used%20with%20INTEGER%20field%20only. https://database.guide/how-autoincrement-works-in-sqlite/ https://www.geeksforgeeks.org/sql-auto-increment/	Yes, this did work eventually, but it took using a few resources in order to fully implement it correctly.
4/15	How offsets works for APIS to make sure we get the correct values when going through the API's pages	https://www.algolia.com/doc/api-reference/api-parameters/offset/#:~:text=Offset%20is%20the%20position%20in,retrieve%20starting%20from%20the%20offset. https://developer.box.com/guides/api-calls/pagination/offset-based/ https://developer.digitalchalk.com/document/rest-api-v5/limit-and-offset/	Yes, through trial and error with testing the actual code and these resources, we were able to get offset to accurately work.